49th EATCS International Conference on Automata, Languages, and Programming

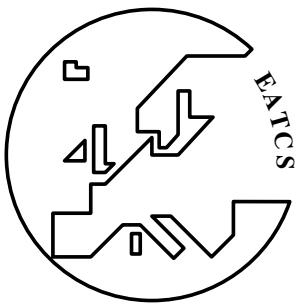
ICALP 2022, July 4–8, 2022, Paris, France

Edited by

Mikołaj Bojańczyk

Emanuela Merelli

David P. Woodruff



Editors

Mikołaj Bojańczyk
University of Warsaw, Poland
bojan@mimuw.edu.pl

Emanuela Merelli
University of Camerino, Italy
emanuela.merelli@unicam.it

David P. Woodruff
Carnegie Mellon University, PA, USA
dwoodruf@andrew.cmu.edu

ACM Classification 2012
Theory of Computation

ISBN 978-3-95977-235-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-235-8.

Publication date
July, 2022

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.ICALP.2022.0

ISBN 978-3-95977-235-8 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Track B: Automata, Logic, Semantics, and Theory of Programming

Preface

This volume contains the papers presented at the *49th EATCS International Conference on Automata, Languages and Programming (**ICALP 2022**)*, held *hybrid* in Paris France, during July 4–8, 2022. ICALP is a series of annual conferences of the *European Association for Theoretical Computer Science (EATCS)*, which first took place in 1972.

This year, the ICALP program consisted of two tracks:

- Track A: Algorithms, Complexity, and Games
- Track B: Automata, Logic, Semantics, and Theory of Programming

In response to the call for papers, a total of 433 submissions were received: 350 for Track A and 83 for Track B. Each submission was assigned to at least three program committee members, aided by 641 external subreviewers or experts providing a quick opinion. The committees decided to accept 127 papers for inclusion in the scientific program: 103 papers for Track A and 24 for Track B. The selection was made by the program committees based on originality, quality, and relevance to theoretical computer science. The quality of the manuscripts was very high indeed, and many deserving papers could not be selected.

The EATCS sponsored awards for both a best paper and a best student paper in each of the two tracks, selected by the program committees.

The **best paper awards** were given to the following papers:

Track A: Ian Newman and Nithin Varma. *Strongly Sublinear Algorithms for Testing Pattern Freeness.*

Track B: Jakub Gajarský, Michał Pilipczuk, Wojciech Przybyszewski and Szymon Toruńczyk. *Twin-width and types.*

The **best student paper awards**, for papers that are solely authored by students, were given to the following papers:

Track A: Joakim Blikstad. *Sublinear-round Parallel Matroid Intersection*, and Jakub Tětek. *Approximate Triangle Counting via Sampling and Fast Matrix Multiplication.*

Track B: Gaëtan Douéneau-Tabot. *Hiding pebbles when the output alphabet is unary.*

Apart from the contributed talks, ICALP 2022 included invited presentations by Leslie Ann Goldberg (Professor of Computer Science at the University of Oxford), Madhu Sudan (Gordon MacKay Professor of Computer Science at Harvard University), Albert Atserias (Professor at the Universitat Politècnica de Catalunya), Constantinos Daskalakis (Professor in MIT's Electrical Engineering and Computer Science Department), Stéphan Thomassé (Professor at the Computer Science Department at Ecole Normale Supérieure de Lyon), Santosh Vempala (Frederick Storey Chair in Computing and Professor at Georgia Tech).

This volume contains all the contributed papers presented at the conference, and an abstract or paper accompanying each of the invited talks by Albert Atserias, Constantinos Daskalakis, Leslie Ann Goldberg, Madhu Sudan, Stéphan Thomassé, and Santosh Vempala.

The program of ICALP 2022 also included presentations of the EATCS Award 2022 to Patrick Cousot, the Alonzo Church Award 2022 to Dexter Kozen, the Presburger Award 2022 to Dor Minzer, the EATCS Distinguished Dissertation Awards 2022 to Alexandros Hollender, Jason Li and Jan van den Brand, as well as the announcement of new EATCS Fellows Samson Abramsky and Orna Kupferman.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The following workshops were held as satellite events of ICALP 2022 on July 4, 2022:

- Parameterized Approximation Algorithms (PAAW)
- Combinatorial Reconfiguration
- Recent Advances on Total Search Problems
- LearnAut: 4th edition of the Learning and Automata
- Algorithmic Aspects of Temporal Graphs V
- Trends in Arithmetic Theories
- Structure Meets Power 2022
- Straight-Line Programs, Word Equations and their Interplay
- Graph Width Parameters: from Structure to Algorithms (GWP 2022)

We wish to thank all authors who submitted extended abstracts for consideration, the program committees for their scholarly effort, and all the referees who assisted the program committees in the evaluation process.

We are also grateful to the Conference General Chair, Thomas Colcombet, and his colleagues from the Research Institute on the Foundations of Computer Science, Université Paris Cité, and Fondation Sciences Mathématiques de Paris, for organizing ICALP 2022, and to CNRS, Inria, and Nomadic Lab. for sponsorships.

We would like to thank Anca Muscholl, the Chair of the ICALP Steering Committee, for her continuous support and Artur Czumaj, the president of EATCS, for his generous advice on the organization of the conference.

July 2022

Mikołaj Bojańczyk
Emanuela Merelli
David P. Woodruff

Organization

Program Committees

Track A

David Woodruff	CMU, (chair)
Petra Berenbrink	University of Hamburg
Sergio Cabello	University of Ljubljana
Yixin Cao	Hong Kong Polytechnic University
Sitan Chen	University of California Berkeley
Xi Chen	Columbia University
Ilias Diakonikolas	University of Wisconsin-Madison
David Doty	University of California Davis
Yuval Filmus	Technion
Cyril Gavoille	Université de Bordeaux
Sevag Gharibian	Paderborn University
Seth Gilbert	National University of Singapore
Nick Gravin	Shanghai University of Finance and Economics
Kasper Green Larsen	Aarhus University
Abhradeep Guha Thakurta	Google Research
Hamed Hatami	McGill University
Sandy Irani	University of California Irvine
Yuval Ishai	Technion
Aayush Jain	NTT Research/CMU
Ken-ichi Kawarabayashi	National Institute of Informatics
Yuqing Kong	Peking University
Michal Koucký	Charles University
Stefano Leonardi	Sapienza Universita di Roma
Nutan Limaye	IT University of Copenhagen
Frederic Magniez	CNRS
Audra Mcmillan	Apple
Slobodan Mitrovic	MIT / University of California Davis
Wolfgang Mulzer	Freie Universitat Berlin
Cameron Musco	University of Massachusetts Amherst
Anand Natarajan	MIT
Jelani Nelson	University of California Berkeley
Debmalya Panigrahi	Duke University
Richard Peng	Georgia Tech
Vijaya Ramachandran	University of Texas at Austin
Saket Saurabh	Institute of Mathematical Sciences, Chennai
Christian Sohler	University of Cologne
Thomas Steinke	Google Research
Vasilis Syrgkanis	Microsoft Research
Emanuele Viola	Northeastern University
Adrian Vladu	CNRS
Jan Vondrak	Stanford
Hoeteck Wee	NTT Research / ENS
Christian Wulf-Nilsen	University of Copenhagen

Track B

Mikołaj Bojańczyk	University of Warsaw,(chair)
Luca Aceto	Reykjavik University
Isolde Adler	University of Leeds
Antoine Amarilli	Télécom Paris
Pablo Barcelo	Catholic University of Chile
Libor Barto	Charles University
Laura Ciobanu	Heriot-Watt University
Erich Grädel	RWTH Aachen University
Christoph Haase	University of Oxford
Marcin Jurdziński	University of Warwick
Benjamin Kaminski	Saarland University
Joost-Pieter Katoen	RWTH Aachen University
Bartek Klin	University of Oxford
Naoki Kobayashi	University of Tokyo
Dexter Kozen	Cornell University
Orna Kupferman	Hebrew University
Jérôme Leroux	CNRS / University of Bordeaux
Nathan Lhote	Aix-Marseille University
Markus Lohrey	University of Siegen
Joël Ouaknine	Max Planck Institute
Prakash Panangaden	McGill University
Michael Pinsker	Vienna University of Technology
Sven Schewe	University of Liverpool
Jeffrey Shallit	University of Waterloo
Mahsa Shirmohammadi	CNRS / University of Paris
Sebastian Siebertz	University of Bremen
Alex Simpson	University of Ljubljana
Lidia Tendera	University of Opole

Organizing Committee

Thomas Colcombet, IRIF, Chair

Sandrine Cadet	Olivier Carton	Geoffroy Couteau
Hugo Férée	Irène Guessarian	Natalia Hacquart
Florian Horn	Maximilien Lesellier	Simon Mauras
Valia Mitsou	Sylvain Perifel	Amaury Pouly
Arnaud Sangnier	Sylvain Schmitz	Mahsa Shirmohammadi
Laurent Viennot		

Steering Committee

Nikhil Bansal	University of Michigan, US
Artur Czumaj	Warwick University, UK
Javier Esparza	TUM Munich, Germany
Simon Gay	University of Glasgow, UK
Leslie Ann Goldberg	Oxford University, UK
Thore Husfeldt	Lund University, Sweden & IT University of Copenhagen, Denmark
Giuseppe Italiano	Luiss University, Italy
Emanuela Merelli	University of Camerino, Italy
Anca Muscholl	Bordeaux University, France, Steering Committee Chair
Yuval Rabani	Hebrew University, Israel
Paul Spirakis	University of Liverpool, UK and University of Patras, Greece
James Worrell	University of Oxford, UK

Financial Sponsors

IRIF, CNRS, Université Paris Cité

Additional Reviewers

Anders Aamand
Scott Aaronson
Amir Abboud
Dorna Abdolazimi
Antonis Achilleos
Nidia Obscura Acosta
Raghavendra Addanki
Foto Afrati
Peyman Afshani
Saba Ahmadi
Susanne Albers
Maryam Aliakbarpour
Josh Alman
Helmut Alt
Saeed Akhoondian Amiri
Hyung-Chan An
Alexandr Andoni
Anurag Anshu
Antonios Antoniadis
Simon Apers
Eugene Asarin
Albert Atserias
Nuttapong Attrapadung
Kyriakos Axiotis
Yossi Azar
Arturs Backurs
Saikrishna Badrinarayanan
Mitali Bafna
Mirza Galib Anwarul Husain Baig
Ainesh Bakshi
Nikhil Balaji
Eric Balkanski
Nikhil Bansal
James Bartusek
Sanjoy Baruah
Mohammadhossein Bateni
Jatin Batra
Tugkan Batu
Kevin Batz
Aleksandrs Belovs
Shalev Ben-David
Omri Ben-Eliezer
Michael A. Bender
Benjamin Aram Berendsohn
Helena Bergold
Benjamin Bergougnoux
Olaf Beyersdorff
Siddharth Bhandari
Vishwas Bhargava
Aditya Bhaskara
Hadley Black
Eric Blais
Antonio Blanca
Jannis Blauth
Markus Blumenstock
Hans L. Bodlaender
Greg Bodwin
Niclas Boehmer
Andrej Bogdanov
Martin Böhm
Fritz Bokler
Édouard Bonnet
Michaela Borzechowski
Vitor Bosshard
Olivier Bournez
Florian Bourse
Nicolas Bousquet
Joshua Brakensiek
Marco Bressan
Gavin Brown
Nader Bshouty
Jaroslaw Byrka
Clément Canonne
Ioannis Caragiannis
Charlie Carlson
Marco Carmosino
Arturo Carpi
Margarida Carvalho
Valentina Castiglioni
Matteo Ceccarello
Ruoxu Cen
Deeparnab Chakrabarty
Diptarka Chakraborty
T-H. Hubert Chan
Timothy M. Chan
Sourav Chatterjee
Eshan Chattopadhyay
Vaggos Chatziafratis
Chandra Chekuri
Li Chen
Mingshuai Chen
Kuan Cheng
Pingan Cheng
Siu-Wing Cheng
Markus Chimani
Ashish Chiplunkar
Man Kwun Chiu
Aruni Choudhary
Jonas Cleve
Raphael Clifford
Christian Coester
Gil Cohen
Ilan Cohen
Vincent Cohen-Addad
Spencer Compton
Ty Coon
Colin Cooper
Graham Cormode
Bruno Courcelle
Ágnes Cseh
Radu Curticapean
Artur Czumaj
Yuval Dagan
Ameya Daigavane
Mina Dalirrooyfard
Christoph Damerius
Syamantak Das
Ewan Davies
Gareth T. Davies
Sami Davies
Anindya De
Argyrios Deligkas
Daniele Dell'Erba
Mahsa Derakhshan
Farzaneh Derakhshan
Fernando Hugo Cunha Dias
Martin Dietzfelbinger
Yotam Dikstein
Michael Dinitz
Nikolay Dolbilin
Sally Dong
Ruiwen Dong
Dean Doron
Michal Dory
Jan Dreier
Anne Driemel
Ran Duan
Christoph Durr
Zdenek Dvorak
Franziska Eberle
Talya Eden

Mahsa Eftekhari
Kord Eickmeyer
David Eppstein
Ashkan Norouzi Fard
Dan Feldman
Johannes Fichte
Nathanaël Fijalkow
Ugo Finnendahl
Pierre Fraigniaud
Tom Friedetzky
Frank Fuhlbrück
Travis Gagie
Arun Ganesh
Pawel Gawrychowski
Evangelia Gergatsouli
Riddhi Ghosal
Aravind Gollakota
Gramoz Goranci
Pierre Guillon
Anupam Gupta
Rohit Gurjar
Shai Halevi
Tobias Harks
Zhiyang He
Stefan Hetzl
Petr Hlineny
Martin Hoefer
Jacob Holm
Florian Horn
Xin Huang
Thore Husfeldt
Zvonko Iljazović
Siddhartha Jain
Emmanuel Jeandel
Ce Jin
Charanjit Jutla
John Kallaugher
Adam Karczmarz
Nathaniel Kell
Rasmus Killmann
Robert Kleinberg
Katharina Klost
Frederic Koehler
Leszek Kolodziejczyk
Eitan Kondratovsky
Swastik Kopparty
Robin Kothari
Mario Krenn
Ariel Kulik
Akash Kumar
Kazuhiro Kurita

Kord Eickmeyer
Marek Elias
Rolf Fagerberg
Dorsa Fathollahi
Vitaly Feldman
Hendrik Fichtenberger
Emmanuel Filiot
Carsten Fischer
Adrian Francalanza
Zachary Friggstad
Takuro Fukunaga
Andreas Galanis
Ankit Garg
Romain Gay
Shayan Oveis Gharan
Ludmila Glinskih
Stefan Göller
Lee-Ad Gottlieb
Xiangyu Guo
Manoj Gupta
Gregory Gutin
Thekla Hamm
Pooya Hatami
Markus Hecher
Chris Heunen
Quang Minh Hoang
Lukáš Holík
Felix Hommelsheim
Lingxiao Huang
Xuangui Huang
John Iacono
Neil Immerman
Klaus Jansen
Xinrui Jia
Zhengzhong Jin
Dominik Kaaser
Andrzej Kamisiński
Tarun Kathuria
Dominik Kempa
Andrey Kim
Max Klimm
Kristin Knorr
Martins Kokainis
Michael Kompatscher
Christian Konrad
Tuukka Korhonen
Laszlo Kozma
Ravishankar Krishnaswamy
Janardhan Kulkarni
Neeraj Kumar
O-Joung Kwon

Kord Eickmeyer
Yuval Emek
Martin Farach-Colton
Sándor Fekete
Guillaume Fertin
Nathanael Fijalkow
Arnold Filtser
Kyle Fox
Dominik D. Freydenberger
Vincent Froese
Nicole Funk
Waldo Gálvez
Sanjam Garg
Ran Gelles
Reza Gheissari
Shay Golan
Mika Goos
Benoit Groz
Zeyu Guo
Varun Gupta
Mohammadtaghi Hajiaghayi
Sariel Har-Peled
Meng He
D. Ellis Hershkowitz
Shuichi Hirahara
Jan Hockendorff
Alexandros Hollender
Sam Hopkins
Shang-En Huang
Sophie Huiberts
Rahul Ilango
Ayush Jain
Rajesh Jayaram
Shaofeng Jiang
Daniel Jost
Sagar Kale
Michael Kapralov
Alexander Kauer
Stefan Kiefer
Sam Kim
Nina Klobas
Yusuke Kobayashi
Gillat Kol
Christian Komusiewicz
Vasilis Kontonis
Pravesh Kothari
Robert Krauthgamer
Dominik Krupke
Rucha Kulkarni
Rajendra Kumar
Rasmus Kyng

Arnaud Labourel
Mina Latifi
Luca Laurenti
Thomas Lavastida
Hung Le
Chin Ho Lee
Joon-Woo Lee
Yin Tat Lee
Stefan Lendl
Michael Levet
Amit Levi
Reut Levi
Bo Li
Jason Li
Jiaao Li
Ray Li
Shi Li
Xin Li
Yi Li
Yuhao Li
Moritz Lichter
Noam Lifshitz
Jeck Lim
Paloma de Lima
Honghao Lin
We-Kai Lin
Jingcheng Liu
Kuikui Liu
Qipeng Liu
Quanquan Liu
Tianren Liu
Tianyu Liu
Yang Liu
Vasilis Livanos
Bruno Loff
Maarten Löffler
Daniel Lokshtanov
Hsueh-I Lu
Marco Lübbecke
Aleksander Łukasiewicz
Stian Lybech
Jayson Lynch
Xin Lyu
Sepideh Mahabadi
Tung Mai
Konstantin Makarychev
Johann Makowsky
Giulio Malavolta
Guillaume Malod
Ryan Mann
Nathan Manohar
Vignesh Manoharan
Mathieu Mari
Simon Mauras
Guillaume Maurras
Richard Mayr
Dylan McDermott
Andrew McGregor
Nicole Megow
Saeed Mehraban
Arturo Merino
Arnaud de Mesmay
Julian Mestre
Jakub Michaliszyn
Pascal Michel
Victor Milenkovic
Neeldhara Misra
Dieter Mitsche
Michael Mitzenmacher
Chandra Kanta Mohapatra
Hendrik Molter
Tobias Mömke
Morteza Monemizadeh
Jonathan Mosheiff
Amer Mouawad
Dave Mount
Loay Mualem
Partha Mukhopadhyay
Ian Munro
Alexander Munteanu
Richard Mycroft
Viswanath Nagarajan
Danupon Nanongkai
Meghana Nasre
Gonzalo Navarro
Inbal Livni Navon
Amir Nayyeri
Yakov Nekrich
Daniel Neuen
Stefan Neumann
Ngoc Khanh Nguyen
Zipei Nie
Prajakta Nimbhorkar
Chinmay Nirkhe
Thomas Noll
Klara Nosan
Krzysztof Nowicki
André Nusser
Zeev Nutov
Pierre Ohlmann
Pierre Ohlmann
Olga Ohrimenko
Alexander Okhotin
Andre Oliveira
Sebastian Ordyniak
Piotr Ostropolski -Nalewaja
Yota Otachi
Sang-il Oum
Luca Padovani
Rasmus Pagh
Dömötör Pálvölgyi
Irene Parada
Fanny Pascual
Francesco Pasquale
Zuzana Patáková
Matthew Patitz
Subhasree Patro
Jarkko Peltomäki
Pan Peng
Guillermo Perez
Will Perkins
Josh Petrack
Seth Pettie
Ulrich Pferschy
Huy Tuan Pham
Jeff Phillips
Marta Piecyk
Michał Pilipczuk
Solon Pissis
Madhusudhan Pittu
Adam Polak
Igor Potapov
Aditya Potukuch
Amaury Pouly
Marc Pouzet
Anupam Prakash
Maximilian Probst
Kirk Pruhs
Pavel Pudlak
Gabriele Puppis
Qi Qi
Willy Quach
Willy Quach
Harald Räcke
Jaikumar Radhakrishnan
Bader Abu Radi
Klaus Radke
Jakub Radoszewski
Mustazee Rahman
Saladi Rahul
Ashutosh Rai
Rajeev Raman
Timothy Randolph
Michael Rao
Cyrus Rashtchian
Nidhi Rathi
Malin Rau
R Ravi
Elizaveta Rebrova

Oded Regev
Marc Renault
Susanna F. de Rezende
Liam Roddity
Dana Ron-Goldreich
Günter Rote
Ben Rubinstein
Seyran Saeedi
Mohammad Salavatipour
Sai Sandeep
Jayshree Sarathy
Kevin Schewior
Sylvain Schmitz
Pascal Schweitzer
Saeed Seddighin
Alexander Shen
Rahul Shome
Sebastian Siebertz
Jin Sima
Mateusz Skomra
Jack Snoeyink
Zhao Song
Jonathan Spreer
Piyush Srivastava
Rob Van Stee
John Stell
Ana Stoica
Eijiro Sumii
Ziteng Sun
Céline Swennenhuis
Suguru Tamaki
Biaoshuai Tao
Prasad Tetali
Dimitrios Thilikos
Csaba Toth
Madhur Tulsiani
Shashanka Ubaru
Chris Umans
John Van de Wetering
Virginia Vassilevska Williams
Pavel Veselý
Cosimo Vinci
Mate Vizer
Tjark Vredeveld
David Wajc
Guanghui Wang
Kunihiro Wasa
Alexander Wei
Omri Weinstein
Michael Whitmeyer
Lucia Williams

Rogério Reis
Clément Requilé
Joao Ribeiro
Heiko Röglin
Adi Rosén
Thomas Rothvoss
Matteo Russo
Chandan Saha
Ville Salo
Laura Sanita
David Saulpic
Ildikó Schlotter
Jason Schoeters
Chris Schwiegelshohn
Rik Sengupta
Abhishek Shetty
Aaron Sidford
Jad Silbak
Vikrant Singhal
Michał Skrzypczak
Dmitry Sokolov
Zhuoqing Song
Ramanujan M. Sridharan
Georgios Stamoulis
Matej Stehlik
Frank Stephan
Christoph Striecks
Xiaorui Sun
Ola Svensson
Wai Ming Tai
Zihan Tan
Jakab Tardos
Jakub Tětek
Mikkel Thorup
Patrick Totzke
Emilio Tuosto
Ryuhei Uehara
Jalaj Upadhyay
Jan Van den Brand
Sergei Vassilvitskii
Laurent Viennot
Marc Vinyals
Ilya Volkovich
Duong Vuong
Hendrik Waldner
Haitao Wang
Adam Bene Watts
Alex Wein
Philip Wellnitz
Andreas Wiese
Tobias Winkler

Lisheng Ren
Mohsen Rezapour
Robert Robere
Lars Rohwedder
Benjamin Rossman
Aviad Rubinstein
Karthik C, S.
Cenk Sahinalp
Robert Samal
Thatchaphol Saranurak
Nicolas Schabanel
Melanie Schmidt
Roy Schwartz
Ziv Scully
Mingfu Shao
David Shmoys
Tasos Sidiropoulos
Sandeep Silwal
Sahil Singla
Michał Skrzypczak
Shuang Song
Paul Spirakis
Nikhil Srivastava
Tatiana Starikovskaya
Alex Steiger
Frank Stephan
Hang Su
Yuxin Sun
Chaitanya Swamy
Avishay Tal
Erasmo Tani
Jakub Tarnawski
Justin Thaler
Ivan Tjuawinata
Noam Touitou
Iddo Tzameret
Jonathan Ullman
Ali Vakilian
Nithin Varma
László Végh
Alexandre Vigny
Ellen Vitercik
Tijn de Vos
Erik Waingarten
Di Wang
Kangning Wang
Karol Węgrzycki
Nicole Wein
Linda Brown Westrick
Sebastian Wild
R. Teal Witter

Damien Woods
Mary Wootters
David Wu
Hongxun Wu
Jinzhao Wu
Pei Wu
Xuan Wu
Mingyu Xiao
Jeff Xu
Jie Xue
Yutaro Yamaguchi
Xiang Yan
Elizabeth Yang
Mihalis Yannakakis
Taisuke Yasuda
Yitong Yin
Yusuke Yoshida
Huacheng Yu
Jing Yu
Pei Yuan
Yang Yuan
Viktor Zamaraev
Nikos Zarifis
Meirav Zehavi
Mark Zhandry
Chen-Da Liu Zhang
Chihao Zhang
Peng Zhang
Qiankun Zhang
Shufan Zhang
Yuhao Zhang
Hongyu Zheng
Hang Zhou
Samson Zhou
Kaiyuan Zhu

List of Authors

Amir Abboud (7)
Weizmann Institute of Science, Rehovot, Israel

Shweta Agrawal (8)
Indian Institute of Technology, Madras, India

Xavier Allamigeon (110)
INRIA, Palaiseau, France;
CMAP, École polytechnique, IP Paris, CNRS, Palaiseau, France

Josh Alman (9)
Department of Computer Science, Columbia University, New York, NY, USA

Omar Alrabiah (10)
EECS Department, University of California, Berkeley, CA, USA

Djamel Eddine Amir (111)
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Sepehr Assadi (11)
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Albert Atserias (1)
Universitat Politècnica de Catalunya, Centre de Recerca Matemàtica, Barcelona, Spain

Nikhil Ayyadevara (12)
Indian Institute of Technology Delhi, India

Yossi Azar (13)
School of Computer Science, Tel Aviv University, Israel

Christel Baier (129)
Technische Universität Dresden, Germany

Nikhil Bansal (14)
University of Michigan, Ann Arbor, MI, USA

David Barozzini (112)
Institute of Informatics, University of Warsaw, Poland

Surender Baswana (15)
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Calvin Beideman (16)
University of Illinois Urbana-Champaign, IL, USA

Omri Ben-Eliezer (17)
Massachusetts Institute of Technology, Cambridge, MA, USA

Aaron Berger (19)
MIT, Cambridge, MA, USA

Pierre Bergé (18)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Aaron Bernstein (11, 20)
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Nathalie Bertrand (113)
Univ Rennes, Inria, CNRS, IRISA, France

Ivona Bezáková (21)
Department of Computer Science, Rochester Institute of Technology, NY, USA

Koustav Bhanja (15)
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Davide Bilò (22)
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy

Mitchell Black (23)
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Guy Blanc (24)
Stanford University, CA, USA

Joakim Blikstad (25)
KTH Royal Institute of Technology, Sweden

Jeremiah Blocki (26)
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Niclas Boehmer (27)
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

León Bohn (114)
RWTH Aachen University, Germany

Édouard Bonnet (18)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Benjamin Bordais (115)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Zvika Brakerski (28)
Weizmann Institute of Science, Rehovot, Israel

Marcin Briański (29)
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Léonard Brice (116)
Université libre de Bruxelles, Brussels, Belgium

Karl Bringmann (30, 31, 32)
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany;
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Caroline Brosse (33)
Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, Limos, F-63000 Clermont-Ferrand, France

Damien Busatto-Gaston (115)
Université libre de Bruxelles, Brussels, Belgium

David Caballero (34)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Antonio Casares (117)
LaBRI, Université de Bordeaux, France

Alejandro Cassis (30, 31, 32)
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany;
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Diptarka Chakraborty (35)
National University of Singapore, Singapore

Sourav Chakraborty (36)
Indian Statistical Institute, Kolkata, India

Parinya Chalermsook (37)
Aalto University, Espoo, Finland

Karthekeyan Chandrasekaran (16)
University of Illinois Urbana-Champaign, IL, USA

Moses Charikar (38)
Stanford University, CA, USA

Kushagra Chatterjee (35)
National University of Singapore, Singapore

Eshan Chattopadhyay (10)
Computer Science Department, Cornell University, Ithaca, NY, USA

Lin Chen (39)
Department of Computer Science, Texas Tech University, Lubbock, TX, USA

Qingyun Chen (89)
University of California, Merced, CA, USA

Ryder Chen (40)
The University of Sydney, Australia

Chi-Ning Chou (41)
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Keerti Choudhary (22, 35)
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Aleksander B. G. Christiansen (42)
Technical University of Denmark, Lyngby, Denmark

Gil Cohen (43, 44)
Department of Computer Science, Tel Aviv University, Israel

Sarel Cohen (22)
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Israel

Vincent Cohen-Addad (7)
Google Research, Zürich, Switzerland

Amin Coja-Oghlan (45)
Faculty of Computer Science, TU Dortmund, Germany

Thomas Colcombet (117)
CNRS, IRIF, Université Paris Cité, France

Jacobus Conradi (46)
Department of Computer Science, Universität Bonn, Germany

Artur Czumaj (47)
University of Warwick, Coventry, UK

Rajni Dabas (12)
Department of Computer Science, University of Delhi, India

Varsha Dani (48)
Department of Computer Science, Rochester Institute of Technology, Rochester, NY, USA

Debarati Das (49)
Pennsylvania State University,
University Park, PA, USA

Constantinos Daskalakis (2)
EECS and CSAIL, MIT, Cambridge, MA, USA

Samir Datta (118)
Chennai Mathematical Institute, India

Niel de Beaudrap (119)
University of Sussex, UK

Bart de Keijzer (76)
Department of Informatics,
King's College London, UK

Tijn de Vos (61)
Universität Salzburg, Austria

Mingyang Deng (50)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Amit Deshpande (51)
Microsoft Research, Bengaluru, India

Shyam S. Dhamapurkar (52)
Southern University of Science and Technology,
Shenzhen, China

Ming Ding (53, 54)
ETH Zürich, Switzerland

Dean Doron (55)
Department of Computer Science, Ben-Gurion
University of the Negev, Beer-Sheva, Israel

Amina Doumane (121)
CNRS, LIP, ENS Lyon, France

Gaëtan Douéneau-Tabot (120)
IRIF, Université Paris Cité & Direction générale
de l'armement – Ingénierie de projets, France

Anne Driemel (46)
Hausdorff Center for Mathematics,
Universität Bonn, Germany

Aditi Dudeja (11)
Department of Computer Science,
Rutgers University, Piscataway, NJ, USA

Hugues Déprés (18)
Univ Lyon, CNRS, ENS de Lyon, Université
Claude Bernard Lyon 1, LIP UMR5668, France

Josep Díaz (48)
Department of Computer Science, Polytechnic
University of Catalonia, Barcelona, Spain

Nico Döttling (28)
CISPA Helmholtz Center for Information
Security, Saarbrücken, Germany

Talya Eden (56)
Boston University, MA, USA;
MIT, Cambridge, MA, USA

Charilaos Efthymiou (57)
Computer Science, University of Warwick,
Coventry, UK

Louis Esperet (58)
Univ. Grenoble Alpes, CNRS, Laboratoire
G-SCOP, Grenoble, France

Léo Exibard (122)
Reykjavik University, Iceland

Moran Feldman (59)
University of Haifa, Israel

Emmanuel Filiot (122)
Université libre de Bruxelles, Brussels, Belgium

Nick Fischer (30, 32)
Universität des Saarlandes, Saarland Informatics
Campus, Saarbrücken, Germany;
Max Planck Institute for Informatics, Saarland
Informatics Campus, Saarbrücken, Germany

Fedor V. Fomin (60)
Department of Informatics,
University of Bergen, Norway

Sebastian Forster (61)
Universität Salzburg, Austria

Tobias Friedrich (22, 62)
Hasso Plattner Institute,
University of Potsdam, Germany

Jakub Gajarský (123)
University of Warsaw, Poland

Andreas Galanis (21, 45, 63)
Department of Computer Science,
University of Oxford, UK

Luyining Gan (64)
Department of Mathematics and Statistics,
University of Nevada, Reno, NV, USA

Moses Ganardi (124)
Max Planck Institute for Software Systems
(MPI-SWS), Kaiserslautern, Germany

Arnab Ganguly (65)
Dept. of Computer Science, University of
Wisconsin, Whitewater, WI, USA

Robert Ganian (66)
Algorithms and Complexity Group,
TU Wien, Austria

Sanjam Garg (28)
University of California, Berkeley, CA, USA;
NTT Research, Sunnyvale, CA, USA

Stéphane Gaubert (110)
INRIA, Palaiseau, France;
CMAP, École polytechnique, IP Paris, CNRS,
Palaiseau, France

Hans Gawendowicz (62)
Hasso Plattner Institute,
University of Potsdam, Germany

Paweł Gawrychowski (67)
Institute of Computer Science,
University of Wrocław, Poland

Elahe Ghasemi (68)
Sharif University of Technology, Teheran, Iran;
Univ Gustave Eiffel, CNRS, LIGM, F-77454
Marne-la-Vallée, France

Arijit Ghosh (69)
Indian Statistical Institute, Kolkata, India

Hugo Gimbert (125)
Université de Bordeaux, CNRS, France

Leslie Ann Goldberg (3, 21, 45)
University of Oxford, UK

Petr A. Golovach (60)
Department of Informatics,
University of Bergen, Norway

Timothy Gomez (34)
Department of Computer Science, University of
Texas Rio Grande Valley, TX, USA

Jesse Goodman (10)
Computer Science Department,
Cornell University, Ithaca, NY, USA

Elena Grigorescu (26)
Department of Computer Science,
Purdue University, West Lafayette, IN, USA

Martin Grohe (70)
RWTH Aachen University, Germany

Shibashis Guha (115)
Tata Institute of Fundamental Research,
Mumbai, India

Chetan Gupta (118)
Aalto University, Finland

Maximilian Probst Gutenberg (53)
ETH Zürich, Switzerland

Mika Göös (126)
EPFL, Lausanne, Switzerland

Thekla Hamm (66)
Algorithms and Complexity Group,
TU Wien, Austria

Jie Han (64)
School of Mathematics and Statistics and Center
for Applied Math, Beijing Institute of
Technology, China

Nathaniel Harms (71)
University of Waterloo, Canada

Ishay Haviv (72)
School of Computer Science, The Academic
College of Tel Aviv-Yaffo, Tel Aviv, Israel

Thomas P. Hayes (48)
Department of Computer Science, University of
New Mexico, Albuquerque, NM, USA

Dean Hirsch (9)
Department of Computer Science,
Columbia University, New York, NY, USA

Justin Holmgren (73)
NTT Research, Sunnyvale, CA, USA

Jakob Bæk Tejs Houen (74)
BARC, Department of Computer Science,
University of Copenhagen, Denmark

Mathieu Hoyrup (111)
Université de Lorraine, CNRS, Inria, LORIA,
F-54000 Nancy, France

Chien-Chung Huang (37)
École Normale Supérieure, Paris, France

Ziyun Huang (75)
Department of Computer Science and Software
Engineering, Penn State Erie, The Behrend
College, USA

Paweł M. Idziak (127)
Jagiellonian University, Kraków, Poland

Tanmay Inamdar (60)
Department of Informatics,
University of Bergen, Norway

Stavros D. Ioannidis (76)
Department of Informatics,
King's College London, UK

Rahul Jain (118)
Fernuniversität in Hagen, Germany

Arun Jambulapati (77)
Stanford University, CA, USA

Klaus Jansen (78)
Universität Kiel, Germany

Haotian Jiang (14)
University of Washington, Seattle, WA, USA

Shaofeng H.-C. Jiang (47)
Peking University, Beijing, China

Shunhua Jiang (79)
Columbia University, New York, NY, USA

Yujia Jin (77)
Stanford University, CA, USA

Vincent Jugé (68)
Univ Gustave Eiffel, CNRS, LIGM,
F-77454 Marne-la-Vallée, France

Shunsuke Kanda (99)
RIKEN Center for Advanced Intelligence
Project, Tokyo, Japan

Ricardo D. Katz (110)
CIFASIS-CONICET, Rosario, Argentina

Piotr Kawałek (127)
Jagiellonian University, Kraków, Poland

Chandrima Kayal (36)
Indian Statistical Institute, Kolkata, India

Ghazal Khalighinejad (68)
Duke University, Durham, NC, USA; Sharif
University of Technology, Teheran, Iran

Ayrat Khalimov (122)
Université libre de Bruxelles, Brussels, Belgium

Arindam Khan (12, 78, 80)
Department of Computer Science and
Automation, Indian Institute of Science,
Bengaluru, India

Jahanvi Khatkar (40)
The University of Sydney, Australia

Sandra Kiefer (81)
Max Planck Institute for Software Systems,
Saarland Informatics Campus, Saarbrücken,
Germany

Stefan Kiefer (126)
University of Oxford, UK

Yael Kirkpatrick (50)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Aleks Kissinger (119)
University of Oxford, UK

Tomohiro Koana (27)
Algorithmics and Computational Complexity,
Technische Universität Berlin, Germany

Tomasz Kociumaka (49)
Max Planck Institute for Informatics,
Saarbrücken, Germany

Shimon Kogan (82)
Weizmann Institute of Science, Rehovot, Israel

Balagopal Komarath (83)
Indian Institute of Technology Gandhinagar,
India

Viktoriia Korchemna (66)
Algorithms and Complexity Group,
TU Wien, Austria

Martin Koutecký (29)
Computer Science Institute, Charles University,
Prague, Czech Republic

Robert Krauthgamer (47)
Weizmann Institute of Science, Rehovot, Israel

Jacek Krzaczkowski (127)
Maria Curie-Sklodowska University,
Lublin, Poland

Daniel Král' (29)
Faculty of Informatics, Masaryk University,
Brno, Czech Republic

Akash Kumar (84)
School of Computer and Communication
Sciences, EPFL, Lausanne, Switzerland

William Kuszmaul (19, 85)
MIT, Cambridge, MA, USA

Rasmus Kyng (53, 54)
ETH Zürich, Switzerland

Marvin Künnemann (30)
TU Kaiserslautern, Germany

Jakub Łącki (86)
Google Research, New York, NY, USA

Bundit Laekhanukit (89)
Shanghai University of Finance and Economics,
China

Jane Lange (24)
MIT, Cambridge, MA, USA

Alexandra Lassota (87)
EPFL, Lausanne, Switzerland

Euiwoong Lee (7)
University of Michigan, Ann Arbor, MI, USA

Yin Tat Lee (4)
University of Washington, Seattle, WA, USA;
Microsoft Research, Seattle, WA, USA

Clément Legrand-Duchesne (88)
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI,
UMR 5800, F-33400 Talence, France

Karoliina Lehtinen (117)
CNRS, Aix-Marseille Université,
Université de Toulon, LIS, France

Pascal Lenzner (62)
Hasso Plattner Institute,
University of Potsdam, Germany

Shoham Letzter (17)
University College London, UK

Xin Li (10)
Computer Science Department,
Johns Hopkins University, Baltimore, MD, USA

Chao Liao (89)
Shanghai Jiao Tong University, China

Vincent Limouzy (33)
Université Clermont Auvergne, Clermont
Auvergne INP, CNRS, Mines Saint-Etienne,
Limos, F-63000 Clermont-Ferrand, France

Bingkai Lin (90)
Nanjing University, China

Andrea Lincoln (73)
University of California Berkeley, CA, USA

Marvin Lira (78)
Universität Kiel, Germany

Paul Liu (59)
Stanford University, CA, USA

Daniel Lokshtanov (91)
University of California,
Santa Barbara, CA, USA

Aditya Lonkar (80)
Department of Computer Science and
Automation, Indian Institute of Science,
Bangalore, India

Anand Louis (84)
Computer Science and Automation Department,
IISc, Bangalore, India

Peter J. Love (41)
Department of Physics and Astronomy,
Tufts University, Medford, MA, USA

Zhenjian Lu (92)
University of Warwick, Coventry, UK

Aleksander Łukasiewicz (87)
University of Wrocław, Poland

Christof Löding (114)
RWTH Aachen University, Germany

Chay Machluf (13)
School of Electrical Engineering,
Tel Aviv University, Israel

Arnab Maiti (80)
Indian Institute of Technology, Kharagpur, India

Konrad Majewski (93)
Institute of Informatics, Faculty of Mathematics,
Informatics and Mechanics, University of
Warsaw, Poland

Rupak Majumdar (124)
Max Planck Institute for Software Systems
(MPI-SWS), Kaiserslautern, Germany

Giulio Malavolta (28)
Max Planck Institute for Security and Privacy,
Bochum, Germany

Pasin Manurangsi (7)
Google Research, Mountain View, CA, USA

Nicolas Markey (113)
Univ Rennes, Inria, CNRS, IRISA, France

Arnaud Mary (33)
Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive
UMR 5558, 69622 Villeurbanne, France;
ERABLE team, Inria Grenoble Rhône-Alpes,
Villeurbanne, France

Tomáš Masařík (93)
Institute of Informatics, Faculty of Mathematics,
Informatics and Mechanics, University of
Warsaw, Poland

Corto Mascle (125)
Université de Bordeaux, France

Surya Mathialagan (94)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Claire Mathieu (95)
CNRS Paris, France

Andrew McGregor (96)
University of Massachusetts, Amherst, MA, USA

Raghu Meka (14)
University of California, Los Angeles, CA, USA

Anna Melnichenko (62)
Hasso Plattner Institute,
University of Potsdam, Germany

Dor Minzer (43)
Department of Mathematics, Massachusetts Institute of Technology, Cmabridge, MA, USA

Gopinath Mishra (69)
University of Warwick, Coventry, UK

Cristopher Moore (48)
Santa Fe Institute, NM, USA

Anish Mukherjee (118)
University of Warsaw, Poland; IDEAS NCBR, Warsaw, Poland

Tamalika Mukherjee (26)
Department of Computer Science,
Purdue University, West Lafayette, IN, USA

Anca Muscholl (125)
Université de Bordeaux, France

Amulya Musipatla (97)
Carnegie Mellon University,
Pittsburgh, PA, USA

Tamio-Vesa Nakajima (128)
Department of Computer Science,
University of Oxford, UK

Vasileios Nakos (32)
RelationalAI, Berkeley, CA, USA

Danupon Nanongkai (20, 37)
University of Copenhagen, Denmark;
KTH Royal Institute of Technology,
Stockholm, Sweden

Shyam Narayanan (85)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Bento Natura (79)
London School of Economics, UK

Amir Nayyeri (23)
School of Electrical Engineering and Computer Science, Oregon State University,
Corvallis, OR, USA

Yasamin Nazari (86)
Universität Salzburg, Austria

Daniel Neuen (81)
School of Computing Science,
Simon Fraser University, Burnaby, Canada

Ilan Newman (98)
Department of Computer Science,
University of Haifa, Israel

Takaaki Nishimoto (99)
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Sergey Norin (58)
Department of Mathematics and Statistics,
McGill University, Montreal, Canada

Ashkan Norouzi-Fard (59)
Google Research, Zurich, Switzerland

Jana Novotná (93)
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Ryan O'Donnell (97)
Carnegie Mellon University,
Pittsburgh, PA, USA

Karolina Okrasa (66, 93)
Faculty of Matematics and Information Science, Warsaw University of Technology, Poland;
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Igor C. Oliveira (92)
University of Warwick, Coventry, UK

Abhyuday Pandey (15)
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Anurag Pandey (83)
Department of Computer Science, Universität des Saarlances, Saarland Informatics Campus, Saarbrücken, Germany

Fahad Panolan (91)
Indian Institute of Technology Hyderabad, India

Theodoros Papamakarios (100)
Department of Computer Science,
University of Chicago, IL, USA

Manaswi Paraashar (36)
Aarhus University, Denmark

Merav Parter (82)
Weizmann Institute of Science, Rehovot, Israel

Paweł Parys (112)
Institute of Informatics,
University of Warsaw, Poland

Boaz Patt-Shamir (13)
School of Electrical Engineering,
Tel Aviv University, Israel

Rameesh Paul (84)
Computer Science and Automation Department,
IISc, Bangalore, India

Andreas Pavlogiannis (124)
Aarhus University, Denmark

Shubham Vivek Pawar (52)
Hubli, Karnataka, India

Kristýna Pekárková (29)
Faculty of Informatics, Masaryk University,
Brno, Czech Republic

Shir Peleg (43)
Department of Computer Science,
Tel Aviv University, Israel

Ján Pich (101)
University of Oxford, UK

Marcin Pilipczuk (93)
Institute of Informatics, Faculty of Mathematics,
Informatics and Mechanics, University of
Warsaw, Poland

Michał Pilipczuk (102, 123)
University of Warsaw, Poland

Jakob Piribauer (129)
Technische Universität Dresden, Germany

Karol Pokorski (67)
Institute of Computer Science,
University of Wrocław, Poland

Adam Polak (19, 87)
EPFL, Lausanne, Switzerland

Aaron Potechin (43)
Department of Computer Science,
University of Chicago, IL, USA

Rameshwar Pratap (51)
Indian Institute of Technology,
Mandi, H.P., India

Enguerrand Prebet (130)
Université de Lyon, ENS de Lyon, UCB Lyon 1,
CNRS, INRIA, LIP

Maximilian Probst Gutenberg (20)
ETH Zürich, Switzerland

Wojciech Przybyszewski (123)
University of Warsaw, Poland

Guoliang Qiu (103)
Shanghai Jiao Tong University, China

Jaikumar Radhakrishnan (52)
Tata Institute of Fundamental Research,
Mumbai, India

Chengot Sankaramenon Rahul (83)
School of Mathematics and Computer Science,
Indian Institute of Technology Goa, India

Ashutosh Rai (88)
Department of Mathematics, IIT Delhi,
Hauz Khas, New Delhi, India

M. S. Ramanujan (91)
University of Warwick, Coventry, UK

Jean-François Raskin (115, 116)
Université libre de Bruxelles, Brussels, Belgium

Gaurav Rattan (70)
RWTH Aachen University, Germany

Jean Bernoulli Ravelomanana (45)
Faculty of Computer Science,
TU Dortmund, Germany

Rahul Raychaudhury (69)
Duke University, Durham, NC, USA

Alexander Razborov (100)
University of Chicago, IL, USA;
Steklov Mathematical Institute, Moscow, Russia

Xuandi Ren (90)
Peking University, Beijing, China

Nicolas Resch (104)
Cryptology Group, Centrum Wiskunde &
Informatica, Amsterdam, The Netherlands

João Ribeiro (10)
Computer Science Department, Carnegie Mellon
University, Pittsburgh, PA, USA

Dana Ron (56)
Tel Aviv University, Israel

Victor Rong (50)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Will Rosenbaum (56)
Amherst College, MA, USA

Jurriaan Rot (132)
Institute for Computing and Information
Sciences, Radboud University, Nijmegen,
The Netherlands

Eva Rotenberg (42)
Technical University of Denmark,
Lyngby, Denmark

Ron D. Rothblum (73)
Technion, Haifa, Israel

Wojciech Rozowski (132)
Department of Computer Science,
University College London, UK

Aviad Rubinstein (106)
Computer Science Department,
Stanford University, CA, USA

Ittai Rubinstein (105)
Blavatnik School of Computer Science,
Tel-Aviv University, Israel;
QEDMA Quantum Computing,
Tel-Aviv, Israel

Paweł Rzążewski (93)
Faculty of Mathematics and Information Science,
Warsaw University of Technology, Poland;
Institute of Informatics, Faculty of Mathematics,
Informatics and Mechanics, University of
Warsaw, Poland

Barna Saha (49)
University of California, San Diego, CA, USA

Ville Salo (131)
Department of Mathematics and Statistics,
University of Turku, Finland

Juspreet Singh Sandhu (41)
School of Engineering and Applied Sciences,
Harvard University, Cambridge, MA, USA

Ocan Sankur (113, 129)
Univ Rennes, Inria, CNRS, IRISA, France

Rahul Santhanam (101)
University of Oxford, UK

Thatchaphol Saranurak (20, 37)
University of Michigan, Ann Arbor, MI, USA

Martin Schirneck (22)
Hasso Plattner Institute,
University of Potsdam, Germany

Nicole Schirrmacher (102)
Universität Bremen, Germany

Todd Schmid (132)
Department of Computer Science,
University College London, UK

Tselil Schramm (97)
Stanford University, CA, USA

Felix Schröder (29)
Institute of Mathematics,
Technische Universität, Berlin, Germany

Robert Schweller (34)
Department of Computer Science, University of
Texas Rio Grande Valley, TX, USA

Lia Schütze (124)
Max Planck Institute for Software Systems
(MPI-SWS), Kaiserslautern, Germany

Sayantan Sen (69)
Indian Statistical Institute, Kolkata, India

Rik Sengupta (96)
University of Massachusetts, Amherst, MA, USA

Tim Seppelt (70)
RWTH Aachen University, Germany

Rahul Shah (65)
Dept. of Computer Science, Louisiana State
University, Baton Rouge, LA, USA

Amatya Sharma (80)
Indian Institute of Technology, Kharagpur, India

Vimal Raj Sharma (118)
Indian Institute of Technology, Kanpur, India

Jonathan Shi (41)
Department of Computing Sciences,
Bocconi University, Milan, Italy

Aaron Sidford (20, 77)
Stanford University, CA, USA

Sebastian Siebertz (102)
Universität Bremen, Germany

Alexandra Silva (132)
Department of Computer Science,
Cornell University, Ithaca, NY, USA

Kirill Simonov (66)
Algorithms and Complexity Group,
TU Wien, Austria

Sahil Singla (14)
Georgia Institute of Technology,
Atlanta, GA, USA

Makrand Sinha (14)
Simons Institute and University of California,
Berkeley, CA, USA

Mateusz Skomra (110)
LAAS-CNRS, Université de Toulouse, CNRS,
Toulouse, France

Marek Sokołowski (93)
Institute of Informatics, Faculty of Mathematics,
Informatics and Mechanics, University of
Warsaw, Poland

K. V. N. Sreenivas (12, 78)
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Daniel Štefankovič (21, 45, 63)
Department of Computer Science, University of Rochester, Rochester, NY, USA

Damien Stehlé (8)
ENS de Lyon, France;
Institut Universitaire de France, Paris, France

D. M. Stull (133)
Department of Computer Science, Northwestern University, Evanston, IL, USA

Madhu Sudan (5)
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Pattara Sukprasert (37)
Northwestern University, Evanston, IL, USA

He Sun (20)
University of Edinburgh, UK

Yican Sun (90)
Peking University, Beijing, China

Ola Svensson (59)
EPFL, Lausanne, Switzerland

Amnon Ta-Shma (43)
Department of Computer Science, Tel Aviv University, Israel

Yasuo Tabei (99)
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Li-Yang Tan (24)
Stanford University, CA, USA

Martin Tancer (88)
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Raghunath Tewari (118)
Indian Institute of Technology, Kanpur, India

Sharma V. Thankachan (65)
Dept. of Computer Science, University of Central Florida, Orlando, FL, USA

Stéphan Thomassé (6)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Mikkel Thorup (74)
BARC, Department of Computer Science, University of Copenhagen, Denmark

Kevin Tian (77)
Stanford University, CA, USA

Jonathan Tidor (19)
MIT, Cambridge, MA, USA

Szymon Toruńczyk (102, 123)
University of Warsaw, Poland

Noam Touitou (13)
School of Computer Science, Tel Aviv University, Israel

Ilkka Törmä (131)
Department of Mathematics and Statistics, University of Turku, Finland

Jakub Tětek (107)
Basic Algorithms Research Copenhagen, University of Copenhagen, Denmark

Seeun William Umboh (40)
The University of Sydney, Australia

John van de Wetering (119)
Radboud University Nijmegen, The Netherlands;
University of Oxford, UK

Marie van den Bogaard (116)
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Jan van den Brand (20)
Simons Institute, Berkeley, CA, USA;
University of California Berkeley, CA, USA

Nithin Varma (98)
Chennai Mathematical Institute, India

Virginia Vassilevska Williams (50, 94)
Massachusetts Institute of Technology, Cambridge, MA, USA

Santosh S. Vempala (4)
Georgia Tech, Atlanta, GA, USA

Carmine Ventre (76)
Department of Informatics, King's College London, UK

Pavel Veselý (47)
Charles University, Prague, Czech Republic

Alexandre Vigny (102)
Universität Bremen, Germany

Eric Vigoda (45, 63)
Computer Science, University of California Santa Barbara, CA, USA

Erik Waingarten (17, 38)
Stanford University, CA, USA

Nicolas Waldburger (113)
Univ Rennes, Inria, CNRS, IRISA, France

Igor Walukiewicz (125)
Université de Bordeaux, CNRS, France

Weihang Wang (16)
University of Illinois Urbana-Champaign, IL, USA

Xiuhan Wang (90)
Tsinghua University, Beijing, China

Yanheng Wang (103)
ETH Zürich, Switzerland

Nicole Wein (19)
DIMACS, Piscataway, NJ, USA

Omri Weinstein (79)
The Hebrew University, Jerusalem, Israel; Columbia University, New York, NY, USA

Armin Weiß (127)
Universität Stuttgart, FMI, Germany

Andreas Wiese (80)
Technische Universität München, Germany

Mary Wootters (55)
Departments of Computer Science and Electrical Engineering, Stanford University, CA, USA

Jan Wróblewski (112)
Institute of Informatics, University of Warsaw, Poland

Xiaoyu Wu (39)
School of Mathematical Sciences, Zhejiang University, Hangzhou, China

Xinyu Wu (97)
Carnegie Mellon University, Pittsburgh, PA, USA

Tim Wylie (34)
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Jinhui Xu (75)
Department of Computer Science and Engineering, State University of New York at Buffalo, NY, USA

Yinzhan Xu (94)
Massachusetts Institute of Technology, Cambridge, MA, USA

Anshu Yadav (8)
Indian Institute of Technology, Madras, India

Tal Yankovitz (44)
Department of Computer Science, Tel Aviv University, Israel

Penghui Yao (108)
State Key Laboratory for Novel Software Technology, Nanjing University, China

Yitong Yin (108)
State Key Laboratory for Novel Software Technology, Nanjing University, China

Sorrachai Yingchareonthawornchai (37)
Aalto University, Espoo, Finland

Yuichi Yoshida (71)
National Institute of Informatics, Tokyo, Japan

Chen Yuan (104)
School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China

Weiqiang Yuan (126)
EPFL, Lausanne, Switzerland

Meirav Zehavi (60)
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Rico Zenklusen (59)
ETH Zürich, Switzerland

Georg Zetzsche (124)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Chihao Zhang (103)
Shanghai Jiao Tong University, China

Guochuan Zhang (39)
School of Computer Science, Zhejiang University, Hangzhou, China

Peng Zhang (53, 54)
Rutgers University, Piscataway, NJ, USA

Tianyi Zhang (109)
Tel Aviv University, Israel

Xinyuan Zhang (108)
State Key Laboratory for Novel Software Technology, Nanjing University, China

Yuhao Zhang (89)
Shanghai Jiao Tong University, China

Junyao Zhao (106)
Computer Science Department, Stanford University, CA, USA

Ziqian Zhong (50)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Hang Zhou (95)
École Polytechnique, Institut Polytechnique de
Paris, France

Marius Zimand (92)
Towson University, MD, USA

Stanislav Živný (128)
Department of Computer Science,
University of Oxford, UK

Towards a Theory of Algorithmic Proof Complexity

Albert Atserias
Universitat Politècnica de Catalunya, Centre de Recerca Matemàtica, Barcelona, Spain

Abstract

A possibly unexpected by-product of the mathematical study of the lengths of proofs, as is done in the field of propositional proof complexity, is, I claim, that it may lead to new polynomial-time algorithms. To explain this, I will first recall the origins of proof complexity as a field, and then explain why some of the recent progress in it could lead to some new algorithms.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases proof complexity, logic, computational complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.1

Category Invited Talk

Funding *Albert Atserias*: Partially funded by Spanish State Research Agency, through the Severo Ochoa and María de Maeztu Program for Centers and Units of Excellence in R&D (CEX2020-001084-M) and Ministerio de Ciencia e Innovación (MICIN) through project PID2019- 109137GB-C22 (PROOFS).

As is well known, the formulation of the P vs. NP problem has its origins in mathematical logic. Cook's original interest on the satisfiability problem for logic formulas came from its applications to automated theorem proving; this is clearly reflected in the title of his 1971 paper "On the Complexity of Theorem Proving Procedures" [4]. By viewing the satisfiability of a formula and the tautologyhood of its negation as dual concepts, it can be argued that Cook established the grounds of a general theory of duality for any problem in NP. In this general framework, satisfying assignments and formal proofs would play the roles of dual certificates, with the obvious caveat that proofs typically appear to be exponentially longer than their duals. The question whether this empirical observation about the lengths of proofs is a true fact called for a theory of proof complexity on which to build what later came to be known as Cook's Program [5, 6].

The development of Cook's Program since its formulation in the early 1970's led to many insights on the combinatorial intricacies of formal proofs. The strongest results in the area typically take the form of exponential lower bounds on the length of proofs for proof systems of practical use, including Resolution [9], Cutting Planes [13], and a few others [1, 12, 11]. To achieve this, a deep theory that explains what causes a propositional tautology to not have short proofs was developped. As part of this theory we find certain tight semantic characterizations of resource-bounded proofs [14, 2, 10] much in the same way that the completeness theorem of mathematical logic equalizes truth with proof.

In this talk I want to argue that the semantic study of proof complexity for these proof systems could also lead to new insights of algorithmic nature. Perhaps paradoxically, the starting point for this is the recent discovery that the problem of automating their proof-search is NP-hard [3, 8, 7]. In a nutshell, what these NP-hardness reductions show is that every instance of any problem in NP can be efficiently encoded into a propositional formula that is either a tautology that has a short proof, or is indistinguishable in a formal sense from a falsifiable formula. The existence of a short proof clearly comes with a short certificate. The main point is, however, that the indistinguishability from a falsifiable formula can often,

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 1; pp. 1:1–1:2


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

but not always, be also certified efficiently, no less than through the tools of the theory that studies which formulas fail to have short proofs.

As applications, I will discuss how this phenomenon could be used, potentially, to develop new polynomial-time algorithms for a couple of well-known combinatorial problems not obviously related to propositional logic.

References

1 Miklós Ajtai. The complexity of the pigeonhole principle. *Comb.*, 14(4):417–433, 1994. doi:10.1007/BF01302964.

2 Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *J. Comput. Syst. Sci.*, 74(3):323–334, 2008. Prelim. in CCC 2003. doi:10.1016/j.jcss.2007.06.025.

3 Albert Atserias and Moritz Müller. Automating resolution is np-hard. *J. ACM*, 67(5):31:1–31:17, 2020. Prelim. in FOCS 2019. doi:10.1145/3409472.

4 Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *STOC*, pages 151–158. ACM, 1971.

5 Stephen A. Cook and Robert A. Reckhow. On the Lengths of Proofs in the Propositional Calculus (Preliminary Version). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA*, pages 135–148. ACM, 1974. doi:10.1145/800119.803893.

6 Stephen A. Cook and Robert A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *J. Symb. Log.*, 44(1):36–50, 1979. doi:10.2307/2273702.

7 Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is NP-hard. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 209–222. ACM, 2021. doi:10.1145/3406325.3451080.

8 Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is NP-hard. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 68–77. ACM, 2020. doi:10.1145/3357713.3384248.

9 Armin Haken. The Intractability of Resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

10 Tuomas Hakoniemi. *Size bounds for algebraic and semialgebraic proof systems.* PhD thesis, Universitat Politècnica de Catalunya, 2022.

11 Jan Krajícek, Pavel Pudlák, and Alan R. Woods. An Exponenetioal Lower Bound to the Size of Bounded Depth Frege Proofs of the Pigeonhole Principle. *Random Struct. Algorithms*, 7(1):15–40, 1995. doi:10.1002/rsa.3240070103.

12 Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential Lower Bounds for the Pigeonhole Principle. *Comput. Complex.*, 3:97–140, 1993. doi:10.1007/BF01200117.

13 Pavel Pudlák. Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations. *J. Symb. Log.*, 62(3):981–998, 1997. doi:10.2307/2275583.

14 Pavel Pudlák and Samuel R. Buss. How to Lie Without Being (Easily) Convicted and the Length of Proofs in Propositional Calculus. In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 1994. doi:10.1007/BFb0022253.

Equilibrium Computation, Deep Learning, and Multi-Agent Reinforcement Learning

Constantinos Daskalakis
EECS and CSAIL, MIT, Cambridge, MA, USA

Abstract

Machine Learning has recently made significant advances in challenges such as speech and image recognition, automatic translation, and text generation, much of that progress being fueled by the success of gradient descent-based optimization methods in computing local optima of non-convex objectives. From robustifying machine learning models against adversarial attacks to causal inference, training generative models, multi-robot interactions, and learning in strategic environments, many outstanding challenges in Machine Learning lie at its interface with Game Theory. On this front, however, gradient-descent based optimization methods have been less successful. Here, the role of single-objective optimization is played by equilibrium computation, but gradient-descent based methods commonly fail to find equilibria, and even computing local approximate equilibria has remained daunting. We shed light on these challenges through a combination of learning-theoretic, complexity-theoretic, game-theoretic and topological techniques, presenting obstacles and opportunities for Machine Learning and Game Theory going forward. I will assume no Deep Learning background for this talk and present results from joint works with S. Skoulakis and M. Zampetakis [2] as well as with N. Golowich and K. Zhang [1].

2012 ACM Subject Classification Theory of computation → Multi-agent learning; Theory of computation → Multi-agent reinforcement learning; Theory of computation → Solution concepts in game theory; Theory of computation → Exact and approximate computation of equilibria

Keywords and phrases Deep Learning, Multi-Agent (Reinforcement) Learning, Game Theory, Nonconvex Optimization, PPAD

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.2

Category Invited Talk

Funding This work is supported by NSF Awards CCF-1901292, DMS-2022448 and DMS2134108, a Simons Investigator Award, the Simons Collaboration on the Theory of Algorithmic Fairness, a DSTA grant, and the DOE PhILMs project (DE-AC05-76RL01830).

References

1 Constantinos Daskalakis, Noah Golowich, and Kaiqing Zhang. The complexity of Markov equilibrium in stochastic games. *arXiv preprint*, 2022. `arXiv:2204.03991`.

2 Constantinos Daskalakis, Stratis Skoulakis, and Manolis Zampetakis. The complexity of constrained min-max optimization. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1466–1478, 2021.

Some New (And Old) Results on Contention Resolution

Leslie Ann Goldberg
University of Oxford, UK

Abstract

This is an extended abstract of my talk at ICALP 2022, based on joint work with John Lapinskas.

2012 ACM Subject Classification Mathematics of computing

Keywords and phrases contention resolution, multiple access channel, randomised algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.3

Category Invited Talk

1 Extended Abstract

A *multiple access channel* is a shared channel which is used by processors to access a network, a cloud server, or another shared resource. The processors do not communicate except by listening to the channel. The mechanism of the channel is straightforward: If two or more processors send messages to the channel at the same time, then these messages collide, and are not delivered successfully. However, if during some time step exactly one processor sends a message, then this message is successfully delivered and the processor receives an acknowledgement of successful delivery. A *contention-resolution protocol* is a randomised algorithm that each processor uses to decide when to send its message to the channel (and when to wait because the channel is too busy).

We consider discrete-time contention resolution protocols in which processors communicate by sending discrete messages (packets) to a multiple access channel. During each step, new messages destined for the channel arrive at processors according to a probability distribution with an overall arrival rate λ. The whole process (consisting of the arrivals, the waiting messages, and the sends) can be viewed as a Markov chain. A contention-resolution protocol is *stable* [7] if this Markov chain is positive recurrent, which means that the process has a stationary distribution (so the expected build-up of waiting messages over time is bounded).

Two models of multiple access channels are the *queueing model* and the *queue-free model*. In the queueing model, there are N fixed processors. Each processor maintains a queue of messages which it is waiting to send – messages arrive at the tail of the queue and are sent from the head. The queues assist with stability, and are appropriate for applications with fixed networks. Dynamic networks (such as multiple access channels supporting cloud computing) are better captured by the queue-free model. In the queue-free model, we assume that processors join the network according to a probability distribution and that each processor has a single message to send. Thus, we identify the processor with its message – messages arrive at each step according to a Poisson distribution with rate λ and they stay in the system until they are successfully sent.

Two kinds of contention-resolution protocol are *full-sensing protocols* [12] which constantly listen the channel, obtaining partial information (such as which steps have no sends, which steps have exactly one send, which steps have a collision, or some combination of these) and *acknowledgement-based protocols*, which are appropriate for settings where this listening is not feasible. In acknowledgement-based protocols, the only information that a processor gets is whether its own sends are successful. A popular kind of acknowledgement-based protocol is

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 3; pp. 3:1–3:3


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a *backoff protocol*. A backoff protocol is associated with a send sequence $\mathbf{p} = p_0, p_1, \ldots$ where each p_i is interpreted as a probability in $(0, 1]$. During a given time step, a processor that is trying to send a message that has already had i collisions flips a coin – with probability p_i, it sends. With probability $1 - p_i$, it instead waits for a later time step. One of the best-known backoff protocol is *binary exponential backoff*, which has $p_i = 2^{-i}$ and is the basis for Ethernet [11].

The talk will mention some important recent results [13, 14, 3, 4] about full-sensing protocols, however the main focus will be on acknowledgement-based protocols and specifically on backoff protocols. In the queueing model, binary exponential backoff is known to be stable for sufficiently small arrival rate λ [7, 1]. Unfortunately, this value of λ depends on N and binary exponential backoff is unstable if λ is sufficiently large [8]. In a breakthrough result, Håstad, Leighton and Rogoff [8] showed that there is a stable backoff protocol for every $\lambda \in (0, 1)$ – in particular *polynomial backoff*, with $p_i = i^{-\alpha}$ for some $\alpha > 1$, is stable.

In the queue-free model, it is conjectured [2] that no backoff protocol is stable for any positive arrival rate λ. This question was raised by MacPhee [10] and the same is conjectured for all acknowledgement-based protocols. The conjecture for backoff protocols is known to be true when $\lambda \geq 0.42$ and the conjecture for acknowledgement-based protocols is known to be true when $\lambda \geq 0.531$ [5].

The evidence for the full conjecture concerning backoff protocols is that a backoff protocol is known to be unstable for all positive arrival rates if its send sequence $\mathbf{p} = p_0, p_1, \ldots$ decays smoothly. In particular, for the case where $1/p_j = o(c^j)$ for all $c > 1$, instability follows from work of Kelly and MacPhee [9] (which shows that in this case, with probability 1, only finitely many messages are sucessfully sent). In the case where $1/p_j = \Theta(c^j)$ for some $c > 1$ instability can be proved by generalising the proof of Aldous's seminal instability result for binary exponential backoff [2]. In the case where the send sequences has an infinite subsequence $p_{j_1}, p_{j_2}, \ldots$ satisfying $1/p_{j_k} = \omega(c^{j_k})$ for all $c > 1$ is also easily dealt with. This appears as a lemma in [6] but the method was also known to the authors of [5].

The main issue which makes it difficult to prove the full conjecture is proving instability for backoff protocols that have a send sequence that decays at an inconsistent rate or doesn't decay at all, with p_js that jump back and forth between large and small values as j increase. The main part of this talk describes new work [6] with John Lapinskas that solves this problem except in a special case. This special case has the property that the send sequences alternates between p_j's that are at least $\Omega(1)$ and p_j's that are exponentially small (but no smaller). Moreover, the large p_j's have density $1 - o(1)$ in $\{p_1, \ldots, p_n\}$ as $n \to \infty$. We therefore refer to them as "LCED" send sequences (for "largely constant with exponential decay"). These sequences are defined as follows.

▶ **Definition 1.** *A send sequence* $\mathbf{p}$ *is* LCED *("largely constant with exponential delay") if it satisfies the following properties:*

- **(i) "Largely constant":** *For all* $\eta > 0$*, there exists* $c > 0$ *such that for infinitely many* n*,* $|\{j \leq n : p_j > c\}| \geq (1 - \eta)n$.
- **(ii) "with exponential decay":** $\mathbf{p}$ *has an infinite subsequence* $(p_{\ell_1}, p_{\ell_2}, \ldots)$ *which satisfies* $\log(1/p_{\ell_x}) = \Theta(\ell_x)$ *as* $x \to \infty$.
- **(iii) "(but without super-exponential decay)":** $\log(1/p_j) = O(j)$ *as* $j \to \infty$.

The main Theorem of [6] is as follows.

▶ **Theorem 2** ([6])**.** *Let* $\mathbf{p}$ *be a send sequence which is not LCED. Then for every* $\lambda \in (0, 1)$ *the backoff protocol with arrival rate* λ *and send sequence* $\mathbf{p}$ *is unstable.*

The theorem has the following consequences.

▶ **Corollary 3** ([6]). *For every $\lambda \in (0,1)$ and every monotonically non-increasing send sequence $\mathbf{p} = p_0, p_1, \ldots$, the backoff protocol with arrival rate λ and send sequence $\mathbf{p}$ is unstable.*

▶ **Corollary 4** ([6]). *Let $\mathbf{p}$ be a send sequence. Let $m_{\mathbf{p}}(n)$ be the median of $p_0, \ldots, p_n$. Suppose that $m_{\mathbf{p}}(n) = o(1)$. Then for every $\lambda \in (0,1)$ the backoff protocol with arrival rate λ and send sequence $\mathbf{p}$ is unstable.*

The talk will conclude with a discussion of the prospects for proving the full conjecture.

References

1. Hesham Al-Ammal, Leslie Ann Goldberg, and Philip D. MacKenzie. An improved stability bound for binary exponential backoff. *Theory of Computing Systems*, 34(3):229–244, 2001.
2. David Aldous. Ultimate instability of exponential back-off protocol for acknowledgment-based transmission control of random access communication channels. *IEEE Transactions on Information Theory*, 33(2):219–223, 1987.
3. Michael A. Bender, Tsvi Kopelowitz, William Kuszmaul, and Seth Pettie. Contention resolution without collision detection. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 105–118. ACM, 2020.
4. Haimin Chen, Yonggang Jiang, and Chaodong Zheng. Tight trade-off in contention resolution without collision detection. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing*, pages 139–149. ACM, 2021.
5. Leslie Ann Goldberg, Mark Jerrum, Sampath Kannan, and Mike Paterson. A bound on the capacity of backoff and acknowledgment-based protocols. *SIAM Journal on Computing*, 33(2):313–331, 2004.
6. Leslie Ann Goldberg and John Lapinskas. Instability of backoff protocols with arbitrary arrival rates, 2022. doi:10.48550/ARXIV.2203.17144.
7. Jonathan Goodman, Albert G. Greenberg, Neal Madras, and Peter March. Stability of binary exponential backoff. *Journal of the ACM*, 35(3):579–602, 1988.
8. Johan Håstad, Frank T. Leighton, and Brian Rogoff. Analysis of backoff protocols for multiple access channels. *SIAM Journal on Computing*, 25(4):740–774, 1996.
9. Frank P. Kelly and Iain M. MacPhee. The number of packets transmitted by collision detect random access schemes. *Annals of Probability*, 15:1557–1568, 1987.
10. Iain M. MacPhee. *On Optimal Strategies in Stochastic Decision Processes*. PhD thesis, University of Cambridge, 1987.
11. Robert Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, 1976.
12. Jeannine Mosely and Pierre Humblet. A class of efficient contention resolution algorithms for multiple access channels. *IEEE Transactions on Communications*, 33(2):145–151, 1985.
13. Devavrat Shah and Jinwoo Shin. Randomized scheduling algorithm for queueing networks. *Annals of Applied Probability*, 22:128–171, 2012.
14. Devavrat Shah, Jinwoo Shin, and Prasad Tetali. Medium access using queues. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 698–707. IEEE Computer Society, 2011.

The Manifold Joys of Sampling

Yin Tat Lee
University of Washington, Seattle, WA, USA
Microsoft Research, Seattle, WA, USA

Santosh S. Vempala
Georgia Tech, Atlanta, GA, USA

Abstract

We survey recent progress and many open questions in the field of sampling high-dimensional distributions, with specific focus on sampling with non-Euclidean metrics.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Sampling, Diffusion, Optimization, High Dimension

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.4

Category Invited Talk

Funding This work was supported in part by NSF awards DMS-1839116, DMS-1839323, CCF-1909756, CCF-2007443 and CCF-2134105.

Acknowledgements We thank Yunbum Kook and Andre Wibisono for helpful comments.

1 Introduction

Sampling high-dimensional distributions is a fundamental problem of growing importance in machine learning and related fields [7, 17, 2]. Progress on efficient algorithms for sampling has led to new mathematical connections and insights into a number of areas such as probability, convex geometry and analysis. The focus of this tutorial is to survey the state-of-the-art for the most general results along with open problems. There are many interesting results for special cases that are beyond the scope of this survey.

We begin by stating the most general version of the problem in Euclidean space.

▶ **Definition 1** (Sampling Problem)**.** Given oracle access to an integrable, real-valued function $f : \mathbb{R}^n \to \mathbb{R}_+$, an initial point $x_0 \in \mathbb{R}^n$ with $f(x_0) > \beta \int f(y)dy$ and an error parameter $\varepsilon > 0$, output a point x from a distribution that is within total variation distance ε from the distribution with density proportional to $f(x)$.

A major discovery in the theory of algorithms is that the above problem can be solved in randomized polynomial time for any *logconcave* function f. Recall that a function $f : \mathbb{R}^n \to \mathbb{R}_+$ is logconcave if its logarithm is concave, i.e., for any $\lambda \in [0, 1]$ and any $x, y \in \mathbb{R}^n$,we have $f(\lambda x + (1-\lambda)y) \geq f(x)^\lambda f(y)^{1-\lambda}$. This class includes the important special cases of the uniform density over a convex body and a Gaussian restricted to a convex set. Note that we can alternatively think of a logconcave function as $e^{-f(x)}$ where now f is a convex function. As in optimization, the traditional frontier of polynomial-time algorithms for sampling has to do with convexity. In the past decade there has been progress on going beyond convexity, with appropriate weaker assumptions. Two approaches are (a) to show that isoperimetry of the target distribution suffices and (b) to use the convergence of the continuous time diffusion as a basis for proving the convergence of the discrete-time algorithm. We will illustrate these approaches later in this survey. Let us first introduce the main algorithms for the most general oracle setting.


Ball Walk

The ball walk with step-size parameter $\delta > 0$ is the following Markov Chain: at the current point x, pick a uniform random point y from the ball of radius δ centered at x; go to y with probability $\min\{1, f(y)/f(x)\}$. The starting point is chosen so that it satisfies $f(x_0) > 0$. When f is the indicator of a convex body K, this simply means that we start in K and at each step go to the proposed y only if y is also in K.

Hit-and-Run

The Hit-and-Run Markov chain is the following: at the current point x, pick a uniform random line ℓ through x, and go to a point y on the line with probability proportional to f restricted to ℓ. For a convex body, this means that we pick the next point uniformly from the chord in a random direction through the current point.

2 Mixing rates in the general oracle model

For a more detailed introduction to Markov chains in continuous state spaces, and these walks in particular, the reader is referred to [42, 36]. In this section by mixing rate we mean the rate to halve the χ^2-divergence between the current distribution and the target stationary distribution. We say that a distribution Q_0 is an M-warm start for a distribution Q if $\sup \frac{dQ_0(x)}{dQ(x)} \leq M$. A weak M-warm start is when $\chi^2(Q_0, Q) \leq M$. Recall that

$$\chi^2(P, Q) = \mathbb{E}_Q\left(\left(\frac{dP(x)}{dQ(x)} - 1\right)^2\right).$$

A key parameter in the analysis of a Markov chain is its conductance. For a Markov chain with state space K, transition kernel P and stationary distribution Q, the conductance of any measurable subset A of the state space is defined as

$$\phi(A) = \frac{\int_A P_x(K \setminus A) dQ(x)}{\min\{Q(A), Q(K \setminus A)\}}$$

and the conductance of the Markov chain itself is $\phi = \inf_A \phi(A)$. The conductance directly bounds the mixing rate.

▶ **Theorem 2** ([36]). *For a time-reversible Markov chain with conductance ϕ, the distribution after t steps satisfies*

$$\chi^2(Q_t, Q) \leq \left(1 - \frac{\phi^2}{2}\right)^t \chi^2(Q_0, Q).$$

The paper [36] developed conductance-based analysis of the convergence of Markov chains in the setting of continuous state spaces, including an important extension to the setting when the conductance $\phi(A)$ can only be bounded from below for sets A of measure larger than some threshold.

2.1 Ball walk

The ball walk has the following convergence guarantee for a convex body. We use $\widetilde{O}(\cdot)$ to suppress logarithmic terms.

▶ **Theorem 3** (Ball Walk from Warm Start in Convex Body [22]). *Let $K \subseteq \mathbb{R}^n$ be a convex body containing the unit Euclidean ball with $R^2 = \mathbb{E}_K\left(\|x - \overline{x}\|^2\right)$ for $\overline{x} = \mathbb{E}_K x$. Then the mixing time of the ball walk in K with step size $\delta = 1/\sqrt{n}$ from an $O(1)$-warm start is bounded by $\widetilde{O}(n^2R^2)$.*

This guarantee generalizes cleanly to any logconcave density.

▶ **Theorem 4** (Ball Walk from Warm Start for Logconcave Density [38]). *The mixing time of the ball walk for a target logconcave density ν such that the level set of measure $1/8$ contains a unit ball and $R^2 = \mathbb{E}_\nu\left(\|x - \overline{x}\|^2\right)$, with step size $\delta = 1/\sqrt{n}$ from an $O(1)$-warm start is bounded by $\widetilde{O}(n^2R^2)$.*

The bound above is tight up to polylogarithmic factors; a cylinder with a unit ball cross section and height R shows a lower bound of $\Omega(n^2R^2)$. Can we reduce or eliminate the dependence on R? Classical results tell us that for any convex body there is an affine transformation (the John position) that ensures that the body is sandwiched between balls of radii 1 and at most n. Since all we need to bound is the average squared distance, there is a natural transformation of space that gives a bound of $R = \sqrt{n}$.

▶ **Definition 5.** A distribution with density ν in $\mathbb{R}^n$ is said to be in *isotropic position* if a random point X drawn from ν satisfies $\mathbb{E}_\nu(x) = 0$ and $\mathbb{E}_\nu(xx^\top) = I$.

Note that in isotropic position, $R^2 = \mathbb{E}(\|x\|^2) = n$ and this implies a bound of n^3 on the mixing rate for a logconcave distribution in isotropic position. As it turns out, it is possible to show a better convergence rate, where the dependence on diameter is essentially eliminated. The theorems below work also for *near*-isotropic position, by which we mean that the eigenvalues of the covariance matrix are bounded below and above by constants (rather than all being equal to 1).

▶ **Theorem 6** (Ball Walk from Warm Start with Isotropic Target). *The mixing time of the ball walk applied to a logconcave density in isotropic position with step size $\delta = 1/\sqrt{n}$ from an $O(1)$-warm start is bounded by $\widetilde{O}(n^2)$.*

The above theorem deserves some explanation and is the culmination of a quarter century of progress on geometric isoperimetric inequalities. First, we state the corresponding theorem which establishes the rate of convergence in terms of the *isoperimetry of the target distribution.* For this we define the KLS constant.

▶ **Definition 7.** For a density ν in $\mathbb{R}^n$, the KLS constant of ν is defined as

$$\frac{1}{\psi_\nu} = \inf_{S \subset \mathbb{R}^n} \frac{\nu(\partial S)}{\min\{\nu(S), 1 - \nu(S)\}}$$

and the KLS constant for logconcave densities in $\mathbb{R}^n$ is $\psi_n = \sup\{\psi_\nu : \nu \text{ isotropic logconcave in } \mathbb{R}^n\}$.

We will discuss this constant and its significance presently. The following theorem [22] shows how it bounds the mixing rate of the ball walk.

▶ **Theorem 8** (Ball Walk and KLS [22]). *The mixing time of the ball walk applied to a logconcave density in isotropic position with step size $\delta = 1/\sqrt{n}$ from an $O(1)$-warm start is bounded by $\widetilde{O}(n^2\psi_\nu^2)$.*

Kannan, Lovász and Simonovits [21] conjectured that $\psi_n = O(1)$ for any logconcave isotropic density ν. Recently, Klartag and Lehec [25], following a line of work [21, 13, 16, 12, 32, 3], proved that $\psi_n = O(\log^5 n)$, which implies Theorem 6 above. Another way to state the underlying isoperimetric inequality is the following.

▶ **Theorem 9** (Euclidean Isoperimetry). *For any logconcave density ν in $\mathbb{R}^n$ whose covariance matrix A has largest eigenvalue λ_1 and any two disjoint subsets S_1, S_2 of $\mathbb{R}^n$ we have*

$$\nu(K \setminus S_1 \setminus S_2) \geq \frac{d(S_1, S_2)}{\psi_n \sqrt{\lambda_1}} \min\{\nu(S_1), \nu(S_2)\}$$

where $d(,)$ is Euclidean distance and ψ_n is known to be $O(\log^5 n)$ [25] and conjectured to be $O(1)$ [21].

An equivalent geometric way to state the KLS conjecture is that a hyperplane-induced subset achieves the minimum isoperimetric coefficient up to a universal constant factor. For a detailed discussion of the KLS conjecture, we refer the reader to [31]. Its full resolution remains an intriguing and fruitful open problem.

More recently, it has been shown that the complexity of isotropic transformation and volume computation can be reduced to the KLS constant.

▶ **Theorem 10** (Rounding and Volume Computation [8, 19]). *Given a convex body given in the membership oracle model, it can be brought into near-isotropic position in $\widetilde{O}(n^3\psi_n^2)$ queries and its volume can be computed to within relative error ε using $\widetilde{O}(n^3/\varepsilon^2)$ additional queries.*

We conclude this section with two questions about the analysis of the ball walk, which both seem to require the development of new tools. The first has to do with the convergence of the ball walk without a warm start. Although this does not converge quickly in general (e.g., when starting near a corner), it is conceivable that it does when started from a "nice" point. After all, the result for a warm start effectively states that "most" points are good starting points. But can we get our hands on one of them?

▶ **Question 11.** *Show that the ball walk started from the centroid of a convex body converges to its stationary distribution in polynomial time.*

The second question has to do with verifying that the ball walk has converged. This has value both theoretically and practically (to test convergence on an instance-by-instance basis rather than simply running up to the worst-case bound).

▶ **Question 12.** *Consider the ball walk in a convex body starting from a logconcave initial distribution (e.g., uniform in a ball contained inside the body). Show that the distance between the current distribution and the target can be bounded as a polynomial in the dimension and the distance to stationarity of a random one-dimensional marginal of the current distribution.*

A clean variant of this question, first proposed by Lovász, is whether the expected squared distance of a random point X_t of the ball walk from its starting point X_0 is a monotonic increasing function.

2.2 Hit-and-Run

One advantage of hit-and-run is that there is no step-size parameter. A more important property is that hit-and-run converges rapidly from *any* starting distribution (even a single point), unlike the ball walk. To see that the ball walk does not, consider starting the ball

walk near a corner, e.g., near a vertex of a cube; then the probability of making a proper step (one that moves to a different point) can be arbitrarily small. Another way to see this is to note that the conductance of small sets for the ball walk Markov chain cannot, in general, be bounded from below. For hit-and-run, the picture is dramatically different – every subset can be shown to have large conductance! In particular, hit-and-run started from an arbitrary point in the interior (e.g., close to a corner) will quickly escape the corner.

▶ **Theorem 13** (Conductance of Hit-and-Run [37]). *The conductance of hit-and-run for any convex body in $\mathbb{R}^n$ containing a unit ball and contained in a ball of radius R is $\Omega(1/(nR))$.*

As a consequence it has the same general mixing rate as the ball walk, except that the dependence on the warm start parameter is logarithmic rather than polynomial, and in particular starting from any point x_0 in the interior of K, it is logarithmic in the inverse of the distance of x_0 to the boundary of K. It is conjectured (and observed in practice [17, 2]) that Hit-and-Run also mixes in n^2 steps for a body/logconcave distribution in isotropic position. However, it is an open problem to analyze this and show a bound better than n^3 as implied by the analysis in general position.

▶ **Question 14.** *Analyze the mixing rate of Hit-and-Run for a logconcave density or convex body in isotropic position.*

To understand the difficulty of answering this question, it is useful to see the isoperimetric inequality underlying the conductance bound for hit-and-run. This is our first departure from Euclidean distance, and an indication that the natural underlying geometry is different. For two points u, v in a convex body K, we define the *cross-ratio* distance as follows. Let p, q be the endpoints of the chord induced by u, v inside K, so that their order along the chord is p, u, v, q. Then,

$$d_K(u, v) = \frac{\|u - v\| \, \|p - q\|}{\|p - u\| \, \|v - q\|}.$$

While this distance is not a true distance in that it does not satisfy the triangle inequality, the closely related Hilbert distance, $d_H(u, v) = \ln(1 + d_K(u, v))$, is a metric. We have the following isoperimetric inequality.

▶ **Theorem 15** (Cross-ratio/Hilbert Isoperimetry [35, 38]). *For any logconcave density ν in $\mathbb{R}^n$ whose support is a convex body K and any two disjoint subsets S_1, S_2 of $\mathbb{R}^n$ we have*

$$\nu(K \setminus S_1 \setminus S_2) \geq d_K(S_1, S_2)\nu(S_1)\nu(S_2).$$

This isoperimetric inequality is affine-invariant and already tight. So it is unclear how to derive a more refined inequality that takes advantage of isotropic position.

▶ **Question 16.** *Is there a corresponding KLS conjecture that would imply a better mixing rate for hit-and-run?*

2.3 A bottleneck

For both the ball walk and hit-and-run, the mixing rate from a warm start for an isotropic target is $\Omega(n^2)$; this can be seen for a hypercube. The bottleneck for the ball walk is that the largest value that the step-size parameter δ can be set to is $O(1/\sqrt{n})$. This is because setting it to a larger value leads to most steps being rejected, i.e., WHP proposed steps are outside the body. In the case of hit-and-run, we have the same bottleneck. This is because the length of a random chord through a random point in a hypercube is still only $O(1/\sqrt{n})$, so this is the effective step-size on average.

3 In search of a larger step: the Dikin walk

For the walks discussed so far, the step-size is limited by points near the boundary. This suggests the idea of taking larger steps from points that are deeper inside, e.g., at the current point, use the largest ball that has a constant probability of a random step being accepted. So points deeper inside would take larger steps. However, this has the issue that the resulting stationary distribution is no longer uniform. To correct this, one can use a Metropolis filter based on the ratios of the volumes of the balls at the current point and the proposed point. Unfortunately, this means that this effective radius or step-size must change slowly. A smoother and more elegant approach is inspired by the seminal work of Dikin for convex optimization [10]. To optimize a linear function over a polytope, he proposed constructing a "maximal" ellipsoid around the current point that would be contained in the body, taking a large, objective-improving step within this ellipsoid and repeating. This was the first "interior-point" paradigm (now called Affine Scaling); it predates the central path interior-point method pioneered by Karmarkar [24] and generalized by Nesterov and Nemirovskii [41]. For a point x in the polytope defined by $\{Ax \geq b\}$, the Dikin ellipsoid is defined as follows:

$$E(x,r) = \left\{y : (y-x)^\top \mathbf{H}(x)(y-x) \leq r^2\right\} \qquad \text{where} \qquad \mathbf{H}(x) = \sum_{i=1}^{m} \frac{a_i a_i^\top}{(a_i^\top x - b_i)^2}$$

with a_i being the i'th row (normal vector) of the constraint matrix A for $i = 1, \ldots, m$. This ellipsoid adapts to the local geometry of the polytope, and $E(x,1)$ always fully contained in it. It suggests the following Dikin random walk: at the current point x, pick a random point y in the Dikin ellipsoid $E(x,r)$; if $x \in E(y,r)$, go to y with probability $\min\{1, \mathrm{vol}(E(x,r))/\mathrm{vol}(E(y,r))\}$. Kannan and Narayanan [23] analyzed its mixing rate with a suitable choice of the radius r.

▶ **Theorem 17** (Dikin in Polytope [23]). *The mixing rate of the Dikin walk from a warm start in a polytope in $\mathbb{R}^n$ given by m inequalities is bounded by $O(mn)$.*

Notably, the mixing rate of the walk *does not* depend on the affine position of the body – the process is affine-invariant, i.e., applying the walk to an affine transformation of the input body is the same as applying the transformation to the output of the process on the same body. This means in particular that one can assume for free that the polytope is in isotropic position. This avoids dependence on the outer ball radius parameter R in the previous section.

What should the step-size r be? It is chosen to be $O(1/\sqrt{n})$ and this turns out to be necessary to ensure that the rejection probability is not too high, i.e., that the volumes of Ellipsoids corresponding to nearby points are within a constant factor. The change in the ellipsoid can be bounded using the classical optimization property of *self-concordance*, i.e., when the matrix function $\mathbf{H}(x)$ is the Hessian of a convex function. Self-concordance is a key property in the analysis of the interior-point method for linear programming [41]. In fact, the ellipsoid matrix $\mathbf{H}(x)$ is the Hessian of the convex log barrier function at x,

$$\phi(x) = -\sum_{i=1}^{m} \ln(a_i^\top x - b_i),$$

i.e., $\mathbf{H}(x) = D^2\phi(x)$. So the log-barrier interior-point method for optimization corresponds to the Dikin walk for sampling!

To make this connection a bit more precise, let us define the norm of a vector v induced by a matrix function $\mathbf{H}$ as $\|v\|_x^2 = v^\top \mathbf{H}(x) v$.

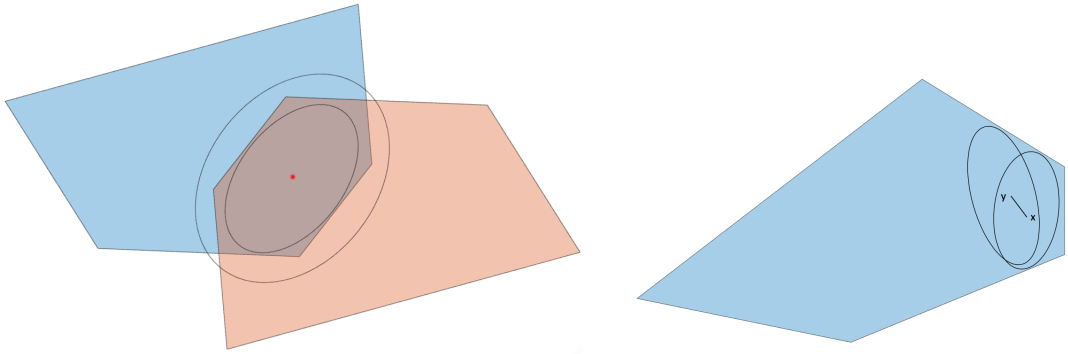


Figure 3.1 (a) $E_u(1) \subseteq K \cap (2u - K) \subseteq E_u(\sqrt{\nu})$. (b) (Strong) self-concordance measures the rate of change of Hessian.

▶ **Definition 18.** For a convex set $K \subset \mathbb{R}^n$, we say a matrix function $\mathbf{H} : K \to \mathbb{R}^{n\times n}$ is *self-concordant* if for any $x \in K$, we have

$$\left\|\mathbf{H}(x)^{-1/2} D\mathbf{H}(x)[h]\mathbf{H}(x)^{-1/2}\right\|_{\text{op}} \leq 2\|h\|_x$$

where $D\mathbf{H}(x)[h]$ is the directional derivative of $\mathbf{H}$ at x in the direction h, i.e., $D\mathbf{H}(x)[h] = \frac{d}{dt}\mathbf{H}(x + th)$. We say $\mathbf{H}$ is symmetric ν-self-concordant if $\mathbf{H}$ is self-concordant and for any $x \in K$,

$$E(x, 1) \subseteq K \cap (2x - K) \subseteq E(x, \sqrt{\nu}).$$

Self-concordance relates the change in $\mathbf{H}$ to the change in x. Many natural self-concordant barriers, including the logarithmic barrier, satisfy a much stronger condition, replacing the operator norm above by the Frobenius norm. While this makes no difference for the worst-case bound for optimization, it turns out to be crucial for sampling.

▶ **Definition 19** (Strong Self-Concordance)**.** For a convex set $K \subset \mathbb{R}^n$, we say a matrix function $\mathbf{H} : K \to \mathbb{R}^{n\times n}$ is *strongly self-concordant* if for any $x \in K$, we have

$$\left\|\mathbf{H}(x)^{-1/2} D\mathbf{H}(x)[h]\mathbf{H}(x)^{-1/2}\right\|_{F} \leq 2\|h\|_x .$$

The canonical barrier [18] satisfies strong self-concordance. The situation with two other classical barriers, namely the universal and entropic barriers has a curious connection.

▶ **Lemma 20** ([27])**.** *Let $\mathbf{H}(x)$ be the Hessian of the universal or entropic barriers. Then,*

$$\left\|\mathbf{H}(x)^{-1/2} D\mathbf{H}(x)[h]\mathbf{H}(x)^{-1/2}\right\|_{F} = O(\psi_n)\|h\|_x .$$

In fact, up to a logarithmic factor, the strong self-concordance of these barriers is *equivalent* to the KLS conjecture.

▶ **Lemma 21.** *Given any strongly self-concordant matrix function $\mathbf{H}$ on $K \subset \mathbb{R}^n$. For any $x, y \in K$ with $\|x - y\|_x < 1$, we have*

$$\|\mathbf{H}(x)^{-\frac{1}{2}}(\mathbf{H}(y) - \mathbf{H}(x))\mathbf{H}(x)^{-\frac{1}{2}}\|_F \leq \frac{\|x - y\|_x}{(1 - \|x - y\|_x)^2}.$$

The following guarantee was shown for the convergence of the generalized Dikin walk in [27].

▶ **Theorem 22** (Convergence of general Dikin walk [27]). *The mixing rate of the Dikin walk for a symmetric, strongly self-concordant matrix function with convex log determinant is* $O(n\bar{\nu})$.

This implies faster mixing and sampling for polytopes using the LS barrier [28], which is strongly self-concordant, has a convex log determinant and has $\bar{\nu} = O(n \log^3 m)$.

▶ **Theorem 23** (Quadratic Convergence of Dikin [27]). *The mixing rate of the Dikin walk based on the LS barrier for any polytope in* $\mathbb{R}^n$ *is* $\tilde{O}(n^2)$ *and each step can be implemented in* $\tilde{O}(mn^{\omega-1})$ *arithmetic operations where* ω *is the matrix multiplication exponent.*

We note that the Dikin walk with the logarithmic barrier for a polytope $\{\mathbf{A}x \geq b\}$ can be implemented in time $O(\text{nnz}(\mathbf{A}) + n^2)$ per step while maintaining the mixing rate of $O(mn)$.

The isoperimetry of the metric induced by the matrix function $\mathbf{H}$ follows by simply connecting it to the cross-ratio distance.

▶ **Lemma 24.** *For* $u, v \in K$, $d_K(u, v) \geq \frac{\|u-v\|_u}{\sqrt{\bar{\nu}}}$.

From this lemma and Theorem 15, we have that the isoperimetric coefficient for the Hessian norm distance is $\Omega(1/\sqrt{\nu})$. This bound, as well as the bound of mn for the Dikin walk with the log barrier are tight as shown by a hypercube with one of its facets duplicated $m - n$ times. However, the situation is far from clear for the weighted Dikin walk, where the log barrier is replaced by one that weights each constraint, as in the LS barrier (and effectively eliminates this bad example).

▶ **Question 25.** *What is the isoperimetric coefficient of the Hessian distance induced by the LS barrier? Does the corresponding weighted Dikin walk mix in* $\widetilde{O}(n)$ *steps?*

4 Large steps via non-Euclidean geometries

The Dikin walk and its weighted version are based on exploiting *local* geometry, where the norm is defined locally. As a direct consequence, this family of algorithms is affine-invariant. However, to ensure the correct stationary distribution and keep the probability of acceptance in the Metropolis filter reasonably large, the effective step-size is again $1/\sqrt{n}$, and the convergence rate is n^2. Is a larger step-size/smaller convergence rate possible?

To understand this, we delve further into the use of non-Euclidean geometry. So far, our random walks have only taken straight line steps in Euclidean space. The distribution of the direction of the next step depends on the current point in the case of the Dikin walk, but the step itself is still a straight line. A more drastic departure would be to use curves instead of straight lines. How?

First note that given a local metric such as the one induced by the Hessian of a convex function defines a manifold and using this metric we can define the length of a curve (a continuous path) and the distance between any two points (as the length of the shortest curve between them). Since the local metric varies, the shortest path between two points can be a curve (e.g., consider flight paths for the sphere metric). In the context of sampling polytopes, we consider the metric defined by the Hessian of the log barrier. The support of this manifold is the original polytope. Distances between points are magnified as the points get closer to the boundary. So geodesic paths tend to avoid the boundary!

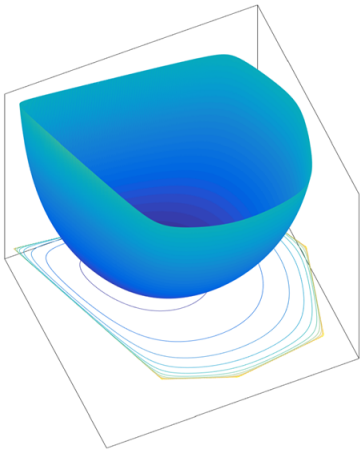


Figure 4.1 The Hessian manifold induced by the log barrier in a polytope.

How do we pick a random next step/direction? For this we can use the notion of tangent space attached to every point in a Riemannian manifold, and pick a random (Gaussian) vector in the tangent space. Then we go along the geodesic (locally shortest path) in the direction of the chosen vector along the manifold. This is the curved step. We make these notions more precise below. The main intuition for considering such a generalization is the possibility of taking larger steps unimpeded by the boundary. This family of walks have been called "geodesic" walks [31]. We will shortly see an even more natural variant. For some quick background on Riemannian geometry, we refer the reader to the appendix.

4.1 Geodesic walks

Consider an explicit polytope P and a manifold M with support P and metric induced by the Hessian of the log barrier in P. Moving to this manifold view allows us to avoid the constraint of small steps near the boundary, as there is no longer a hard boundary constraint. How large can we make the step size? This is limited by another factor, namely, when we take large steps, the filter acceptance probability (to maintain the desired target distribution) can become very small. Could we possibly avoid using a filter? To answer this, we first consider the continuous time limit of the corresponding "diffusion" process.

$$dx_t = \mu(x_t)dt + \left(2D^2\phi(x_t)\right)^{-1/2} dW_t.$$

The first term, called the *drift*, is given by

$$\mu_i(x_t) = \sum_{j=1}^{n} \frac{\partial}{\partial x_j} \left(\left(\nabla^2\phi(x_t)\right)^{-1}\right)_{ij} = D \cdot (D^2\phi(x_t)^{-1})_i$$

where $D\cdot$ is the divergence operator. The drift term is a deterministic vector field biasing the walk locally. Its purpose is to ensure that the process converges to the desired target uniform distribution. Notably, in the continuous setting, it replaces the Metropolis filter. The precise form of the drift can be derived using the Fokker-Planck equation.

▶ **Theorem 26** (Fokker-Planck equation). *For any stochastic differential equation (SDE) of the form*

$$dx_t = \mu(x_t)dt + \sqrt{H(x_t)}dW_t,$$

for a symmetric matrix function H, the probability density of the SDE is given by the diffusion equation

$$\frac{\partial}{\partial t}p(x) = -\sum_{i=1}^{n}\frac{\partial}{\partial x_i}[\mu_i(x)p(x)] + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2}{\partial x_i x_j}[H_{ij}(x)p(x)].$$

In discrete time, we consider the following *geodesic walk*:

$$x^{(j+1)} = \exp_{x^{(j)}}(\sqrt{h}w + \frac{h}{2}\mu(x^{(j)})), \tag{4.1}$$

where $\exp_{x^{(j)}}$ is the exponential map from the tangent space at $x^{(j)}$, $T_{x^{(j)}}M$, back to the manifold, w is a random Gaussian vector in the tangent space $T_{x^{(j)}}M$, $\mu(x^{(j)}) \in T_{x^{(j)}}M$ is the drift term and h is the step size. The Gaussian vector w has mean 0 and variance 1 in the metric at x, i.e., for any u, $\mathbb{E}_w\langle w, u\rangle_x^2 = \|x\|_x^2$. We write it as $w \sim N_x(0, I)$. This discrete walk converges to continuous diffusion as $h \to 0$ and it converges at a rate faster than the walk suggested by Euclidean coordinates, namely, $x^{(j+1)} = x^{(j)} + \sqrt{h}w + h\mu(x^{(j)})$.

Implementing this discretization leads to substantial challenges. The stationary distribution of the geodesic walk is not uniform. To get around this issue, we use rejection sampling. For step-size h chosen in advance, let $p(x \xrightarrow{w} y)$ be the probability density of going from x to y using the local step w.

Algorithm 1 Geodesic Walk.

At current point x:
Pick a Gaussian random vector $w \sim N_x(0, I)$.
Compute $y = \exp_x(\sqrt{h}w + \frac{h}{2}\mu(x))$.
Let $p(x \xrightarrow{w} y)$ be the probability density of going from x to y using the above step w.
Compute a corresponding w' s.t. $x = \exp_y(\sqrt{h}w' + \frac{h}{2}\mu(y))$.
With probability $\min\left(1, \frac{p(y \xrightarrow{w'} x)}{p(x \xrightarrow{w} y)}\right)$, go to y; otherwise, stay at x.

In this algorithm, computing the exponential map and computing the transition probability density are nontrivial steps that need the efficient solution of an ODE. Even though the walk uses a Metropolis filter in the end, the parameter h can be made as large as $n^{-3/4}$, and the overall mixing time is $O(m/h)$, leading to the first sub-quadratic mixing rate for sampling polytopes. Note that this is effectively a step-size of $\sqrt{h} = n^{-3/8} \gg n^{-1/2}$, the limitation of previous methods.

▶ **Theorem 27** (Convergence of Geodesic Walk [29]). *The geodesic walk in a polytope with the log barrier converges to the uniform density in the polytope in $O(mn^{3/4})$ steps and each step can be implemented in $\widetilde{O}(mn^{\omega-1})$ arithmetic operations.*

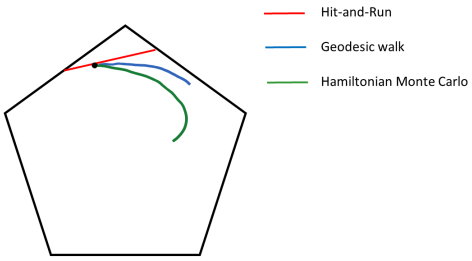


Figure 4.2 An illustration of Hit-and-run/Dikin, Geodesic walk and Hamiltonian Monte Carlo.

4.2 Riemannian Hamiltonian Monte Carlo

In the geodesic walk inspired by diffusion, in each step we choose a random direction and the drift along the entire step is determined by the (current) initial point. An even more natural discretization is to let the drift evolve along the trajectory in the chosen direction. The purpose of this modification would be to maintain the stationarity in spite of going to discrete time without introducing an explicit Metropolis filter. Can this be done? The answer is the method known as Hamiltonian Monte Carlo [40], and its manifold generalization [15]. To sample from a general distribution $e^{-H(x,y)}$. Hamiltonian Monte Carlo uses curves instead of straight lines in a time-reversible manner even if the target distribution is uniform.

▶ **Definition 28.** Given a continuous, twice-differentiable function $H : \mathcal{M} \times \mathbb{R}^n \subset \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ called the *Hamiltonian*, where $\mathcal{M}$ is the x domain of H, we say $(x(t), y(t))$ follows a *Hamiltonian curve* if it satisfies the *Hamiltonian equations*

$$\begin{aligned} \frac{dx}{dt} &= \frac{\partial H(x,y)}{\partial y}, \\ \frac{dy}{dt} &= -\frac{\partial H(x,y)}{\partial x}. \end{aligned} \tag{4.2}$$

We define the map $T_\delta(x, y) \stackrel{\text{def}}{=} (x(\delta), y(\delta))$ where the $(x(t), y(t))$ follows the Hamiltonian curve with the initial condition $(x(0), y(0)) = (x, y)$.

Hamiltonian Monte Carlo is a sequence of randomly generated Hamiltonian curves.

Algorithm 2 Hamiltonian Monte Carlo.

Input: some initial point $x^{(1)} \in \mathcal{M}$.
for $k = 1, 2, \cdots, T$ **do**
  Sample $y^{(k+\frac{1}{2})}$ according to $e^{-H(x^{(k)},y)}/\pi(x^{(k)})$ where $\pi(x) = \int_{\mathbb{R}^n} e^{-H(x,y)}dy$.
  With probability $\frac{1}{2}$, set $(x^{(k+1)}, y^{(k+1)}) = T_\delta(x^{(k)}, y^{(k+\frac{1}{2})})$.
  Otherwise, $(x^{(k+1)}, y^{(k+1)}) = T_{-\delta}(x^{(k)}, y^{(k+\frac{1}{2})})$.
end
Output: $(x^{(T+1)}, y^{(T+1)})$.

▶ **Lemma 29.** *HMC has the following properties:*

1. **Energy conservation.** *For any Hamiltonian curve* $(x(t), y(t))$,

$$\frac{d}{dt}H(x(t), y(t)) = 0.$$

2. **Measure preservation.** *For any $t \geq 0$,*

$$\det\left(DT_t(x,y)\right) = 1$$

where $DT_t(x,y)$ is the Jacobian of the map T_t at the point (x,y).

3. **Time reversibility.** *For $\pi(x) = \int_{\mathbb{R}^n} e^{-H(x,y)}dy$, the probability density $p_x(x')$ of one step of HMC starting at x satisfies $\pi(x)p_x(x') = \pi(x')p_{x'}(x)$.*

Everything so far can be generalized to manifolds. To sample from the distribution $e^{-f(x)}$, we define

$$H(x,v) \stackrel{\text{def}}{=} f(x) + \frac{1}{2}\log((2\pi)^n \det g(x)) + \frac{1}{2}v^T g(x)^{-1}v \tag{4.3}$$

where g is the local metric of the manifold. In the Euclidean case, $g(x) = I$, and the Euclidean Hamiltonian is $H(x,v) = f(x) + \frac{1}{2}\|v\|^2$ and the corresponding dynamics becomes just $\frac{d^2x}{dt^2} = -\nabla f(x)$. One can view x as the position and v as the velocity. The convergence of HMC has been intensively studied in recent years, starting with [39], which established a mixing rate of $O(\kappa^2)$ for the case when f is strongly logconcave. For this setting it is now known that the optimal rate is $O(\kappa)$ [4].

We now restate the HMC equation for manifolds, as Riemannian HMC, noting that derivatives have to take the local metric into account.

▶ **Lemma 30.** *In Euclidean coordinates, the Hamiltonian equation for (4.3) can be rewritten as*

$$\begin{aligned} D_t\frac{dx}{dt} =& \mu(x), \\ \frac{dx}{dt}(0) \sim& N(0, g(x)^{-1}) \end{aligned}$$

where $\mu(x) = -g(x)^{-1}Df(x) - \frac{1}{2}g(x)^{-1}\mathrm{Tr}\left[g(x)^{-1}Dg(x)\right]$, and D_t is the covariant derivative (or Levi-Civita connection) on the manifold $\mathcal{M}$ with metric g.

With the above set-up, as long as we can compute the RHMC ODE accurately and efficiently, there is no need for a filtering step, leading to a significantly simpler implementation than that of the geodesic walk. In fact, it also gives a slightly improved mixing rate by allowing a larger time step in each iteration.

▶ **Theorem 31** (Convergence of RHMC [30])**.** *Riemannian Hamiltonian Monte Carlo in a polytope with the log barrier converges to the uniform density in the polytope in $O(mn^{2/3})$ steps.*

▶ **Question 32.** *Can the step-size of RHMC be improved so that the mixing rate is $O(m\sqrt{n})$? Can the mixing rate be improved to $\widetilde{O}(n^{1.5})$ with a weighted log barrier?*

RHMC appears to be quite practical, with recent implementations being able to sample from constrained distributions in dimension as high as 10^5 [26].

5 From diffusion to sampling: the manifold perspective

Langevin Diffusion (LD) is the following stochastic process[1]:

$$dX_t = -Df(X_t)dt + \sqrt{2}dB_t$$

[1] In this section, we use D for Euclidean derivative and reserve ∇ for manifold derivative.

where B_t is the standard Wiener process. The stationary distribution of this process is the density $\nu(x)$ proportional to $e^{-f(x)}$, a fact that can be verified using the Fokker-Planck equation for the time derivative of the density of X_t. To understand the rate of convergence, we recall two basic notions. First to measure the distance between distributions, we use the relative entropy (or KL divergence), defined as follows:

$$H_\nu(\rho) = \int \rho(x) \log \frac{\rho(x)}{\nu(x)}\, dx = \mathbb{E}_\rho \left(\log \frac{\rho}{\nu} \right).$$

In continuous time, the convergence will depend on what kind of isoperimetry is satisfied by the target density. So far we have seen Cheeger isoperimetry, which results in convergence in the χ^2-divergence. A stronger notion of isoperimetry is given by the Log-Sobolev Inequality. The relative Fisher information is defined as

$$J_\nu(\rho) = \int \rho \|\nabla \log(\frac{\rho}{\nu})\|^2 dx.$$

Note that this definition is for any Riemannian manifold.

▶ **Definition 33.** We say the distribution ν satisfies a Log-Sobolev inequality with constant α (called the log-Sobolev constant) if for every measure with density ρ we have

$$H_\nu(\rho) \leq \frac{1}{2\alpha} J_\nu(\rho).$$

For logconcave densities in $\mathbb{R}^n$, an equivalent definition up to absolute constants is the following:

$$\sqrt{\alpha} \simeq \inf_{S \subseteq \mathbb{R}^n,\, \nu(S) \leq \frac{1}{2}} \frac{\nu(\partial S)}{\nu(S)\sqrt{\log(1/\nu(S))}}$$

which looks like the Cheeger constant except for the additional log factor in the denominator – LSI requires the isoperimetric coefficient to get larger as the measure of the subset gets smaller.

A distribution that satisfies LSI has a strong convergence property in continuous time.

▶ **Theorem 34.** $H_\nu(\rho_t) \leq e^{-2\alpha t} H_\nu(\rho_0)$.

This statement bears a remarkable resemblance to convergence of Gradient Descent to the optimal solution for strongly convex functions. In fact, as noted by Wibisono [45] (see also [20]), *Langevin Diffusion is Gradient Flow in the space of measures with the Wasserstein metric and the objective being the KL-divergence to the target density.*

We discuss this in a bit more detail below, starting from the simple optimization perspective.

▶ **Lemma 35.** *Let F be a function satisfying "Gradient Dominance":*

$$\|\nabla F(x)\|^2 \geq 2\alpha(F(x) - \min_x F(x)).$$

Then, the deterministic process $dx_t = -\nabla F(x_t)dt$ converges exponentially, i.e.,

$$F(x_t) - \min F \leq e^{-2\alpha t}(F(x_0) - \min F).$$

.

Proof. We write

$$\begin{aligned}\frac{d}{dt}(F(x_t) - \min_x F(x)) &= \langle \nabla F(x_t), \frac{dx_t}{dt} \rangle \\ &= -\left\|\nabla F(x_t)\right\|^2 \\ &\leq -2\alpha \cdot (F(x_t) - \min_x F(x)).\end{aligned}$$

The conclusion follows. ◀

For $f : \mathbb{R}^n \to \mathbb{R}_+$, let ν be the density proportional to $e^{-f(x)}$ and $F(\rho) = H_\nu(\rho)$. We will apply the above lemma to this KL-divergence objective.

▶ **Lemma 36.** *For any two densities ρ, ν,*

$$\|\nabla_\rho H_\nu(\rho)\|_\rho^2 = \int \rho(x) \left\| D \log \frac{\rho(x)}{\nu(x)} \right\|^2 dx = J_\nu(\rho).$$

▶ **Theorem 37.** *Let f be a differentiable function with log-Sobolev constant α. Then the Langevin dynamics*

$$dx_t = -Df(x)dt + \sqrt{2}dW_t$$

converges exponentially in KL-divergence to the density $\nu(x) \propto e^{-f(x)}$ with mixing rate $O(1/\alpha)$, i.e., $H_\nu(\rho_t) \leq e^{-2\alpha t} H_\nu(\rho_0)$.

Proof. First, note that the Langevin SDE, by Fokker-Planck, corresponds to the following PDE:

$$\begin{aligned}\frac{d\rho(x)}{dt} &= D \cdot (\rho(x) Df) + \Delta\rho(x) \\ &= D \cdot \left(\rho(x) D \log \frac{\rho(x)}{\nu(x)} \right) \\ &= -\nabla_\rho H_\nu(\rho).\end{aligned}$$

Note that the last step above refers to the derivative with respect to the Wassterstein metric (see [44]). Next, we note that since ν satisfies the log-Sobolev inequality, $F(\rho) = H_\nu(\rho)$ satisfies the Gradient Dominance condition with parameter α. The theorem follows from Lemma 35. ◀

So far, we have only used the log-Sobolev inequality, not convexity or related properties. Will this suffice for an efficient algorithm? It was shown in [43] that this is indeed the case. The Unadjusted Langevin Algorithm (ULA) is the simple Euler discretization of Langevin dynamics, namely,

$$X_{k+1} = X_k - \delta \cdot Df(X_k) + \sqrt{2\delta} Z$$

where $Z \sim N(0, I)$ is standard Gaussian. The next theorem is for $\mathbb{R}^n$.

▶ **Theorem 38** (ULA converges under Isoperimetry [43])**.** *Assume $\nu = e^{-f}$ satisfies LSI with constant $\alpha > 0$ and Df is L-Lipschitz. Then ULA with step size $0 < \delta \leq \frac{\alpha}{4L^2}$ satisfies*

$$H_\nu(\rho_k) \leq e^{-\alpha\delta k} H_\nu(\rho_0) + \frac{8\delta n L^2}{\alpha}.$$

For any $0 < \varepsilon < 4n$, ULA with step size $\delta \leq \frac{\alpha\varepsilon}{16nL^2}$ reaches error $H_\nu(\rho_k) \leq \varepsilon$ after at most $k \geq \frac{1}{\alpha\delta} \log \frac{2H_\nu(\rho_0)}{\varepsilon}$ iterations.

Due to the simplicity and generality of the approach, Langevin Algorithms have been intensively studied in recent years for many special cases and under various additional conditions on the input [9, 5, 11, 34, 46, 6]. Here we consider an extension: what is the generalization of Langevin diffusion from Euclidean space to more general metrics?

▶ **Definition 39** (Riemannian Langevin Diffusion (RLD))**.** Let $\mathcal{M}$ be a manifold with metric g and measure $dv_g(x)$, whose density with respect to the Lebesgue measure is $\sqrt{|\det(g)|}$. For a distribution with density $\nu = e^{-F}$, Riemannian Langevin Diffusion is given by the following SDE whose stationary distribution is νdv_g:

$$dX_t = (\nabla \cdot (g^{-1}(X_t)) - \nabla F(X_t))dt + \sqrt{2g^{-1}(X_t)}dB_t.$$

Here $\nabla\cdot$ denotes the divergence with respect to the manifold and it is applied separately to each row of a matrix. When written in local Euclidean coordinates, this becomes

$$dX_t = (D \cdot (g^{-1}(X_t)) - g^{-1}(X_t)Df(X_t))dt + \sqrt{2g^{-1}(X_t)}dB_t$$

where $f(x) = F(x) - \frac{1}{2}\log\det g(x)$ and the derivative and divergence are Euclidean. When f is constant, then $Df = 0$ and we get the equation for Brownian motion on the manifold.

RLD has the same convergence guarantee in continuous time as stated by Theorem 34, with ν being the manifold stationary measure. While LD can only be applied to smooth target densities, RLD allows us to use the metric g to incorporate constraints and sample from e^{-f} subject to constraints. For example, by letting g be the log-barrier of a polytope we get to sample from densities restricted to the polytope, in continuous time. We next define the extension to a discrete time algorithm.

▶ **Definition 40** (Riemannian Langevin Algorithm (RLA))**.** To sample from the distribution $\nu dv_g(x)$ over the manifold $\mathcal{M}$, for a fixed step size ϵ, repeat:

$$y = \exp_{x_k}(-\epsilon\nabla F(x_k))$$
$$x_{k+1} = \mathfrak{B}(y, \epsilon)$$

where $\mathfrak{B}(x_0, t)$ samples from Brownian motion on the manifold, starting from x_0 after time t.

Recently, following work of [33, 1], [14] made progress on analyzing RLA in this general setting, showing that an extension of self-concordance of the metric suffices to guarantee convergence under the log-Sobolev inequality.

6 Discussion

The study of sampling has led to new tools, both for analysis and for algorithms, surprising connections with convex geometry, functional analysis and optimization, and practical algorithms in high dimension. We conclude by highlighting a few more open problems.

▶ **Question 41.** *Is there a natural KLS conjecture for Hessian manifolds?*

The close connection between diffusion and sampling raises an interesting sampling problem, namely efficiently simulating Brownian motion on manifolds.

▶ **Question 42.** *Given a metric g, an initial point x_0 and a time interval t, give an algorithm to sample from the distribution of Brownian motion on the manifold starting at x_0 for time t. For a manifold with metric g, Brownian motion is given by:*

$$dX_t = \nabla \cdot (g^{-1})dt + \sqrt{2g^{-1}}dB_t.$$

References

1 Kwangjun Ahn and Sinho Chewi. Efficient constrained sampling via the mirror-langevin algorithm. *Advances in Neural Information Processing Systems*, 34, 2021.

2 Apostolos Chalkis and Vissarion Fisikopoulos. volEsti: Volume approximation and sampling for convex polytopes in R. *arXiv preprint*, 2020. arXiv:2007.01578.

3 Yuansi Chen. An almost constant lower bound of the isoperimetric coefficient in the kls conjecture. *Geometric and Functional Analysis*, 31(1):34–61, 2021.

4 Zongchen Chen and Santosh S. Vempala. Optimal convergence rate of hamiltonian monte carlo for strongly logconcave distributions. *Theory of Computing*, 18(9):1–18, 2022. doi:10.4086/toc.2022.v018a009.

5 Xiang Cheng, Niladri S Chatterji, Peter L Bartlett, and Michael I Jordan. Underdamped Langevin MCMC: a non-asymptotic analysis. In *Conference on Learning Theory (COLT)*, pages 300–323. PMLR, 2018.

6 Sinho Chewi, Murat A Erdogdu, Mufan Bill Li, Ruoqi Shen, and Matthew Zhang. Analysis of Langevin Monte Carlo from Poincaré to Log-Sobolev. *arXiv preprint*, 2021. arXiv:2112.12662.

7 Ben Cousins and Santosh Vempala. A practical volume algorithm. *Mathematical Programming Computation*, 8(2):133–160, 2016.

8 Ben Cousins and Santosh Vempala. Gaussian cooling and O*(nˆ3) algorithms for volume and gaussian volume. *SIAM Journal on Computing*, 47(3):1237–1273, 2018.

9 Arnak S Dalalyan. Theoretical guarantees for approximate sampling from smooth and log-concave densities. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(3):651–676, 2017.

10 II Dikin. Iterative solution of problems of linear and quadratic programming. In *Doklady Akademii Nauk*, volume 174(4), pages 747–748. Russian Academy of Sciences, 1967.

11 Alain Durmus, Szymon Majewski, and Błażej Miasojedow. Analysis of langevin monte carlo via convex optimization. *The Journal of Machine Learning Research*, 20(1):2666–2711, 2019.

12 R. Eldan. Thin shell implies spectral gap up to polylog via a stochastic localization scheme. *Geometric and Functional Analysis*, 23:532–569, 2013.

13 B. Fleury. Concentration in a thin Euclidean shell for log-concave measures. *J. Funct. Anal.*, 259(4):832–841, 2010.

14 Khashayar Gatmiry and Santosh S Vempala. Convergence of the riemannian langevin algorithm. *arXiv preprint*, 2022. arXiv:2204.10818.

15 Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

16 Olivier Guedon and Emanuel Milman. Interpolating thin-shell and sharp large-deviation estimates for isotropic log-concave measures. *Geometric and Functional Analysis*, 21(5):1043–1068, 2011. doi:10.1007/s00039-011-0136-5.

17 Hulda S Haraldsdóttir, Ben Cousins, Ines Thiele, Ronan MT Fleming, and Santosh Vempala. Chrr: coordinate hit-and-run with rounding for uniform sampling of constraint-based models. *Bioinformatics*, 33(11):1741–1743, 2017.

18 Roland Hildebrand. Canonical barriers on convex cones. *Mathematics of operations research*, 39(3):841–850, 2014.

19 He Jia, Aditi Laddha, Yin Tat Lee, and Santosh Vempala. Reducing isotropy and volume to KLS: an $O^*(n^3\psi^2)$ volume algorithm. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 961–974, 2021.

20 Richard Jordan, David Kinderlehrer, and Felix Otto. The variational formulation of the Fokker–Planck equation. *SIAM Journal on Mathematical Analysis*, 29(1):1–17, January 1998.

21 R. Kannan, L. Lovász, and M. Simonovits. Isoperimetric problems for convex bodies and a localization lemma. *Discrete & Computational Geometry*, 13:541–559, 1995.

22 R. Kannan, L. Lovász, and M. Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Structures and Algorithms*, 11:1–50, 1997.

23 Ravindran Kannan and Hariharan Narayanan. Random walks on polytopes and an affine interior point method for linear programming. *Mathematics of Operations Research*, 37(1):1–20, 2012.

24 N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

25 Bo'az Klartag and Joseph Lehec. Bourgain's slicing problem and kls isoperimetry up to polylog. *arXiv preprint*, 2022. `arXiv:2203.15551`.

26 Yunbum Kook, Yin Tat Lee, Ruoqi Shen, and Santosh S. Vempala. Sampling with riemannian hamiltonian monte carlo in a constrained space, 2022. `doi:10.48550/ARXIV.2202.01908`.

27 Aditi Laddha, Yin Tat Lee, and Santosh Vempala. Strong self-concordance and sampling. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1212–1222, 2020.

28 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in sqrt(rank) iterations and faster algorithms for maximum flow. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433, 2014.

29 Yin Tat Lee and Santosh Vempala. Geodesic walks in polytopes. *SIAM Journal on Computing*, 51(2):STOC17–400–STOC17–488, 2022. `doi:10.1137/17M1145999`.

30 Yin Tat Lee and Santosh S Vempala. Convergence rate of Riemannian Hamiltonian Monte Carlo and faster polytope volume computation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1115–1121, 2018.

31 Yin Tat Lee and Santosh S Vempala. The Kannan-Lovász-Simonovits conjecture. *arXiv preprint*, 2018. `arXiv:1807.03465`.

32 Yin Tat Lee and Santosh Srinivas Vempala. Eldan's stochastic localization and the KLS hyperplane conjecture: An improved lower bound for expansion. *CoRR*, abs/1612.01507, 2016. `arXiv:1612.01507`.

33 Mufan Bill Li and Murat A Erdogdu. Riemannian langevin algorithm for solving semidefinite programs. *arXiv preprint*, 2020. `arXiv:2010.11176`.

34 Xuechen Li, Yi Wu, Lester Mackey, and Murat A Erdogdu. Stochastic Runge-Kutta accelerates Langevin Monte Carlo and beyond. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

35 L. Lovász. Hit-and-run mixes fast. *Math. Prog.*, 86:443–461, 1998.

36 L. Lovász and M. Simonovits. Random walks in a convex body and an improved volume algorithm. In *Random Structures and Alg.*, volume 4, pages 359–412, 1993.

37 L. Lovász and S. Vempala. Hit-and-run from a corner. *SIAM J. Computing*, 35:985–1005, 2006.

38 L. Lovász and S. Vempala. The geometry of logconcave functions and sampling algorithms. *Random Structures and Algorithms*, 30(3):307–358, 2007.

39 Oren Mangoubi and Aaron Smith. Rapid mixing of hamiltonian monte carlo on strongly log-concave distributions. *arXiv preprint*, 2017. `arXiv:1708.07114`.

40 Radford M Neal et al. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

41 Yurii Nesterov, Arkadii Nemirovskii, and Yinyu Ye. *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM, 1994.

42 S. Vempala. Geometric random walks: A survey. *MSRI Combinatorial and Computational Geometry*, 52:573–612, 2005.

43 Santosh Vempala and Andre Wibisono. Rapid convergence of the Unadjusted Langevin Algorithm: Isoperimetry suffices. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

44 Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009.

45 Andre Wibisono. Sampling as optimization in the space of measures: The Langevin dynamics as a composite optimization problem. In *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018*, pages 2093–3027, 2018. URL: `http://proceedings.mlr.press/v75/wibisono18a.html`.

46 Keru Wu, Scott Schmidler, and Yuansi Chen. Minimax mixing time of the metropolis-adjusted langevin algorithm for log-concave sampling. *arXiv preprint*, 2021. `arXiv:2109.13055`.

A Riemannian metrics

A manifold M can be viewed as an n-dimensional "surface" in $\mathbb{R}^k$ for some $k \geq n$.

1. Riemannian metric: For any $v, u \in T_pM$, the inner product (Riemannian metric) at p is given by $\langle v, u \rangle_p$ and this allows us to define the norm of a vector $\|v\|_p = \sqrt{\langle v, v \rangle_p}$. We call a manifold a Riemannian manifold if it is equipped with a Riemannian metric. When it is clear from context, we define $\langle v, u \rangle = \langle v, u \rangle_p$. In $\mathbb{R}^n$, $\langle v, u \rangle_p$ is the usual ℓ_2 inner product.
2. Tangent space T_pM: For any point p, the tangent space T_pM of M at point p is a linear subspace of $\mathbb{R}^k$ of dimension n. Intuitively, T_pM is the vector space of possible directions that are tangential to the manifold at x. Equivalently, it can be thought as the first-order linear approximation of the manifold M at p. For any curve c on M, the direction $\frac{d}{dt}c(t)$ is tangent to M and hence lies in $T_{c(t)}M$. For any open subset M of $\mathbb{R}^n$, we can identify T_pM with $\mathbb{R}^n$.

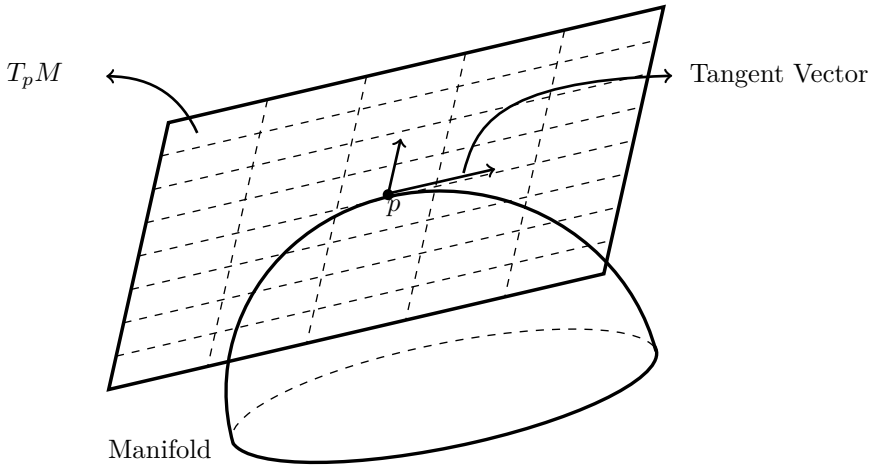


Figure A.1 Riemannian Manifold and Tangent Space.

3. Hessian manifold: We call M a Hessian manifold (induced by ϕ) if M is an open subset of $\mathbb{R}^n$ with the Riemannian metric at any point $p \in M$ defined by

$$\langle v, u \rangle_p = v^\top \nabla^2 \phi(p) u$$

where $v, u \in T_pM$ and ϕ is a smooth convex function on M.

4. Length: For any curve $c : [0, 1] \to M$, we define its length by

$$L(c) = \int_0^1 \left\| \frac{d}{dt} c(t) \right\|_{c(t)} dt.$$

5. Distance: For any $x, y \in M$, we define $d(x, y)$ be the infimum of the lengths of all paths connecting x and y. In $\mathbb{R}^n$, $d(x, y) = \|x - y\|_2$.

6. Geodesic: We call a curve $\gamma(t) : [a, b] \to M$ a geodesic if it satisfies both of the following conditions:

   a. The curve $\gamma(t)$ is parameterized with constant speed. Namely, $\left\|\frac{d}{dt}\gamma(t)\right\|_{\gamma(t)}$ is constant for $t \in [a, b]$.
   b. The curve is the locally shortest length curve between $\gamma(a)$ and $\gamma(b)$. Namely, for any family of curve $c(t, s)$ with $c(t, 0) = \gamma(t)$ and $c(a, s) = \gamma(a)$ and $c(b, s) = \gamma(b)$, we have that $\frac{d}{ds}\big|_{s=0} \int_a^b \left\|\frac{d}{dt}c(t, s)\right\|_{c(t,s)} dt = 0$.

   Note that, if $\gamma(t)$ is a geodesic, then $\gamma(\alpha t)$ is a geodesic for any α. Intuitively, geodesics are local shortest paths. In $\mathbb{R}^n$, geodesics are straight lines.
7. Exponential map: The map $\exp_p : T_pM \to M$ is defined as

   $$\exp_p(v) = \gamma_v(1)$$

   where γ_v is the unique geodesic starting at p with initial velocity $\gamma_v'(0)$ equal to v. The exponential map takes a straight line $tv \in T_pM$ to a geodesic $\gamma_{tv}(1) = \gamma_v(t) \in M$. Note that $\exp_p$ maps v and tv to points on the same geodesic. Intuitively, the exponential map can be thought as point-vector addition in a manifold. In $\mathbb{R}^n$, we have $\exp_p(v) = p + v$.
8. Parallel transport: Given any geodesic $c(t)$ and a vector v such that $\langle v, c'(0)\rangle_{c(0)} = 0$, we define the parallel transport of v along $c(t)$ by the following process: Take h to be infinitesimally small and $v_0 = v$. For $i = 1, 2, \cdots, 1/h$, we let v_{ih} be the vector orthogonal to $c'(ih)$ that minimizes the distance on the manifold between $\exp_{c(ih)}(hv_{ih})$ and $\exp_{c((i-1)h)}(hv_{(i-1)h})$. Intuitively, the parallel transport finds the vectors on the curve such that their end points are closest to the end points of v. For general vector $v \in T_{c'(0)}$, we write $v = \alpha c'(0) + w$ and we define the parallel transport of v along $c(t)$ is the sum of $\alpha c'(t)$ and the parallel transport of w along $c(t)$. For non-geodesic curve, see the definition in Fact 43.
9. Orthonormal frame: Given vector fields $v_1, v_2, \cdots, v_n$ on a subset of M, we call $\{v_i\}_{i=1}^n$ is an orthonormal frame if $\langle v_i, v_j\rangle_x = 1$ if $i = j$ and 0 otherwise. Given a curve $c(t)$ and an orthonormal frame at $c(0)$, we can extend it along the curve by parallel transport and it remains orthonormal on the whole curve.
10. Directional derivatives and the Levi-Civita connection: For a vector $v \in T_pM$ and a vector field u in a neighborhood of p, let γ_v be the unique geodesic starting at p with initial velocity $\gamma_v'(0) = v$. Define

    $$\nabla_v u = \lim_{h \to 0} \frac{u(h) - u(0)}{h}$$

    where $u(h) \in T_pM$ is the parallel transport of $u(\gamma_v(h))$ from $\gamma(h)$ to $\gamma(0)$. Intuitively, Levi-Civita connection is the directional derivative of u along direction v, *taking the metric into account*. In particular, for $\mathbb{R}^n$, we have $\nabla_v u(x) = \frac{d}{dt}u(x + tv)$. When u is defined on a curve c, we define $D_t u = \nabla_{c'(t)} u$. In $\mathbb{R}^n$, we have $D_t u(\gamma(t)) = \frac{d}{dt}u(\gamma(t))$. We reserve $\frac{d}{dt}$ for the usual derivative with Euclidean coordinates.

We list some basic facts about the definitions above.

▶ **Fact 43.** *Given a manifold M, a curve $c(t) \in M$, a vector v and vector fields u, w on M, we have the following:*

1. *(alternative definition of parallel transport) $v(t)$ is the parallel transport of v along $c(t)$ if and only if $\nabla_{c'(t)} v(t) = 0$.*

2. *(alternative definition of geodesic) c is a geodesic if and only if* $\nabla_{c'(t)} c'(t) = 0$.
3. *(linearity)* $\nabla_v(u + w) = \nabla_v u + \nabla_v w$.
4. *(product rule) For any scalar-valued function f,* $\nabla_v(f \cdot u) = \frac{\partial f}{\partial v} u + f \cdot \nabla_v u$.
5. *(metric preserving)* $\frac{d}{dt} \langle u, w \rangle_{c(t)} = \langle D_t u, w \rangle_{c(t)} + \langle u, D_t w \rangle_{c(t)}$.
6. *(torsion free-ness) For any map $c(t, s)$ from a subset of $\mathbb{R}^2$ to M, we have that* $D_s \frac{\partial c}{\partial t} = D_t \frac{\partial c}{\partial s}$ *where* $D_s = \nabla_{\frac{\partial c}{\partial s}}$ *and* $D_t = \nabla_{\frac{\partial c}{\partial t}}$.
7. *(alternative definition of Levi-Civita connection)* $\nabla_v u$ *is the unique linear mapping from the product of vector and vector field to vector field that satisfies (3), (4), (5) and (6).*

Streaming and Sketching Complexity of CSPs: A Survey

Madhu Sudan
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Abstract

In this survey we describe progress over the last decade or so in understanding the complexity of solving constraint satisfaction problems (CSPs) approximately in the streaming and sketching models of computation. After surveying some of the results we give some sketches of the proofs and in particular try to explain why there is a tight dichotomy result for sketching algorithms working in subpolynomial space regime.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Streaming algorithms, Sketching algorithms, Dichotomy, Communication Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.5

Category Invited Talk

Related Version A potentially updated version appears on ECCC as TR 22-065.
Full Version: `https://eccc.weizmann.ac.il/report/2022/065/`

Funding *Madhu Sudan*: Supported in part by a Simons Investigator Award and NSF Awards CCF 1715187 and CCF 2152413.

Acknowledgements I want to thank Santhoshini Velusamy for introducing me to this wonderful line of research. I want to thank her and all other co-authors – Chi-Ning Chou, Sasha Golovnev, Noah Singer, Raghuvansh Saxena, Amirbehshad Shahrasbi and Ameya Velingker – for the collaborations and discussions which informed this survey. They also caught numerous errors and gave suggestions that have hopefully made this survey more readable. I want to thank the organizers of ICALP 2022 for inviting me to write this survey, and to David Woodruff in particular for the additional encouragement and nudges that were crucial to get me going.

1 Introduction

In this article we survey the current state of knowledge on the *approximability* of *constraint satisfaction problems (CSPs)* using small space *streaming* and *sketching* algorithms. We start by reviewing the definitions below before turning to the survey of results.

2 CSPs: What and Why

CSPs form an infinite class of optimization problems where the goal is to assign n variables values from a finite set while maximizing the number of constraints that can be satisfied, where each constraint looks locally at the assignment of a few variables to determine if it is satisfied on not. Different problems in the class differ based on which type of constraints are allowed. Different instances of the problem arise by applying the constraints to different subsets (or subsequences) of variables. Algorithms aim to compute, or approximate, the maximum, over all assignments, of the fraction of constraints that can be simultaneously satisfied. Resource restrictions on the algorithm (time, space, number of passes in the streaming setting) as well

as the type of constraints allow to determine the level of approximability that is feasible. In this survey we describe our knowledge of the approximability of CSPs when restricted to streaming and sketching algorithms with limited space.

We start by describing CSPs more formally. For positive integer n we use $[n]$ to denote the set $\{1, \ldots, n\}$ and $\mathbb{Z}_n$ to denote the set $\{0, \ldots, n-1\}$. A CSP problem is described by positive integers k, q and a family of functions $\mathcal{F} \subseteq \{f : \mathbb{Z}_q^k \to \{0,1\}\}$. Since k, q are implicit in $\mathcal{F}$, we refer to this problem as $\mathsf{Max\text{-}CSP}(\mathcal{F})$. Given variables $X_1, \ldots, X_n$, an assignment to the variables is a sequence $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$. A constraint C on these variables is given by a pair $(f, (j_1, \ldots, j_k))$ where the first element of the pair $f \in \mathcal{F}$ is the choice of the type of constraint and the second element is a sequence of k *distinct* indices with $j_i, \ldots, j_k \in [n]$. An assignment $\mathbf{a}$ satisfies $C = (f, (j_1, \ldots, j_k))$ if and only if $f(a_{j_1}, \ldots, a_{j_k}) = 1$. We use $C(\mathbf{a})$ to denote the quantity $f(X_{j_1}, \ldots, X_{j_k})$. An instance of $\mathsf{Max\text{-}CSP}(\mathcal{F})$ on n variables and m constraints is given by $\Psi = (C_1, \ldots, C_m)$ with C_i being a constraint on $X_1, \ldots, X_n$ for every $i \in [m]$. Given a assignment $\mathbf{a}$ to the variables, the value on the instance Ψ at $\mathbf{a}$, denoted $\mathsf{val}_\Psi(\mathbf{a})$, is the quantity $\frac{1}{m}\sum_{i\in[m]} C_i(\mathbf{a})$, i.e., the value is the fraction of constraints of Ψ that are satisfied by $\mathbf{a}$. The value of the instance val_Ψ is defined to be the maximum value over all assignments, i.e., $\mathsf{val}_\Psi = \max_{\mathbf{a}\in[q]^n}\{\mathsf{val}_\Psi(\mathbf{a})\}$. The goal of CSP optimization algorithms is to compute, or approximate, val_Ψ given Ψ.[1]

Example. We illustrate the definition with the example of the Max-CUT problem, where given an undirected graph on vertex set $[n]$, the goal is to find a "cut" $S \sqcup \overline{S} = [n]$ that maximizes the number of edges crossing the cut (i.e, with one endpoint each in S and $\overline{S}$). This problem is captured by $q = k = 2$ and the family $\mathcal{F} = \{\oplus\}$ where $\oplus(u, v) = u + v \pmod 2$.

Note that in the example above, we could get positive integer weighted graphs also since there is no requirement that the constraints themselves be distinct. For simplicity we will assume the length on the stream is polynomial in n so that the weights are also polynomial, though as we point out later this restriction does not alter the complexity of the approximation problems much. But the graphs will not have any self-loops due to the "restriction" that the variables in a constraint have to be distinct. If one wished to consider the Max-CUT problem where self-loops are also allowed (though in the case of Max-CUT this would make no sense - since a self-loop can never be cut), then one could consider instances $\mathsf{Max\text{-}CSP}(\mathcal{F}')$ where $\mathcal{F}' = \{\oplus, F\}$ and $F(u, v) = \oplus(u, u)$ for all u, v. (So v is just a dummy variable and u is a variable which supplies both arguments to the cut function $\oplus$.) Thus while the requirement that the variables are distinct may appear as a restriction, it is not: For every family $\mathcal{F}$ there exists a family $\mathcal{F}'$ such that $\mathsf{Max\text{-}CSP}(\mathcal{F}')$ captures the $\mathsf{Max\text{-}CSP}(\mathcal{F})$ problem where variables are allowed to repeat.

We also remark that in some prior works in the Boolean setting, i.e., when $q = 2$, constraints may be applied to "literals", rather than "variables". We refer to these results as applying to Boolean CSPs over literals. In our setting we apply constraints only to variables. Again, our setting is more general than the setting of Boolean CSPs over literals in that for every family $\mathcal{F} \subseteq \{f : \mathbb{Z}_2^k \to \{0,1\}\}$ there is a family $\mathcal{F}'$ such that CSP with constraints from $\mathcal{F}$ applied to literals is the same problem as $\mathsf{Max\text{-}CSP}(\mathcal{F}')$. For example consider the Max-2-LIN problem, i.e., the CSP whose constraints are given by linear equations

[1] Throughout this paper we assume that $\mathcal{F}$ does not include the all 0 function. Such a function corresponds to placing constraints that are never satisfiable. Inclusion of such constraints in the family does not change the complexity of any of the tasks we consider since these constraints are easy to ignore.

modulo 2 on two distinct variables. Max-2-LIN is the Boolean CSP over literals over the family $\mathcal{F} = \{\oplus\}$. But if we let $\mathcal{F}' = \{\oplus, \overline{\oplus}\}$ where $\overline{\oplus}(u, v) = u + v + 1 \pmod 2$, then Max-2-LIN = Max-CSP($\mathcal{F}'$). On the other hand Max-CSP($\mathcal{F}$) is the Max-CUT problem which can not be expressed as a Boolean CSP over literals. Thus our setting is strictly richer in expressibility.

2.1 Why study CSPs?

Before going on to giving more precise descriptions of the approximation versions of CSPs and models of streaming algorithms, we digress to comment on why study CSPs at all.

CSPs do capture a host of natural optimization problems: Some familiar names of problems include the Maximum Cut problem in (undirected) graphs, the Maximum Dicut problem in directed graphs, the Maximum q-colorability problem in graphs, the Unique Games problem, etc. Each one of these problems on their own right has probably been the topic of multiple papers, and the umbrella of CSPs unifies their study. That being said this reason on its own is not as compelling as some of the other reasons we describe next – after all the study of CSPs does exclude many other natural optimization problems including problems based on connectivity in graphs such as flow maximization or congestion minimization. It also excludes global considerations such as balanced cuts or sparse cuts; and of course there are a host of non-graph-theoretic problems.

To this author, the real reason to study CSPs is that they tend to allow for finite classification. The first such result dates back to Schaefer [29] who studied the satisfiability of Boolean CSPS and showed that they exhibit a dichotomy. Feder and Vardi [17] explored the expressibility of different logics and arrived at a morally "broadest" set of problems ("Monotone Monadic SNP") that could potentially exhibit a dichotomy, and showed that this set of problems was essentially equivalent to CSPs over arbitrary finite alphabets. They posed the dichotomy of this class as an open question which was eventually resolved by Bulatov [7] and Zhuk [31] after many years of sustained attack. Subsequent works extended such classification quests to other classes of problems including optimization and counting. (See Creignou, Khanna and Sudan [15] for some of the early lines of work.) Many of these bodies are extensive, see e.g. the recent monograph by Cai and Chen [8] and references therein for vast explorations of counting problems. In optimization and approximation the work of Raghavendra [28] gives a fine dichotomy, under the "Unique Games Conjecture", that inspires some of the streaming work we describe in this survey.

Finite classifications are interesting in that they highlight the generality of some algorithms. Even the weak classification of the approximability of CSPs pointed to the general utility of the randomized rounding and Max Flow algorithms [25]. The sharp characterizations in [28] point to the power of semidefinite programming and specifically to the sum-of-squares framework of algorithms. One could also ask similar questions in the context of streaming: The dichotomy work presented in this survey does highlight the role of norm estimation algorithms in streaming optimization. Other algorithms might emerge with further exploration.

Finite classifications also point to interesting phenomena. For instance in the context of polynomial time approximability most natural problems have approximability in well separated bands of functions: constant factor approximations, polylogarithmic approximations, and polynomial approximations are common whereas very few have intermediate approximability, say to within a factor of $2^{\sqrt{\log n}}$. A finite classification implies this phenomenon - the entire infinite class of functions only shows finitely many distinct behaviors. A similar phenomenon again seems to occur with streaming algorithms – many problems have

polylogarithmic space approximation algorithms while others require polynomially growing space. Intermediate complexity is rare. CSPs again seems to validate this separation, at least in the context of sketching algorithms, as we will see in the rest of this survey.

3 Preliminaries: Approximation and Streaming

We formalize some basic notions related to approximation problems and streaming algorithms. While a reader familiar with the notion might skip ahead we recommend they make sure they understand the notions (and notations) of: (i) trivial approximation algorithms, (ii) gapped optimization problems and the notation: (γ, β)-Max-CSP($\mathcal{F}$) (iii) approximability and approximation-resistance and (iv) sketching algorithms.

3.1 Approximating CSPs

Since solving Max-CSP($\mathcal{F}$) exactly can be quite hard for most $\mathcal{F}$ we turn often to algorithms that produce approximate solutions. We discuss some basic definitions regarding these in this section. All definitions restrict algorithms to come from some resource-bounded class $\mathcal{C}$. While we defer the discussion of the specific classes considered to later sections, here we consider definitions for a generic such class $\mathcal{C}$.

The most common notion of approximation is an α-approximation algorithm for some $\alpha \in [0, 1]$. An α-approximation algorithm $\mathbf{ALG}$ for Max-CSP($\mathcal{F}$) is one that for every instance Ψ outputs a value $\mathbf{ALG}(\Psi)$ satisfying $\alpha \cdot \mathsf{val}_\Psi \leq \mathbf{ALG}(\Psi) \leq \mathsf{val}_\Psi$.[2] We say Max-CSP($\mathcal{F}$) is α-approximable in $\mathcal{C}$ if there exists $\mathbf{ALG} \in \mathcal{C}$ that is an α-approximation algorithm for Max-CSP($\mathcal{F}$).

A more refined notion of approximation that is more common in the literature proving non-existence of algorithms is associated with gapped problems. Given $0 \leq \beta < \gamma \leq 1$, we say that an algorithm $\mathbf{ALG}$ solves the "(γ, β)-approximation version of Max-CSP($\mathcal{F}$)", abbreviated (γ, β)-Max-CSP($\mathcal{F}$), if the following two conditions hold: (1) For every Ψ such that $\mathsf{val}_\Psi \geq \gamma$, we have $\mathbf{ALG}(\Psi) = 1$ and (2) For every Ψ such that $\mathsf{val}_\Psi \leq \beta$, we have $\mathbf{ALG}(\Psi) = 0$. We say that (γ, β)-Max-CSP($\mathcal{F}$) is solvable in $\mathcal{C}$ if there exists an $\mathbf{ALG} \in \mathcal{C}$ solving (γ, β)-Max-CSP($\mathcal{F}$).

Assuming $\mathcal{C}$ satisfies mild closure properties the latter notion roughly captures α-approximability precisely, while giving more detailed information. To see some flavor of the translation between the two notions, suppose Max-CSP($\mathcal{F}$) is α-approximated by $\mathbf{ALG}$. Now consider a pair γ, β with $\beta < \alpha\gamma$. Then the algorithm $\mathbf{ALG}'$ that outputs $\mathbf{ALG}'(\Psi) = 1$ if $\mathbf{ALG}(\Psi) > \beta$ and 0 otherwise solves (γ, β)-Max-CSP($\mathcal{F}$). And if $\mathcal{C}$ satisfies the closure property that $\mathbf{ALG}' \in \mathcal{C}$ whenever $\mathbf{ALG} \in \mathcal{C}$ then it follows that α-approximability in $\mathcal{C}$ implies (γ, β)-Max-CSP($\mathcal{F}$) is solvable in $\mathcal{C}$ for every $\beta < \alpha\gamma$. A rough converse is also true, again assuming some (this time more stronger) closure properties of $\mathcal{C}$ that we leave unspecified: If for some α we have that for every $\gamma \in [0, 1]$, the (γ, β)-Max-CSP($\mathcal{F}$) is solvable in $\mathcal{C}$ for $\beta = \alpha\gamma$, then for every $\epsilon > 0$ we have that Max-CSP($\mathcal{F}$) is $\alpha - \epsilon$ approximable in $\mathcal{C}$. (See [13, Proposition 2.21] for a detailed statement and proof.)

The discussion above explains why the study of (γ, β)-Max-CSP($\mathcal{F}$) (for every γ, β and $\mathcal{F}$) is at least as rich as the study of α-approximability of Max-CSP($\mathcal{F}$). But it can provide more detailed information. For instance researchers are often interested in approximating the value

[2] Note that a small space streaming algorithm has very little hope of outputting a near-optimal solution which might take $\Omega(n)$ space to represent. So we require our algorithms only to output the value achieved by the optimal, or approximately-optimal, solution.

on satisfiable, or nearly satisfiable, instances. (See for instance, [16, 4, 27] for such works in the setting of $\mathcal{C}$ being all polynomial time algorithms.) We can understand these corner cases by focussing on $\gamma = 1$ or $\gamma \to 1$ and exploring the maximal β such that (γ, β)-Max-CSP($\mathcal{F}$) is solvable in $\mathcal{C}$. For instance recent results show that for every $\beta < 1$, $(1, \beta)$-Max-DICUT is solvable in $\mathcal{C}$ when $\mathcal{C}$ is the class of polylog space bounded sketching algorithms – a result that is not captured by the single parameter approximability of the problem.

Before concluding we also highlight what is a "non-trivial" approximation. For families $\mathcal{F}$ where every constraint has at least one satisfying assignment this notion is quite simple. We say that an algorithm that outputs a constant (independent of the input Ψ) is a trivial algorithm. Note that trivial algorithms are still legitimate approximation algorithms. For instance the algorithm that always outputs $1/2$ is a $1/2$-approximation for Max-CUT – this is so since no instance Ψ of Max-CUT has $\mathsf{val}_\Psi < 1/2$. We say that an approximation ratio is non-trivial if it is not achieved by a trivial algorithm. Similarly we say that a (γ, β)-approximation is non-trivial if $\gamma < 1$ and there exists an instance Ψ with $\mathsf{val}_\Psi \leq \beta$. To quantify this notion of non-triviality we define $\rho_{\min}(\mathcal{F})$ to be $\inf_{\Psi \text{ instance of Max-CSP}(\mathcal{F})}\{\mathsf{val}_\Psi\}$. We say that a Max-CSP($\mathcal{F}$) is *approximation resistant* to $\mathcal{C}$ if for every $\alpha > \rho_{\min}(\mathcal{F})$ no algorithm $\mathbf{ALG} \in \mathcal{C}$ is an α-approximation to Max-CSP($\mathcal{F}$). Equivalently for every $\rho_{\min}(\mathcal{F}) < \beta < \gamma < 1$ it is the case that (γ, β)-Max-CSP($\mathcal{F}$) is not solvable in $\mathcal{C}$. We say Max-CSP($\mathcal{F}$) is *approximable* in $\mathcal{C}$ if it is not *approximation resistant* to $\mathcal{C}$.

In what follows we will describe works exploring the various approximation factors achievable for Max-CSP($\mathcal{F}$) for different $\mathcal{F}$ with streaming algorithms that have bounded space. We shall also explore some restrictions of streaming algorithms known as sketching algorithms. We introduce these terms below.

3.2 Streaming and Sketching algorithms

We consider the approximability of Max-CSP($\mathcal{F}$) when the input instance $\Psi = (C_1, \ldots, C_m)$ is presented as sequence of constraints to the approximating algorithm. The algorithm is restricted in the amount of space it is provided. We allow the algorithm to be randomized: in all upper bounds the algorithm will be expected to generate and store any randomness in the restricted space it is given, while in the lower bounds we will rule out algorithms that are a distribution over deterministic algorithms (and so strictly more general).

Formally a space $s(n)$-streaming algorithm for (γ, β)-Max-CSP($\mathcal{F}$) on n variables is given by a pair of functions (τ, ν) where $\Gamma : \{0,1\}^{s(n)} \times \Lambda_n \to \{0,1\}^{s(n)}$ is the state transition function and output function $\nu : \{0,1\}^{s(n)} \to \{0,1\}^{s(n)}$, where $\Lambda_n = \Lambda_n(\mathcal{F})$ denotes the set of all possible constraints of Max-CSP($\mathcal{F}$) on n variables.On input a stream $\boldsymbol{\sigma} = (C_1, \ldots, C_m) \in \Lambda_n^m$ the algorithm first computes the state $S(\boldsymbol{\sigma}) = S_m$ where $S_i = \Gamma(S_{i-1}, C_i)$ for $i \in [m]$ and $S_0 = 0$ in the deterministic case. It then outputs $\nu(S(\boldsymbol{\sigma}))$. A randomized streaming algorithm is the same except that now the initial state S_0 is distributed uniformly randomly over $\{0,1\}^{s(n)}$.

In this paper we will also focus on a restriction of streaming algorithms known as sketching algorithms. To define this notion consider the concatenation of two streams $\boldsymbol{\sigma} \circ \boldsymbol{\tau}$. By definition of a streaming algorithm the final state $S(\boldsymbol{\sigma} \circ \boldsymbol{\tau})$ can be determined from $S(\boldsymbol{\sigma})$ and $\boldsymbol{\tau}$. A sketching algorithm is one that is restricted even further in that $S(\boldsymbol{\sigma} \circ \boldsymbol{\tau})$ can be determined from $S(\boldsymbol{\sigma})$ and $S(\boldsymbol{\tau})$, i.e., there exists a composer function $C : \{0,1\}^{s(n)} \times \{0,1\}^{s(n)} \to \{0,1\}^{s(n)}$ such that for every $\boldsymbol{\sigma}, \boldsymbol{\tau} \in \Lambda_n^*$, we have $S(\boldsymbol{\sigma} \circ \boldsymbol{\tau}) = C(S(\boldsymbol{\sigma}), S(\boldsymbol{\tau}))$.

Most common sketching algorithms are obtained from so called "linear-sketching algorithms" where $C \in \Gamma_n$ is viewed as a vector in $v \in \mathbb{R}^N$ for some large N, and a stream $(C_1, \ldots, C_m)$ represents the sum of the m corresponding vectors $v_1 + \cdots + v_m$. The sketch

of a vector v is given by Av where $A \in \mathbb{R}^{N \times s}$ projects v down to some low-dimensional subspace. Ignoring bit precision issues this compresses large N dimensional inputs into small s dimensional sketches that end up giving significant information about the original input, surprisingly often. It is easy to see that such linear sketching algorithms indeed satisfy the definition of sketching.

In the rest of this article we describe the surprising effectiveness of sketching algorithms in approximating Max-CSP($\mathcal{F}$). We also describe matching lower bounds for sketching algorithms that often generalize also to give streaming lower bounds.

4 Results on Streaming CSPs

Prior to 2010, despite extensive work on streaming algorithms and lower bounds for other problems, there were no works covering CSPs. This was even noted at a workshop at Bertinoro in 2011 [21].

Lower bounds for Max-CUT

The first works focussed on lower bounds for the Max-CUT problem in independent works by Kogan and Krauthgamer [26] and Khanna, Kapralov and Sudan [22]. The former showed that there existed $\alpha < 1$ such that α-approximating Max-CUT requires $\Omega(\sqrt{n})$-space. The latter showed the tighter result showing that for every $\alpha > 1/2$, α-approximating Max-CUT requires $\Omega(\sqrt{n})$-space in the streaming setting. In other words Max-CUT is approximation resistant to $o(\sqrt{n})$ space streaming algorithms. Subsequent works focussed on the space complexity and pushed it higher. Khanna, Kapralov, Sudan and Velingker [23] pushed the space requirement up to linear at the cost of a weaker approximation; specifically they showed that there exists $\alpha < 1$ such that α-approximating Max-CUT requires $\Omega(n)$-space. Finally, in a tour-de-force work, Kapralov and Krachun [24] settled the approximability of Max-CUT essentially completely by showing it is approximation resistant to $o(n)$ space streaming algorithms.

An aside is in order here: The input to a Max-CSP($\mathcal{F}$) has length $O(m \log n)$ and even if we forbid repeated constraints m can be as large as n^k. So, a priori one could imagine space complexities of streaming algorithms being much higher than $O(n)$. But a folklore observation shows that it suffices for the streaming algorithm to maintain a random sample of $O(n/\epsilon^2)$ constraints and the optimum value on the sampled constraints is a $(1-\epsilon)$ approximation to the optimum value on the input instance.[3] Since this sample of constraints takes only $O(n \log n)$ space to store and the optimal value for this sample can be computed in $O(n)$ space (though using exponential time), it follows that $O(n \log n)$ space suffices to get α-approximation for every Max-CSP($\mathcal{F}$) problem for every $\alpha < 1$. In particular, returning to the Max-CUT problem, the space lower bound from [24] is optimal to within a logarithmic factor.

Upper bounds for Max-DICUT

While the lower bounds for Max-CUT are technically hard (especially [24]) arguably these results are not very surprising: A streaming algorithm with limited (say polylogarithmic) space seems hardly capable of understanding the global structure imposed by the many

[3] This observation also relies on the fact that the value of every instance is bounded away from 0, which in turn relies on the fact that 0 is not a constraint function in $\mathcal{F}$.

different constraints and understanding how well they can be satisfied simultaneously. Indeed the lower bounds of [22] show that $o(\sqrt{n})$ space streaming algorithms can not distinguish random graphs from random bipartite graphs with a planted bipartition. In view of such limited power to understand the global structure it would not have surprised some researchers (notably this author) if every Max-CSP($\mathcal{F}$) problem had turned out to be approximation resistant to $o(n)$-space streaming algorithms. In other words – it was conceivable in 2015 that there were no non-trivial streaming algorithms for CSPs.

A striking paper of Guruswami, Velingker and Velusamy [19] changed the picture by giving an elegant and simple algorithm for approximating Max-DICUT, the problem whose input is a directed graph on vertex set $[n]$ and the goal is to find a cut $S \sqcup \overline{S}$ that maximized the number of edges going from S to $\overline{S}$. (This problem is expressible as Max-CSP($\mathcal{F}$) for $\mathcal{F} = \{u \wedge \overline{v}\}$.) The "trivial" approximability of this problem is 1/4, but [19] gave a non-trivial 0.4-approximation algorithm for this problem using polylogarithmic space.[4] The key insight to their algorithm is that one can estimate some non-trivial global information about the input by appealing to norm estimation algorithms that have been well explored in the sketching community. In particular their work relies on algorithms for estimating the ℓ_1 norm of a vector in the "turnstile" model which go back to the work of Indyk [20]. As we will discuss later, this algorithm can be generalized arguably quite surprisingly to many other CSPs.

Tight bounds and classification of Boolean binary CSPs

While the lower bound for Max-CUT is obviously tight, the Max-DICUT approximability of .4 from [19] was not known to be tight. From the $1/2 + \epsilon$-inapproximability of Max-CUT one can deduce a $(1/2 + \epsilon)$-inapproximability, for every $\epsilon > 0$, for Max-DICUT as well (by a reduction which maps every edge from an instance of Max-CUT to a pair of directed edges between the same vertices). Neither the algorithm nor the analysis appear tight. Indeed in a subsequent work, Chou, Golovnev and Velusamy [14] managed to improve both the algorithm and the lower bound to get a tight approximability of 4/9 for Max-DICUT. Specifically they give a polylog space algorithm achieving this approximation ratio and also prove that no streaming algorithm with $o(\sqrt{n})$ space can do better! This tight result for Max-DICUT may appear accidental, but [14] go further and classify the approximability of every Boolean (i.e., with $q = 2$) CSP on literals on binary constraints (i.e., $k = 2$). In doing so their work points to some remarkable phenomena: For every $\alpha \in [0, 1]$, every CSP in the finite, but nevertheless diverse, class they consider either is α-approximable in polylogarithmic space, or is not $\alpha - \epsilon$ approximable (for every $\epsilon > 0$) with $o(\sqrt{n})$ space. And in all cases the approximation algorithm uses the ℓ_1-norm approximator in a manner similar to [19]. Together these results suggest a broader phenomenon explored and somewhat confirmed in the further work reported next.

Sketching complexity of CSPs

In joint work with Chou, Golovnev and Velusamy [13] we give a dichotomy result for all (γ, β)-Max-CSP($\mathcal{F}$) (i.e., for every $k, q, \mathcal{F}$ and every $\gamma, \beta \in [0, 1]$) for $o(\sqrt{n})$-space sketching algorithms, as described below.

[4] Throughout this article we will not spell out the exponent in polylogarithmic terms though of course the original papers give more detailed answers.

▶ **Theorem 1** ([13, Theorem 1.1]). *For every $q, k \in \mathbb{N}$, $0 \leq \beta < \gamma \leq 1$, and every $\mathcal{F} \subseteq \{\mathbb{Z}_q^k \to \{0,1\}\}$, one of the following two conditions holds: Either (γ,β)-Max-CSP($\mathcal{F}$) can be solved by a polylogarithmic space sketching algorithm, or for every $\epsilon > 0$, every sketching algorithm for $(\gamma-\epsilon,\beta+\epsilon)$-Max-CSP($\mathcal{F}$) requires $\Omega(\sqrt{n})$-space. Furthermore there is a polynomial space algorithm that decides, given γ, β and $\mathcal{F}$, which of the two conditions holds.*

A corollary to approximation resistance is the following: For every $\mathcal{F}$, either Max-CSP($\mathcal{F}$) is approximable by a polylogarithmic space sketching algorithm, or it is approximation resistant to $o(\sqrt{n})$-space sketching algorithms.[5] In the special case of $k = q = 2$ the lower bound above extends beyond sketching algorithms to all streaming algorithms ([13, Theorem 1.3]). Put together these results subsume all previous works with the exception of the linear space lower bound for Max-CUT from [24]. Even in the case of $k = q = 2$ the dichotomy is more detailed than the one in [14] in that it covers all CSPs, not just CSPs on literals, and it also talks about the solvability of all (γ,β)-Max-CSP($\mathcal{F}$) and not only the best approximation ratio. For example the results show that for every sufficiently small $\epsilon > 0$, $(1-\epsilon, 1-2\epsilon)$-Max-DICUT is solvable by a polylogarithmic space sketching algorithm while $(1-\epsilon, 1-2\epsilon+\delta)$-Max-DICUT requires $\Omega(\sqrt{n})$ space for every streaming algorithm for every $\delta > 0$. In particular it asserts that nearly satisfiable instances are detectable by small space sketching algorithms.

The sketching algorithms used for the positive result in Theorem 1 builds on the algorithm of [14], which we refer to as a "bias-based algorithm" here. We will discuss that algorithm further later, but highlight one major difference. Rather than appealing to ℓ_1-norm estimation algorithms, the new algorithm appeals to a matrix norm estimation algorithm, this time from the work of Andoni, Krauthgamer and Onak [1]. (Roughly the ℓ_1 norm given by $\|(x_1,\ldots,x_n)\|_1 = \max_{b_1,\ldots,b_n \in \{-1,+1\}} \sum_{i=1} b_i x_i$ optimizes over a Boolean domain. The matrix norm estimators allow us to optimize some problems over q-ary domains.)

While the theorem holds out the possibility that there are non-trivial approximation algorithms for (infinitely) many CSPs, this is not immediate from the theorem statement due to the lack of "explicitness" of the classification. Specifically there is no simple relationship that says given $\mathcal{F}$ what range of γ and β are "easy" (i.e., solvable in polylog space) and which ones are not. This is unfortunately inevitable. As $\mathcal{F}$ gets more complex the relationships do seem to get more complex. The results of [13] show that γ and β are determined by optimizing some $O(q^k)$ real variable linear function over the reals subject to some degree k polynomial constraints. Even in the case of Max-DICUT this leads to some degree 2 polynomials in γ and β that determine the complexity. (See [13, Example 1, Pages 21-23] for more details.) Nevertheless the conditions can be analyzed computationally, and in particular using the quantified theory of reals (using only the existential theory does not seem to suffice) to understand the complexity of (γ,β)-Max-CSP($\mathcal{F}$) for any given $\gamma, \beta, \mathcal{F}$.

Remarkably some subsequent work has managed to extract explicit results, even for infinite families of functions, by exploring the decision conditions arising from the proof of Theorem 1. For instance, Boyland, Hwang, Prasad, Singer and Velusamy [6], analyze the approximability of Max-kAND for every $k \in \mathbb{N}$ – the problem where constraints are the conjunctions of k-literals – and give an exact expression for the approximation ratio of Max-kAND. (They show Max-kAND is approximable to within a factor that roughly looks like $2^{-(k-1)}(1-O(1/k))$ - see [6] for an exact expression.) In particular this gives an infinite subfamily of CSPs that is non-trivially approximable by the algorithm from [13]. [6] also pin down the approximability of some other symmetric functions. Another work, by Chou,

[5] This corollary is not immediate from the theorem statement, but uses some additional aspects of the proof. See [13, Theorem 2.14] for details.

Golovnev, Shahrasbi, Sudan and Velusamy [10], also analyzes the sketching approximability of some linear threshold functions, giving some infinite families that are approximation-resistant to $o(\sqrt{n})$-space sketching algorithms and other infinite families that are approximable by polylog space sketching algorithms.

Streaming Lower Bounds

While the classification essentially only rules out sketching algorithms using $o(\sqrt{n})$-space for the hard problems, for a broad class of problems it even rules out non-trivial streaming algorithms. In fact for all problems it pins down the polylogarithmic space approximability to within a factor of q – we expand on this later below, but first speak about broad classes of approximation resistant problems. We start with some definitions.

We say that a distribution $\mathcal{D}$ on $\mathbb{Z}_q^k$ is one-wise independent if for every $i \in [k]$ we have that when $X = (X_1, \ldots, X_k)$ is sampled according to $\mathcal{D}$, then X_i is distributed uniformly over $\mathbb{Z}_q$. We say that $f : \mathbb{Z}_q^k \to \{0,1\}$ supports one-wise independence if there is a one-wise independent distribution $\mathcal{D}$ supported on a subset of the satisfying assignments of f, i.e., if $\mathbf{a} \in \mathbb{Z}_q^k$ has positive probability under $\mathcal{D}$ then $f(\mathbf{a}) = 1$. We say that $\mathcal{F}$ *supports one-wise independence* if every function $f \in \mathcal{F}$ supports one-wise independence. We say that $\mathcal{F}$ *weakly supports one-wise independence* if there exists $\mathcal{F}' \subseteq \mathcal{F}$ such that $\rho_{\min}(\mathcal{F}') = \rho_{\min}(\mathcal{F})$ and $\mathcal{F}'$ supports one-wise independence.

▶ **Theorem 2** ([13, Theorem 2.17]). *If $\mathcal{F}$ weakly supports one-wise independence, then $\mathcal{F}$ is approximation-resistant to $o(\sqrt{n})$-space streaming algorithms.*

Many natural families support one-wise independence. For readers familiar with some of these problems, we name some here without definitions: Max-Exact-kSAT for $k \geq 2$, Max-kOR, Max-CUT, Max-qColoring, Max-Unique-Games$_q$ to name a few. All of these problems turn out to be approximation-resistant by the above theorem.

Linear space lower bounds

Another direction of work has tried to extend the results of [24], i.e., $\Omega(n)$-space streaming lower bounds, to problems beyond Max-CUT. Here, we are far from a full understanding, but we do get approximation resistance for a (strict) subclass of families supporting one-wise independence. We define the families next.

For a function $f : \mathbb{Z}_q^k \to \{0,1\}$ and $\mathbf{a} \in \mathbb{Z}_q^k$ we define the *width of f at $\mathbf{a}$* to be $\omega_{\mathbf{a}}(f) = \frac{1}{q}|\{\theta \in \mathbb{Z}_q | f(\mathbf{a} + (\theta, \theta, \cdots, \theta)) = 1\}|$. We define the *width* of f to be the quantity $\omega(f) = \max_{\mathbf{a} \in \mathbb{Z}_q^k}\{\omega_{\mathbf{a}}(f)\}$, i.e. the maximum over $\mathbf{a}$ of the width of f at $\mathbf{a}$. (Roughly the set $L_{\mathbf{a}} = \{\mathbf{a} + (\theta, \theta, \cdots, \theta) | \theta \in \mathbb{Z}_q\}$ is a *line* through $\mathbb{Z}_q^k$ and $\omega_{\mathbf{a}}(f)$ measures the density of the intersection of this line with $f^{-1}(1)$, and the width of f is the widest such intersection.) We define the *width* of $\mathcal{F}$, denoted $\omega(\mathcal{F})$, to be the minimum over $f \in \mathcal{F}$ of the width of f. Note that $1/q \leq \omega(\mathcal{F}) \leq 1$ for every $\mathcal{F}$. Finally we say that $\mathcal{F}$ is *wide* if $\omega(\mathcal{F}) = 1$, i.e., the width is maximal.

A simple example of a wide family is the k-equality function $f_{k\mathsf{EQ}}$ where $f_{k\mathsf{EQ}}(u_1, \ldots, u_k) = 1$ if and only $u_1 = \cdots = u_k$. Note that every wide family supports one-wise independence. But there exist functions supporting one-wise independence that are not wide: For example $\oplus_3 : \mathbb{Z}_2^3 \to \{0,1\}$ given by $\oplus_3(a,b,c) = a + b + c \pmod 2$ supports one-wise independence but has width $1/2$.

The following theorem is shown in joint work with Chou, Golovnev, Velingker and Velusamy [11].

▶ **Theorem 3** ([11, Theorem 1.1]). *For every wide family $\mathcal{F}$, Max-CSP($\mathcal{F}$) is approximation-resistant to $o(n)$-space streaming algorithms.*

We will not cover any aspects of the proof of this theorem in this article, except to say that it builds on the proof of [24] following exactly the same sequence of steps, while replacing every step in their proof with ingredients needed to handle k-ary functions over non-Boolean alphabets. While the class of functions covered by this theorem is even smaller than the set covered by Theorem 2, it suffices to imply the following theorem which pins down the approximability of every Max-CSP($\mathcal{F}$) to within a factor of q.

▶ **Theorem 4** ([11, Theorem 4.3]). *For every family $\mathcal{F}$ and every $\epsilon > 0$, $(\omega(\mathcal{F}) - \epsilon, \rho(\mathcal{F}) + \epsilon)$-Max-CSP($\mathcal{F}$) requires $\Omega(n)$ space for every streaming algorithm. Consequently, for every $\mathcal{F}$ the largest α for which Max-CSP($\mathcal{F}$) is α-approximable by a $o(n)$-space streaming algorithm satisfies $\alpha \in \left[\rho_{\min}(\mathcal{F}), \frac{\rho_{\min}(\mathcal{F})}{\omega(\mathcal{F})}\right]$.*

We remark that while Theorem 4 immediately implies Theorem 3, the proof in [11] essentially derives the former from the latter using simple arguments.

4.1 Aside: Ordering CSPs

Before turning to some of the technical ingredients in the proofs we take a brief detour to cover an application of the results described above to a somewhat different class of optimization problems called ordering CSPs. We describe this class informally first: Recall that the solution space of the standard CSPs (the ones we work with in the rest of this paper) comes from a product set, namely an n-tuple of variables $(X_1, \ldots, X_n)$ takes values from $\mathbb{Z}_q \times \mathbb{Z}_q \times \cdots \times \mathbb{Z}_q$. A variation of this theme considers the setting where the variables need to be ordered, i.e., the $(X_1, \ldots, X_n)$ take on values from $\mathrm{Sym}_n = \{\pi : [n] \to [n] \mid \pi \text{ is one-to-one}\}$. (I.e., $X_i = \pi(i)$ where π is a permutation.) The natural notion of local constraints on ordering problems pick sequences of k distinct variables out of the n variables (as in standard CSPs) and look at the ordering from Sym_k induced by these k variables and constrain them. Thus a constraint function in ordering CSPs is given by $\Pi : \mathrm{Sym}_k \to \{0, 1\}$ and constraint families are a set of constraint functions. Thus for every k and every family $\mathcal{F} \subseteq \{\Pi : \mathrm{Sym}_k \to \{0, 1\}\}$ we get an ordering CSP, denoted Max-OCSP($\mathcal{F}$). (Note that unlike in standard CSPs, there is no notion of an alphabet or q in the case of ordering CSPs.)

Two examples of ordering CSPs include the Maximum Acyclic Subgraph (MAS) problem and the Betweenness problem. The former asks, given a directed graph G, to find the largest acyclic subgraph in it. This problem is captured as Max-OCSP($\{<\}$) where $<: \mathrm{Sym}_2 \to \{0, 1\}$ satisfies $<(\pi) = 1$ if and only if $\pi(1) < \pi(2)$. By placing the constraint $<(i, j)$ for every directed edge (i, j) in a graph G, we get an Max-OCSP($<$) instance that exactly captures the MAS instance. Betweenness is the ordering problem where constraints are given by a triple of variables and require that the ordering place the middle variable between the first and third (though allowing either of the first or the third to be the higher ranked variable). Once again it can be naturally formulated as an ordering CSP.

With ordering CSPs again, one can ask what is the trivial approximability of an ordering CSP and when can a ordering CSP be solved non-trivially. Both questions turn out to have simple answers though somewhat disappointing ones from the algorithmic point of view. Note that a random ordering satisfies a constraint Π with probability $\rho(\Pi) \stackrel{\text{def}}{=} \frac{1}{k!} \cdot |\Pi^{-1}(1)|$. Letting $\rho(\mathcal{F}) = \min_{\Pi \in \mathcal{F}}\{\rho(\Pi)\}$ we get that every instance of Max-OCSP($\mathcal{F}$) has value at least $\rho = \rho(\mathcal{F})$, and thus ρ-approximation is trivial. It turns out that there are no algorithms (that can do better for any $\mathcal{F}$ running in $o(n)$-space), as shown in the following theorem from joint work with Singer and Velusamy [30].

▶ **Theorem 5.** *For every k, every family $\mathcal{F} \subseteq \{\Pi : \mathrm{Sym}_k \to \{0, 1\}\}$ and every $\epsilon > 0$, every streaming $(\rho(\mathcal{F}) + \epsilon)$-approximation algorithm for Max-OCSP($\mathcal{F}$) requires $\Omega(n)$ space.*

5 Some ideas behind the proofs

5.1 The $\Omega(\sqrt{n})$ space lower bound for Max-CUT

We start with the lower bound from [22] on the Max-CUT problem. We start with some basic ideas about lower bounds. Lower bounds in streaming are typically "distributional". To prove a lower bound on (γ,β)-Max-CSP($\mathcal{F}$) for some $\gamma,\beta,\mathcal{F}$, for every sufficiently large n we construct two distributions of instances on n variables – the **YES** and **NO** distributions. The **YES** distributions are supported with probability $1-o(1)$ on instances from the set $\Gamma=\{\Psi|\mathsf{val}_\Psi\geq\gamma\}$. Similarly, the **NO** distributions are supported with probability $1-o(1)$ on instances from the set $B=\{\Psi|\mathsf{val}_\Psi\leq\beta\}$. In the case of Max-CUT we will thus consider a **YES** distribution supported (with probability one) on bipartite graphs, and **NO** instances will have cut value at most $1/2+o(1)$ (with probability $1-o(1)$). The goal is to prove that for any space s algorithm **ALG** with $s=o(\sqrt{n})$ the distribution of the final state of **ALG** in the **YES** and **NO** cases are very close in total variation distance. (For distributions $\mathcal{D},\mathcal{D}'$ supported on some set Ω the total variation distance, denoted $\|\mathcal{D}-\mathcal{D}'\|_{\mathrm{tv}}$, is the quantity $\frac{1}{2}\sum_{\omega\in\Omega}|\mathcal{D}(\omega)-\mathcal{D}'(\omega)|$.) Since the inputs are random, it suffices to consider deterministic $s(n)$-space bounded algorithms.

Both distributions are parameterized by two constants: a small $\alpha\in(0,1)$ and large, but constant, integer T. The graphs are defined on vertex set $[n]$ and have roughly $(\alpha/2)\cdot T\cdot n$ edges. These edges come as the union of T matchings $M'_1,\ldots,M'_T$, each with roughly $\alpha n/2$ edges.In the **NO** distribution these matchings will just be uniform matchings of the right size (we will get to the exact distribution of size shortly). In the **YES** distribution a random cut of $[n]$ is chosen by picking a vector $\mathbf{x}\in\{0,1\}^n$ uniformly at random and letting the cut be $\{i|x_i=1\}$. The matchings $M_1,\ldots,M_t$ are uniform subject to the condition that every matched edge crosses the cut. The lower bound is proved by a "hybrid argument" involving T steps. For $t\in\{0,\ldots,T\}$ let S^Y_t denote the state of **ALG** after seeing the first t matchings from the **YES** distribution, and similarly let S^N_t denote the state of **ALG** after the first t matchings from the **NO** distribution. By definition we have $S^Y_0=S^N_0$. The key step is to prove that for every t,

$$\|S^Y_t-S^N_t\|_{\mathrm{tv}}\text{ is small assuming }\|S^Y_{t-1}-S^N_{t-1}\|_{\mathrm{tv}}\text{ is small,}\tag{1}$$

and to use this result inductively to conclude $\|S^Y_T-S^N_T\|_{\mathrm{tv}}$ is small which shows that the two distributions are not distinguishable by small space algorithms. By construction the **YES** distribution is supported on bipartite graphs. If αT is sufficiently large then it can be argued by a standard Chernoff plus union bound that with probability $1-o(1)$, a graph from the **NO** distribution also has value at most $1/2+o(1)$ and together these suffice for the lower bound on Max-CUT. We thus turn to the proof of Equation (1).

The upper bound works by designing two-party one way communication problem that captures the added distinguishability of **YES** from **NO** conditioned on knowing $S^Y_{t-1}\approx_d S^N_{t-1}$ (where $\approx_d$ indicates that the two random variables are close in terms of total variation distance). A rough abstraction of this problem is as follows: Alice, who knows $\mathbf{x}$ must send some information about it to Bob. This information may capture information such as S^Y_{t-1} and/or S^N_{t-1}, both of which may in principle depend on $\mathbf{x}$, but should be limited to $o(\sqrt{n})$ bits. Now Bob, who gets to see M_t which is either (in the **YES** case) a random matching crossing the cut given by $\mathbf{x}$ or (in the **NO** case) a random matching, must distinguish the two.

It turns out a problem very similar to this was already defined and studied in the literature. Specifically, Gavinksy, Kempe, Kerenedis, Raz and de Wolf [18] define the Boolean Hidden Matching (BHP) problem where Alice is given a uniform vector $\mathbf{x}\in\mathbb{Z}_2^n$ and Bob is given a

matching $\widetilde{M}$ with $m = \alpha n$ edges on vertex set $[n]$ drawn uniformly among all such matchings, and a 0/1 labelling $\mathbf{w} \in \mathbb{Z}_2^m$ on the edges where in the **YES** case, the label w_e of an edge $e = (i, j)$ satisfies $w_e = x_i + x_j \pmod 2$, while in the **NO** case w_e's are uniformly random and independent. The goal of the communication is for Bob to distinguish the **YES** case from the **NO** case. [18] show that this problem requires $\Omega(\sqrt{n})$ bits of communication to achieve constant advantage in distinguishing. (The advantage of a protocol is the probability that the protocol outputs 1 in the **YES** distribution minus the probability it does so in the **NO** distribution. Specifically the [18] result shows that for every $\delta > 0$ there exists $\tau > 0$ such that for every $\alpha < 1/2$ and every sufficiently large n, every protocol that achieves advantage δ must communicate at least $\tau\sqrt{n}$ bits. These quantifiers are somewhat important as we will see below.)

To use the BHM lower bound from [18] we need to address two issues. First the input to Bob in the BHM problem is not the same as coming from the streaming problem. This problem is easy to deal with – Bob is getting more information in the BHM problem than in the motivating Max-CUT based problem, and this only makes the lower bound even stronger. Formally Bob can reduce an instance of BHM to the streaming inspired-problem by dropping all the edges e that have label $w_e = 0$. This gives Bob roughly $\alpha n/2$ edges (since each edge crosses the cut with probability roughly $1/2$ and these are roughly independent events) reducing exactly to the setting in the streaming-inspired problem.

The second and more important issue is that the BHM problem was only "roughly" motivated by the streaming problem above – we need a more careful and formal argument connecting the two. Formally we consider the random variables, S_t^Y, S_t^N and a hybrid variable $\widetilde{S}$, where $\widetilde{S}$ is the state of **ALG** on receiving $M_1, \ldots, M_{t-1}$ from the **YES** distribution and M_t from the **NO** distribution. The BHM lower bound immediately implies that $\widetilde{S} \approx_d S_t^Y$: The only difference between the two states is the tth input which comes from the **YES** distribution for S_t^Y and from the **NO** distribution for $\widetilde{S}$; and the setup of BHM allows Alice to generate and communicate S_{t-1}^Y to Bob allowing Bob to compute the final state and use **ALG** to distinguish them. To complement we also have $\|\widetilde{S} - S_t^N\|_{\text{tv}} \leq \|S_{t-1}^Y - S_{t-1}^N\|_{\text{tv}}$ by the data processing inequality: $\widetilde{S}$ is determined by S_{t-1}^Y and $M_t \sim$ **NO** while S_t^N is determined from S_{t-1}^N and M_t. We stress a subtle point here: It is crucial that M_t is independent of S_{t-1}^Y and S_{t-1}^N for this inequality to be applicable, and this does hold in our case since the **NO** distribution is independent of $\mathbf{x}$ which is the only variable connecting the different matchings in the **YES** case. (This subtlety is the reason why extensions of this proof apply only to families supporting one-wise independence, or only give sketching lower bounds.)

We also comment briefly on the choice of various parameters such as α, T, ϵ (where our goal is to prove hardness of $(1, 1/2 + \epsilon)$-Max-CUT), δ (the advantage allowed in BHM) and τ (where the space lower bound is $\tau\sqrt{n}$). We want our bound to hold for every $\epsilon > 0$ so given ϵ, we first pick α small enough for the BHM lower bound to hold. In our case it holds for every $\alpha < 1/2$. Given this choice of α we pick T large enough so that a graph from the **NO** distribution with $\alpha T n$ edges is very likely not to have a Max-CUT of fractional size more that $1/2 + \epsilon$. Given this choice we pick δ small enough so that T applications of the hybrid argument still lead to negligible advantage in distinguishing **YES** from **NO**. Finally the τ we obtain is whatever is guaranteed by the BHM lower bound for this choice of δ.

Our eventual streaming and sketching lower bounds will extend the ideas from above, but we will return to those after describing the algorithms for Max-DICUT from [19, 14].

5.2 Bias-based algorithms for Max-DICUT

The key ingredient in the algorithm of [19] for Max-DICUT is the notion of the "bias" of a graph on vertex set $[n]$. For a vertex v in a directed graph, let in-deg(v) denote the number of incoming edges into v and let out-deg(v) denote the number of outgoing edges. Now define $\mathsf{bias}(v) = \text{in-deg}(v) - \text{out-deg}(v)$, and define $\mathsf{bias}(G) = \frac{1}{2m}\sum_{v\in[n]} |\mathsf{bias}(v)|$. Thus if we term the vector $(\mathsf{bias}(v))_{v\in[n]} \in \mathbb{R}^n$ to be the bias-vector of the graph, then the bias of the graph is essentially the ℓ_1 norm of this vector up to normalization. As mentioned already, the ℓ_1-norm and hence the bias of a graph can be estimated arbitrarily well by a streaming algorithm presented with a stream of edges using an algorithm from [20]. The key to the algorithms of [19] and [14] are inequalities relating the bias of a graph to the dicut value, that allow them to output lower bounds of the value of the dicut. (For uniformity we will only talk about the fractional value here and later and use val_G to denote this quantity.)

Note that by definition $0 \leq \mathsf{bias}(G) \leq 1$ for every graph G on m vertices. [19] show that $\mathsf{val}_G \leq \frac{1+\mathsf{bias}(G)}{2}$. This inequality follows easily from the observation that every cut must leave at least $|\text{in-deg}(v) - \text{out-deg}(v)|$ of the edges incident to v uncut. Since every uncut edge may be counted twice by this process, we get a lower bound of $\frac{1}{2}\sum_v |\text{in-deg}(v) - \text{out-deg}(v)|$ on the number of uncut edges.

[19] complement upper bound above with a lower bound: For every G we must have $\mathsf{val}_G \geq \mathsf{bias}(G)$. This is "constructive" (though not in streaming sublinear space) – the greedy cut which puts all vertices with positive bias on the sink side of the cut and the rest on the source side achieves this. (A simple argument to see is iterative: Remove directed cycles from the graph one at a time till we get a DAG. This does not alter the bias. Now remove maximal length directed paths - each such path contributes one to the non-normalized bias, and also contributes at least one edge to the dicut since by maximality the source of the path must have zero indegree and the sink must have zero out degree.)

Combining the two bounds above with the lower bound $\mathsf{val}_G \geq 1/4$ for every G gives a .4 approximation algorithm: The algorithm computes $\mathsf{bias}(G)$ and outputs $\max\{\mathsf{bias}(G), 1/4\}$. To improve on this [14] give an improved lower bound on val_G when $\mathsf{bias}(G) \leq 1/3$. Their bound is also "constructive" - they consider a random dicut where each vertex of positive bias is placed on the sink side with probability $1/2 + \delta$ independently (for some parameter δ that we will optimize later). Remaining vertices are placed on the sink side with probability $1/2 - \delta$ independently. They analyze the cut produced by this rounding after optimizing over δ and use the expected size as an additional lower bound. We won't reproduce their bound or analysis here, but only comment that the analysis involves optimizing degree two rational functions in δ. This already gives them a 4/9 approximation algorithm.

The choice of a single rounding probability for all vertices in the graph is somewhat surprising. (This probability may depend on the graph and bias, but once the graph is fixed all vertices get rounded with the same probability.) It seems like a choice made for ease of analysis - optimizing a single variable δ is easier than optimizing n variables! One could nevertheless ask – could we have done better with more careful choices? The surprising result from [14] is that this won't help and indeed no $o(\sqrt{n})$-space algorithm can improve on the bound above! So somehow $\mathsf{bias}(G)$ is the right quantity to compute, and rounding independently with the same probability for all vertices (upto the choice of the preferred side) is the right algorithm!

5.3 The framework of [13]

To extend the algorithm of the previous section to problems beyond Max-DICUT, we need to understand what are notions of bias of a variable and of the whole instance for Max-CSP($\mathcal{F}$) for general $\mathcal{F}$. (In this discussion we will assume $\mathcal{F}$ has a single function f though extensions to more functions are straightforward.) Recall that in the Max-DICUT problem constraints arrive as pairs (i, j) where the edge goes from vertex i to vertex j. Thus the in-degree of a vertex could be abstract as the number of constraints in which it is the second variable, and out-degree as the number of constraints where it is the first variable. We use this to motivate a new notion of bias of a variable X_i, denoted $\mathsf{d\text{-}bias}(i)$ (for detailed bias): This will be a k dimensional vector whose jth coordinate $\mathsf{d\text{-}bias}(i)_j$ records the number of constraints in which X_i appears as the jth variable in a constraint. Considering all the biases of all vertices gives us an $n \times k$ matrix $B = B(\Psi)$ with $B(i, j) = \mathsf{d\text{-}bias}(i)_j$ that "represents" an instance Ψ.

The main idea in [13] can roughly be captured as follows: *If there exists $t \in \mathbb{N}$ and instances Ψ_g and Ψ_b on t variables with $B(\Psi_g) = B(\Psi_b)$ such that $\mathsf{val}_{\Psi_g} \geq \gamma$ and $\mathsf{val}_{\Psi_b} \leq \beta$ then Max-CSP($\mathcal{F}$) can not be solved in $o(\sqrt{n})$ space by a sketching algorithm. Else it can be solved by a polylogarithmic space linear sketching algorithm.* A priori neither statement should be obvious and we will give some idea below as to why they are true. Furthermore even if the statements are true it is not clear how to decide which of the two conditions hold (since a priori one may have to enumerate over all n and all pairs of instances to determine if the condition is true). It turns out all the issues get answered rather nicely jointly. It turns out that it suffices to consider (weighted) instances on kq variables to answer the final question, and studying the space of these instances also leads to the algorithms and the lower bounds.

Below we elaborate on this and in particular why it suffices to consider instances on a finite number of variables. We work with the simpler setting of Boolean CSPs (so $q = 2$) on literals, i.e., when constraints can be applied to variables as well as their negations. We note that the resulting setting ($|\mathcal{F}| = 1$, $q = 2$ and constraints on literals) is the case considered in a preliminary work [12], whereas the more general result comes from [13]. While the latter is a stronger result, the former offers more intuition into the proofs.

In this setting of constraints over Boolean literals, we show we only need to consider instances involving k variables – and we explain how this happens. Suppose there are two instances on n variables: Ψ_g with $\mathsf{val}_{\Psi_g} \geq \gamma$ and Ψ_b with $\mathsf{val}_{\Psi_b} \leq \beta$ satisfying $B(\Psi_g) = B(\Psi_b)$. We show how to simplify the two CSPs. From now onwards it will be convenient to think of a weighted CSP instance as being a distribution on constraints - where a constraint is chosen with probability proportional to its weight. Now, since we are considering Boolean CSPs on *literals* we can flip variables as necessary (by flipping literals in all constraints) till we get that 1^n is the assignment achieving $\mathsf{val}_{\Psi_g}(1^n) \geq \gamma$. To preserve $B(\Psi_g) = B(\Psi_b)$ we flip variables in Ψ_g and Ψ_b together. Note that this flipping preserves $\mathsf{val}_{\Psi_b} \leq \beta$. Next we observe that we can assume Ψ_g and Ψ_b are symmetric under permutations: I.e. if some constraint $C(X_1, \ldots, C_n)$ appears in Ψ_g with some probability p then for every permutation $\pi : [n] \to [n]$ the constraint $C(X_{\pi(1)}, \ldots, X_{\pi(n)})$ also appears in Ψ_g with the same probability p. (We can convert any Ψ to a Ψ' satisfying this feature as follows: To pick a random constraint of Ψ', pick a random constraint C of Ψ and a uniformly random permutation and let the constraint produced by $C(X_{\pi(1)}, \ldots, X_{\pi(n)})$.) This transformation preserves $\mathsf{val}_{\Psi_g} \geq \gamma$ since 1^n, the assignment achieving the maximum value is fixed under permutations. We also have that Ψ_b is closed under permutations since the (empty!) set of assignments that achieves value greater than β is also closed under permutations. The fact that Ψ_g and Ψ_b are symmetric under permutations make them very simple: All that determines these instances

is the distribution supported on $\mathbb{Z}_2^k$ indicating the pattern of negations of the k variables in a randomly chosen constraint. The names of the variables are no longer relevant – since they are just a uniformly random sequence of k distinct variables! Suppose $\mathcal{D}_Y$ represents the distribution on $\mathbb{Z}_2^k$ given by Ψ_g and $\mathcal{D}_N$ the distribution given by Ψ_b. We now study these distributions further and they will lead us to the answers to the three issues raised earlier.

The space of $\mathcal{D}_Y$ and $\mathcal{D}_N$

We can go back from distributions $\mathcal{D}$ over $\mathbb{Z}_2^k$ to instances $\Psi_{\mathcal{D}}$ of $\mathsf{Max\text{-}CSP}(\mathcal{F})$ on k variables $X_1, \ldots, X_k$ as follows: A random constraint of $\Psi_{\mathcal{D}}$ is of the form $f(X_1 \oplus b_1, \ldots, X_k \oplus b_k)$ where $\mathbf{b} = (b_1, \ldots, b_k) \sim \mathcal{D}$. Now the fact that $\mathcal{D}_Y$ came from an instance Ψ_g of value at least γ implies that the all 1's assignment satisfies $\Psi_{\mathcal{D}_Y}$. The fact that $B(\Psi_g) = B(\Psi_b)$ implies that $\mathcal{D}_Y$ and $\mathcal{D}_N$ have the same marginals. It remains to interpret the implication that $\Psi_b \leq \beta$: We stress that it does not mean $\Psi_{\mathcal{D}_N}$ has value less than β - indeed $\Psi_{\mathcal{D}_N}$ can have value much larger than that or even γ! The implication turns out to be exactly the following: "For every $p \in [0,1]$ if $X_1, \ldots, X_k$ are assigned values identically and independently according to $\mathsf{Bern}(p)$ (i.e., they take values in $\mathbb{Z}_2$ with $\Pr[X_i = 1] = p$), then the expected value $\mathsf{val}_{\Psi_{\mathcal{D}_N}}(X_1, \ldots, X_k) \leq \beta$." I.e., no identical and independent probabilistic assignment to the variables satisfies many constraints.

It turn out we can now capture these considerations on $\mathcal{D}_Y$ and $\mathcal{D}_N$ in a nice mathematical framework and that will lead to matching algorithms and lower bounds. Note that a distribution on $\mathbb{Z}_2^k$ can be viewed as a vector in $\mathbb{R}^{2^k}$ in a natural way, and the space of all distributions is a convex set in $\mathbb{R}^{2^k}$. Now let $S_\gamma^Y(f)$ denote the subset of this set representing distributions $\mathcal{D}$ such that $\mathsf{val}_{\Psi_{\mathcal{D}}}(1^k) \geq \gamma$. Similarly let $S_\beta^N(f)$ denote the subset of distributions $\mathcal{D}$ such that for every $p \in [0,1]$, $\mathbb{E}_{\mathbf{b} \in \mathsf{Bern}(p)^k}[\mathsf{val}_{\Psi_{\mathcal{D}}}(\mathbf{b})] \leq \beta$. Both these sets are convex sets! (In particular for every p, the constraint $\mathbb{E}_{\mathbf{b} \in \mathsf{Bern}(p)^k}[\mathsf{val}_{\Psi_{\mathcal{D}}}(\mathbf{b})] \leq \beta$ is a linear constraint on $\mathcal{D}$, though we have infinitely many such constraints.) By construction the two sets are disjoint for $\beta < \gamma$, but they may still contain distributions with matching marginals! To see this we may project these two sets to their marginals: So let $K_\gamma^Y(f) \subseteq \mathbb{R}^k$ be the set of marginals of all distributions in $S_\gamma^Y(f)$ and similarly let $K_\beta^N(f)$ be the marginals of $S_\beta^N(f)$. The discussion thus far has reduced the question: "Do there exist n and instances Ψ_1 and Ψ_2 on n variables with $B(\Psi_1) = B(\Psi_2)$ such that $\mathsf{val}_{\Psi_1} \geq \gamma$ and $\mathsf{val}_{\Psi_2} \leq \beta$?" to the much simpler and finite dimensional question "Do $K_\gamma^Y(f)$ and $K_\beta^N(f)$ intersect?". (An affirmative answer to one question implies an affirmative answer to the other.)

Before turning to show why this leads to algorithms or lower bounds we first point out that the question of the intersection of these two sets is decidable. Specifically the intersection question can be posed as polynomial inequalities in $2^k + 1$ variables (2^k from $\mathcal{D}$ and one from p) of degree at most $k + 1$ with one variable (p) being universally quantified and the rest being existentially quantified. Results in the quantified theory of reals [5] easily show how to decide this question in space polynomial in the input size, which in our case is roughly 2^k to represent the function f (and whatever else is needed to specify γ and β).

Sketching lower bound when $K_\gamma^Y(f) \cap K_\beta^N(f) \neq \emptyset$

It turns out that the existence of two distributions with matching marginals is the crux of the $\mathsf{Max\text{-}CUT}$ lower bound of [22] and so extending to other settings is a reasonable hope. Specifically the $\mathsf{Max\text{-}CUT}$ lower bound relies on $\mathcal{D}_Y = \mathsf{Unif}(\{00, 11\})$ and $\mathcal{D}_N = \mathsf{Unif}(\mathbb{Z}_2^2)$. To extend to other problems and distributions, we use the same approach of dividing a long stream of constraints into T substreams of length αn. A communication problem captures

the additional information gained by a substream while a hybrid argument combines the information gained from the substreams. Both steps turn out to be different though and we elaborate on them below.

The BHM problem could be interpreted as arising from the associated distributions above in two different ways. In both Alice gets $\mathbf{x} \in \mathbb{Z}_2^n$ and Bob's first input is a matching on $[n]$, which specifies potential constraints: Bob's second input can be interpreted in two ways: (1) For each constraint, he gets information on whether $\mathbf{x}$ satisfies the constraint or not, (2) Using the fact that $\mathcal{D}_Y$ is uniform on a subgroup of $\mathbb{Z}_2^2$, Bob gets input on which coset the variables in the constraint come from. The first interpretation doesn't seem naturally amenable to the lower bound techniques which seem more tailored to understanding inputs that are uniform in the **NO** case. The second interpretation seems restricted to groups and cosets and in particular does not seem to support $\mathcal{D}_Y$ not being uniform on a set, leave alone a subgroup. However it is possible to extend this approach beyond such algebraic settings and this is what is done in [13]. To do so they introduce the $(\mathcal{D}_Y, \mathcal{D}_N)$-Randomized Mask Detection Problem (RMD) which is again a distribution distinguishability problem in the one-way communication setting: Here Alice gets a vector $\mathbf{x} \in \mathbb{Z}_2^n$ and Bob gets a k hypermatching with $m = \alpha n$ edges. Additionally Bob gets a vector $\mathbf{w} \in \mathbb{Z}_2^{km}$, or equivalently, one vector in $\mathbb{Z}_2^k$ associated with each hyperedge of the matching. In the **YES** case this vector associated with a hyperedge is the labels of $\mathbf{x}$ restricted to the vertices incident to the hyperedge masked (i.e., xor-ed, or summed in $\mathbb{Z}_2$) by a vector $\mathbf{b} \in \mathbb{Z}_2^k$ drawn according to $\mathcal{D}_Y$. Each mask vector $\mathbf{b}$ is drawn independently for every hyperedge. The **NO** distribution is similar with the difference that now $\mathbf{b} \sim \mathcal{D}_N$ independently for each edge.

[13] give a $\Omega(\sqrt{n})$ communication lower bound for this problem to achieve any constant advantage (this time for $\alpha < 1/k$). The lower bound works in two parts. First the extend the proof from [18] to general k in the setting where $\mathcal{D}_N$ is uniform on $\mathbb{Z}_2^k$. (As noted above this setting seems amenable to their proof technique). The second part of the proof shows how to use the first part to show hardness of RMD on distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ that differ in a "simple" way (in particular they differ in probabilities of at most four structured points in their support). They then complement this by showing that one can move from every $\mathcal{D}_Y$ to every $\mathcal{D}_N$ (with matching marginals) using a finite number of steps (as a function of q and k) where each step creates a "simple" difference in the sense above. A series of triangle inequalities now shows that $(\mathcal{D}_Y, \mathcal{D}_N)$-RMD is also indistinguishable to $o(\sqrt{n})$-communication protocols.

To convert the RMD lower bound into a lower bound on $\mathsf{Max\text{-}CSP}(\mathcal{F})$ we first need to interpret the RMD inputs as constraints of a $\mathsf{Max\text{-}CSP}(\mathcal{F})$ problem, and then to prove that combining T substreams preserves indistinguishability by streaming algorithms. The first step is natural: We apply constraints so that the hidden vector $\mathbf{x}$ is expected to satisfy γ fraction of the constraints in the **YES** case: Specifically if a hyperedge gives Bob the information $\mathbf{x}|_S + \mathbf{b}$ corresponding to the restriction of $\mathbf{x}$ to some sequence S of k variables masked by $\mathbf{b}$, then the resulting constraint negates literals according to $\mathbf{x}|_S + \mathbf{b}$, so that after the negations are applied, the input to the constraint is $\mathbf{b}$ which, by the condition that $\mathcal{D}_Y \in S_\gamma^Y(f)$ is expected to satisfy the constraint with probability γ. Similarly in the **NO** case every assignment is expected to satsify the constraint with probability at most β. Taking sufficiently many constraints (i.e., $\alpha T \to \infty$) allows us to apply Chernoff bounds and the union bound to conclude tight bounds on the value of the resulting CSPs in the **YES** and **NO** case.

The tricky part turns out to be the combination. When $\mathcal{D}_N$ is uniform, the same hybrid argument as in the $\mathsf{Max\text{-}CUT}$ case works and using this twice we conclude that if $\mathcal{D}_Y$ and $\mathcal{D}_N$ have uniform marginals then the resulting CSP instances are indistinguishable by $o(\sqrt{n})$ space

streaming algorithms. [13] also cover some slight extensions that allow them to cover hardness of Max-DICUT where the underlying distributions do not have uniform marginals. But for general $\mathcal{D}_Y$ and $\mathcal{D}_N$ with non-uniform marginals the method truly breaks down and produces instances of Max-CSP(f) that are distinguishable by polylogspace algorithms as pointed out by [9]. However in such cases it is possible to show that no sketching algorithm can work. This relies on an easy reduction from RMD to a T-player simultaneous communication problem where T players each get inputs independently according to the distribution of Bob's input and then need to communicate short messages to a referee whose goal is to distinguish the inputs being all from the **YES** distribution or from the **NO** distribution. This yields the lower bound of Theorem 1.

A sketching algorithm when $K^Y_\gamma(f) \cap K^N_\beta(f) = \emptyset$

We now turn to the complementary result, giving a sketching algorithm when $K^Y_\gamma(f) \cap K^N_\beta(f) = \emptyset$. If the sets do not intersect then there must be a hyperplane in $\mathbb{R}^k$ separating them. Let this plane be given by $\lambda_1, \ldots, \lambda_k$ and τ so that $\{\mathbf{a} \in \mathbb{R}^k | \sum_{i\in[k]} \lambda_i a_i \geq \tau\}$ contains $K^Y_\gamma(f)$. Since $K^Y_\gamma(f)$ and $K^N_\beta(f)$ are closed sets if they are disjoint there must be a gap separating them and so we also have $\theta > 0$ so that $\{\mathbf{a} \in \mathbb{R}^k | \sum_{i\in[k]} \lambda_i a_i \leq \tau - \theta\}$ contains $K^N_\beta(f)$.

It is natural to think of λ_i as representing a preference that the ith variable in this constraint has for taking the value 1 with higher λ_i's representing higher preferences. When the ith variable in a constraint is negated we let $-\lambda_i$ capture its preference. These preferences allow aggregation across constraints and yield the definition: For $j \in [n]$, let $\mathsf{bias}(\Psi, j)$ be the sum of the appropriate λ values over all constraints that variable j participates in. Define $\mathsf{bias}(\Psi) = \frac{1}{m}\sum_{j=1}^n |\mathsf{bias}(\Psi, j)|$. (We note that these definitions extend the Max-DICUT notions exactly). $\mathsf{bias}(\Psi)$ can be estimated as previously by appealing to ℓ_1 norm estimation algorithms. The algorithm for (γ, β)-Max-CSP($\mathcal{F}$) now reports **YES** if and only if $\mathsf{bias}(\Psi) \geq \tau - \theta/2$. This algorithm turns out to be correct, using analysis ideas that follow in a straightforward way from the construction of the convex sets. In case the reader wonders where the ℓ_1 estimator is suggested in the construction of the convex sets, this happens in the step where we passed from a general Ψ_1 and Ψ_2 with matching detailed bias matrix B, to assuming 1^n achieves the maximum value of Ψ_1. The computational challenge behind this vertex is to compute the maximal satisfying assignment, flipping literals of Ψ_1 according to this, and then computing its bias. This step is achieved computationally by the ℓ_1 norm estimation algorithm!

The general case

Up to now we focussed on the simpler case of $\mathcal{F} = \{f\}$, $q = 2$ and constraints being applied to literals. It turns out that this is the exact setting considered in the early version [12]. The extension to the general case, where $\mathcal{F}$ is not a singleton, constraints are applied only to variables, and q is general, appears in [13]. The extensions do manage to work out with no surprises (at least no unpleasant ones).

The elimination of the need to work with literals is the conceptually hard step but works out by working with qk variables which is different from k even in the Boolean case. Roughly our simplified picture of the extremal examples Ψ_1 and Ψ_2, uses two variables for each of the k positions in constraint that a variable can appear in: one corresponding to the unnegated variable X and one to the negated variable. To extend to general q we now use q variables per coordinate $i \in [k]$ – with variable $X_{i,\sigma}$ roughly capturing the "literal" $X_i + \sigma \pmod q$. The

extension to larger sets $\mathcal{F}$ is simple, we augment the detailed-bias information as well as the marginals to include information about which function $f \in \mathcal{F}$ the constraint is working with. This leads to sets $S^Y_\gamma(f)$, $S^N_\beta(f)$ that are extended to capture distributions over $\mathcal{F} \times \mathbb{Z}_q^k$ and the marginals $K^Y_\gamma(f)$, $K^N_\beta(f)$ now project to $\mathcal{F} \times [k] \times \mathbb{Z}_q$ dimensions. Somewhat surprisingly both the algorithm and the lower bounds extend to this setting with the ℓ_1 norm estimator replaced by an $\|\cdot\|_{1,\infty}$-norm estimator[6] of [1]. Deciding intersection of the two sets reduces to quantified systems with 2 alternations, and roughly $|\mathcal{F}|q^k$ variables and degree k. Perhaps the most complex part of the extension is the extension of the streaming lower bounds which work with two variants of the RMD problem. Also the reduction of the communication problems to the streaming problems is a bit delicate due to the absence of literals but works out in the end. We omit the many details, referring the reader to the original paper [13] for those.

6 Future directions

One can hope for many possible extensions to the dichotomy reported in Theorem 1. Perhaps the dichotomy extends as is to streaming algorithms (i.e., beyond sketching), perhaps even for linear space, perhaps even for randomly ordered streams, and maybe even for multipass algorithms. Unfortunately, while several extensions are still possible, the clean dichotomy does seem to fray quite a bit for each possible extension.

At the moment it is still plausible that the dichotomy extends as is to all $o(\sqrt{n})$ space streaming algorithms though there is no strong evidence in either direction. For space beyond $\sqrt{n}$ there do seem to be a number of new candidate algorithms so our expectation would be that the current dichotomy won't hold and there may exist more than two classes of problems. We note that we don't have concrete theorems proving this though. For randomly ordered streams as well as multipass algorithms there seem to be new algorithms in polylogarithmic space. This is the subject of an upcoming work by the author with Saxena, Singer and Velusamy. Finally the multipass setting seems to be the most challenging for the lower bounds. Here some remarkable works [2, 3], have shown strong space lower bounds but for progressively weak approximations. Here an interesting challenge is to establish a tight lower bound for any non-trivial CSP for arbitrarily large (but constant) number of passes.

References

1 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming Algorithms via Precision Sampling. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011, Palm Springs, CA, USA, October 23-25, 2011)*, pages 363–372, October 2011. doi:10.1109/FOCS.2011.82.

2 Sepehr Assadi, Gillat Kol, Raghuvansh R. Saxena, and Huacheng Yu. Multi-Pass Graph Streaming Lower Bounds for Cycle Counting, MAX-CUT, Matching Size, and Other Problems. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS 2020, Virtual, November 16-19, 2020)*, pages 354–364, November 2020. doi:10.1109/FOCS46700.2020.00041.

3 Sepehr Assadi and Vishvajeet N. Graph streaming lower bounds for parameter estimation and property testing via a streaming XOR lemma. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 612–625. ACM, 2021. doi:10.1145/3406325.3451110.

[6] For matrix M with rows indexed by $[n]$ and columns by $\mathbb{Z}_q$ the $(1,\infty)$-norm is given by $\|M\|_{1,\infty} = \sum_{i=1}^{n} \max_{j \in \mathbb{Z}_q} |M_{ij}|$.

4 Libor Barto and Marcin Kozik. Robust satisfiability of constraint satisfaction problems. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 931–940. ACM, 2012. doi:10.1145/2213977.2214061.

5 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Springer, 2006.

6 Joanna Boyland, Michael Hwang, Tarun Prasad, Noah Singer, and Santhoshini Velusamy. Closed-form expressions for the sketching approximability of (some) symmetric Boolean CSPs. *CoRR*, abs/2112.06319, 2021. arXiv:2112.06319.

7 Andrei A. Bulatov. A Dichotomy Theorem for Nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS 2017, Berkeley, CA, USA, October 15-17, 2017)*, pages 319–330, October 2017. doi:10.1109/FOCS.2017.37.

8 Jin-Yi Cai and Xi Chen. *Complexity Dichotomies for Counting Problems: Volume 1, Boolean Domain*. Cambridge University Press, 2017.

9 Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh Saxena, Zhao Song, and Huacheng Yu. Personal communication, March 2021.

10 Chi-Ning Chou, Alexander Golovnev, Amirbehshad Shahrasbi, Madhu Sudan, and Santhoshini Velusamy. Sketching approximability of (weak) monarchy predicates. Manuscript, May 2022.

11 Chi-Ning Chou, Alexander Golovnev, Madhu Sudan, Ameya Velingker, and Santhoshini Velusamy. Linear space streaming lower bounds for approximating CSPs. *CoRR*, abs/2106.13078, 2021. Extended abstract in Proc. STOC 2022. arXiv:2106.13078.

12 Chi-Ning Chou, Alexander Golovnev, Madhu Sudan, and Santhoshini Velusamy. Approximability of all Boolean CSPs with linear sketches. *CoRR*, abs/2102.12351, 2021. arXiv:2102.12351.

13 Chi-Ning Chou, Alexander Golovnev, Madhu Sudan, and Santhoshini Velusamy. Approximability of all finite CSPs with linear sketches. *CoRR*, abs/2105.01161, 2021. Extended abstract appears in Proc. FOCS 2021. arXiv:2105.01161.

14 Chi-Ning Chou, Alexander Golovnev, and Santhoshini Velusamy. Optimal Streaming Approximations for all Boolean Max-2CSPs and Max-kSAT. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS 2020, Virtual, November 16-19, 2020)*, pages 330–341. IEEE Computer Society, November 2020. doi:10.1109/FOCS46700.2020.00039.

15 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2001. doi:10.1137/1.9780898718546.

16 Víctor Dalmau and Andrei A. Krokhin. Robust satisfiability for CSPs: Hardness and algorithmic results. *ACM Trans. Comput. Theory*, 5(4):15:1–15:25, 2013. doi:10.1145/2540090.

17 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

18 Dmitry Gavinsky, Julia Kempe, Iordanis Kerenidis, Ran Raz, and Ronald de Wolf. Exponential separation for one-way quantum communication complexity, with applications to cryptography. *SIAM Journal on Computing*, 38(5):1695–1708, December 2008. Conference version in STOC 2007. doi:10.1137/070706550.

19 Venkatesan Guruswami, Ameya Velingker, and Santhoshini Velusamy. Streaming Complexity of Approximating Max 2CSP and Max Acyclic Subgraph. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX 2017, Berkeley, CA, USA, August 16-18, 2017)*, volume 81 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, August 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.8.

20 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM*, 53(3):307–323, May 2006. Conference version in FOCS 2000. doi:10.1145/1147954.1147955.

21 Piotr Indyk, Andrew McGregor, Ilan Newman, and Krzysztof Onak. Open problems in data streams, property testing, and related topics, June 2011. Compiled from IITK Workshop on Algorithms for Processing Massive Data Sets (2009) and Bertinoro Workshop on Sublinear Algorithms (2011).

22 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Streaming lower bounds for approximating MAX-CUT. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015, San Diego, California, USA, January 4-6, 2015)*, pages 1263–1282. Society for Industrial and Applied Mathematics, January 2015. doi:10.1137/1.9781611973730.84.

23 Michael Kapralov, Sanjeev Khanna, Madhu Sudan, and Ameya Velingker. $(1 + \omega(1))$-approximation to MAX-CUT requires linear space. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017, Barcelona, Spain, January 16-19, 2017)*, pages 1703–1722. Society for Industrial and Applied Mathematics, January 2017. doi:10.5555/3039686.3039798.

24 Michael Kapralov and Dmitry Krachun. An optimal space lower bound for approximating MAX-CUT. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019, Phoenix, AZ, USA, June 23-26, 2019)*, pages 277–288. Association for Computing Machinery, June 2019. doi:10.1145/3313276.3316364.

25 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The Approximability of Constraint Satisfaction Problems. *SIAM Journal on Computing*, 30(6):1863–1920, January 2001. Conference versions in STOC 1997 and CCC 1997. doi:10.1137/S0097539799349948.

26 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 6th Annual Conference on Innovations in Theoretical Computer Science (ITCS 2015, Rehovot, Israel, January 11-13, 2015)*, pages 367–376. Association for Computing Machinery, 2015. doi:10.1145/2688073.2688093.

27 Gábor Kun, Ryan O'Donnell, Suguru Tamaki, Yuichi Yoshida, and Yuan Zhou. Linear programming, width-1 CSPs, and robust satisfaction. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 484–495. ACM, 2012. doi:10.1145/2090236.2090274.

28 Prasad Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008, Victoria, BC, Canada, May 17-20, 2008)*, pages 245–254, 2008. doi:10.1145/1374376.1374414.

29 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978, San Diego, CA, USA, May 1-May 3, 1978)*, pages 216–226. Association for Computing Machinery, May 1978. doi:10.1145/800133.804350.

30 Noah Singer, Madhu Sudan, and Santhoshini Velusamy. Streaming approximation resistance of every ordering CSP. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX 2021, August 16-18, 2021)*, volume 207 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, September 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.17.

31 Dmitriy Zhuk. A Proof of the CSP Dichotomy Conjecture. *Journal of the ACM*, 67(5):30:1–30:78, August 2020. Conference version in FOCS 2017. doi:10.1145/3402029.

A Brief Tour in Twin-Width

Stéphan Thomassé ✉
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

This is an introduction to the notion of twin-width, with emphasis on how it interacts with first-order model checking and enumerative combinatorics. Even though approximating twin-width remains a challenge in general graphs, it is now well understood for ordered graphs, where bounded twin-width coincides with many other complexity gaps. For instance classes of graphs with linear FO-model checking, small classes, or NIP classes are exactly bounded twin-width classes. Some other applications of twin-width are also presented.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Twin-width, matrices, ordered graphs, enumerative combinatorics, model theory, algorithms, computational complexity, Ramsey theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.6

Category Invited Talk

1 Introduction

One of the most natural ways to understand discrete structures is to measure their complexity. A reasonable expectation is that the class of structures with bounded measure is not too difficult to understand and manipulate. Ideally, bounded measure classes should be simple with respect to several points of view such as "computationally hard problems can be solved efficiently" or "the number of structures of size n is a small function of n" and should also enjoy some stability like "the measure should stay bounded if one performs moderate deterministic changes to the structures".

The first difficulty to provide a general purpose complexity measures on graphs is that it quickly boils down to the basic question: What is a simple 01-matrix? Fortunately this problem has already been addressed long ago and the answer is very simple: a matrix M is complex if it contains all small matrices up to some (large) size. Consequently, the complexity measure $vc(M)$ could be the maximum k for which all 01-valued $k \times k$-matrices appear in M. This is equivalent to the well-known Vapnik-Cervonenkis dimension and indeed classes of matrices with bounded VC-dimension have moderate growth $O(2^{n^{2-\varepsilon}})$ and some problems are computationally easier (for instance the minimum hitting set problem can be approximated). Interestingly, these two properties characterize bounded VC-dimension for classes of matrices closed under submatrices. This is the bounded/unbounded VC-dimension gap, which is (arguably) the first question one should ask when investigating a class of structures.

The exact same idea can be used to measure the complexity of a permutation matrix M (exactly one 1 per row and per column): let $mt(M)$ be the maximum k for which all $k \times k$ permutation matrices appear in M. Marcus and Tardos [17], proving the Stanley-Wilf conjecture, showed that the growth of a class of permutation matrices with bounded measure mt is c^n. Using their method, Guillemot and Marx [14] showed that checking if a fixed $k \times k$ permutation matrix F is contained in an $n \times n$ permutation matrix P can be done in linear time $f(k)n$ (when both coded as permutations). Their breakthrough method was a completely new win/win scheme: they showed that unless one can detect F in P, then P can be iteratively contracted in linear time and the contraction scheme allows *then* to test if indeed F is a subpermutation of P. Note that permutation matrices are ordered

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 6; pp. 6:1–6:5


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

matrices, where rows and columns are linearly ordered. We showed in Twin-width IV [9] that the natural generalization of the mt parameter to general ordered matrices is the following: a matrix M has *grid rank* k if this is the maximum value for which M has a $k \times k$ block division in which every block has rank at least k.

Guillemot and Marx concluded their paper by asking if their technique for permutations could be generalized for graphs. This was the goal of our paper Twin-width I [12] where we defined the twin-width of a graph G as the minimum degree of error in a contraction sequence of G (we iteratively contract pairs of vertices, two contracted groups of vertices forming an error edge if there is both an edge and a non edge between them). Precisely, the twin-width $tww(G)$ of a graph G on n vertices is the minimum k such that: there exists a sequence of partitions $P_n, \dots, P_1$ of $V(G)$ where each P_{i-1} is obtained from P_i by merging two parts, and such that for every part X in P_j, the number of parts Y in P_j which are not homogeneous with X is at most k. Here two disjoint sets X, Y are *homogeneous* if the relation between $x \in X$ and $y \in Y$ does not depend of the choices of x, y (therefore homogeneity, and hence twin-width, is also defined for binary multirelations). So bounded twin-width corresponds to maximum degree in every *error graph* G_j which vertices are the parts of P_j and edges are the non homogeneous pairs. If we impose further that all components of graphs G_j have bounded size, we have shown in Twin-width VI [10] that the obtained parameter is equivalent to rank-width. Hence twin-width generalizes rank-width but also captures strict minor closed classes, or strict permutation graphs.

Mimicking Guillemot and Marx argument for permutations, it is not hard to show that if one has access to such a sequence $P_n, \dots, P_1$ certifying twin-width k, then one can test if some fixed graph H of size t is an induced subgraph of G in linear time $f(k,t).n$. One of the main result of Twin-width I is that we can moreover test any FO-formula of depth t in time $f(k,t).n$. Also, generalizing Marcus-Tardos' result on permutations, we could prove in Twin-width II [6] that the number of (labelled) graphs of size n with twin-width bounded by some constant is at most $c^n.n!$ (we call this a *small class*). This result implies in particular that the class of (sub)cubic graphs (degree at most 3) does not have bounded twin-width, since it is not small. But so far we have no "deterministic" construction of a cubic graph with arbitrarily high twin-width. One can naturally wonder if these two implications (easyness of FO-model checking and small property) could be equivalent to bounded twin-width.

This is unfortunately not the case: since FO-model checking can be solved in linear time on bounded degree graphs, there are classes of graphs on which FO-model checking is tractable and for which twin-width is unbounded. From the counting point of view, we conjectured in [6] the equivalence between bounded twin-width and being a small class. Sadly again, we could prove in Twin-width VII [8] that there are (countable) Cayley graphs of finitely generated groups with unbounded twin-width, while any such Cayley graph defines a small class. Thus bounded twin-width for general graphs does not seem to be equivalent to some computational complexity class, nor it seems to be definable via counting. Nevertheless, bounded twin-width is a particularly stable notion since every first order interpretation of a bounded twin-width class has also bounded twin-width. For instance squares of planar graphs have bounded twin-width.

We still have a very limited understanding of twin-width, and especially for bounded degree graphs: not only we do not have an algorithm to approximate it, but we do not know what could be a certificate of high twin-width, and we are not even able to construct by hand a cubic graph of high twin-width. So why twin-width is so hard to handle, given that it enjoys so many nice properties? The answer is that twin-width indeed corresponds to a crucial complexity gap, but for ordered graphs (a binary birelation consisting of a graph and a linear order on its vertices) rather than for graphs.

Indeed, in Twin-width IV [9] we could show that for classes of ordered graphs, bounded twin-width, linear FO-model checking and being a small class are equivalent. Moreover these three characterizations are in turn equivalent to the fact that the adjacency matrices of the graphs, ordered by their linear order, have bounded grid-rank. Finally, approximating twin-width for ordered graphs can be done in polynomial time. In particular, the bounded/unbounded twin-width gap for ordered matrices is as fundamental as the one of VC-dimension as it has many equivalent formulations coming from other domains. For instance, for ordered graphs, the NIP property in model theory coincides with bounded twin-width. Since any graph G with twin-width k has a linear order L for which (G, L) has twin-width k, the difficulty of twin-width for general graphs seems to come from the fact that we have lost the information encoded in the linear order.

Hence an appealing strategy to show that a graph G has (reasonably) bounded twin-width is to be able to guess a suitable linear order T on its vertices since we have the machinery to efficiently approximate the twin-width of the ordered graph (G, T). For instance, we could prove that minor closed classes have bounded twin-width by using a Lex-DFS to provide the linear order. We can also take advantage of the stability of twin-width by FO-interpretation. Let us illustrate this on some example: Assume that we want to approximate the twin-width of a bipartite graph G with bipartition A, B in which B is linearly ordered by $<$. We would not have a clue if B would not have been ordered, and we can directly conclude if both A and B are ordered, so what about this "semi-ordered" case? The answer is quite easy: associate to each vertex $a \in A$ the characteristic 01 vector of its neighbors in B, ordered by $<$, and sort A by lexicographic order. This is a first-order interpretation, so the order we find on A cannot increase twin-width too much. Hence now A is ordered, and twin-width can be approximated. Note that if several vertices of A have the same neighborhood, we can pairwise contract them since this does not affect twin-width. This is probably the best advice to try to compute twin-width: find an order. Is there a general algorithm to find it?

To conclude, let us observe that as a way to characterize simple matrices, twin-width is a very general tool which can apply to many topics. For instance bounded twin-width is a group invariant and finitely generated groups can have either bounded or unbounded twin-width (but again we have no explicit presentation of any unbounded twin-width group). Another field where matrices are central is linear programming. We have showed in Twin-width III [7] that when a matrix has bounded twin-width, there is a constant duality gap between minimum hitting set and maximum packing (while bounded VC-dimension only bridges the fractional gap for hitting set). We provide in the references a list of recent publications with better bounds on twin-width of classes ([1, 2, 3, 15, 18]), on computing twin-width ([4, 5]), on using twin-width for algorithms ([11, 16]) and for data-structures [20], and more topics ([13, 19, 21, 22]) that we cannot unfortunately cover here. We believe that there are many other aspects of twin-width waiting to be discovered.

References

1 Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Bounds for the twin-width of graphs. *CoRR*, abs/2110.03957, 2021. `arXiv:2110.03957`.

2 Jakub Balabán and Petr Hlinený. Twin-width is linear in the poset width. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.IPEC.2021.6`.

3 Jakub Balabán, Petr Hlinený, and Jan Jedelský. Twin-width and transductions of proper k-mixed-thin graphs. *CoRR*, abs/2202.12536, 2022. arXiv:2202.12536.

4 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. *CoRR*, abs/2112.08953, 2021. arXiv:2112.08953.

5 Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-width VIII: delineation and win-wins. *CoRR*, abs/2204.00722, 2022. doi:10.48550/arXiv.2204.00722.

6 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1977–1996. SIAM, 2021. doi:10.1137/1.9781611976465.118.

7 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.35.

8 Édouard Bonnet, Colin Geniet, Romain Tessera, and Stéphan Thomassé. Twin-width VII: groups. *CoRR*, abs/2204.12330, 2022. doi:10.48550/arXiv.2204.12330.

9 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021. arXiv:2102.03117.

10 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1036–1056. SIAM, 2022. doi:10.1137/1.9781611977073.45.

11 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.10.

12 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

13 Édouard Bonnet, O-joung Kwon, and David R. Wood. Reduced bandwidth: a qualitative strengthening of twin-width in minor-closed classes (and beyond). *CoRR*, abs/2202.11858, 2022. arXiv:2202.11858.

14 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101. SIAM, 2014. doi:10.1137/1.9781611973402.7.

15 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. *CoRR*, abs/2201.09749, 2022. arXiv:2201.09749.

16 Stefan Kratsch, Florian Nelles, and Alexandre Simon. On triangle counting parameterized by twin-width. *CoRR*, abs/2202.06708, 2022. arXiv:2202.06708.

17 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the stanley-wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. doi:10.1016/j.jcta.2004.04.002.

18 William Pettersson and John Sylvester. Bounds on the twin-width of product graphs. *CoRR*, abs/2202.11556, 2022. arXiv:2202.11556.

19 Michal Pilipczuk and Marek Sokolowski. Graphs of bounded twin-width are quasi-polynomially χ-bounded. *CoRR*, abs/2202.07608, 2022. arXiv:2202.07608.

20 Michal Pilipczuk, Marek Sokolowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.52.

21 Wojciech Przybyszewski. VC-density and abstract cell decomposition for edge relation in graphs of bounded twin-width. *CoRR*, abs/2202.04006, 2022. arXiv:2202.04006.

22 André Schidler and Stefan Szeider. A SAT approach to twin-width. *CoRR*, abs/2110.06146, 2021. arXiv:2110.06146.

Improved Approximation Algorithms and Lower Bounds for Search-Diversification Problems

Amir Abboud ✉
Weizmann Institute of Science, Rehovot, Israel

Vincent Cohen-Addad ✉
Google Research, Zürich, Switzerland

Euiwoong Lee ✉
University of Michigan, Ann Arbor, MI, USA

Pasin Manurangsi ✉
Google Research, Mountain View, CA, USA

Abstract

We study several questions related to diversifying search results. We give improved approximation algorithms in each of the following problems, together with some lower bounds.

1. We give a polynomial-time approximation scheme (PTAS) for a diversified search ranking problem [9] whose objective is to minimizes the discounted cumulative gain. Our PTAS runs in time $n^{2^{O(\log(1/\epsilon)/\epsilon)}} \cdot m^{O(1)}$ where n denotes the number of elements in the databases and m denotes the number of constraints. Complementing this result, we show that no PTAS can run in time $f(\epsilon) \cdot (nm)^{2^{o(1/\epsilon)}}$ assuming Gap-ETH and therefore our running time is nearly tight. Both our upper and lower bounds answer open questions from [9].
2. We next consider the Max-Sum Dispersion problem, whose objective is to select k out of n elements from a database that maximizes the dispersion, which is defined as the sum of the pairwise distances under a given metric. We give a quasipolynomial-time approximation scheme (QPTAS) for the problem which runs in time $n^{O_\epsilon(\log n)}$. This improves upon previously known polynomial-time algorithms with approximate ratios 0.5 [35, 16]. Furthermore, we observe that reductions from previous work rule out approximation schemes that run in $n^{\tilde{o}_\epsilon(\log n)}$ time assuming ETH.
3. Finally, we consider a generalization of Max-Sum Dispersion called Max-Sum Diversification. In addition to the sum of pairwise distance, the objective also includes another function f. For monotone submodular function f, we give a quasipolynomial-time algorithm with approximation ratio arbitrarily close to $(1-1/e)$. This improves upon the best polynomial-time algorithm which has approximation ratio 0.5 [16]. Furthermore, the $(1-1/e)$ factor is also tight as achieving better-than-$(1-1/e)$ approximation is NP-hard [26].

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation Algorithms, Complexity, Data Mining, Diversification

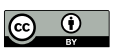
Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.7

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2203.01857` [2]

Funding *Amir Abboud*: Supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.
Euiwoong Lee: Partially supported by a gift from Google.

Acknowledgements We are grateful to Karthik C.S. for insightful discussions, and to Badih Ghazi for encouraging us to work on the problems.



1 Introduction

A fundamental task in databases in general and in search engines in particular is the selection and ordering of the results to a given query. Suppose that we have already retrieved the set of appropriate answers S_q to a query q by a certain preliminary process. Which item from the (possibly huge) set S_q should be presented *first*? Which should be the first ten?

Besides the obvious approach of ranking the *most relevant* answers first, perhaps the second most important consideration is that the output set should satisfy certain *diversity* requirements. If a user searches for "Barcelona" it would be desirable that the first ten results contain a mix of items containing, e.g. general details of the city, tourist information, and news about the associated soccer team, even though the most relevant items in certain absolute terms may only pertain to the latter. There are various natural ways to formalize what makes a set of results diverse, and much research has gone into this *Search Diversification* topic in the past two and a half decades in various context (see e.g. [19, 3, 33, 15, 9, 39, 50, 16, 24, 12, 31, 46, 34, 1, 36, 25, 52]). Recently, there have also been extensive research efforts into algorithmic fairness (see e.g. a survey [47]). Some of these fairness notions (e.g. [21, 7]) are also closely related to diversity: a set of results that is not diverse enough (e.g. returning only pictures of members of one group when a user searches for "scientists") could be problematic in terms of fairness.

A well-known work on search diversification [19] suggests that a diverse set of results is one that satisfies the following: The k^{th} result in the list should maximize the sum[1] of: (1) the relevance to the query, and (2) the total distance to the first $k-1$ results in the list. The success of this natural notion of diversification may be attributed to the fact that it can be computed efficiently with a greedy algorithm. However, it may be a bit too simplistic and the objectives that real-world search engines seem to optimize for are actually closer to other, more complicated (to compute) notions of diversity that have been proposed in follow-up works (e.g. [9, 33, 16]).

The goal of this paper is to investigate the time complexity of computing these latter, more intricate definitions of the search diversification task. Since such problems are NP-Hard even for restricted settings, and since approximate solutions are typically acceptable in this context, our focus is on understanding their time vs. approximation trade-offs. Our results reduce the gaps in the literature, completely resolving the complexity of some of the most natural notions.

1.1 Diversified Search Ranking

The first problem we study is a diversified search ranking problem formulated by Bansal et al. [9]. Here we are given a collections $\mathcal{S}$ of subsets of $[n]$ and, for each $S \in \mathcal{S}$, a positive integer k_S. Our goal is to find a permutation $\pi : [n] \to [n]$ that minimizes the *discounted cumulative gain (DCG)* defined as

$$\mathrm{DCG}_{\mathcal{S},\mathbf{k}}(\pi) := \sum_{S \in \mathcal{S}} \frac{1}{\log(t_\pi(S)+1)}, \tag{1}$$

where $t_\pi(S)$ is defined as the earliest time the set S is covered k_S times, i.e. $\min\{i \in [n] | S \cap \pi([i])| \geq k_S\}$.

[1] To be more precise, it is a weighted average of the two terms.

This formulation relates to diversification by viewing the output π as the ranking of the documents to be shown, and each topic corresponds to a set S of documents related to that topic. With this interpretation, the DCG favors rankings that display "diverse topics as early in the ranking as possible". Bansal et al. [9] gave a polynomial-time approximation scheme (PTAS) for the problem in the special case that $k_S = 1$ for all $S \in \mathcal{S}$ with running time $n^{2^{O(\log(1/\epsilon)/\epsilon)}} m^{O(1)}$. On the other hand, for the case of general k_S's, they give a quasipolynomial-time approximation scheme with running time $n^{(\log\log n)^{O(1/\epsilon)}} m^{O(1)}$ and left as an open question whether a PTAS exists. We resolve this open question by giving a PTAS for the more general problem; the running time we obtain for this more general problem is similar to the running time obtained by Bansal et al.'s PTAS for the special case $k_S = 1$. We then show that this is indeed the best possible (under some complexity assumption).

▶ **Theorem 1.** *There is a randomized PTAS for maximizing DCG that runs in time* $n^{2^{O(\log(1/\epsilon)/\epsilon)}} \cdot m^{O(1)}$.

The above running time is doubly exponential in $1/\epsilon$, and Bansal et al. [9] asked whether this dependency is necessary even for the special case $k_S = 1$. We also answer this question by showing that the doubly exponential is necessary, assuming the Gap Exponential Time Hypothesis (Gap-ETH)[2]:

▶ **Theorem 2.** *Assuming Gap-ETH, for any function g, there is no PTAS for maximizing DCG that runs in time $g(\epsilon) \cdot (nm)^{2^{o(1/\epsilon)}}$. Moreover, this holds even when restricted to instances with $k_S = 1$ for all $S \in \mathcal{S}$.*

1.2 Max-Sum Dispersion

The second problem we consider is the so-called *Max-Sum Dispersion* problem where we are given a metric space (U, d) where $|U| = n$ and an integer $p \geq 2$. The goal is to select $S \subseteq U$ of size p that maximizes

$$\mathrm{Disp}(S) := \sum_{\{u,v\} \subseteq S} d(u, v).$$

Roughly speaking, if the metric determines how different the items are, then our goal is to pick items that are "as diverse as possible" according to the Disp objective.

The Max-Sum Dispersion problem is a classic problem that has been studied since the 80s [45, 38, 49, 35, 16]. Previous works have given 0.5-approximation algorithm for the problem in polynomial time [35, 16]. We observe that the known NP-hardness reduction, together with newer hardness of approximation results for the Densest k-Subgraph problem with perfect completeness, yields strong lower bounds for the problem. (Details are deferred to the full version [2].) For example, if we assume the Strongish Planted Clique Hypothesis [43], then no $(0.5 + \epsilon)$-approximation algorithm is possible in $n^{o(\log n)}$ time. In other words, to achieve an improvement over the known approximation ratio, the algorithm must run in $n^{\Omega(\log n)}$ time. Complementing this, we provide a quasipolynomial-time approximation scheme that runs in time $n^{O_\epsilon(\log n)}$:

▶ **Theorem 3.** *There is a QPTAS for Max-Sum Dispersion that runs in time $n^{O(\log n/\epsilon^4)}$.*

[2] Gap-ETH [23, 42] asserts that there is no $2^{o(n)}$-time algorithm to distinguish between a satisfiable n-variable 3SAT formula and one which is not even $(1 - \epsilon)$-satisfiable for some $\epsilon > 0$

1.3 Max-Sum Diversification

Finally, we consider a generalization of Max-Sum Dispersion where, in addition to the metric space (U, d), we are now also given a monotone set function f (which we can access via a value oracle) and the goal is to select a set $S \subseteq U$ of size p that maximizes

$$\mathrm{Div}(S) := \mathrm{Disp}(S) + f(S).$$

This problem is referred to as *Max-Sum Diversification*.

The Max-Sum Diversification problem is more expressive than Max-Sum Dispersion. For example, the value $f(S)$ in the objective may be used to encode how relevant the selected set S is to the given query, in addition to the diversity objective expressed by $\mathrm{Disp}(S)$.

Borodin et al. [16] gave a 0.5-approximation algorithm for the problem when f is a monotone submodular function. Since Max-Sum Diversification is a generalization of Max-Sum Dispersion, our aforementioned lower bounds also imply that improving on this 0.5 factor requires at least $n^{\Omega(\log n)}$ time. Furthermore, submodular Max-Sum Diversification is also a generalization of maximizing monotone submodular function subject to a cardinality constraint. For this problem, an $(1 - 1/e)$-approximation algorithm is known and it is also known that achieving better than this ratio is NP-hard [26]. Therefore, it is impossible to achieve a better-than-$(1 - 1/e)$ approximation even in (randomized) quasi-polynomial time, assuming NP $\not\subseteq RTIME(n^{O(\log n)})$. Here we manage to provide such a tight quasi-polynomial time approximation algorithm:

▶ **Theorem 4.** *For any $\epsilon > 0$, there exists a randomized $n^{O(\log n/\epsilon^4)}$-time $(1 - 1/e - \epsilon)$-approximation algorithm for submodular Max-Sum Diversification.*

We remark that an interesting special case of submodular Max-Sum Diversification is when f is linear, i.e. $f(S) = \sum_{u \in S} f(u)$. In this case, Gollapudi and Sharma [33] provided an approximation-preserving reduction from the problem to the Max-Sum Dispersion. Therefore, our QPTAS for the latter (Theorem 3) also yields a QPTAS for this special case of Max-Sum Dispersion.

2 Preliminaries

For a natural number n, we use $[n]$ to denote $\{1, \ldots, n\}$. We say that a randomized algorithm for a maximization problem is an α-approximation if the expected objective of the output solution is at least α times the optimum; note that we can easily get a high-probability bound with approximation guarantee arbitrarily close to α by repeating the algorithm multiple times and pick the best solution.

2.1 Concentration Inequalities

For our randomized approximation algorithms, we will need some standard concentration inequalities. First, we will use the following version of Chernoff bound which gives a tail bound on the sum of i.i.d. random variables. (See e.g. [44] for a proof.)

▶ **Lemma 5** (Chernoff bound). *Let $X_1, \ldots, X_r \in [0, 1]$ be independent random variables, $S := X_1 + \cdots + X_r$ and $\mu := \mathbb{E}[S]$. Then, for any $\delta \in [0, 1]$, we have*

$$\Pr[|S - \mu| > \delta\mu] \leq 2 \exp\left(-\frac{\delta^2 \mu}{3}\right).$$

Furthermore, for any $\delta \geq 0$, we have

$$\Pr[S > (1+\delta)\mu] \leq \exp\left(-\frac{\delta^2\mu}{2+\delta}\right).$$

It will also be convenient to have a concentration of sums of random variables that are drawn without replacement from a given set. For this, we will use (a without-replacement version of) the Hoeffding's inequality, stated below. (See e.g. [10].)

▶ **Lemma 6** (Hoeffding's inequality). *Let $X_1, \ldots, X_r$ be random variables drawn without replacement from a multiset $\mathcal{X} \subseteq [0,1]$, $A := \frac{1}{r}(X_1 + \cdots + X_r)$ and $\mu := \mathbb{E}[A]$. Then, for any $\delta \in [0,1]$, we have*

$$\Pr[|A - \mu| > \delta] \leq 2\exp\left(-2\delta^2 r\right).$$

2.2 Densest k-Subgraph

For both our Max-Sum Dispersion and Max-Sum Diversification problems, we will use as a subroutine algorithms for (variants of) the *Densest k-Subgraph (DkS)* problem. In DkS, we are given a set V of nodes, weights $w : \binom{V}{2} \to [0,1]$ and an integer k, the goal is to find a subset $T \subseteq V$ with $|T| = k$ that maximizes $\mathrm{Den}(T) := \frac{1}{|T|(|T|-1)/2}\sum_{\{u,v\}\subseteq T} w(\{u,v\})$. An *additive QPTAS* is an algorithm running in quasipolynomial time for any fixed $\epsilon > 0$ such that its output T satisfies $\mathrm{Den}(T) \geq \mathrm{OPT} - \epsilon$; Barman [11] gave such an algorithm for DkS.

We will in fact use a slightly generalized version of the problem where a subset $I \subseteq V$ of vertices is given as an input and these vertices must be picked in the solution T (i.e. $I \subseteq T$). To avoid cumbersomeness, we also refer to this generalized version as DkS. It is not hard to see[3] that Barman's algorithm [11] extends easily to this setting:

▶ **Theorem 7.** *There is an additive QPTAS for DkS that runs in time $n^{O(\log n/\epsilon^2)}$.*

DkS is a classic problem in approximation algorithms literature, and many approximation algorithms [30, 51, 29, 28, 6, 32, 13, 11] and hardness results [27, 37, 48, 4, 14, 17, 40, 20] have been proved over the years. Most of these works focus on *multiplicative* approximation; the best known polynomial-time algorithm in this setting has an approximation ratio of $n^{1/4+\epsilon}$ for any constant $\epsilon > 0$ [13] and there are evidences that achieving subpolynomial ratio in polynomial time is unlikely [40, 14, 22]. As for *additive* approximation, it is known that an approximation scheme that runs in time $n^{\tilde{o}(\log n)}$ would break the exponential time hypothesis (ETH) [17]; therefore, the running time in Theorem 7 (in terms of n) is tight up to $\mathrm{poly}\log\log n$ factor in the exponent. We provide additional discussions on related results in the full version [2].

2.3 Submodular Maximization over a Matroid Constraint

For our approximation algorithm for Max-Sum Diversification, we will also need an approximation algorithm for *monotone submodular maximization under a matroid constraint*. In this problem, we are given a monotone submodular set function $f : 2^X \to \mathbb{R}_{\geq 0}$ over a ground set X together with a matroid $\mathcal{M} = (X, \mathcal{I})$. The function f is given via a value oracle and $\mathcal{M}$ can be accessed via a membership oracle (which answers questions of the form "does

[3] In fact, in Section 5.1, we also give a more general algorithm than the one stated in Theorem 7 which can also handle an additional monotone submodular function.

S belong to $\mathcal{I}$?"). The goal is to find $S \in \mathcal{I}$ that maximizes $f(S)$. Călinescu et al. gave a randomized algorithm with approximation ratio $(1 - 1/e)$ for the problem, which we will use in our algorithm.

▶ **Theorem 8** ([18]). *There exists a randomized polynomial-time $(1 - 1/e)$-approximation algorithm for maximizing a montone submodular function over a matroid constraint.*

3 Diversified Search Ranking

In this section, we consider the diversified search ranking question as proposed in [9] and prove our upper and lower bounds (Theorems 1 and 2).

3.1 Polynomial-time Approximation Scheme

We will start by presenting our PTAS. At a high-level, our PTAS is similar to that of Bansal et al.'s: our algorithm use bruteforce to try every possible values of $\pi(1), \ldots, \pi(\exp(\tilde{O}(1/\epsilon)))$. Once these are fixed, we solve the remaining problem using linear programming (LP). We use the same LP as Bansal et al., except with a slightly more refined rounding procedure, which allows us to achieve a better approximation guarantee.

The remainder of this section is organized as follows. In Section 3.1.1, we present our LP rounding algorithm and its guarantees. Then, we show how to use it to yield our PTAS in Section 3.1.2.

3.1.1 Improved LP Rounding

For convenience in the analysis below, let us also define a more generic objective function where $\frac{1}{\log(t_\pi(S))+1}$ in Equation (1) can be replaced by any non-increasing function $f : [n] \to (0, 1]$:

$$\mathrm{DCG}^f_{\mathcal{S},\mathbf{k}}(\pi) := \sum_{S \in \mathcal{S}} f(t_\pi(S)).$$

The main result of this subsection is the following polynomial time LP rounding algorithm for the above general version of DCG:

▶ **Lemma 9.** *There exists an absolute constant C such that for any $\alpha \in (0, 0.5)$ the following holds: there is a polynomial-time algorithm that computes a ranking with expected DCG at least $(1 - \alpha) \cdot \tau_{f,\alpha}$ times that of the optimum where*

$$\tau_{f,\alpha} := \min_{t \in [n]} \frac{f\left(\frac{C \log(1/\alpha)}{\alpha} \cdot \frac{t}{f(t)}\right)}{f(t)}.$$

Informally speaking, the term $\tau_{f,\alpha}$ somewhat determines "how fast f increases". In the next section, once we fix the first u elements of the ranking, f will become $f(t) := 1/\log(t+u)$ which is "slowly growing" when u is sufficiently large. This allows us to ensure that the guarantee in Lemma 9 yields an $(1 - O(\epsilon))$-approximation as desired.

3.1.1.1 LP Formulation

To prove Lemma 9, we use the same knapsack constraint-enhanced LP as in [9], stated below. Note that the number of knapsack constraints can be super-polynomial. However, it is known that such an LP can be solved in polynomial time; see e.g. [8, Section 3.1] for more detail.

$$\text{Maximize} \qquad \sum_{S \in \mathcal{S}} \sum_{t \in [n]} (y_{S,t} - y_{S,t-1}) \cdot f(t)$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{e\in[n]} x_{e,t} = 1 && \forall t \in [n] \\
& \sum_{t\in[n]} x_{e,t} = 1 && \forall e \in [n] \\
& \sum_{e\in S\subseteq A} \sum_{t'<t} x_{e,t'} \geq (k_S - |A|)\cdot y_{S,t} && \forall S \in \mathcal{S}, A \subseteq S, t \in [n] \\
& y_{S,t} \geq y_{S,t-1} && \forall S \in \mathcal{S}, t \in \{2,\dots,n\} \\
& x_{e,t}, y_{S,t} \in [0,1] && \forall e,t \in [n], S \in \mathcal{S}.
\end{aligned}$$

3.1.1.2 Rounding Algorithm

Let $\gamma \in (0, 0.1)$ be a parameter to be chosen later. Our rounding algorithm works as follows:

1. $\pi \leftarrow \emptyset$
2. For $i = 1, \dots, \lceil \log n \rceil$ do:
   a. Let $t_i = \min\{n, 2^i\}$.
   b. Let $z_{e,i} = \sum_{t\leq t_i} x^*_{e,t}$ and $p_{e,i} = \min\{1, \frac{z_{e,i}}{\gamma \cdot f(t_i)}\}$ for all $e \in [n]$.
   c. Let A_i be the set such that $e \in [n]$ is independently included w.p. $p_{e,i}$.

Finally, our permutation π is defined by adding elements from $A_1, \dots, A_{\lceil \log n \rceil}$ in that order, where the order within each A_i can be arbitrary and we do not add an element if it already appears in the permutation.

Once again, we remark that our algorithm closely follows that of [9], except that Bansal et al. simply chose their $p_{e,i}$ to be $\min\{1, O(\log^2 n) \cdot z_{e,i}\}$, whereas our $p_{e,i}$ is a more delicate $\min\{1, \frac{z_{e,i}}{\gamma \cdot f(t_i)}\}$. This allows our analysis below to produce a better approximation ratio.

3.1.1.3 Analysis

We will now proceed to analyze our proposed randomized rounding procedure. Let $\eta \in (0, 0.1)$ be a parameter to be chosen later, and let $(\mathbf{x}^*, \mathbf{y}^*)$ denote an optimal solution to the LP. For each S, let $t^*(S)$ be the largest positive integer t^* such that

$$y^*_{S,t^*-1} \leq \eta \cdot f(t^*). \tag{2}$$

We start with the following lemma, which is a refinement of [9, Lemma 1].

▶ **Lemma 10.** $\text{OPT} \leq (1+\eta) \cdot \sum_{S\in\mathcal{S}} f(t^*(S))$.

Proof. We have

$$\begin{aligned}
\text{OPT} &\leq \sum_{S\in\mathcal{S}} \sum_{t\in[n]} (y^*_{S,t} - y^*_{S,t-1}) \cdot f(t) \\
&= \sum_{S\in\mathcal{S}} \left(\sum_{t=1}^{t^*(S)-1} (y^*_{S,t} - y^*_{S,t-1}) \cdot f(t) + \sum_{t=t^*(S)}^{n} (y^*_{S,t} - y^*_{S,t-1}) \cdot f(t) \right) \\
&\leq \sum_{S\in\mathcal{S}} \left(\sum_{t=1}^{t^*(S)-1} (y^*_{S,t} - y^*_{S,t-1}) + \sum_{t=t^*(S)}^{n} (y^*_{S,t} - y^*_{S,t-1}) \cdot f(t^*(S)) \right) \\
&\leq \sum_{S\in\mathcal{S}} \left(y^*_{S,t^*(S)-1} + f(t^*(S)) \right) \\
&\overset{(2)}{\leq} \sum_{S\in\mathcal{S}} (1+\eta) \cdot f(t^*(S)).
\end{aligned}$$

◀

Next, we show via standard concentration inequalities that $|A_i|$'s has small sizes with a large probability.

▶ **Lemma 11.** *With probability* $1 - 2\exp\left(-\frac{1}{3\gamma}\right)$*, we have* $|A_i| \leq \frac{2t_i}{\gamma f(t^*)}$ *for all* $i \in [\lceil \log n \rceil]$.

Proof. Notice that $\sum_{e\in[n]} p_{e,i} \leq \frac{\sum_{e\in[n]} z_{e,i}}{\gamma f(t_i)} = \frac{t_i}{\gamma f(t_i)}$. As a result, by Chernoff bound (Lemma 5), we have

$$\Pr\left[|A_i| > \frac{2t_i}{\gamma f(t^*)}\right] \leq \exp\left(-\frac{t_i}{3\gamma f(t^*)}\right) \leq \exp\left(-\frac{t_i}{3\gamma}\right).$$

By union bound, we thus have $|A_i| \leq \frac{2t_i}{\gamma f(t^*)}$ for all $i \in [\lceil \log n \rceil]$ with probability at least

$$1 - \sum_{i\in[\lceil \log n \rceil]} \exp\left(-\frac{t_i}{3\gamma}\right) \leq 1 - 2\exp\left(-\frac{1}{3\gamma}\right). \qquad \blacktriangleleft$$

Let $i^*(S)$ denote the smallest i such that $t_i \geq t^*(S)$. We now bound the probability that S is covered (k_S times) by the end of the $i^*(S)$-th iteration of the algorithm. Our bound is stated below. We note that our bound here is not with high probability, unlike that of the analysis of [9] which yields a bound of $1 - o(1/n)$. We observe here that such a strong bound is not necessary for the analysis because we are working with a maximization problem and therefore such a high probability bound is not necessary to get a bound on the expectation of the DCG.

▶ **Lemma 12.** *Assume that* $\eta \geq 2\gamma$*. For each* $S \in \mathcal{S}$*, we have* $t_\pi(S) \leq |A_1| + \cdots + |A_{i^*(S)}|$ *with probability* $1 - \exp\left(\frac{\eta}{8\gamma}\right)$.

Proof. It suffices to show that at least k_S elements of S are selected in $A_{i^*(S)}$. Let S_g denote the set of elements $e \in S$ for which $p_{e,i^*(S)} = 1$. If $|S_g| \geq k_S$, then we are done. Otherwise, from knapsack constraint, we have

$$\sum_{e\in S\setminus S_g} z_{e,i^*(S)} \geq (k_S - |S_g|) y^*_{S,t_{i^*(S)}} \geq (k_S - |S_g|) y^*_{S,t^*(S)} \geq \eta \cdot f(t^*(S)) \cdot (k_S - |S_g|)$$
$$\geq \eta \cdot f(t_{i^*(S)}) \cdot (k_S - |S_g|),$$

where the third inequality follows from our choice of $t^*(S)$. This implies that

$$\sum_{e\in S\setminus S_g} p_{e,i^*(S)} \geq \eta/\gamma \cdot (k_S - |S_g|).$$

Recall that $\eta/\gamma \geq 2$. This means that the probability that at least k_S elements of S are selected in $A_{i^*(S)}$ is at least

$$1 - \Pr[|(S \setminus S_g) \cap A_{i^*(S)}| \leq 0.5\eta/\gamma \cdot (k_S - |S_g|)]$$
$$\leq 1 - \exp\left(-\frac{1}{8} \cdot \eta/\gamma \cdot (k_S - |S_g|)\right)$$
$$\leq 1 - \exp\left(-\frac{\eta}{8\gamma}\right),$$

where the first inequality follows from the Chernoff bound. ◀

Applying the union bound to the two previous lemmas, we immediately arrive at the following:

▶ **Lemma 13.** *Assume that $\eta \geq 2\gamma$. For all $S \in \mathcal{S}$, we have*

$$\mathbb{E}_\pi[f(t_\pi(S))] \geq \left(1 - 2\exp\left(-\frac{1}{3\gamma}\right) - \exp\left(\frac{\eta}{8\gamma}\right)\right) \cdot f\left(\frac{8t^*(S)}{\gamma f(t^*(S))}\right)$$

Finally, combining Lemmas 10 and 13 and selecting $\eta = 2\alpha, \gamma = O(\eta/\log(1/\eta))$ yields Lemma 9.

3.1.2 From LP Rounding to PTAS

As stated earlier, we may now use bruteforce to try all possible values of the first few elements in the ranking and then use our LP rounding to arrive at the PTAS:

Proof of Theorem 1. For any $\epsilon < 0.1$, we use bruteforce for the first $u = (4C/\epsilon)^{100/\epsilon}$ elements and then use Lemma 9 on the remaining instance but with $f(t) := \frac{1}{\log(t+u)}$. The expected approximation ratio we have is at least

$$\begin{aligned}
&(1-0.5\epsilon) \cdot \tau_{f,0.5\epsilon} \\
&\geq (1-0.5\epsilon) \cdot \min_{t\in[n]} f\left(\frac{4C\log(1/\epsilon)}{\epsilon} \cdot \frac{t}{f(t)}\right)/f(t) \\
&= (1-0.5\epsilon) \cdot \min_{t\in[n]} \frac{\log(t+u)}{\log\left(\frac{4C\log(1/\epsilon)}{\epsilon} \cdot \frac{t}{f(t)} + u\right)} \\
&\geq (1-0.5\epsilon) \cdot \min_{t\in[n]} \frac{\log(t+u)}{\log\left(\frac{4C\log(1/\epsilon)}{\epsilon} \cdot (t+u)\log(t+u)\right)} \\
&= (1-0.5\epsilon) \cdot \min_{t\in[n]} \frac{1}{1 + \frac{\log\left(\frac{4C\log(1/\epsilon)}{\epsilon}\right)}{\log(t+u)} + \frac{\log\log(t+u)}{\log(t+u)}} \\
&= (1-0.5\epsilon) \cdot \frac{1}{1 + \frac{\log\left(\frac{4C\log(1/\epsilon)}{\epsilon}\right)}{\log(u)} + \frac{\log\log(u)}{\log(u)}} \\
&\geq (1-0.5\epsilon) \cdot \frac{1}{1+0.1\epsilon+0.1\epsilon} \\
&\geq 1-\epsilon,
\end{aligned}$$

as desired. ◀

3.2 Running Time Lower Bound

To prove our running time lower bound, we will reduce from the *Maximum k-Coverage* problem. Recall that in Maximum k-Coverage, we are given a set $\mathcal{T} \subseteq [M]$ and an integer k; the goal is to find $T_1^*, \cdots T_k^* \in \mathcal{T}$ that maximizes $|T_1^* \cup \cdots \cup T_k^*|$. We write $\mathrm{Cov}(\mathcal{T}, k)$ to denote this optimum. Furthermore, we say that a Maximum k-Coverage is *regular* if $|T| = M/k$ for all $T \in \mathcal{T}$. Finally, we use N to denote $|\mathcal{T}| \cdot M$ which upper bound the "size" of the problem.

Manurangsi [41] showed the following lower bound for this problem:

▶ **Theorem 14** ([41]). *Assuming the Gap Exponential Time Hypothesis (Gap-ETH), for any constant $\delta > 0$, there is no $N^{o(k)}$-time algorithm that can, given a regular instance $(\mathcal{T}, k)$ distinguish between the following two cases:*

- *(YES)* $\mathrm{Cov}(\mathcal{T}, k) \geq M$.
- *(NO)* $\mathrm{Cov}(\mathcal{T}, k) \leq (1 - 1/e + \delta)M$.

Proof of Theorem 2. Fix $\delta = 0.1$. We reduce from the Maximum k-Coverage problem. Suppose that $(\mathcal{T}, k)$ is a regular Maximum k-Coverage instance; we assume w.l.o.g. that k is divisible by 10.

We construct the instance $(\mathcal{S}, \{k_S\}_{S\in\mathcal{S}})$ of the DCG maximization as follows:

- Let $n = |\mathcal{T}|$ where we associate each $j \in [n]$ with $T_j \in \mathcal{T}$.
- Let $\mathcal{S} = \{S_1, \ldots, S_M\}$ where $S_i = \{j \in [n] \mid i \in T_j\}$.
- Let $k_S = 1$ for all $S \in \mathcal{S}$.

In the YES case, let $T_{j_1}, \ldots, T_{j_k}$ be such that $|T_{j_1} \cup \cdots \cup T_{j_k}| = M$. Let $\pi^* : [n] \to [n]$ be any permutation such that $\pi^*(\ell) = j_\ell$ for all $\ell \in [k]$. From regularity of $(\mathcal{T}, k)$, there are exactly $q := M/k$ sets $S \in \mathcal{S}$ such that $t_{\pi^*}(S) = i$. Therefore, we have

$$\mathrm{DCG}_{\mathcal{S},\mathbf{k}}(\pi^*) = \sum_{i\in[k]} \frac{M}{k} \cdot \frac{1}{\log(i+1)}.$$

Let OPT^* denote the RHS quantity. Notice that

$$\mathrm{OPT}^* \le \frac{M}{\log(k+1)}. \tag{3}$$

In the NO case, consider any permutation $\pi : [n] \to [n]$. Let t_i denote the i-th smallest value in the multiset $\{t_\pi(S)\}_{S\in\mathcal{S}}$. Regularity of $(\mathcal{T}, k)$ implies that

$$t_i \ge t_{i-q} + 1 \tag{4}$$

for all $i > q$. This in turn implies that

$$t_i \ge \lceil i/q \rceil . \tag{5}$$

Furthermore, $\mathrm{Cov}(\mathcal{T}, k) \le (1 - 1/e - \delta)M \le 0.8M$ implies that

$$t_{0.8M} > k.$$

Furthermore, applying (4) to the above, we have

$$t_{0.9M} \ge t_{0.8M} + \left\lfloor \frac{0.1M}{q} \right\rfloor = k + 0.1k = 1.1k. \tag{6}$$

With the above notion, we may write $\mathrm{DCG}_{\mathcal{S},\mathbf{k}}(\pi) - \mathrm{OPT}^*$ as

$$\begin{aligned}
\mathrm{DCG}_{\mathcal{S},\mathbf{k}}(\pi) - \mathrm{OPT}^* &= \sum_{i=1}^{M} \frac{1}{\log(t_i+1)} - \sum_{i=1}^{M} \frac{1}{\log(\lceil i/q \rceil + 1)} \\
&\overset{(5)}{\ge} \sum_{i=0.9M}^{M} \left(\frac{1}{\log(t_i+1)} - \frac{1}{\log(\lceil i/q \rceil + 1)} \right) \\
&\overset{(6)}{\ge} \sum_{i=0.9M}^{M} \left(\frac{1}{\log(1.1k+1)} - \frac{1}{\log(\lceil i/q \rceil + 1)} \right) \\
&\ge \sum_{i=0.9M}^{M} \left(\frac{1}{\log(1.1k+1)} - \frac{1}{\log(k+1)} \right) \\
&= 0.1M \cdot \left(\frac{1}{\log(1.1k+1)} - \frac{1}{\log(k+1)} \right) \\
&= \Theta\left(\frac{M}{\log^2 k} \right).
\end{aligned}$$

Finally, observe also that

$$\mathrm{OPT}^* = \frac{M}{k} \cdot \sum_{i \in [k]} \frac{1}{\log(i+1)} = \frac{M}{k} \Theta\left(\frac{k}{\log k}\right) = \Theta\left(\frac{M}{\log k}\right).$$

Combining the above two inequalities, we have

$$\mathrm{DCG}_{\mathcal{S},\mathbf{k}}(\pi) \geq \left(1 + \Theta\left(\frac{1}{\log k}\right)\right) \cdot \mathrm{OPT}^*.$$

Now, suppose that there is a PTAS for maximizing DCG that runs in time $f(\epsilon) \cdot (nm)^{2^{o(1/\epsilon)}}$. If we run the algorithm with $\epsilon = \gamma / \log k$ where $\gamma > 0$ is sufficiently small constant, then we can distinguish between the YES case and the NO case in time $f(1/\log k) \cdot (nm)^{2^{o(\log k)}} \leq f(1/\log k) \cdot (nm)^{o(k)} = g(k) \cdot N^{o(k)}$ which, from Theorem 14, violates Gap-ETH. ◀

4 Max-Sum Dispersion

In this section, we provide a QPTAS for Max-Sum Dispersion (Theorem 3).

As alluded to earlier, our algorithm will reduce to the Densest k-Subgraph (DkS) problem, for which an *additive* QPTAS is known [11]. Notice here that DkS is a generalization of the Max-Dispersion problem because we may simply set $V = U, k = p$ and $w(\{u, v\}) = d(u, v)/D$ where $D := \max_{u,v} d(u, v)$ denote the diameter of the metric space. Note however that we cannot apply Theorem 7 yet because the QPTAS in that theorem offers an *additive* guarantee. E.g. if the optimum is $o(1)$, then the QPTAS will not yield anything at all unless we set $\epsilon = o(1)$, which then gives a running time $n^{\omega(\log n)}$. This example can happen when e.g. there is a single pair u, v that are very far away and then all the other pairs are close to u.

Our main technical contribution is to give a simple structural lemma that allows us to avoid such a scenario. Essentially speaking, it allows us to pick a vertex and selects all vertices that are "too far away" from it. Once this is done, the remaining instance can be reduced to DkS without encountering the "small optimum" issue described in the previous paragraph.

4.1 A Structural Lemma

Henceforth, we write $\mathrm{Disp}(S, T)$ to denote $\sum_{u \in S, v \in T} d(u, v)$ and $\mathrm{Disp}(u, T)$ as a shorthand for $\mathrm{Disp}(\{u\}, T)$. Furthermore, we use $\mathcal{B}(u, D)$ to denote $\{z \in U \mid d(z, u) \leq D\}$ and let $\overline{\mathcal{B}(u, D)} := U \setminus \mathcal{B}(u, D)$.

We now formalize our structural lemma. It gives a lower bound on the objective based on a vertex in the optimal solution and another vertex *not* in the optimal solution. Later on, by guessing these two vertices, we can reduce to DkS while avoiding the "small optimum" issue.

▶ **Lemma 15.** *Let S^{OPT} be any optimal solution of Max-Sum Dispersion and let $u^{\min}$ be the vertex in S^{OPT} that minimizes $\mathrm{Disp}(u^{\min}, S^{\mathrm{OPT}})$. Furthermore, let v be any vertex* not *in S^{OPT} and let $\Delta = d(u^{\min}, v)$. Then, we have*

$$\mathrm{Disp}(S^{\mathrm{OPT}}) \geq \frac{p(p-1)\Delta}{16}.$$

Proof of Lemma 15. Let $S^{\mathrm{OPT}}_{\mathrm{close}} := S^{\mathrm{OPT}} \cap \mathcal{B}(u^{\min}, 0.5\Delta)$. Consider two cases, based on the size of $S^{\mathrm{OPT}}_{\mathrm{close}}$:

- Case I: $|S^{\mathrm{OPT}}_{\mathrm{close}}| \leq p/2$. In this case, we have

$$\mathrm{Disp}(u^{\min}, S^{\mathrm{OPT}}) \geq \mathrm{Disp}(u^{\min}, S^{\mathrm{OPT}} \setminus S^{\mathrm{OPT}}_{\mathrm{close}}) \geq (p/2)(\Delta/2) = \Delta p/4.$$

  Furthermore, by our definition of $u^{\min}$, we have

$$\mathrm{Disp}(S^{\mathrm{OPT}}) = \frac{1}{2}\sum_{u \in S} \mathrm{Disp}(u, S^{\mathrm{OPT}}) \geq \frac{p}{2}\,\mathrm{Disp}(u^{\min}, S^{\mathrm{OPT}}).$$

  Combining the two inequalities, we have $\mathrm{Disp}(S^{\mathrm{OPT}}) \geq p^2\Delta/8$.
- Case II: $|S^{\mathrm{OPT}}_{\mathrm{close}}| > p/2$. In this case, since S^{OPT} is an optimal solution, replacing any $z \in S^{\mathrm{OPT}}_{\mathrm{close}}$ with v must not increase the solution value, i.e.

$$\begin{aligned}\mathrm{Disp}(z, S^{\mathrm{OPT}}) &\geq \mathrm{Disp}(v, S^{\mathrm{OPT}} \setminus \{z\}) \\ &\geq \mathrm{Disp}(v, S^{\mathrm{OPT}}_{\mathrm{close}} \setminus \{z\}) \\ &\geq ((p-1)/2)(0.5\Delta),\end{aligned}$$

  where the second inequality uses the fact that for any $z' \in S^{\mathrm{OPT}}_{\mathrm{close}}$ we have $d(v, z') \geq d(u, v) - d(u, z') \geq \Delta - 0.5\Delta$. From this, we once again have

$$\begin{aligned}\mathrm{Disp}(S^{\mathrm{OPT}}) = \frac{1}{2}\sum_{u \in S} \mathrm{Disp}(u, S^{\mathrm{OPT}}) \geq \frac{1}{2}\sum_{z \in S^{\mathrm{OPT}}_{\mathrm{close}}} \mathrm{Disp}(z, S^{\mathrm{OPT}}) &\geq |S^{\mathrm{OPT}}_{\mathrm{close}}| \cdot \frac{(p-1)\Delta}{8} \\ &> \frac{p(p-1)}{16\Delta},\end{aligned}$$

  where the last inequality follows from our assumption of this case. ◀

4.2 QPTAS for Max-Sum Dispersion

We now present our QPTAS, which simply guesses $u^{\min}$ and $v = \mathrm{argmax}_{z \notin S^{\mathrm{OPT}}} d(z, u)$ and then reduces the problem to DkS. By definition of v, if we let $\Delta = d(u, v)$, every point outside $\mathcal{B}(u^{\min}, \Delta)$ must be in S^{OPT}. The actual reduction to DkS is slightly more complicated than that described at the beginning of this section. Specifically, among points $\overline{\mathcal{B}(u^{\min}, \Delta)}$ that surely belong to S^{OPT}, we ignore all points outside $\mathcal{B}(u^{\min}, 20\Delta/\epsilon)$ (i.e., they do not appear in the DkS instance) and we let $\mathcal{B}(u^{\min}, 20\Delta/\epsilon) \setminus \mathcal{B}(u^{\min}, \Delta)$ be the "must pick" part. Ignoring the former can be done because the contribution to the objective from those points can be approximated to within $(1 \pm O(\epsilon))$ regardless of the points picked in the ball $\mathcal{B}(u^{\min}, \Delta)$. This is not true for the latter, which means that we need to include them in our DkS instance.

Proof of Theorem 3. Our algorithm works as follows:

1. For every distinct $u, v \in U$ do:
   a. Let $\Delta := d(u, v)$ and $\Delta^* = 20\Delta/\epsilon$.
   b. If $|\overline{\mathcal{B}(u, \Delta)}| \geq p$, then skip the following steps and continue to the next pair u, v.
   c. Otherwise, create a DkS instance where $V := \mathcal{B}(u, \Delta^*), I := V \setminus \mathcal{B}(u, \Delta)$, $k = p - |\overline{\mathcal{B}(u, \Delta^*)}|$ and w is defined as $w(\{y, z\}) := 0.5d(y, z)/\Delta^*$ for all $y, z \in V$.
   d. Use the additive QPTAS from Theorem 7 to solve the above instance to within an additive error of $\epsilon' := 0.00005\epsilon^2$. Let T be the solution found.
   e. Finally, let $S^{u,v} := T \cup \overline{\mathcal{B}(u, \Delta^*)}$.
2. Output the best solution among $S^{u,v}$ considered.

It is obvious that the running time is dominated by the running time of the QPTAS which takes $n^{O(\log n/(\epsilon')^2)} = n^{O(\log n/\epsilon^4)}$ as desired.

Next, we show that the algorithm indeed yields a $(1-\epsilon)$-approximation. To do this, let us consider $S^{\mathrm{OPT}}, u^{\min}$ as defined in Lemma 15, and let $u = u^{\min}, v := \operatorname{argmax}_{z \notin S^{\mathrm{OPT}}} d(u,z)$. Let T be the solution found by the DkS algorithm for this u, v and let $T' := T \setminus I$. We have

$$\begin{aligned}
&\mathrm{Disp}(S^{u,v}) \\
&= \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}) + \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}, T) + \mathrm{Disp}(T) \\
&= \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}) + \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}, I) + \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}, T') + \mathrm{Disp}(T). \qquad (7)
\end{aligned}$$

Similarly, letting $S := S^{\mathrm{OPT}} \cap \mathcal{B}(u,\Delta^*)$ and $S' := S^{\mathrm{OPT}} \setminus I$, we have

$$\begin{aligned}
&\mathrm{Disp}(S^{\mathrm{OPT}}) \\
&= \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}) + \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}, I) + \mathrm{Disp}(\overline{\mathcal{B}(u,\Delta^*)}, S') + \mathrm{Disp}(S). \qquad (8)
\end{aligned}$$

Now, observe from the definition of the DkS instance (for this u, v) that for any J such that $I \subseteq J \subseteq V$, we have

$$\mathrm{Den}(J) = \frac{1}{k(k-1)/2} \cdot \frac{0.5}{\Delta^*} \mathrm{Disp}(J).$$

The additive approximation guarantee from Theorem 7 implies that $\mathrm{Den}(T) \geq \mathrm{Den}(S) - \epsilon'$. Using the above equality, we can rewrite this guarantee as

$$\mathrm{Disp}(S) - \mathrm{Disp}(T) \leq \epsilon' \cdot \Delta^* \cdot k(k-1). \qquad (9)$$

Taking the difference between Equation (8) and Equation (7) and applying Equation (9), we have

$$\begin{aligned}
\mathrm{Disp}(S^{\mathrm{OPT}}) - \mathrm{Disp}(S^{u,v}) &\leq \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, S') - \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, T') + \epsilon' \cdot \Delta^* \cdot k(k-1). \\
(\text{Our choice of } \epsilon') &\leq \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, S') - \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, T') + 0.001\epsilon\Delta \cdot p(p-1) \\
(\text{Lemma 15}) &\leq \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, S') - \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, T') + 0.1\epsilon\,\mathrm{Disp}(S^{\mathrm{OPT}}).
\end{aligned}$$

Now, since $|S'| = |T'| \leq p$ and $S', T' \subseteq \mathcal{B}(z,\Delta)$, we have

$$\begin{aligned}
\mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, S') - \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, T') &\leq |\overline{\mathcal{B}(z,\Delta^*)}| \cdot |S'| \cdot ((\Delta^* + \Delta) - (\Delta^* - \Delta)) \\
&\leq 2|\overline{\mathcal{B}(z,\Delta^*)}| \cdot |S'| \cdot \Delta \\
(\text{Our choice of } \Delta^*) &\leq 0.1\epsilon \cdot |\overline{\mathcal{B}(z,\Delta^*)}| \cdot |S'| \cdot (\Delta^* - \Delta) \\
&\leq 0.1\epsilon \cdot \mathrm{Disp}(\overline{\mathcal{B}(z,\Delta^*)}, S') \\
&\leq 0.1\epsilon \cdot \mathrm{Disp}(S^{\mathrm{OPT}}).
\end{aligned}$$

Combining the above two inequalities, we get $\mathrm{Disp}(S^{u,v}) \geq (1 - 0.2\epsilon) \cdot \mathrm{Disp}(S^{\mathrm{OPT}})$, as desired. ◀

5 Max-Sum Diversification

In this section, we give our quasipolynomial-time approximation algorithm for the Max-Sum Diversification with approximation ratio arbitrarily close to $(1 - 1/e)$ (Theorem 4). In fact, we prove a slightly stronger version of the theorem where the approximation ratio for the dispersion part is arbitrarily close to 1 and that of the submodular part is arbitrarily close to $1 - 1/e$. This is stated more precisely below; note that this obviously implies Theorem 4.

▶ **Theorem 16.** *Let S^{OPT} be any optimal solution of Max-Sum Diversification. There exists a randomized $n^{O(\log n/\epsilon^4)}$-time algorithm that finds a p-size set S such that*

$$\mathbb{E}[\mathrm{Div}(S)] \geq (1-\epsilon)\,\mathrm{Disp}(S^{\mathrm{OPT}}) + (1-1/e-\epsilon)f(S^{\mathrm{OPT}}).$$

At a high-level, our algorithm for Max-Sum Diversification is very similar to that of Max-Sum Dispersion presented in the previous section. Specifically, we use a structural lemma (akin to Lemma 15) to reduce our problem to a variant of DkS. This variant of DkS additionally has a submodular function attached to it. Using techniques from DkS approximation literature, we give an algorithm for this problem by in turn reducing it to the submodular maximization problem over a partition matroid, for which we can appeal to Theorem 8.

5.1 Approximating Densest Subgraph and Submodular Function

We will start by giving an algorithm for the aforementioned extension of the DkS problem, which we call *Submodular DkS*:

▶ **Definition 17** (Submodular DkS)**.** *Given (V, I, w, k) (similar to DkS) together with a monotone submodular set function h on the ground set V (accessible via a value oracle), the goal is to find a size-k subset T where $I \subseteq T \subseteq V$ that maximizes $h(T) + \mathrm{Den}(T)$.*

We give a quasipolynomial-time algorithm with an approximation guarantee similar to QPTAS for the original DkS (i.e. Theorem 7) while also achiving arbritrarily close to $(1-1/e)$ approximation ratio for the submodular part of the objective:

▶ **Theorem 18.** *For any set T^{OPT} of size k such that $I \subseteq T^{\mathrm{OPT}} \subseteq V$, there is an $n^{O(\log n/\gamma^2)}$-time algorithm that output a size-k T such that $I \subseteq T \subseteq V$ and*

$$\mathbb{E}[h(T) + \mathrm{Den}(T)] \geq (1-1/e-\gamma)\,h(T^{\mathrm{OPT}}) + \mathrm{Den}(T^{\mathrm{OPT}}) - \gamma. \tag{10}$$

In order to facilitate the subsequent discussion and proof, it is useful to define additional notations. (Throughout, we view vectors as column vectors.)

- Let $\mathbf{W} \in \mathbb{R}^{V\times V}$ denote the vector where $\mathbf{W}_{u,v} = w(\{u,v\})$ for $u \neq v$ and $\mathbf{W}_{u,u} = 0$.
- For every $U \subseteq V$, let $\mathbf{1}(U) \in \mathbb{R}^V$ denote the indicator vector of U, i.e.

$$\mathbf{1}(U)_v = \begin{cases} 1 & \text{if } v \in U, \\ 0 & \text{otherwise.} \end{cases}$$

- For every $U \subseteq V$, let $\mathbf{w}(U) = \mathbf{W} \cdot \mathbf{1}(U) \in \mathbb{R}^V$.
- Finally, for every non-empty $U \subseteq V$, let $\overline{\mathbf{w}}(U) := \frac{1}{|U|} \cdot w(U)$ and $\overline{\mathbf{1}}(U) := \frac{1}{|U|} \cdot \mathbf{1}(U)$.

To understand our reduction, we must first describe the main ideas behind the QPTAS of [11]. (Some of these ideas also present in previous works, e.g. [5].) Let us assume for simplicity of presentation that $I = \emptyset$. Observe that DkS is, up to an appropriate scaling, equivalent to find a size-k subset T that maximizes $\overline{\mathbf{1}}(T)^T \cdot \mathbf{W} \cdot \overline{\mathbf{1}}(T) = \overline{\mathbf{1}}(T)^T \overline{\mathbf{w}}(T)$. The main observation is that, if we randomly pick a subset $U \subseteq T^{\mathrm{OPT}}$ of size $\Theta_\gamma(\log n)$, then with high probability $\|\overline{\mathbf{w}}(U) - \overline{\mathbf{w}}(T^{\mathrm{OPT}})\|_\infty \leq O(\gamma)$ and $|\overline{\mathbf{1}}(T^{\mathrm{OPT}})^T \overline{\mathbf{w}}(T^{\mathrm{OPT}}) - \overline{\mathbf{1}}(U)^T \overline{\mathbf{w}}(U)| < O(\gamma)$. Roughly speaking, [11] exploits this by "guessing" such a set U and then solves for T such that $\|\overline{\mathbf{w}}(U) - \overline{\mathbf{w}}(T)\|_\infty \leq O(\gamma)$ and $|\overline{\mathbf{1}}(T)^T \overline{\mathbf{w}}(U) - \overline{\mathbf{1}}(U)^T \overline{\mathbf{w}}(U)| < O(\gamma)$; note that (the fractional version of) this is a linear program and can be solved efficiently. [11] then shows that a fractional solution to such a linear program can be rounded to an actual size-k set without any loss in the objective function.

We further push this idea by noting that, if we randomly partition V into $V_1, \ldots, V_s$ part where $s = O_\gamma(k/\log n)$, then the intersections $U_i^{\mathrm{OPT}} := V_i \cap T^{\mathrm{OPT}}$ satisfy the two conditions from the previous paragraphs (for $T = U_i^{\mathrm{OPT}}$). Therefore, we may enumerate all sets $U_i \subseteq V_i$ of roughly expected size to construct a collection $\mathcal{P}_i$ of subsets that satisfies these two conditions. Our goal now become picking $U_1 \in \mathcal{P}_1, \ldots, U_s \in \mathcal{P}_s$ that maximizes $h(U_1 \cup \cdots \cup U_s)$. This is simply monotone submodular maximization subject to a partition matroid constraint and therefore we may appeal to Theorem 8. We remark here that the two conditions that all subsets in $\mathcal{P}_i$ satisfy already ensure that the DkS objective is close to optimum. The full proof of Theorem 18 is deferred to the full version [2].

5.2 From Submodular DkS to Max-Sum Diversification

Having provided an approximation algorithm for Submodular DkS, we can use it to approximate Max-Sum Diversification via a similar approach to the reduction from Max-Sum Dispersion to DkS in the previous section. In particular, we can prove a structural lemma for Max-Sum Diversification that is analogous to Lemma 15 for Max-Sum Dispersion. We can then use the reduction nearly identical to the one in the proof of Theorem 3 to arrive at Theorem 16. The full details are deferred to the full version [2].

6 Conclusion

In this work, we consider three problems related to diversification: DCG in diversified search ranking, Max-Sum Dispersion and Max-Sum Diversification. For DCG, we give a PTAS and prove a nearly matching running time lower bound. For Max-Sum Dispersion, we give a QPTAS and similarly provide evidence for nearly matching running time lower bounds. Finally, we give a quasi-polynomial time algorithm for Max-Sum Diversification that achieves an approximation ratio arbitrarily close to $(1 - 1/e)$, which is also tight given the $(1 - 1/e + o(1))$ factor NP-hardness of approximating Maximum k-Coverage [26]. Our algorithms for DCG and Max-Sum Diversification are randomized and it remains an interesting open question whether there are deterministic algorithms with similar running times and approximation ratios.

References

1 Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. Diversity maximization under matroid constraints. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 32–40. ACM, 2013. doi:10.1145/2487575.2487636.

2 Amir Abboud, Vincent Cohen-Addad, Euiwoong Lee, and Pasin Manurangsi. Improved approximation algorithms and lower bounds for search-diversification problems. *CoRR*, abs/2203.01857, 2022. doi:10.48550/arXiv.2203.01857.

3 Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In Ricardo Baeza-Yates, Paolo Boldi, Berthier A. Ribeiro-Neto, and Berkant Barla Cambazoglu, editors, *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 5–14. ACM, 2009. doi:10.1145/1498759.1498766.

4 Noga Alon, Sanjeev Arora, Rajsekar Manokaran, Dana Moshkovitz, and Omri Weinstein. Inapproximability of densest κ-subgraph from average case hardness, 2011.

5 Noga Alon, Troy Lee, Adi Shraibman, and Santosh S. Vempala. The approximate rank of a matrix and its algorithmic applications: approximate rank. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 675–684. ACM, 2013. doi:10.1145/2488608.2488694.

6 Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1–3):15–26, 2002. doi:10.1016/S0166-218X(01)00243-8.

7 Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 405–413. PMLR, 2019. URL: http://proceedings.mlr.press/v97/backurs19a.html.

8 Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA*, pages 1539–1545, 2010. doi:10.1137/1.9781611973075.125.

9 Nikhil Bansal, Kamal Jain, Anna Kazeykina, and Joseph Naor. Approximation algorithms for diversified search ranking. In *ICALP*, pages 273–284, 2010. doi:10.1007/978-3-642-14162-1_23.

10 Rémi Bardenet and Odalric-Ambrym Maillard. Concentration inequalities for sampling without replacement. *Bernoulli*, 21(3):1361–1385, 2015.

11 Siddharth Barman. Approximating nash equilibria and dense subgraphs via an approximate version of carathéodory's theorem. *SIAM J. Comput.*, 47(3):960–981, 2018. doi:10.1137/15M1050574.

12 Julien Baste, Lars Jaffke, Tomás Masarík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019. doi:10.3390/a12120254.

13 Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 201–210, 2010. doi:10.1145/1806689.1806718.

14 Aditya Bhaskara, Moses Charikar, Aravindan Vijayaraghavan, Venkatesan Guruswami, and Yuan Zhou. Polynomial integrality gaps for strong SDP relaxations of densest k-subgraph. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 388–405, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2095116.2095150.

15 Aditya Bhaskara, Mehrdad Ghadiri, Vahab S. Mirrokni, and Ola Svensson. Linear relaxations for finding diverse elements in metric spaces. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4098–4106, 2016. URL: https://proceedings.neurips.cc/paper/2016/hash/d79c6256b9bdac53a55801a066b70da3-Abstract.html.

16 Allan Borodin, Aadhar Jain, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Trans. Algorithms*, 13(3):41:1–41:25, 2017. doi:10.1145/3086464.

17 Mark Braverman, Young Kun-Ko, Aviad Rubinstein, and Omri Weinstein. ETH hardness for densest-k-subgraph with perfect completeness. In *SODA*, pages 1326–1341, 2017.

18 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

19 Jaime G. Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In W. Bruce Croft, Alistair Moffat, C. J. van Rijsbergen, Ross Wilkinson, and Justin Zobel, editors, *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 335–336. ACM, 1998. doi:10.1145/290941.291025.

20 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-exponential time hypothesis to fixed parameter tractable inapproximability: Clique, dominating set, and more. *SIAM J. Comput.*, 49(4):772–810, 2020. doi:10.1137/18M1166869.

21 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5029–5037, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/978fce5bcc4eccc88ad48ce3914124a2-Abstract.html.

22 Eden Chlamtác, Pasin Manurangsi, Dana Moshkovitz, and Aravindan Vijayaraghavan. Approximation algorithms for label cover and the log-density threshold. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 900–919. SIAM, 2017. doi:10.1137/1.9781611974782.57.

23 Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016. URL: http://eccc.hpi-web.de/report/2016/128.

24 Marina Drosou, H. V. Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. Diversity in big data: A review. *Big Data*, 5(2):73–84, 2017. doi:10.1089/big.2016.0054.

25 Alessandro Epasto, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Scalable diversity maximization via small-size composable core-sets (brief announcement). In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 41–42. ACM, 2019. doi:10.1145/3323165.3323172.

26 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

27 Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 534–543, New York, NY, USA, 2002. ACM. doi:10.1145/509907.509985.

28 Uriel Feige, Guy Kortsarz, and David Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001. doi:10.1007/s004530010050.

29 Uriel Feige and Michael Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *J. Algorithms*, 41(2):174–211, November 2001. doi:10.1006/jagm.2001.1183.

30 Uriel Feige and Michael Seltser. On the densest k-subgraph problem. Technical report, Weizmann Institute of Science, Rehovot, Israel, 1997.

31 Fedor V. Fomin, Petr A. Golovach, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. Diverse collections in matroids and graphs. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.31.

32 Doron Goldstein and Michael Langberg. The dense k subgraph problem. *CoRR*, abs/0912.5327, 2009. URL: http://arxiv.org/abs/0912.5327, arXiv:0912.5327.

33 Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009. doi:10.1145/1526709.1526761.

34 Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, See Woo Lee, and Yota Otachi. Computing diverse shortest paths efficiently: A theoretical and experimental study. *CoRR*, abs/2112.05403, 2021. arXiv:2112.05403.

35 Refael Hassin, Shlomi Rubinstein, and Arie Tamir. Approximation algorithms for maximum dispersion. *Oper. Res. Lett.*, 21(3):133–137, 1997. doi:10.1016/S0167-6377(97)00034-5.

36 Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 100–108. ACM, 2014. doi:10.1145/2594538.2594560.

37 Subhash Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput.*, 36(4):1025–1071, 2006. doi:10.1137/S0097539705447037.

38 Michael J. Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987. doi:10.1111/j.1538-4632.1987.tb00133.x.

39 Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

40 Pasin Manurangsi. Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph. In *STOC*, pages 954–961, 2017. doi:10.1145/3055399.3055412.

41 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In *SODA*, pages 62–81, 2020. doi:10.1137/1.9781611975994.5.

42 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs. In *ICALP*, pages 78:1–78:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.78.

43 Pasin Manurangsi, Aviad Rubinstein, and Tselil Schramm. The strongish planted clique hypothesis and its consequences. In *ITCS*, pages 10:1–10:21, 2021. doi:10.4230/LIPIcs.ITCS.2021.10.

44 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

45 I. Douglas Moon and Sohail S. Chaudhry. An analysis of network location problems with distance constraints. *Management Science*, 30(3):290–307, 1984. URL: http://www.jstor.org/stable/2631804.

46 Zafeiria Moumoulidou, Andrew McGregor, and Alexandra Meliou. Diverse data selection under fairness constraints. In Ke Yi and Zhewei Wei, editors, *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*, volume 186 of *LIPIcs*, pages 13:1–13:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICDT.2021.13.

47 Dana Pessach and Erez Shmueli. Algorithmic fairness. *CoRR*, abs/2001.09784, 2020. arXiv:2001.09784.

48 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 755–764, New York, NY, USA, 2010. ACM. doi:10.1145/1806689.1806792.

49 S. S. Ravi, Daniel J. Rosenkrantz, and Giri Kumar Tayi. Heuristic and special case algorithms for dispersion problems. *Oper. Res.*, 42(2):299–310, 1994. doi:10.1287/opre.42.2.299.

50 LT Rodrygo, Craig Macdonald, and Iadh Ounis. Search result diversification. *Foundations and Trends in Information Retrieval*, 9(1):1–90, 2015.

51 Anand Srivastav and Katja Wolf. Finding dense subgraphs with semidefinite programming. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '98, pages 181–191, London, UK, UK, 1998. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646687.702946.

52 Sepehr Abbasi Zadeh, Mehrdad Ghadiri, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Scalable feature selection via distributed diversity maximization. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 2876–2883. AAAI Press, 2017. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14914.

Round-Optimal Lattice-Based Threshold Signatures, Revisited

Shweta Agrawal ✉
Indian Institute of Technology, Madras, India

Damien Stehlé ✉
ENS de Lyon, France
Institut Universitaire de France, Paris, France

Anshu Yadav ✉
Indian Institute of Technology, Madras, India

Abstract

Threshold signature schemes enable distribution of the signature issuing capability to multiple users, to mitigate the threat of signing key compromise. Though a classic primitive, these signatures have witnessed a surge of interest in recent times due to relevance to modern applications like blockchains and cryptocurrencies. In this work, we study round-optimal threshold signatures in the post-quantum regime and improve the only known lattice-based construction by Boneh et al. [CRYPTO'18] as follows:

- *Efficiency.* We reduce the amount of noise flooding used in the construction from $2^{\Omega(\lambda)}$ down to $\sqrt{Q}$, where Q is the bound on the number of generated signatures and λ is the security parameter. By using lattice hardness assumptions over polynomial rings, this allows to decrease the signature bit-lengths from $\widetilde{O}(\lambda^3)$ to $\widetilde{O}(\lambda)$, bringing them significantly closer to practice. Our improvement relies on a careful analysis using Rényi divergence rather than statistical distance in the security proof.
- *Instantiation.* The construction of Boneh et al. requires a standard signature scheme to be evaluated homomorphically. To instantiate this, we provide a homomorphism-friendly variant of Lyubashevsky's signature [EUROCRYPT '12] which achieves low circuit depth by being "rejection-free" and uses an optimal, moderate noise flooding of $\sqrt{Q}$, matching the above.
- *Towards Adaptive Security.* The construction of Boneh et al. satisfies only selective security, where all the corrupted parties must be announced before any signing query is made. We improve this in two ways: in the Random Oracle Model, we obtain *partial adaptivity* where signing queries can be made before the corrupted parties are announced but the set of corrupted parties must be announced *all at once*. In the standard model, we obtain full adaptivity, where parties can be corrupted at any time but this construction is in a weaker *pre-processing* model where signers must be provided correlated randomness of length proportional to the number of signatures, in an offline preprocessing phase.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases Post-Quantum Cryptography, Lattices, Threshold Signatures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.8

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eprint.iacr.org/2022/634.pdf`

Funding This work was partly supported by the DST "Swarnajayanti" fellowship, National Blockchain Project, European Union Horizon 2020 Research and Innovation Program Grant 780701, BPI-France in the context of the national project RISQ (P141580), and the ANR AMIRAL project (ANR-21-ASTR-0016). The work is also partially supported by Microsoft Research Travel Grants to travel and present the paper at the conference.

Acknowledgements Part of the research corresponding to this work was conducted while the authors were visiting the Simons Institute for the Theory of Computing.

1 Introduction

A threshold signature [23] distributes the signature issuing capacity among several users, so that a signature can be generated only if a sufficient number of users collaborate to sign a message. In more detail, each of N parties holds a partial signing key, and any set of parties at least as large as a given threshold $t \leq N$ can participate in a protocol to generate a signature. Security requires that a valid signature cannot be generated if fewer than t parties cooperate.

A central motivation for constructing threshold signatures is to decentralize the trust placed in the signing authority, thus reducing the risk of the signing key being compromised. While threshold signatures have been studied for a long time [43, 24, 14, 33, 31, 44, 25, 20, 15, 13, 30, 21, 32, 7], they have received renewed attention in recent years due to numerous applications in modern topics such as cryptocurrencies and blockchains. Most prior work has focused on creating distributed versions of ECDSA or Schnorr signatures [44, 31, 25, 14, 15] which are not quantum secure. From conjectured post-quantum assumptions such as those related to Euclidean lattices, much less is known, especially with optimal round complexity.

1.1 Prior Work

The thresholdisation of lattice-based signatures from the NIST post-quantum cryptography project has been investigated in [19] but the resulting candidates incur several rounds of communication. A threshold signature restricted to $t = N$ was proposed in [22] but it also involves possibly many rounds, because of aborts. To the best of our knowledge, the only lattice-based, round-optimal threshold signature construction is by Boneh et al. [9] (henceforth BGGJKRS), relying on the Learning With Errors problem (LWE). However, while this construction provided the first feasibility result for a long-standing open problem, it suffers from the following drawbacks:

1. *Noise Flooding and Impact on Parameters.* It makes use of the so-called "noise flooding" technique [34, 6, 37], which aims to hide a noise term $e \in \mathbb{Z}$ that possibly contains sensitive information, by adding to it a fresh noise term e' whose distribution has a standard deviation that is much larger than an a priori upper bound on $|e|$. To get security against attackers with success probability $2^{-o(\lambda)}$ where λ is the security parameter, the standard deviation of e' must be a factor $2^{\Omega(\lambda)}$ larger than the upper bound on $|e|$.
   Unfortunately, this precludes the use of an efficient LWE parametrisation. Concretely, one has to set the LWE noise rate α as $2^{-\Omega(\lambda)}$ so that $|e'|$ remains small compared to the working modulus q. As the best known algorithms for attacking LWE with (typical) parameters n, q, α have run-times that grow as $\exp(\widetilde{O}(n \log q / \log^2 \alpha))$ (see, e.g., [38]) this leads to setting $n \log q = \widetilde{\Omega}(\lambda^3)$. As the signature shares have bit-sizes that grow as $\Omega(n \log q)$, this leads to $\widetilde{\Omega}(\lambda^3)$-bit signature sizes – prohibitively expensive in practice.

2. *Instantiating Underlying Signature.* It requires a standard signature scheme to be evaluated homomorphically. BGGJKRS do not suggest a candidate and existing lattice based signatures are not suitable – the GPV signature scheme [35] and its practical versions [27, 51, 28] seem ill-suited, as the signing algorithm is very sequential, and the required 1-dimensional Gaussian samples are obtained via algorithms based on rejection sampling (see, e.g., [39, 56]) that are costly to transform into circuits. The other candidate is Lyubashevsky's signature scheme [46, 47]. It has the advantage of being far less sequential, but it also relies on rejection sampling: when some rejection test does not pass, then one needs to restart the signing process.

3. *Selective Security.* It only achieves a very restricted notion of *selective* security, where all the corrupted parties must be announced before any partial signing query is made. To obtain security in the more realistic *adaptive* setting, one option is to invoke complexity leveraging, which consists in guessing at the outset which parties will be corrupted. This is not only dissatisfying as a solution but also leads to a further degradation of the parameters.

1.2 Our Contributions

In this work, we improve the construction from [9] as follows:

- *Efficiency.* We decrease the noise flooding ratio from $2^{\Omega(\lambda)}$ down to $\sqrt{Q}$, where Q is the bound on the number of generated signatures. This gives a one-round threshold signature of bit-length growing as $\widetilde{O}(\lambda \log^2 Q)$, which is $\widetilde{O}(\lambda)$ for any polynomially bounded Q,[1] in contrast with $\widetilde{O}(\lambda^3)$ for the construction from [9]. These bit-lengths are obtained when relying on the ring variants of SIS and LWE [48, 50, 54, 49]. Additionally, we show that the amount of noise flooding used in this construction is *optimal*, by exhibiting an attack when a smaller noise flooding ratio is used.
- *Instantiation.* To instantiate the signature underlying BGGJKRS, we provide a homomorphism friendly variant of Lyubashevsky's signature [EUROCRYPT '12] which achieves low circuit depth. We remove the rejection sampling at the expense of adding moderate noise of size $\sqrt{Q}$, matching the above. Again, we show that this amount of flooding is optimal by demonstrating an attack when smaller flooding is used.
- *Selective versus Adaptive.* As discussed above, the construction BGGJKRS satisfies only selective security. We improve this in two ways: in the Random Oracle Model (ROM), in which a hash function is being modeled as a uniformly sampled function with the same domain and range, we obtain a notion of *partial adaptivity* where signing queries can be made before the corrupted parties are announced. However, the set of corrupted parties must be announced *all at once*. In the standard model, we obtain a construction with full adaptivity, where parties can be corrupted at any stage in the protocol. However, this construction is in a weaker *pre-processing* model where signers must be provided correlated randomness of length proportional to the number of signing queries. The informed reader may notice similarities with the "MPC with Preprocessing" model, please see [29] and references therein [2].

1.3 Technical Overview

Recap of BGGJKRS Threshold Signatures. The round-optimal threshold signatures provided by [9] are designed using a "universal thresholdizer" which enables the thresholdizing of a number of primitives. This thresholdizer is itself instantiated using a threshold version of "special" fully homomorphic encryption (FHE), which in turn can be constructed using the

[1] For many applications, the bound Q is quite limited and can be considered to be a small polynomial in λ. For example, for applications pertaining to cryptocurrencies, the bound Q may capture the total number of transactions made with a user's wallet during the lifetime of a signing key. According to statistics available at the URLs below, one transaction per day and per user is a generous upper bound. This suggests that number of signing queries in the lifecycle of the key will be quite limited. `https://www.blockchain.com/charts/n-transactions`, `https://www.statista.com/statistics/647374/worldwide-blockchain-wallet-users/`

[2] Note that we can trade the offline sharing of correlated randomness with an additional communication round in the signing protocol – however, this would destroy round optimality.

LWE assumption. In threshold fully homomorphic encryption (TFHE), the setup algorithm takes as input a threshold t and produces a set of decryption key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ for the parties such that every party can perform a partial decryption using its own decryption key and any t out of N partial decryptions can be combined into a complete decryption of the ciphertext in a single round.

In more detail, the TFHE construction of BGGJKRS leverages the fact that the decryption in LWE based FHE schemes [12, 11, 36] requires to compute an inner product of the ciphertext ct with the secret key sk, followed by a rounding operation. Since inner product is a linear operation, a natural approach to thresholdize FHE decryption is by applying a Shamir t-out-of-N secret sharing to sk. This will yield N keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$, which can be distributed to the N users. Now, to decrypt a ciphertext ct, each user can compute the inner product with its individual secret key sk_i as its partial decryption m_i. To combine any t partial decryptions into the final decryption, the combiner chooses Lagrange coefficients $\gamma_1, \ldots, \gamma_t$ so that $\sum_i \gamma_i \mathsf{sk}_i = \mathsf{sk}$. Then, she computes

$$\sum_i \gamma_i m_i = \sum_i \gamma_i \langle \mathsf{ct}, \mathsf{sk}_i \rangle = \langle \mathsf{ct}, \sum_i \gamma_i \mathsf{sk}_i \rangle = \langle \mathsf{ct}, \mathsf{sk} \rangle,$$

followed by rounding, as desired. However, this appealingly simple construction turns out to be insecure. This is because each time a party computes a partial decryption, it leaks information about its secret share sk_i via the inner product with (the public) value ct.

To get around this insecurity, a natural approach is to add noise to the partial decryption which quickly transforms a simple computation to intractable. However, care must be taken to ensure that this added noise does not affect correctness, since it is later multiplied by the Lagrange coefficients during reconstruction: the previous $\sum_i \gamma_i m_i$ will now become $\sum_i \gamma_i (m_i + e_i)$ for some noise terms e_i. BGGJKRS propose two solutions – one to use a secret sharing scheme whose reconstruction coefficients are binary, and another, to "clear the denominators" by observing that since the Lagrange coefficients are rational numbers, it is possible to scale them to be integers. The exact details are irrelevant for the current discussion and hence omitted (please refer to [9] for more details).

To use this technique to construct threshold signatures, the authors propose the following. Choose a signature scheme Sig, compute an FHE encryption $\mathsf{ct}_{\mathsf{sk}}$ of its signing key $\mathsf{Sig.sk}$ and let each signer homomorphically evaluate the signing algorithm for a message μ on this ciphertext. In more detail, given $\mathsf{ct}_{\mathsf{sk}} = \mathsf{FHE.Enc}(\mathsf{Sig.sk})$, each party first computes $\mathsf{FHE.Eval}(C, \mathsf{ct}_{\mathsf{sk}})$ where C is the circuit $\mathsf{Sig.Sign}(\mu, \cdot)$. By correctness of FHE, this yields an FHE encryption of the signature $\sigma = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mu)$. To this ciphertext, the thresholdization trick described above may now be applied.

Modeling the Adversary and Effect on Parameters. In their analysis, BGGJKRS consider the complexity-theory security requirement of "no polynomial time attacks", corresponding to assuming attacks with advantage $\epsilon = \lambda^{-O(1)}$ and run-time $\lambda^{O(1)}$. However, for practically motivated primitives like threshold signatures, it is more meaningful to consider attackers with advantage $2^{-o(\lambda)}$ and run-time $2^{o(\lambda)}$. We choose our adversarial model so that all attacks should be exponential while all honest algorithms run in polynomial time. Compared to the complexity-theory definition of security, this provides a much more significant (and practically meaningful) hardness gap between honest and malicious parties.

For subexponentially strong attackers as described above, the noise flooding used in BGGJKRS is exponential, severely damaging the practicality of the scheme, despite the exciting developments in practical FHE [18, 57, 41, 17, 16, 26]. In more detail, the proof

requires to make the statistical distance between some noise terms e' and $e + e'$ small, so that knowing $e + e'$ is essentially the same as knowing e', which does not carry sensitive information. To get security against attackers with advantage $2^{-o(\lambda)}$, the statistical distance must be set to $2^{-\Omega(\lambda)}$ and, as a result, the standard deviation of e' must be a factor $2^{\Omega(\lambda)}$ larger than the upper bound on $|e|$.

Tightening Analysis via Rényi divergence. In this work, we examine whether this flooding noise can be improved so that the impact of flooding e by e' on efficiency is minimised. To this end, we explore using *Rényi divergence* rather than statistical distance to bound the distance between distributions in the security proof. Rényi divergence has been used in prior work as a replacement to the statistical distance in lattice based cryptography [42, 45, 8, 52, 40, 4, 10, 3, 5]. To understand why this may be beneficial, let us first see how statistical distance is used in typical security proofs of cryptography. Let $\mathcal{P}$ and $\mathcal{Q}$ be two non-vanishing probability distributions over a common measurable support X. Typical security proofs consider a hard problem relying on some ideal distribution $\mathcal{Q}$, and then replace this ideal distribution by a real world distribution $\mathcal{P}$. When the statistical distance $\Delta(\mathcal{Q}, \mathcal{P})$ between the two distributions is small, the problem remains hard, implying security. This is made rigorous by the so-called "probability preservation" property which says that for any measurable event $E \subseteq X$, we have $\mathcal{Q}(E) \geq \mathcal{P}(E) - \Delta(\mathcal{Q}, \mathcal{P})$.

Let us now define Rényi Divergence (RD). For $a \in (1, \infty)$, the RD of order a is defined by $R_a(\mathcal{P}||\mathcal{Q}) = \left(\sum_{x \in X} \frac{\mathcal{P}(x)^a}{\mathcal{Q}(x)^{a-1}}\right)^{\frac{1}{a-1}}$. It enjoys an analogous probability preservation property, though multiplicative as against additive. For $E \subseteq X$, we have $\mathcal{Q}(E) \geq \mathcal{P}(E)^{\frac{a}{a-1}} / R_a(\mathcal{P}||\mathcal{Q})$. Thus, if an event E occurs with significant probability under $\mathcal{P}$, and if the SD or the RD is small, then the event E also occurs with significant probability under $\mathcal{Q}$. As discussed in [5], probability preservation in SD is meaningful when the distance is smaller than any $\mathcal{P}(E)$ that the security proof is required to deal with – if $\mathcal{P}(E) \geq \epsilon$ for some ϵ, then we require that $\Delta(\mathcal{Q}, \mathcal{P}) < \epsilon$. The analogous requirement for RD is $R_a(\mathcal{P}||\mathcal{Q}) \leq \mathsf{poly}(1/\epsilon)$. Bai et al. [5] observed that RD is often less demanding than SD in proofs. This is because RD between distributions may be small enough to suffice for RD probability preservation while SD may be too large for the SD probability preservation to be applicable. Thus, RD can often serve as a better tool for security analysis, especially in applications with search-type security definitions, like signatures.

In this work, we study the applicability of RD analysis in the construction of threshold signatures. Building upon the above approach, we show that a limited flooding growing as $\sqrt{Q}$ suffices in BGGJKRS, where Q is the number of signing queries made by the attacker. We note that this is a substantial improvement in practice, since the number of sign queries is typically very different, and much smaller, than the run time of the adversary. Note that signature queries require active participation by an honest user and there is no reason for an honest user to keep replying after an overly high number of queries that clearly shows adversarial behavior. As a concrete example, in the NIST post quantum project [1], adversarial runtimes can go up to 2^{256} in some security levels, but the number of signature queries is always bounded by 2^{64} (which is itself an overly conservative bound in many scenarios). Thus, dependence on the number of queries is significantly better than exponential dependence on the security parameter, and this leads to a significant improvement in the signature bit size.

Optimality of our Moderate Flooding. We also show that this magnitude of flooding is necessary for this construction, by exhibiting a statistical attack when smaller noise is used. At a high level, our attack proceeds as follows. First we show that using legitimate

information available to her, the adversary can compute $\mathsf{err}_M + e_{1,M}$ where err_M is the error that results from homomorphically evaluating the signing algorithm for message M and $e_{1,M}$ is the flooding noise that is used in the partial signature of the first party. As a warmup, consider the setting where the flooding noise is randomized. Now, since the signature scheme is deterministic, the term err_M depends only on M and remains fixed across multiple queries for the same message. On the other hand, the term $e_{1,M}$ keeps changing. Using Hoeffding's bound, it is possible to estimate the average of $e_{1,M}$ across multiple queries and use this to recover err_M, leading to an attack.

This attack may be avoided by making the flooding noise a deterministic function of the message, e.g., by using a pseudo-random function evaluated on the message to generate the noise. We show that this modification is not sufficient to make the threshold signature construction secure. For this purpose, we design a signature scheme which includes "useless" information in the signature: this information does not affect correctness nor security of the signature itself, but allows us to recreate the attack described above on the resulting threshold signature. We start with a secure signature scheme $\mathsf{Sig} = (\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ whose signing key is a uniform bit-string among those with the same number of 0's and 1's. Now, let us consider a special signature scheme $\mathsf{Sig}' = (\mathsf{Sig}'.\mathsf{KeyGen}, \mathsf{Sig}'.\mathsf{Sign}, \mathsf{Sig}'.\mathsf{Verify})$ derived from Sig by modifying the signing algorithm as follows: for $i \in [|\mathsf{Sig.sk}|]$, if $\mathsf{Sig.sk}_i = 0$, then append a 0 to the signature. Since our signing key has exactly half as many 0's as 1's, this leads to a string of $|\mathsf{Sig.sk}|/2$ zeroes being appended to every signature: this does not leak any information and does not affect correctness (it is simply ignored during verification). Now, consider using Sig' to instantiate our threshold signature scheme. Then, for any message M, the FHE ciphertext CT_{σ_M} now additionally includes homomorphically evaluated encryptions of $\{\mathsf{Sig.sk}_i\}_{i \in [|\mathsf{Sig.sk}|]:\mathsf{Sig.sk}_i = 0}$. Note that these extra encryptions are designed to be a deterministic function of the secret key so that across multiple messages, the corresponding error term (obtained via homomorphic evaluation) will not change. On the other hand, the message-dependent error terms can be assumed to change across messages. Due to this, the error term recovered by the adversary will be a sum of a fixed term (dependent only on the secret key) plus a fresh term per signature, which allows it to recreate the first attack. Please see Section 3 for more details.

Homomorphism-Friendly Signature. Next, we provide a variant of Lyubashevsky's signature scheme [47] which enjoys low circuit depth and is homomorphism friendly. As discussed above, Lyubashevsky's signature contains a rejection sampling step, whose purpose is to make the distribution of the resultant signature canonical, but this step is cumbersome to implement homomorphically. We show that by using RD analysis in place of statistical distance, analogously to the case of threshold signatures discussed above, the rejection sampling step can be replaced by noise flooding of moderate magnitude $\sqrt{Q}$. Additionally, we show that this flooding is optimal – please see Section 4 for details.

Towards Adaptive Security. Another limitation of the construction of $\mathsf{BGGJKRS}$ is that security is proved in the weak "selective" model where the adversary must announce all corrupted users before receiving the public parameters and verification key. In contrast, the more reasonable adaptive model allows the adversary to corrupt users based on the public parameters, the verification key and previous user corruptions it may have made. We briefly describe the difficulty in achieving adaptive security. At a high level, in the selective game, the challenger proceeds by simulating the partial keys corresponding to the honest parties in a "special way". The challenge in the adaptive setting is that without knowing who are the honest/corrupted parties, the challenger does not know which partial keys to program.

For more details, let us consider the case of an N-out-of-N threshold signature. In the simulation, the challenger knows which party is honest at the start of the game, e.g., player N. Now, the challenger can sample FHE secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{N-1}$ randomly, implicitly setting the last share as $\mathsf{sk} - \sum_{i\in[N-1]} \mathsf{sk}_i$. To invoke the unforgeability of the underlying signature scheme Sig, the challenger must "forget" the signing key $\mathsf{Sig.sk}$ at some point in the proof, and rely on the Sig challenger to return signatures, which it then encrypts using the (public key) FHE scheme. By correctness of FHE, this is the same as computing the signing circuit for a given message on the ciphertext containing the secret key, which is what happens in the real world. However, the FHE encryption of signing key $\mathsf{Sig.sk}$ is provided as part of the public parameters in the real world, which in turn means that the FHE secret key must be "forgotten" so that the FHE ciphertext of $\mathsf{Sig.sk}$ is indistinguishable from a dummy value. Yet the challenger must return legitimate partial signatures of queried messages m_j in the security game, which in turn are (noisy) partial decryptions of the FHE ciphertexts $\widehat{\sigma_j}$ of signatures σ_j. Knowing all the corrupt secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{N-1}$ from the outset enables the challenger to walk this tightrope successfully – it obtains σ_j from the Sig challenger, computes an FHE encryption $\widehat{\sigma_j}$ of this, computes partial decryptions using $\mathsf{sk}_1, \ldots, \mathsf{sk}_{N-1}$, floods these with appropriate noise and returns these to the adversary.

In the adaptive game, the honest player is not known at the beginning of the game so the challenger is unable to sample FHE secret key shares as described above. When requested for a partial signatures for message m_j, it can obtain the corresponding signature σ_j and can FHE encrypt it, but cannot decrypt it using secret key shares which are unavailable. To preserve correctness and indistinguishability from the real world, it is forced to return (noisy) random secret shares $\{\sigma_{i,j}\}_{i\in[N],j\in\mathsf{poly}}$ of σ_j as partial signatures, for unbounded j. Later if user 1 is corrupted (say), the adversary receives the secret key share sk_1. Now, to preserve indistinguishability, the challenger must explain the partial signatures $\{\sigma_{1,j}\}_{j\in\mathsf{poly}}$ corresponding to user 1 as $\langle \widehat{\sigma}_j, \mathsf{sk}_1 \rangle \approx \sigma_{1,j}$, which seems impossible for unbounded j.

We overcome this hurdle in the ROM by having the challenger simulate all partial keys as though corresponding to a corrupt user and when the list of corrupted parties becomes available, "program" the ROM to "explain" the returned keys in a consistent way. This yields an intermediate notion of "partial adaptivity", in which the attacker can make signing queries before corruption, but must announce its corrupted users all at once. In more detail, we modify the signing key to additionally contain a random secret share of $\mathbf{0}$, i.e., each party is provided a vector $\mathbf{v}_i$ of length N, such that $\sum_{i\in[N]} \mathbf{v}_i = \mathbf{0}$. In the scheme, to compute a partial signature for a message m_j, the partial signing algorithm first computes $r_{i,j} = H(j,K)^{\mathsf{T}}\, \mathbf{v}_i$ where $H(j,K)$ is a random vector of length N, and K is a secret value required for a technical reason that we will not discuss here. It then returns $\langle \widehat{\sigma}_j, \mathsf{sk}_i \rangle + \mathsf{noise}_{i,j} + r_{i,j}$. By linearity, it holds that $\sum_{i\in[N]} H(j,K)^{\mathsf{T}}\, \mathbf{v}_i = \mathbf{0}$, so correctness is not affected. But the unbounded programmability of the ROM helps us overcome the aforementioned impasse in the proof. Now, the challenger answers partial signature queries by returning (noisy) random secret shares $\{\sigma_{i,j}\}_{i\in[N],j\in\mathsf{poly}}$ of σ_j. When later, user 1 is corrupted, it can correctly explain the returned signatures as follows: it samples sk_1, computes $d_{1,j} = \langle \widehat{\sigma}_j, \mathsf{sk}_1 \rangle + \mathsf{noise}$ and sets $r_{1,j} = \sigma_{1,j} - d_{1,j}$. Now we may program $H(j,K)$ so that $r_{i,j} = H(j,K)^{\mathsf{T}}\, \mathbf{v}_i$ for all j – it can be checked that there are enough degrees of freedom to satisfy these equations. However, since all secrets of a user are revealed when it is corrupted, the value $H(j,K)$ is fixed when even a single user is corrupted. This is why we require that all corruptions be made simultaneously and only achieve the restricted notion of "partial" adaptivity.

We also provide a construction in the standard model which achieves full adaptivity where users can be corrupted at arbitrary points in the security game. But this construction is only secure in a weaker *pre-processing* model where the signers must be provided correlated

randomness of length proportional to the number of signing queries, in an offline pre-processing phase. We emphasize that the correlated randomness is independent of the messages to be signed later. This model is reminiscent of the "MPC with Preprocessing" model (please see [29] and references therein). We refer the reader to Section 5 for more details.

2 Preliminaries

In this section, we define some preliminaries used in our work. Please refer to the full version of the paper [2] for additional preliminaries.

▶ **Definition 1** (Threshold Signatures). *Let $P = \{P_1, \ldots, P_N\}$ be a set of N parties. A threshold signature scheme for a class of efficient access structures $\mathbb{S}$ on P is a tuple of PPT algorithms denoted by* $\mathsf{TS} = (\mathsf{TS.KeyGen}, \mathsf{TS.PartSign}, \mathsf{TS.PartSignVerify}, \mathsf{TS.Combine}, \mathsf{TS.Verify})$ *defined as follows:*

- $\mathsf{TS.KeyGen}(1^\lambda, \mathsf{A}) \to (\mathsf{pp}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i=1}^N)$*: On input the security parameter λ and an access structure* A*, the* KeyGen *algorithm outputs public parameters* pp*, verification key* vk *and a set of key shares* $\{\mathsf{sk}_i\}_{i=1}^N$*.*
- $\mathsf{TS.PartSign}(\mathsf{pp}, \mathsf{sk}_i, m) \to \sigma_i$*: On input the public parameters* pp*, a partial signing key* sk_i *and a message $m \in \{0,1\}^*$, the partial signing algorithm outputs a partial signature σ_i.*
- $\mathsf{TS.PartSignVerify}(\mathsf{pp}, m, \sigma_i) \to \mathsf{accept/reject}$*: On input the public parameters* pp*, a message $m \in \{0,1\}^*$ and a partial signature σ_i, the partial signature verification algorithm outputs accept or reject.*
- $\mathsf{TS.Combine}(\mathsf{pp}, \{\sigma_i\}_{i\in S}) \to \sigma_m$*: On input the public parameters* pp *and the partial signatures $\{\sigma_i\}_{i\in S}$ for $S \in \mathsf{A}$, the combining algorithm outputs a full signature σ_m.*
- $\mathsf{TS.Verify}(\mathsf{vk}, m, \sigma_m) \to \mathsf{accept/reject}$*: On input a verification key* vk*, a message m and a signature σ_m, the verification algorithm outputs accept or reject.*

A TS scheme should satisfy the following requirements.

▶ **Definition 2** (Compactness). *A* TS *scheme for $\mathbb{S}$ satisfies compactness if there exist polynomials $\mathrm{poly}_1(\cdot), \mathrm{poly}_2(\cdot)$ such that for all λ, $\mathsf{A} \in \mathbb{S}$ and $S \in \mathsf{A}$, the following holds. For $(\mathsf{pp}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i=1}^N) \leftarrow \mathsf{TS.KeyGen}(1^\lambda, \mathsf{A})$, $\sigma_i \leftarrow \mathsf{TS.PartSign}(\mathsf{pp}, \mathsf{sk}_i, m)$ for $i \in S$, and $\sigma_m \leftarrow \mathsf{TS.Combine}(\mathsf{pp}, \{\sigma_i\}_{i\in S})$, we have that $|\sigma_m| \le \mathrm{poly}_1(\lambda)$ and $|\mathsf{vk}| \le \mathrm{poly}_2(\lambda)$.*

▶ **Definition 3** (Evaluation Correctness). *A signature scheme* TS *for $\mathbb{S}$ satisfies evaluation correctness if for all $\lambda, \mathsf{A} \in \mathbb{S}$ and $S \in \mathsf{A}$, the following holds. For $(\mathsf{pp}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i=1}^N) \leftarrow \mathsf{TS.KeyGen}(1^\lambda, \mathsf{A})$, $\sigma_i \leftarrow \mathsf{TS.PartSign}(\mathsf{pp}, \mathsf{sk}_i, m)$ for $i \in [N]$ and $\sigma_m \leftarrow \mathsf{TS.Combine}(\mathsf{pp}, \{\sigma_i\}_{i\in S})$, we have that $\Pr[\mathsf{TS.Verify}(\mathsf{vk}, m, \sigma_m) = accept] \ge 1 - \lambda^{-\omega(1)}$.*

▶ **Definition 4** (Partial Verification Correctness). *A signature scheme* TS *for $\mathbb{S}$ satisfies partial verification correctness if for all λ and $\mathsf{A} \in \mathbb{S}$, the following holds. For $(\mathsf{pp}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i=1}^N) \leftarrow \mathsf{TS.KeyGen}(1^\lambda, \mathsf{A})$, $\Pr[\mathsf{TS.PartSignVerify}(\mathsf{pp}, m, \mathsf{TS.PartSign}(\mathsf{pp}, \mathsf{sk}_i, m)) = 1] = 1 - \lambda^{-\omega(1)}$.*

▶ **Definition 5** (Unforgeability). *A* TS *scheme is unforgeable if for any adversary $\mathcal{A}$ with runtime $2^{o(\lambda)}$, the output of the following experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{TS},uf}(1^\lambda)$ is 1 with probability $2^{-\Omega(\lambda)}$:*

1. *On input the security parameter λ, the adversary outputs an access structure $\mathsf{A} \in \mathbb{S}$.*
2. *Challenger runs the $\mathsf{TS.KeyGen}(1^\lambda)$ algorithm and generates public parameters* pp*, verification key* vk *and set of N key shares $\{\mathsf{sk}_i\}_{i=1}^N$. It sends* pp *and* vk *to $\mathcal{A}$.*
3. *Adversary $\mathcal{A}$ then issues polynomial number of following two types of queries in any order*
   - *Corruption queries: $\mathcal{A}$ outputs a party $i \in [N]$ which it wants to corrupt. In response, the challenger returns the key share sk_i.*

- *Signature queries: $\mathcal{A}$ outputs a query of the form (m, i), where m is a message and $i \in [N]$, to get partial signature σ_i for m. The challenger computes σ_i as $\mathsf{TS.PartSign}(\mathsf{pp}, \mathsf{sk}_i, m)$ and provides it to $\mathcal{A}$.*

4. *At the end of the experiment, adversary $\mathcal{A}$ outputs a message-signature pair (m^*, σ^*).*
5. *The experiment outputs 1 if both of the following conditions are met: (i) Let $S \subseteq [N]$ be the set of corrupted parties, then S is a an invalid party set, i.e. $S \notin \mathsf{A}$ (ii) m^* was not queried previously as a signing query and $\mathsf{TS.Verify}(\mathsf{vk}, m^*, \sigma^*) = accept$.*

We also consider following weaker notions of unforgeability.

▶ **Definition 6** (Partially Adaptive Unforgeability). *Here, all the corruptions are done all at once. That is, Step 3, is now changed as follows:*

- *$\mathcal{A}$ issues polynomial number of signing queries of the form (m, i) adaptively and gets corresponding σ_i's.*
- *$\mathcal{A}$ outputs a set $S \subseteq [N]$ such that $S \notin \mathsf{A}$. The challenger returns $\{\mathsf{sk}_i\}_{i \in S}$.*
- *$\mathcal{A}$ continues to issue polynomial number of more signing queries of the form (m, i) adaptively, and gets corresponding σ_i.*

Rest of the steps remain the same.

▶ **Definition 7** (Selective Unforgeability). *In this case, all the corruptions happen before any signing query. That is, Step 3, is now further changed as follows:*

- *$\mathcal{A}$ outputs a set $S \subseteq [N]$ such that $S \notin \mathsf{A}$. The challenger returns $\{\mathsf{sk}_i\}_{i \in S}$.*
- *$\mathcal{A}$ issues polynomial number of signing queries of the form (m, i) adaptively, and gets corresponding σ_i.*

Rest of the steps remain the same.

▶ **Definition 8** (Robustness). *A TS scheme for $\mathbb{S}$ satisfies robustness if for all λ, the following holds. For any adversary $\mathcal{A}$ with run-time $2^{o(\lambda)}$, the following experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{TS},rb}(1^\lambda)$ outputs 1 with probability $2^{-\Omega(\lambda)}$:*

- *On input the security parameter 1^λ, the adversary outputs an access structure $\mathsf{A} \in \mathbb{S}$.*
- *The challenger samples $(\mathsf{pp}, \mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{TS.KeyGen}(1^\lambda, \mathsf{A})$ and provides $(\mathsf{pp}, \mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$ to $\mathcal{A}$.*
- *Adversary $\mathcal{A}$ outputs a partial signature forgery (m^*, σ_i^*, i).*
- *The experiment outputs 1 if $\mathsf{TS.PartSignVerify}(\mathsf{pp}, m^*, \sigma_i^*) = 1$ and $\sigma_i^* \neq \mathsf{TS.PartSign}(\mathsf{pp}, \mathsf{sk}_i, m^*)$.*

2.1 Rényi Divergence

▶ **Definition 9** (Rényi Divergence). *Let P and Q be any two discrete probability distributions such that $Supp(P) \subseteq Supp(Q)$. Then for $a \in (1, \infty)$, the Rényi Divergence of order a is defined by: $R_a(P||Q) = \left(\sum_{x \in Supp(P)} \frac{P(x)^a}{Q(x)^{a-1}}\right)^{\frac{1}{a-1}}$.*

The following lemma is borrowed from [5, Lemma 2.9], with the exception of the multiplicativity property for non-independent variables, which is borrowed from [53, Proposition 2].

▶ **Lemma 10.** *Let $a \in [1, \infty]$. Let P and Q denote distributions with $Supp(P) \subseteq Supp(Q)$. Then the following properties hold*

- ***Log Positivity:*** *$R_a(P||Q) \geq R_a(P||P) = 1$.*
- ***Data Processing Inequality:*** *$R_a(P^f||Q^f) \leq R_a(P||Q)$ for any function f, where P^f (resp. Q^f) denotes the distribution of $f(y)$ induced by sampling $y \leftarrow P$ (resp. $y \leftarrow Q$).*

- ***Probability preservation:*** *Let $E \subseteq Supp(Q)$ be an arbitrary event. If $a \in (1, \infty)$, then $Q(E) \geq P(E)^{\frac{a}{a-1}} / R_a(P\|Q)$.*
- ***Multiplicativity:*** *Assume that P and Q are two distributions of a pair of random variables (Y_1, Y_2). For $i \in \{1, 2\}$, let P_i (resp. Q_i) denote the marginal distribution of Y_i under P (resp. Q), and let $P_{2|1}(\cdot|y_1)$ (resp. $Q_{2|1}(\cdot|y_1)$) denote the conditional distribution of Y_2 given that $Y_1 = y_1$. Then we have:*
  - *$R_a(P||Q) = R_a(P_1||Q_1) \cdot R_a(P_2||Q_2)$ if Y_1 and Y_2 are independent for $a \in [1, \infty]$.*
  - *$R_a(P||Q) \leq R_a(P_1||Q_1) \cdot \max_{y_1 \in Y_1} R_a(P_{2|1}(\cdot|y_1)||Q_{2|1}(\cdot|y_1))$.*

We will use the following RD bounds. Note that proof tightness can often be improved by optimizing over a, as suggested in [55].

▶ **Lemma 11** ([5]). *For any n-dimensional lattice, $\Lambda \subseteq \mathbb{R}^n$ and $s > 0$, let P be the distribution $\mathcal{D}_{\Lambda,s,\mathbf{c}}$ and Q be the distribution $\mathcal{D}_{\Lambda,s,\mathbf{c}'}$ for some fixed $\mathbf{c}, \mathbf{c}' \in \mathbb{R}^n$. If $\mathbf{c}, \mathbf{c}' \in \Lambda$, let $\epsilon = 0$. Otherwise fix $\epsilon \in (0, 1)$ and assume that $s > \eta_\epsilon(\Lambda)$. Then for any $a \in (1, +\infty)$*

$$R_a(P||Q) \in \left[\left(\frac{1-\epsilon}{1+\epsilon}\right)^{\frac{2}{a-1}}, \left(\frac{1+\epsilon}{1-\epsilon}\right)^{\frac{2}{a-1}} \right] \cdot \exp\left(a\pi \frac{||\mathbf{c}-\mathbf{c}'||^2}{s^2}\right).$$

3 More Efficient Threshold Signatures from Lattices

In this section, we show how to drastically decrease the exponential flooding used in the scheme by Boneh et al. [9]. We also show that the limited flooding that we use is in fact optimal, and smaller noise would lead to an attack. For ease of exposition, the construction below is for the special case of N out of N threshold and restricted to selective security. We extend it to t out of N threshold in the full version of the paper [2] and to adaptive security in Section 5. In Section 4, we show how to instantiate the underlying signature scheme using a variant of Lyubashevsky's signature [47] with matching moderate flooding.

3.1 Optimizing the Boneh et al. scheme using the Rényi Divergence

Our scheme is similar to the one in [9] and is provided in Figure 1. The construction uses the following building blocks:

- A PRF $F : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^r$, where $\mathcal{K}$ is the PRF key space and r is the bit-length of randomness used in sampling from discrete Gaussian $\mathcal{D}_s$.
- A fully homomorphic encryption scheme $\mathsf{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$. As in [9], we also assume that the $\mathsf{FHE.Dec}$ can be divided into two sub-algorithms: $\mathsf{FHE.decode}_0$ and $\mathsf{FHE.decode}_1$.
- A UF-CMA signature scheme $\mathsf{Sig} = (\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$.
- A context hiding homomorphic signature scheme $\mathsf{HS} = (\mathsf{HS.PrmsGen}, \mathsf{HS.KeyGen}, \mathsf{HS.Sign}, \mathsf{HS.SignEval}, \mathsf{HS.Process}, \mathsf{HS.Verify}, \mathsf{HS.Hide}, \mathsf{HS.HVerify})$ to provide robustness.
- An N out of N secret sharing scheme Share.

Correctness. From the correctness of $\mathsf{FHE.Eval}$, CT_σ is an encryption of $\mathcal{C}_M(\mathsf{Sig.sk}) = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, M) = \sigma_M$, which decrypts with $\mathsf{FHE.sk}$. So, $\mathsf{FHE.decode}_0(\mathsf{FHE.sk}, \mathsf{CT}_\sigma) = \sigma_M \lfloor q/2 \rceil + e$. The signature computed by the $\mathsf{TS.Combine}$ algorithm is

$$\begin{aligned}
\mathsf{FHE.decode}_1\Big(\sum_{i=1}^{N} \sigma_i\Big) &= \mathsf{FHE.decode}_1\Big(\textstyle\sum_{i=1}^{N} \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_\sigma) + \sum_{i=1}^{N} e'_i\Big) \\
&= \mathsf{FHE.decode}_1\Big(\mathsf{FHE.decode}_0\Big(\textstyle\sum_{i=1}^{N} \mathsf{sk}_i, \mathsf{CT}_\sigma\Big) + \sum_{i=1}^{N} e'_i\Big) \\
&= \mathsf{FHE.decode}_1\Big(\mathsf{FHE.decode}_0(\mathsf{FHE.sk}, \mathsf{CT}_\sigma) + \textstyle\sum_{i=1}^{N} e'_i\Big) \\
&= \mathsf{FHE.decode}_1\Big(\sigma_M \lfloor q/2 \rceil + e + \textstyle\sum_{i=1}^{N} e'_i\Big) = \sigma_M.
\end{aligned}$$

$\mathsf{TS.KeyGen}(1^\lambda)$: Upon input the security parameter λ, do the following.

1. For each party P_i, sample a PRF key $\mathsf{sprf}_i \leftarrow \mathcal{K}$.
2. Generate the signature scheme's keys $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$.
3. Generate the FHE keys $(\mathsf{FHE.pk}, \mathsf{FHE.sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$ and compute an FHE encryption of $\mathsf{Sig.sk}$ as $\mathsf{CT}_{\mathsf{Sig.sk}} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}, \mathsf{Sig.sk})$.
4. Generate the HS public parameters $\mathsf{HS.pp} \leftarrow \mathsf{HS.PrmsGen}(1^\lambda, 1^n)$ and the public and the signing keys $(\mathsf{HS.pk}, \mathsf{HS.sk}) \leftarrow \mathsf{HS.KeyGen}(1^\lambda, \mathsf{HS.pp})$. Here n is the bit-length of $(\mathsf{FHE.sk}, \mathsf{sprf}_i)$.
5. Share $\mathsf{FHE.sk}$ as $\{\mathsf{sk}_i\}_{i=1}^N \leftarrow \mathsf{Share}(\mathsf{FHE.sk})$ such that $\sum_{i=1}^N \mathsf{sk}_i = \mathsf{FHE.sk}$.
6. For each party P_i, randomly choose a tag $\tau_i \in \{0,1\}^*$ and compute $(\pi_{\tau_i}, \pi_i) \leftarrow \mathsf{HS.Sign}(\mathsf{HS.sk}, (\mathsf{sk}_i, \mathsf{sprf}_i), \tau_i)$.
7. Output $\mathsf{TSig.pp} = \{\mathsf{FHE.pk}, \mathsf{CT}_{\mathsf{Sig.sk}}, \mathsf{HS.pp}, \mathsf{HS.pk}, \{\tau_i, \pi_{\tau_i}\}_{i=1}^N\}$, $\mathsf{TSig.vk} = \mathsf{Sig.vk}$, $\mathsf{TSig.sk} = \{\mathsf{TSig.sk}_i = (\mathsf{sk}_i, \mathsf{sprf}_i, \pi_i)\}_{i=1}^N$.

$\mathsf{TS.PartSign}(\mathsf{TSig.pp}, \mathsf{TSig.sk}_i, M)$: Upon input the public parameters $\mathsf{TSig.pp}$, a partial signing key $\mathsf{TSig.sk}_i$ and a message M, parse $\mathsf{TSig.pp}$ as $(\mathsf{FHE.pk}, \mathsf{CT}_{\mathsf{Sig.sk}}, \mathsf{HS.pp}, \mathsf{HS.pk}, \{\tau_i, \pi_{\tau_i}\}_{i=1}^N\})$ and $\mathsf{TSig.sk}_i$ as $(\mathsf{sk}_i, \mathsf{sprf}_i, \pi_i)$ and do the following.

1. Compute $u = F(\mathsf{sprf}_i, M)$ and sample $e_i' \leftarrow \mathcal{D}_s(u)$, where $\mathcal{D}_s(u)$ represents sampling from $\mathcal{D}_s$ using u as the randomness.
2. Let $\mathcal{C}_M$ be the signing circuit, with message M being hardwired. Compute $\mathsf{CT}_\sigma = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, \mathcal{C}_M, \mathsf{CT}_{\mathsf{Sig.sk}})$.
3. Compute $\sigma_i = \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_\sigma) + e_i'$.
4. This step computes a homomorphic signature $\tilde{\pi}_i$ on partial signature σ_i to provide robustness. Let $\mathcal{C}_{\mathsf{PS}}$ be the circuit to compute $\mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_\sigma) + e_i'$ in which CT_σ is hardcoded and the FHE key share sk_i and the PRF key sprf_i are given as inputs.
   - Compute $\pi_i^* = \mathsf{HS.SignEval}(\mathsf{HS.pp}, \mathcal{C}_{\mathsf{PS}}, \pi_{\tau_i}, (\mathsf{sk}_i, \mathsf{sprf}_i), \pi_i)$.
   - Compute $\tilde{\pi}_i = \mathsf{HS.Hide}(\mathsf{HS.pk}, \sigma_i, \pi_i^*)$.
5. Output $y_i = (\sigma_i, \tilde{\pi}_i)$.

$\mathsf{TS.Combine}(\mathsf{TSig.pp}, \{y_i\}_{i\in[N]})$: Upon input the public parameters $\mathsf{TSig.pp}$ and a set of partial signatures $\{y_i\}_{i\in[N]}$, parse y_i as $(\sigma_i, \tilde{\pi}_i)$ and output $\sigma_M = \mathsf{FHE.decode}_1(\sum_{i=1}^N \sigma_i)$.

$\mathsf{TS.PartSignVerify}(\mathsf{TSig.pp}, M, y_i)$: Upon input the public parameters $\mathsf{TSig.pp}$, message M, and a partial signature y_i, parse y_i as $(\sigma_i, \tilde{\pi}_i)$ and do the following.

1. Compute $\mathsf{CT}_\sigma = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, \mathcal{C}_M, \mathsf{CT}_{\mathsf{Sig.sk}})$.
2. Compute $\alpha = \mathsf{HS.Process}(\mathsf{HS.pp}, \mathcal{C}_{\mathsf{PS}})$, where $\mathcal{C}_{\mathsf{PS}}$ is as described above.
3. Parse y_i as $(\sigma_i, \tilde{\pi}_i)$ and output $\mathsf{HS.HVerify}(\mathsf{HS.pk}, \alpha, \sigma_i, \tau_i, (\pi_{\tau_i}, \tilde{\pi}_i))$.

$\mathsf{TS.Verify}(\mathsf{TSig.vk}, M, \sigma_M)$: Upon input the verification key $\mathsf{TSig.vk}$, a message M and a signature σ_M, output $\mathsf{Sig.Verify}(\mathsf{TSig.vk}, M, \sigma_M)$.

In the above, we set $s = B_{eval} \cdot \sqrt{Q\lambda}$, where $B_{eval} \leq \mathrm{poly}(\lambda)$ is a bound on the FHE decryption noise after homomorphic evaluation of the signing circuit $\mathcal{C}_M$, and Q is the bound on the number of signatures.

Figure 1 Optimization of Boneh et al Threshold Signature Scheme.

3.1.1 Unforgeability

For security, we prove the following theorem.

▶ **Theorem 12.** *Assume F is a secure PRF,* Sig *is UF-CMA secure,* FHE *satisfies semantic security ,* Share *satisfies privacy and* HS *is context hiding. Then the construction of threshold signatures in Figure 1 satisfies selective unforgeablity (Definition 7) if the flooding noise is of the size* $\mathrm{poly}(\lambda) \cdot \sqrt{Q}$*, where Q is the number of the signing queries.*

The security of the construction can be argued using a sequence of hybrids. We assume w.l.o.g. that the adversary $\mathcal{A}$ queries for all but the first key share, i.e., $S = [N] \setminus \{1\}$.

Hybrid_0: This is the real world.

Hybrid_1: Same as Hybrid_0, except that $\tilde{\pi}_1$ in $\mathsf{PartSign}$ is now generated using HS simulator as $\tilde{\pi}_1 = \mathsf{HS.Sim}(\mathsf{HS.sk}, \alpha, \sigma_1, \tau_1, \pi_{\tau_1})$, where $\alpha = \mathsf{HS.Process}(\mathsf{HS.pp}, \mathcal{C}_{\mathsf{PS}})$.

Hybrid_2: Same as Hybrid_1 except that to compute $\sigma_1 = \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_\sigma) + e'_1$, the randomness u used to sample $e'_1 \leftarrow \mathcal{D}_s(u)$ is chosen uniformly randomly instead of computing it using the PRF.

Hybrid_3: Same as Hybrid_2, except that now, for signing query for $(M, 1)$, the challenger simulates σ_1 as follows:

1. Computes $\mathsf{CT}_\sigma = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, \mathcal{C}_M, \mathsf{CT}_{\mathsf{Sig.sk}})$ and $\{\sigma'_i = \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_\sigma)\}_{i\in[2,N]}$.
2. Computes $\sigma_M = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, M)$ and set $\sigma_1 = \sigma_M \lfloor \frac{q}{2} \rceil - \sum_{i=2}^{N} \sigma'_i + e'_1$, where $e'_1 \leftarrow \mathcal{D}_s$.

Hybrid_4: Same as Hybrid_3 except that instead of sharing $\mathsf{FHE.sk}$, now the challenger generates the FHE key shares as $\{\mathsf{sk}_i\}_{i=1}^{N} \leftarrow \mathsf{Share}(\mathbf{0})$.

Hybrid_5: Same as Hybrid_4, except that $\mathsf{CT}_{\mathsf{Sig.sk}}$ in $\mathsf{TSig.pp}$ is replaced by $\mathsf{CT}_{\mathbf{0}} = \mathsf{FHE.Enc}(\mathsf{FHE.pk}, \mathbf{0})$.

Detailed proofs of indistinguishability are provided in the full version [2]. Below, we provide the proof for the main new claim in our work.

▶ **Claim 13.** *If there is an adversary that can win the unforgeability game in* Hybrid_2 *with probability* ϵ*, then its probability of winning the game in* Hybrid_3 *is at least* $\epsilon^2/2$.

Proof. Let the number of signing queries that the adversary makes be Q. The two hybrids differ only in the error term in σ_1, as shown below. In Hybrid_2, we have $\sigma_1 = \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_\sigma) + e'_1$, for $e'_1 \leftarrow \mathcal{D}_s$. In Hybrid_3, we have:

$$\begin{aligned}
\sigma_1 &= \sigma_M . \lfloor q/2 \rceil - \sum_{i=2}^{N} \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_\sigma) + e'_1 \\
&= \sigma_M . \lfloor q/2 \rceil - \sum_{i=1}^{N} \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_\sigma) + \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_\sigma) + e'_1 \\
&= \sigma_M . \lfloor q/2 \rceil - \mathsf{FHE.decode}_0(\sum_{i=1}^{N} \mathsf{sk}_i, \mathsf{CT}_\sigma) + \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_\sigma) + e'_1 \\
&= \sigma_M . \lfloor q/2 \rceil - \mathsf{FHE.decode}_0(\mathsf{sk}, \mathsf{CT}_\sigma) + \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_\sigma) + e'_1 \\
&= \sigma_M . \lfloor q/2 \rceil - \sigma_M . \lfloor q/2 \rceil + e + \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_\sigma) + e'_1 \\
&= \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_\sigma) + (e'_1 + e),
\end{aligned}$$

for some e satisfying $|e| \leq B_{eval}$. Thus, in Hybrid_2, the error term in σ_1 is e'_1, while in Hybrid_3, it is $e'_1 + e$, where, $e'_1 \leftarrow \mathcal{D}_s$, and e is the error in FHE ciphertext CT_σ.

Recall the distribution seen by the adversary – the public parameters $\mathsf{TSig.pp}$, the verification key $\mathsf{TSig.vk}$, the corrupted secret key shares $\mathsf{TSig.sk}_i$, the messages M_j and corresponding partial signatures $(\sigma_j, \tilde{\pi}_j)$. Note that since messages are chosen adaptively, their distribution depends on previous signature queries and responses, and in particular on the differently generated error terms in both hybrids. On the other hand $\mathsf{TSig.pp}$, $\mathsf{TSig.vk}$, $\{\mathsf{TSig.sk}_i\}$, $\{\tilde{\pi}_j\}$ are constructed identically in both hybrids and independently from the rest (in particular these error terms): we implicitly assume that they are fixed and known, and exclude them from the analysis. We refer to the distribution to be considered in Hybrid_2 as D_2 and in Hybrid_3 as D_3.

Let E_j be the random variables corresponding to the error term in CT_{σ_j} in the j-th response and $\mathcal{E}_j^{(2)}$ and $\mathcal{E}_j^{(3)}$ be their distributions in Hybrids 2 and 3, respectively. Similarly, let M_j be the random variable corresponding to the queried message in j-th query and $\mathcal{M}_j^{(2)}$

and $\mathcal{M}_j^{(3)}$ be their distributions in Hybrids 2 and 3, respectively. Then, from the discussion above, we have $\mathcal{E}_j^{(2)} = \mathcal{D}_s$ and $\mathcal{E}_j^{(3)} = \mathcal{D}_{s,e_j}$ for all $j \in [Q]$, where e_j is the error in CT_{σ_j} and can depend upon previous queries and responses.

Overall, we have $D_k = (\mathcal{E}_Q^{(k)}, \mathcal{M}_Q^{(k)}, \mathcal{E}_{Q-1}^{(k)}, \mathcal{M}_{Q-1}^{(k)}, \ldots, \mathcal{E}_1^{(k)}, \mathcal{M}_1^{(k)})$ for $k \in \{2,3\}$ and

$$R_a(D_2\|D_3) = R_a(\mathcal{E}_Q^{(2)}, \mathcal{M}_Q^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{E}_Q^{(3)}, \mathcal{M}_Q^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}). \tag{3.1}$$

Applying the multiplicativity property of the Rényi divergence (Lemma 10), we obtain that $R_a(D_2\|D_3)$ is bounded from above by

$$\begin{aligned} &\max_{x\in X} R_a(\mathcal{E}_Q^{(2)}|X=x \parallel \mathcal{E}_Q^{(3)}|X=x) \cdot R_a(\mathcal{M}_Q^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{M}_Q^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}) \\ = &\max_{x\in X} R_a(\mathcal{D}_s|X=x \parallel \mathcal{D}_{s,e_Q}|X=x) \cdot R_a(\mathcal{M}_Q^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{M}_Q^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}), \end{aligned} \tag{3.2}$$

where $X = (M_Q, E_{Q-1}, \ldots, E_1)$ and e_Q is the error term in CT_{σ_Q}; note that e_Q may depend on the sample from X (which differs in Hybrids 2 and 3) and is bounded by B_{eval}. Then applying Lemma 11 in Equation (3.2), we get

$$\begin{aligned} R_a(D_2\|D_3) &\leq \exp(a\pi\|e_Q\|^2/s^2) \cdot R_a(\mathcal{M}_Q^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{M}_Q^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}) \\ &\leq \exp(a\pi B_{eval}^2/s^2) \cdot R_a(\mathcal{M}_Q^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{M}_Q^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}). \end{aligned}$$

Further, since M_Q is a function of $E_{Q-1}, M_{Q-1}, \ldots, E_1, M_1$, the data processing inequality (Lemma 10) gives

$$\begin{aligned} &R_a(\mathcal{M}_Q^{(2)}, \mathcal{E}_{Q-1}^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{M}_Q^{(3)}, \mathcal{E}_{Q-1}^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}) \\ &\qquad\qquad \leq R_a(\mathcal{E}_{Q-1}^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{E}_{Q-1}^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}), \end{aligned}$$

Hence, we get

$$\begin{aligned} R_a(D_2\|D_3) &\leq \exp(a\pi B_{eval}^2/s^2) \cdot R_a(\mathcal{E}_{Q-1}^{(2)}, \ldots, \mathcal{E}_1^{(2)}, \mathcal{M}_1^{(2)} \parallel \mathcal{E}_{Q-1}^{(3)}, \ldots, \mathcal{E}_1^{(3)}, \mathcal{M}_1^{(3)}) \\ &\leq \exp(a\pi B_{eval}^2 Q/s^2), \end{aligned}$$

where the last inequality follows from induction.

As $s = B_{eval} \cdot \sqrt{Q\lambda}$, we get $R_a(D_2\|D_3) \leq \exp(a\pi/\lambda)$. Therefore, from the probability preservation property of the Rényi divergence (Lemma 10), we have $D_3(\mathsf{E}) \geq \frac{D_2(\mathsf{E})^{\frac{a}{a-1}}}{R_a(D_2\|D_3)} \geq D_2(\mathsf{E})^{\frac{a}{a-1}} \exp(-\frac{a\pi}{\lambda})$. The result is obtained by setting $a = 2$. ◀

3.2 On the Optimality of Our Flooding

We show that the flooding amount that we achieved is optimal for our threshold signature scheme. To argue this, we show how to attack it if the flooding amount is below $\Omega(\sqrt{Q})$. For simplicity, we restrict to the case of $N = 2$. Recall that in our construction, $\mathsf{TS.PartSign}(\mathsf{TSig.pp}, \mathsf{TSig.sk}_i, M)$ outputs $\sigma_{i,M} = \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_{\sigma_M}) + e'_{i,M}$, where $\mathsf{TSig.sk}_i = (\mathsf{sk}_i, \mathsf{sprf}_i)$.[3] W.l.o.g, assume that the adversary gets the partial signing key $\mathsf{TSig.sk}_2$ and the response for any signing query is a partial signature corresponding to party P_1. For any message M of its choice, the adversary receives $\sigma_{1,M} = \mathsf{FHE.decode}_0(\mathsf{sk}_1, \mathsf{CT}_{\sigma_M}) + e'_{1,M}$. From this the adversary can compute:

[3] We focus only on the $\sigma_{i,M}$ component of PartSign's output since the second component, the HS signature of $\sigma_{i,M}$, is not relevant here.

$$\begin{aligned}\sigma_{1,M} + \mathsf{FHE.decode}_0(\mathsf{sk}_2, \mathsf{CT}_{\sigma_M}) &= \mathsf{FHE.decode}_0(\mathsf{FHE.sk}, \mathsf{CT}_{\sigma_M}) + e'_{1,M} \\ &= \sigma_M + \mathsf{err}_M + e'_{1,M},\end{aligned}$$

where err_M is the error in CT_{σ_M}. Note that if the adversary succeeds in computing err_M for polynomially many M's, then it can compute $\mathsf{FHE.sk}$.

As a warm-up, we show that if the error $e'_{1,M}$ is randomized, small and of center 0, then the adversary can indeed compute err_M. Later, we will show that even for deterministic flooding $e'_{1,M}$, there exist secure signature schemes for which the attack can be extended. Since the adversary knows the key share sk_2, it can compute $\sigma_{2,M}$ on its own and hence can compute $\sigma_M = \mathsf{TS.Combine}(\mathsf{TSig.pp}, \sigma_{1,M}, \sigma_{2,M})$. Hence, from $\sigma_M + \mathsf{err}_M + e'_{1,M}$, the adversary can compute $\mathsf{err}_M + e'_{1,M}$. Since, the signature scheme is deterministic, err_M depends only on M. Thus, if the same message is queried for signature multiple times, then each time the term err_M remains the same, but since flooding is randomized, the term $e'_{1,M}$ is different.

To compute err_M, the adversary issues all Q signing queries for the same message M and receives $\sigma^{(1)}_{1,M}, \ldots, \sigma^{(Q)}_{1,M}$, where $\sigma^{(i)}_{1,M}$ denotes the partial signature returned for message M in the ith query. From these responses the adversary gets Q different values of the form

$$w^i = \mathsf{err}_M + e'^i_{1,M} \tag{3.3}$$

Since err_M is same, taking average on both sides of Equation (3.3) over all the Q samples, we get $\frac{\sum_{i\in[Q]} w^i}{Q} = \mathsf{err}_M + \frac{\sum_{i\in[Q]} e'^i_{1,M}}{Q}$. If $|\frac{1}{Q}\sum_{i\in[Q]} e'^i_{1,M}| < 1/2$, then the adversary can recover err_M as $\mathsf{err}_M = \left\lfloor \frac{1}{Q}\sum_{i\in[Q]} w^i \right\rceil$. As $e'^1_{1,M}, \ldots, e'^Q_{1,M}$ are independently and identically distributed with mean 0, by Hoeffding's inequality, we have

$$\Pr\left[\left|\frac{\sum_{i\in[Q]} e'^i_{1,M}}{Q}\right| < 1/2\right] \geq 1 - 2\exp\left(-\frac{Q}{2s^2}\right).$$

If $Q \geq \Omega(s^2 \log\lambda)$, then $1 - 2\exp(-Q/(2s^2)) \geq 1 - \lambda^{-\Omega(1)}$, in which case the adversary can recover err_M with probability sufficiently close to 1 to recover sufficiently many err_M's to compute $\mathsf{FHE.SK}$. To prevent this, we do need s to grow at least proportionally to $\sqrt{Q}$.

3.2.1 Attack for Deterministic Error

In the argument for randomized error, the fact that $e'^i_{1,M}$ is randomized is crucial. However, as discussed in Section 1, we can extend the attack for the case of deterministic flooding as well, by exhibiting a secure signature scheme (with deterministic flooding) for which a variant of the attack can apply.

Consider a special (contrived) signature scheme $\mathsf{Sig}' = (\mathsf{Sig}'.\mathsf{KeyGen}, \mathsf{Sig}'.\mathsf{Sign}, \mathsf{Sig}'.\mathsf{Verify})$ derived from a secure signature scheme $\mathsf{Sig} = (\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ as follows:

1. $\mathsf{Sig}'.\mathsf{KeyGen}$ is identical to $\mathsf{Sig.KeyGen}$. Let $(\mathsf{Sig.sk}, \mathsf{Sig.vk})$ be the signing and verification keys, respectively, and $\mathsf{Sig.sk}_i$ denote the ith bit of $\mathsf{Sig.sk}$ for $i \in [\ell]$, where ℓ is the bit-length of $\mathsf{Sig.sk}$.
2. $\mathsf{Sig}'.\mathsf{Sign}(\mathsf{Sig.sk}, M)$ is modified as follows:
   - Compute $\sigma_M = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, M)$. Set $\sigma'_M := \sigma_M$.
   - For i from 1 to ℓ: if $\mathsf{Sig.sk}_i = 0$, then set $\sigma'_M := \sigma'_M \| \mathsf{Sig.sk}_i$.
   - Output σ'_M.

3. $\mathsf{Sig}'.\mathsf{Verify}(\mathsf{Sig.vk}, M, \sigma'_M)$ is defined as $\mathsf{Sig.Verify}(\mathsf{Sig.vk}, M, \sigma_M)$, where σ_M is obtained from σ'_M by removing all the bits after the kth bit, where k is the bit-length of signatures in Sig.

Above, we assume that the signing key of Sig is a uniform bit-string among those with the same number of 0's and 1's. Since $\mathsf{Sig.sk}$ has always $\ell/2$ bits equal to 0, the number of zeroes appended to the signature will be $\ell/2$ and hence does not leak any extra information to the adversary. Hence, it follows easily that if Sig is a secure signature scheme, then so is Sig'. However, as discussed in Section 1, our attack can be generalized to work for this setting.

3.2.1.1 The Attack

Now, consider using Sig' to instantiate our threshold signature scheme. Then, for any message M, the FHE ciphertext CT_{σ_M} now additionally includes homomorphically evaluated encryptions of $\{\mathsf{Sig.sk}_i\}_{i\in[\ell]:\mathsf{Sig.sk}_i=0}$. Let $\mathsf{CT}_{\sigma_M}, \mathsf{err}_M, e'_M$ respectively denote the encryption of σ_M, the error in CT_{σ_M} and the flooding noise added to partial decryption of CT_{σ_M}. Let $\mathsf{CT}^*, \mathsf{err}^*$ and e^*_M denote the components corresponding to $\{\mathsf{Sig.sk}_i\}_{i\in[\ell]:\mathsf{Sig.sk}_i=0}$.

For any message M, the adversary can compute $\mathsf{err}_M + e'_M$ as described previously, from which the adversary gets $\mathsf{err}^* + e^*_M$. If the adversary manages to compute err^* (for sufficiently many instances), then it can also recover $\mathsf{FHE.sk}$.

Note that err^* is independent of any message and hence is constant across different messages, while e^*_M does depend on M and is different for different messages. This gives an attack strategy. To compute err^*, the adversary issues Q signing queries on different messages $\{M_j\}_{j\in[Q]}$, and from the received partial signatures, derives the values for $w^*_j = \mathsf{err}^* + e^*_{M_j}$ for $j \in [Q]$.

Observe that the above equation is of the same form as Equation (3.3). Heuristically, one would expect the $e^*_{M_j}$ to behave as independent and identically distributed random variables with centre 0. Hence, we can argue in similar way that if $Q \geq \Omega(s^2 \log \lambda)$ then the adversary can recover err^* with probability $1 - 1/\operatorname{poly}(\lambda)$. This implies that for hiding err^*, the standard deviation parameter s must grow at least proportionally to $\sqrt{Q}$.

4 Instantiating Threshold Signatures: Rejection-Free Lyubashevsky

Here, we provide an FHE friendly variant of Lyubashevsky's signature scheme from [47]. Our construction uses a hash function $H : \{0,1\}^* \to D_H := \{\mathbf{v} : \mathbf{v} \in \{-1, 0, 1\}^k; \|\mathbf{v}\|_1 \leq \alpha\}$, modeled as a random oracle. Here α is a parameter, typically much smaller than k. The signature scheme is described in Figure 2.

Correctness. Since $\mathbf{z} = \mathbf{y} + \mathbf{Sc}$, where $\mathbf{y} \leftarrow \mathcal{D}_{\mathbf{Z}^m,\sigma}$, we have $\|\mathbf{z}\| \leq 2\sigma\sqrt{m} + \|\mathbf{Sc}\|$ with probability $1 - 2^{-\Omega(\lambda)}$, using standard Gaussian tail bounds . Since $\|\mathbf{S}\|_\infty \leq d$ and $\|\mathbf{c}\|_1 \leq \alpha$, we have $\|\mathbf{Sc}\| \leq d\alpha\sqrt{m}$. This gives us $\|\mathbf{z}\| \leq (2\sigma + d\alpha)\sqrt{m}$ with overwhelming probability. Finally, note that $H(\mathbf{Az} - \mathbf{Tc}, \mu) = H(\mathbf{A}(\mathbf{y} + \mathbf{Sc}) - \mathbf{ASc}, \mu) = H(\mathbf{Ay}, \mu) = \mathbf{c}$.

Security. We establish security via the following theorem. Because of space constraints, proof of the theorem is given in the full version of the paper [2].

▶ **Theorem 14.** *Assume that $m > \lambda + (n \log q)/\log(2d+1)$, $\sigma \geq \alpha d\sqrt{mQ}$ where Q is the maximum number of signing queries an attacker can make and $|D_H| \geq 2^\lambda$. Assume further that $\mathsf{SIS}_{q,n,m,\beta}$ is hard for $\beta = 2\gamma + 2d\alpha\sqrt{m}$. Then the construction in Figure 2 satisfies* UF-CMA *in the random oracle model.*

$\mathsf{KeyGen}(1^\lambda)$: Upon input the security parameter λ, set $q, n, m, \beta, k, d, \sigma$ such that $n = \Omega(\lambda)$ and the scheme is secure (see Theorem 14); then do the following:

1. Sample $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{S} \leftarrow \{-d, \ldots, 0, \ldots, d\}^{m \times k}$.
2. Set $\mathbf{T} = \mathbf{AS}$.
3. Output $\mathsf{vk} = (\mathbf{A}, \mathbf{T})$, $\mathsf{sk} = \mathbf{S}$.

$\mathsf{Sign}(\mathsf{sk}, \mu)$: Upon input the signing key sk and a message μ, do the following:

1. Sample $\mathbf{y} \leftarrow \mathcal{D}_{\mathbf{Z}^m, \sigma}$.
2. Set $\mathbf{c} = H(\mathbf{Ay}, \mu)$.
3. Set $\mathbf{z} = \mathbf{y} + \mathbf{Sc}$.
4. Output $(\mathbf{z}, \mathbf{c})$.

$\mathsf{Verify}(\mathsf{vk}, \mu, (\mathbf{z}, \mathbf{c}))$: Upon input the verification key vk, a message μ, and a signature $(\mathbf{z}, \mathbf{c})$, do the following:

1. Check if $\|\mathbf{z}\| \leq \gamma$, where $\gamma = (2\sigma + \alpha d)\sqrt{m}$.
2. Check if $H(\mathbf{Az} - \mathbf{Tc}, \mu) = \mathbf{c}$.
3. If both checks pass, then accept, else reject.

Figure 2 Lyubashevsky's Signature Without Aborts.

5 Adaptive Security for Threshold Signatures

As discussed in Section 1, we provide two constructions to improve the selective security achieved by [9]. Below, we describe our construction in the ROM, which satisfies partially adaptive unforgeability (Definition 6). Due to space constraints, we provide our construction in the standard model with *pre-processing* that satisfies fully adaptive unforgeability (Definition 5) in the full version.

5.1 Partially Adaptive Unforgeability

We use the same building blocks as in Section 3.1 for construction. We also use two keyed hash function modelled as random oracles: $H : \{0,1\}^\lambda \times \{0,1\}^* \to \mathbb{Z}_q^N$ and $H_1 : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^r$. The construction is provided in Figure 3.

Correctness and Unforgeability. The proof of correctness is similar to that in Section 3.1 and is provided in the full version of the paper [2]. We prove partially adaptive unforgeability via the following theorem.

▶ **Theorem 15.** *Assume* Sig *satisfies unforgeability,* FHE *is semantically secure ,* HS *is context hiding and* Share *satisfies privacy. Then the* TS *construction in Figure 3 satisfies partially adaptive unforgeability (Definition 6) in ROM if the flooding error is of size* $\mathrm{poly}(\lambda)\sqrt{Q}$*, where* Q *is the upper bound on the number of signing queries.*

Proof. The security of the construction can be argued using the following hybrids:

Hybrid_0 and Hybrid_1 are the same as that in the proof of Theorem 12.

Hybrid_2: Same as Hybrid_1, except that the randomness u_i used in sampling e'_i in $\sigma_{i,M}$ is chosen uniformly randomly from $\{0,1\}^r$ and then H_1 is programmed as $H_1(\mathsf{hkey}_i, M) = u_i$. For random oracle queries for hash H_1 by the adversary $\mathcal{A}$ on an input x, the challenger first checks if $H_1(x)$ is already set. If so, then returns that value else chooses a value uniformly randomly from $\{0,1\}^r$ and saves and returns it.

$\mathsf{TS.KeyGen}(1^\lambda)$: Upon input the security parameter λ, do the following:

1. Randomly choose $K \leftarrow \{0,1\}^\lambda$ and N vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N \in \mathbb{Z}_q^N$ such that $\sum_{i=1}^N \mathbf{v}_i = \mathbf{0}$.
2. Generate $(\mathsf{Sig.vk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ and $(\mathsf{FHE.pk}, \mathsf{FHE.sk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$ and share $\mathsf{FHE.sk}$ into N shares as $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{Share}(\mathsf{FHE.sk})$ such that $\sum_{i=1}^N \mathsf{sk}_i = \mathsf{FHE.sk}$.
3. Compute an FHE encryption of the signing key as $\mathsf{CT}_{\mathsf{Sig.sk}} = \mathsf{FHE.Enc}(\mathsf{FHE.pk}, \mathsf{Sig.sk})$.
4. For each party P_i, randomly choose a tag $\tau_i \in \{0,1\}^*$, a hash key $\mathsf{hkey}_i \leftarrow \{0,1\}^\lambda$ and generate HS public parameters $\mathsf{HS.pp} \leftarrow \mathsf{HS.PrmsGen}(1^\lambda, 1^n)$ and HS public and signing keys as $(\mathsf{HS.pk}, \mathsf{HS.sk}) \leftarrow \mathsf{HS.KeyGen}(1^\lambda, \mathsf{HS.pp})$. Here, n is the length of input to $\mathsf{PartSign}$ circuit which depends on $(\mathsf{FHE.sk}, K, \mathbf{v}_i, \mathsf{hkey}_i)$.
5. Compute $(\pi_{\tau_i}, \pi_i) = \mathsf{HS.Sign}(\mathsf{HS.sk}, (\mathsf{sk}_i, K, \mathbf{v}_i, \mathsf{hkey}_i), \tau_i)$.
6. Output $\mathsf{TSig.pp} = (\mathsf{FHE.pk}, \mathsf{HS.pp}, \mathsf{HS.pk}, \mathsf{CT}_{\mathsf{Sig.sk}}, \{\tau_i, \pi_{\tau_i}\}_{i=1}^N)$, $\mathsf{TSig.vk} = \mathsf{Sig.vk}$, $\mathsf{TSig.sk} = \{\mathsf{TSig.sk}_i = (\mathsf{sk}_i, K, \mathbf{v}_i, \mathsf{hkey}_i, \pi_i)\}_{i=1}^N$.

$\mathsf{TS.PartSign}(\mathsf{TSig.pp}, \mathsf{TSig.sk}_i, M)$: Upon input the public parameters $\mathsf{TSig.pp}$, the key share $\mathsf{TSig.sk}_i = (\mathsf{sk}_i, K, \mathbf{v}_i, \mathsf{hkey}_i, \pi_i)$ and a message M, do the following:

1. Compute $u_i = H_1(\mathsf{hkey}_i, M)$ and sample $e_i' \leftarrow \mathcal{D}_s(u_i)$.
2. Let $\mathcal{C}_M$ be the signing circuit, with message M being hardcoded. Compute an FHE encryption of signature σ_M as $\mathsf{CT}_{\sigma_M} = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, \mathcal{C}_M, \mathsf{CT}_{\mathsf{Sig.sk}})$.
3. Compute $r_{i,M} = H(K, M)^T \mathbf{v}_i$ and $\sigma_{i,M} = \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_{\sigma_M}) + r_{i,M} + e_i'$.
4. This step computes a homomorphic signature $\tilde{\pi}_{i,M}$ on $\sigma_{i,M}$ to provide robustness. Let $\mathcal{C}_{\mathsf{PS}}$ be the circuit to compute $\mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_{\sigma_M}) + H(K, M)^T \mathbf{v}_i + e_i'$ in which CT_{σ_M} is hardcoded and the key share $\mathsf{TSig.sk}_i$ is given as the input. Compute $\pi^*_{i,M} = \mathsf{HS.SignEval}(\mathsf{HS.pp}, \mathcal{C}_{\mathsf{PS}}, \pi_{\tau_i}, (\mathsf{sk}_i, K, \mathbf{v}_i, \mathsf{hkey}_i), \pi_i)$ and $\tilde{\pi}_{i,M} = \mathsf{HS.Hide}(\mathsf{HS.pk}, \sigma_{i,M}, \pi^*_{i,M})$.
5. Output $y_{i,M} = (\sigma_{i,M}, \tilde{\pi}_{i,M})$.

Algorithms $\mathsf{TS.PartSignVerify}$, $\mathsf{TS.Combine}$ and $\mathsf{TS.Verify}$ are identical to those in Section 3.1.

Figure 3 Partially Adaptive Threshold Signature Scheme.

Hybrid_3: Same as Hybrid_2 except that the value of $H(K, M)$ for each M in pre corruption signing query is set in the reverse order, i.e., firstly partial signatures are computed and then $H(K, M)$ is set accordingly as follows:

1. The challenger computes $\mathsf{CT}_{\sigma_M} = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, \mathcal{C}_M, \mathsf{CT}_{\mathsf{Sig.sk}})$.
2. It then computes $\mathsf{FHE.decode}_0(\mathsf{FHE.sk}, \mathsf{CT}_{\sigma_M})$ and generates N shares as $\{s_{i,M}\}_{i=1}^N \leftarrow \mathsf{Share}(\mathsf{FHE.decode}_0(\mathsf{FHE.sk}, \mathsf{CT}_{\sigma_M}))$.
3. Returns partial signatures as $\{\sigma_{i,M} = s_{i,M} + e_i'\}_{i=1}^N$. Also, if a message M is repeated for signing query, then the challenger uses same $\{s_{i,M}\}_{i=1}^N$ shares of $\mathsf{FHE.decode}_0(\mathsf{FHE.sk}, \mathsf{CT}_{\sigma_M})$ again.
4. When the adversary $\mathcal{A}$ outputs the set S of corrupted parties, the challenger first programs the value of $H(K, M)$ for each M in pre corruption signing queries as described next, and then provides key shares for $i \in S$ to $\mathcal{A}$.
   - Programming $H(K, M)$: $\forall i \in [N]$, compute $r_{i,M} = s_{i,M} - \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_{\sigma_M})$ and solve for vector $\mathbf{b}_M \in \mathbb{Z}_q^N$ such that $\forall\ i \in [N], \mathbf{b}_M^T \mathbf{v}_i = r_{i,M}$. Set $H(K, M) = \mathbf{b}_M$. Note that since there are $N-1$ independent equations in N unknowns, such a $\mathbf{b}_M$ exists and can be computed.
5. To answer a random oracle query for hash function H on input x, the challenger first checks if the value is already set, if so then returns that value, else randomly samples a fresh vector $\mathbf{r}_x$ and sets and returns $H(x) = \mathbf{r}_x$.

Hybrid_4: Same as Hybrid_3, except that now the signing queries are answered differently. For each pre-corruption signing query for a message M, $\sigma_{i,M}$ is computed as follows:

1. The challenger computes $\sigma_M = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, M)$ and generates random shares of $\sigma_M \lfloor q/2 \rceil$ as $\{s_{i,M}\}_{i=1}^N \leftarrow \mathsf{Share}(\sigma_M \lfloor q/2 \rceil)$ such that $\sum_{i=1}^N s_{i,M} = \sigma_M \lfloor q/2 \rceil$.
2. Returns $\sigma_{i,M} = s_{i,M} + e'_i$, where $e'_i \leftarrow \mathcal{D}_s$

When $\mathcal{A}$ outputs the set S of corrupted parties, the challenger does the following:

1. Let $PreQ$ be the set of messages for which signing queries were made before. Then for each $M \in PreQ$ it does the following. For each $i \in S$, computes $r_{i,M} = s_{i,M} - \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_{\sigma_M})$. Computes $\mathbf{b}_M$ such that $\forall\ i \in S,\ \mathbf{b}_M^T \mathbf{v}_i = r_{i,M}$. Sets $H(K, M) = \mathbf{b}_M$. Such a $\mathbf{b}_M$ exists and can be computed since there are only $N-1$ equations to satisfy in N unknowns.
2. Returns the secret key shares $\{\mathsf{TSig.sk}_i\}_{i \in S}$.

For each post corruption signing query on message M, the challenger does the following. Let the honest party be P_a, i.e. $S = [N] \setminus \{a\}$.

1. Computes $\mathsf{CT}_{\sigma_M} = \mathsf{FHE.Eval}(\mathsf{FHE.pk}, \mathcal{C}_M, \mathsf{CT}_{\mathsf{Sig.sk}})$ and $\sigma_M = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, M)$.
2. For each $i \in S$, computes $\sigma'_{i,M} = \mathsf{FHE.decode}_0(\mathsf{sk}_i, \mathsf{CT}_{\sigma_M}) + H(K, M)^T \mathbf{v}_i$ and $\sigma_{i,M} = \sigma'_{i,M} + e'_i$, where $e'_i \leftarrow \mathcal{D}_s$.
3. Returns $\sigma_{a,M} = \sigma_M \lfloor q/2 \rceil - \sum_{i \in S} \sigma'_{i,M} + e'_a$, where $e'_a \leftarrow \mathcal{D}_s$.

Hybrid_5 and Hybrid_6: are the same as Hybrid_4 and Hybrid_5, respectively, defined in the proof of Theorem 12.

In the full version of the paper we show that consecutive hybrids are indistinguishable and that the probability of the adversary winning the unforgeability game (Definition 6) is negligible in Hybrid_6. ◀

References

1 Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf, 2017.

2 Shweta Agrawal, Damien Stehlé, and Anshu Yadav. Round-optimal lattice-based threshold signatures, revisited. Cryptology eprint Archive, 2022.

3 Martin R Albrecht and Amit Deo. Large modulus ring-LWE ≥ module-LWE. In *ASIACRYPT*, 2017.

4 Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange—a new hope. In *USENIX Security*, 2016.

5 Shi Bai, Tancrède Lepoint, Adeline Roux-Langlois, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: using the Rényi divergence rather than the statistical distance. *J. Cryptol.*, 2018.

6 Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *TCC*, 2010.

7 Rikke Bendlin, Sara Krehbiel, and Chris Peikert. How to share a lattice trapdoor: threshold protocols for signatures and (H) IBE. In *ACNS*, 2013.

8 Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In *TCC*, 2016.

9 Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, 2018.

10 Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *ACM CCS*, 2016.

11 Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

12 Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, 2011.

13 Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *CCS*, 2020.

14 Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO*, 2019.

15 Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC*, 2020.

16 Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT*, 2018.

17 Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.

18 Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 2020.

19 Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: threshold post-quantum signatures. In *IMACC*, 2019.

20 Anders Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS*, 2020.

21 Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergård. Fast threshold ECDSA with honest majority. In *SCN*, 2020.

22 Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In *PKC*, 2021.

23 Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 1994.

24 Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *S& P*, 2018.

25 Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *S&P*, 2019.

26 Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT*, 2015.

27 Léo Ducas and Thomas Prest. Fast Fourier orthogonalization. In *ISSAC*, 2016.

28 Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. Specification v1.0, available at `https://falcon-sign.info/`.

29 Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In *ASIACRYPT*, 2015.

30 Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. Threshold ECDSA for decentralized asset custody. *Cryptology ePrint Archive*, 2020.

31 Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *CCS*, 2018.

32 Rosario Gennaro and Steven Goldfeder. One round threshold ECDSA with identifiable abort. *IACR Cryptol. ePrint Arch.*, 2020.

33 Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *ACNS*, 2016.

34 Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. URL: `crypto.stanford.edu/craig`.

35 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

36 Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

37 Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *ICS*, 2010.

38 Gottfried Herold, Elena Kirshanova, and Alexander May. On the asymptotic complexity of solving LWE. *Des. Codes Cryptogr.*, 2018.

39 James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In *PQCrypto*, 2020.

40 Andreas Hülsing, Tanja Lange, and Kit Smeets. Rounded gaussians – fast and secure constant-time sampling for lattice-based crypto. In *PKC*, 2018.

41 Kamil Kluczniak and Leonard Schild. Fdfb: Full domain functional bootstrapping towards practical fully homomorphic encryption. *arXiv preprint*, 2021. `arXiv:2109.02731`.

42 Adeline Langlois, Damien Stehlé, and Ron Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In *EUROCRYPT*, 2014.

43 Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO*, 2017.

44 Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, 2018.

45 San Ling, Duong Hieu Phan, Damien Stehlé, and Ron Steinfeld. Hardness of k-LWE and applications in traitor tracing. In *CRYPTO*, 2014.

46 Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, 2009.

47 Vadim Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, 2012.

48 Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP*, 2006.

49 Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.

50 Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, 2006.

51 Thomas Pornin and Thomas Prest. More efficient algorithms for the NTRU key generation using the field norm. In *PKC*, 2019.

52 Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In *ASIACRYPT*, 2017.

53 Mélissa Rossi. *Extended Security of Lattice-Based Cryptography.* PhD thesis, Université PSL, 2020. URL: `https://www.di.ens.fr/~mrossi/docs/thesis.pdf`.

54 Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, 2009.

55 Katsuyuki Takashima and Atsushi Takayasu. Tighter security for efficient lattice cryptography via the Rényi divergence of optimized orders. In *ProvSec*, 2015.

56 Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: compact and scalable arbitrary-centered discrete Gaussian sampling over integers. In *PQCrypto*, 2020.

57 Ruiyu Zhu, Changchang Ding, and Yan Huang. Practical MPC+ FHE with applications in secure multi-partyneural network evaluation. *IACR Cryptol. ePrint Arch.*, 2020.

Parameterized Sensitivity Oracles and Dynamic Algorithms Using Exterior Algebras

Josh Alman
Department of Computer Science, Columbia University, New York, NY, USA

Dean Hirsch
Department of Computer Science, Columbia University, New York, NY, USA

Abstract

We design the first efficient sensitivity oracles and dynamic algorithms for a variety of parameterized problems. Our main approach is to modify the algebraic coding technique from static parameterized algorithm design, which had not previously been used in a dynamic context. We particularly build off of the "extensor coding" method of Brand, Dell and Husfeldt [STOC'18], employing properties of the exterior algebra over different fields.

For the k-Path detection problem for directed graphs, it is known that no efficient dynamic algorithm exists (under popular assumptions from fine-grained complexity). We circumvent this by designing an efficient sensitivity oracle, which preprocesses a directed graph on n vertices in $2^k \operatorname{poly}(k) n^{\omega+o(1)}$ time, such that, given ℓ updates (mixing edge insertions and deletions, and vertex deletions) to that input graph, it can decide in time $\ell^2 2^k \operatorname{poly}(k)$ and with high probability, whether the updated graph contains a path of length k. We also give a deterministic sensitivity oracle requiring $4^k \operatorname{poly}(k) n^{\omega+o(1)}$ preprocessing time and $\ell^2 2^{\omega k+o(k)}$ query time, and obtain a randomized sensitivity oracle for the task of approximately counting the number of k-paths. For k-Path detection in undirected graphs, we obtain a randomized sensitivity oracle with $O(1.66^k n^3)$ preprocessing time and $O(\ell^3 1.66^k)$ query time, and a better bound for undirected bipartite graphs.

In addition, we present the first fully dynamic algorithms for a variety of problems: k-Partial Cover, m-Set k-Packing, t-Dominating Set, d-Dimensional k-Matching, and Exact k-Partial Cover. For example, for k-Partial Cover we show a randomized dynamic algorithm with $2^k \operatorname{poly}(k) \operatorname{polylog}(n)$ update time, and a deterministic dynamic algorithm with $4^k \operatorname{poly}(k) \operatorname{polylog}(n)$ update time. Finally, we show how our techniques can be adapted to deal with natural variants on these problems where additional constraints are imposed on the solutions.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Fixed parameter tractability; Mathematics of computing → Paths and connectivity problems

Keywords and phrases sensitivity oracles, k-path, dynamic algorithms, parameterized algorithms, set packing, partial cover, exterior algebra, extensor, algebraic algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.9

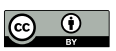
Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2204.10819`

Funding This research was supported in part by a grant from the Simons Foundation (Grant Number 825870 JA), by NSF grant CCF-2107187, by a grant from the Columbia-IBM center for Blockchain and Data Transparency, by JPMorgan Chase & Co., and by LexisNexis Risk Solutions. Any views or opinions expressed herein are solely those of the authors listed.

1 Introduction

The area of dynamic algorithms studies how to quickly and efficiently solve computational problems when the input data is changing. For example, if P is a property of a graph, then a dynamic graph algorithm for P is a data structure which maintains an n-node graph G,



and can handle updates which insert or remove an edge of G, and queries which ask whether G currently satisfies P. Efficient dynamic algorithms, which handle updates and queries in $n^{o(1)}$ time, have been designed for many important problems, and they are used in many applications, both as ways to analyze evolving data, and as subroutines of larger algorithms which need to iterate over and check many similar possibilities. See, for instance, the recent survey [16].

However, there are many prominent dynamic problems which would have many applications, but for which we do not have efficient algorithms. Often times, we even have conditional lower bounds from fine-grained complexity, showing that efficient dynamic algorithms for these problems are unlikely to exist (see e.g. [1, 17, 23] and [16, Section 2.1]). It has recently become popular to circumvent such lower bounds by instead designing a *sensitivity oracle* for the problem, a weaker notion which can still be used in many applications.

Let ℓ be a positive integer. A sensitivity oracle for a dynamic problem, with sensitivity ℓ, preprocesses an initial input, and must answer queries where $\leq \ell$ changes are made to the *initial* input. For example, if P is a property of a graph, then a graph algorithm for P with sensitivity ℓ is a data structure which preprocesses an initial graph G, and can handle queries where ℓ edges are updated (inserted or removed) in the initial graph G, and asks whether P is still satisfied. One can imagine "resetting" G back to its original state after each query[1].

In this paper, we study dynamic algorithms and sensitivity oracles for *parameterized* problems. Consider, for instance, the k-Path problem: given a positive integer k, in an n-node graph G (directed or undirected), determine whether there is a path of length k. This problem is NP-complete, so we should not hope for a dynamic algorithm with update time $n^{o(1)}$ (such a dynamic algorithm could be used to solve the static problem in $n^{2+o(1)}$ time!). However, k-Path is known to be fixed-parameter tractable (FPT), and can be solved in time $2^{O(k)} \cdot n^2$ [8, 26, 25], which is sufficiently efficient when k is small. We can thus hope for dynamic parameterized algorithms for the problem, with update time $f(k) \cdot n^{o(1)}$. And indeed, a recent line of work has designed efficient dynamic parameterized algorithms with such a running time for many different problems, typically by using dynamic variants on classic techniques from the parameterized algorithms literature like kernelization and color coding. For the k-Path problem in *undirected* graphs, such an algorithm is known with update time $k! \cdot 2^{O(k)} \cdot \text{polylog}(n)$ [2], and another with amortized update time $2^{O(k^2)}$ [11].

By contrast, no efficient dynamic parameterized algorithm for k-Path in *directed* graphs is known. Moreover, Alman, Mnich and Vassilevska [2] proved a conditional lower bound, that it does not have such an efficient dynamic parameterized algorithm assuming any one of three popular conjectures from fine-grained complexity theory (the 3SUM conjecture, the Triangle conjecture, and a "Layered Reachability Oracle" conjecture they introduce, which concerns a special case of a more popular hypothesis about reachability oracles).

This leads naturally to the two main questions we address in this paper. The first asks whether there is an analogue of the aforementioned line of work on sensitivity oracles for problems without efficient dynamic algorithms in the parameterized setting.

▶ **Question 1.** *Is there an efficient parameterized sensitivity oracle for k-Path in directed graphs?*

[1] Sensitivity oracles are sometimes referred to as "fault-tolerant" or "emergency planning" algorithms in the literature. For graph problems, these terms also sometimes refer to the decrement-only case (where edge updates only remove edges), but following [18, Section A.1], we use "sensitivity oracle" to refer to the fully dynamic case, where edges can be inserted and deleted.

We say that a sensitivity oracle for a parameterized problem, with parameter k, is *efficient* if its preprocessing time is $f(k) \cdot \text{poly}(n)$ for some computable function f, and its query time is $\text{poly}(\ell) \cdot g(k) \cdot n^{o(1)}$ for some computable function g (where ℓ is the sensitivity parameter, i.e., the number of updates allowed per query). In the case of k-Path in directed graphs, and other graph problems, we specifically seek such an efficient sensitivity oracle in the *fully dynamic* setting where queries can change any ℓ edges by inserting and deleting them.

It is natural to ask that the query time has a *polynomial* dependence on ℓ (rather than, say, just a $f(\ell)$ dependence), as we do here, for two reasons. First, this is the dependence one would get by converting an efficient dynamic algorithm into a sensitivity oracle. Second, with our definition, any parameterized problem with an efficient sensitivity oracle is in FPT via the algorithm where the sensitivity oracle preprocesses an empty graph and then gets the full input graph as a query (but this would not be true if an arbitrary $f(\ell)$ term were allowed in the query time). Along the way to answering Question 1, we will also address more precisely the relationship between the classes of parameterized problems with efficient dynamic algorithms, efficient sensitivity oracles, and efficient static algorithms (a.k.a. the class FPT).

To our knowledge, such a *fully dynamic* notion of sensitivity oracles for parameterized problems has not been previously studied. The closest prior work is very recent [6] which considered a similar but *only decremental* setting, wherein queries may only delete edges from the graph, and not insert new edges. They design very elegant decremental sensitivity oracles for directed k-Path and for k-Vertex Cover, but their preprocessing and query times have exponential dependence on ℓ and hence are not "efficient" as we defined above. We also give evidence that the techniques of [6] cannot extend to the fully-dynamic setting; see Section 1.2 below for more details.

The second question we address is inspired by prior work on static algorithms for k-Path. Many fundamental techniques in the literature on parameterized algorithms were first introduced to study the k-Path problem. One such technique, algebraic coding (sometimes called "monomial testing" or "multilinear monomial detection"), is used in the current fastest static randomized algorithms for k-Path, and has also been used in other applications in algebraic complexity theory [22, 25, 10, 9, 8, 21]. Nonetheless, to our knowledge, these techniques have not been used in a dynamic or sensitivity setting before.

▶ **Question 2.** *Can algebraic coding techniques from the design of parameterized algorithms be used to design efficient dynamic algorithms or sensitivity oracles?*

A positive answer to Question 2 could lead to efficient dynamic algorithms or sensitivity oracles for a host of parameterized problems.

1.1 Our results

Let $\omega < 2.373$ be such that we can multiply two $n \times n$ matrices in $O(n^\omega)$ arithmetic operations [3]. Our first main result gives a positive answer to Question 1.

▶ **Theorem 3.** *The k-Path problem in directed graphs has an efficient parameterized sensitivity oracle. It can be solved with*[2]*:*

- *a Monte Carlo randomized algorithm with preprocessing time $2^k \,\text{poly}(k) n^\omega$ and query time $\ell^2 2^k \,\text{poly}(k)$, or*

[2] We work in the word-RAM model of computation with w-bit words for $w = O(\log n)$. Hence, only $O(\ell)$ words are needed to specify the ℓ edges to change in a query, and we can achieve query times independent of n.

- *a deterministic algorithm with preprocessing time $4^k \operatorname{poly}(k) n^\omega$ and query time $\ell^2 2^{\omega k}$.*

In addition to edge insertion and deletions, these algorithms also allow for vertex failures as part of the ℓ updates per query.

Although k-Path is known to not have an efficient dynamic parameterized algorithm (assuming the aforementioned 3SUM, Triangle, or Layered Reachability Oracle hardness assumptions from fine-grained complexity), Theorem 3 shows it does have an efficient parameterized sensitivity oracle. Since the reductions used in [2] are still valid in the sensitivity setting, one corollary is that the sensitivity versions of the 3SUM, Triangle, and Layered Reachability problems have efficient algorithms (although there are simple algorithms showing this for 3SUM and Triangle which do not go through k-Path; see Section 4 for more details).

It follows from prior work [18] that assuming another popular conjecture, the Strong Exponential Time Hypothesis (SETH), there are problems in FPT which do not have efficient parameterized sensitivity oracles (one example is the counting version of the single-source reachability problem; see again Section 4 for more details). Hence, assuming popular conjectures from fine-grained complexity, it follows that the class of parameterized problems with an efficient sensitivity oracle lies *strictly between* the class of parameterized problems with an efficient dynamic algorithm, and the class FPT (i.e., both class inclusions are strict).

Our sensitivity oracle for Theorem 3 uses $\Theta(n^2)$ space (for constant k), and it is natural to wonder whether this can be improved, especially since known dynamic parameterized algorithms for many other problems use much smaller space (e.g., many which dynamically maintain a small kernel [2]). However, we prove unconditionally that the space usage of our algorithm cannot be improved:

▶ **Theorem 4.** *Any randomized or deterministic sensitivity oracle for k-Path in directed graphs which handles edge insertion queries must use $\Omega(n^2)$ space.*

We also give three additional algorithmic results to complement Theorem 3. First, we extend Theorem 3 to give an algorithm for approximately counting k-paths:

▶ **Theorem 5.** *There is a randomized efficient parameterized sensitivity oracle which approximately counts the number of k-paths in an n-node, m-edge directed graph. For any $\epsilon > 0$, it produces an estimate to the number of k-paths in the graph that, with probability $> 99\%$, is within ϵ relative error, with preprocessing time $\epsilon^{-2} \cdot 4^k \operatorname{poly}(k) \cdot \min\{mn, n^\omega\}$ and update time $O\left(\epsilon^{-2} \cdot \ell^2 \cdot 2^{\omega k}\right)$. In addition to edge insertion and deletions, it also allows for vertex failures as part of the ℓ updates per query.*

Second, we present a randomized sensitivity oracle with a better dependence on k, at the cost of a worse dependence on ℓ, but only for undirected graphs:

▶ **Theorem 6** (Undirected graphs)**.** *For the k-Path detection problem on an undirected graph G on n vertices, there exists a randomized sensitivity oracle with preprocessing time $O(1.66^k n^3)$ and query time $O(\ell^3 1.66^k)$.*

Third, we obtain the following corollary for bipartite graphs:

▶ **Corollary 7** (Undirected bipartite graphs)**.** *For the k-Path detection problem on an undirected bipartite graph, where the partition of the vertices V into the two sides $V = S \cup T$ is known in advance and holds after any updates, there exists a sensitivity oracle with $2^{k/2} \operatorname{poly}(k) n^3$ preprocessing time and $\ell^3 2^{k/2} \operatorname{poly}(k)$ query time.*

Our algorithm for Theorem 3 uses a new dynamic version of the algebraic coding technique, and more specifically, a recent implementation called "extensor coding" by Brand, Dell, and Husfeldt [10]. See Section 2.5 and 3 below for more background on this technique, and the new ideas we introduce to be able to use it in a dynamic or sensitivity setting. We are then able to use and further modify our approach to design efficient dynamic parameterized algorithms for many problems for which no such algorithm was previously known, positively answering Question 2. We present the definitions for the problems we consider, followed by the results.

▶ **Definition 8** (k-Partial Cover). *Given a collection of subsets $S_1, ..., S_n \subseteq [N]$, find the minimum size of a sub-collection T of these, for which $\left|\bigcup_{S\in T} S\right| \geq k$, or declare that no such T exists.*

▶ **Definition 9** (m-Set k-Packing). *Given subsets $S_1, ..., S_n \subseteq [N]$, all with size $|S_i| = m$, decide whether there exists a sub-collection of k sets, which are all pairwise disjoint.*

▶ **Definition 10** (t-Dominating Set). *Given an undirected graph G on n vertices, find the minimum size of a set of vertices $S \subseteq V(G)$ such that $|S \cup N(S)| \geq t$ where $N(S)$ is the set of neighbors of vertices in S, i.e., $N(S) = \bigcup_{v\in S} N(v)$.*

▶ **Definition 11** (d-Dimensional k-Matching). *Fixing a universe $U = U_1 \times U_2 \times \ldots \times U_d$, where U_i are pairwise disjoint of combined size $|\biguplus_i U_i| = N$, and given a collection of tuples $T \subseteq U$, decide whether T contains a sub-collection of k pairwise disjoint tuples. (We say the tuples $a, b \in U$ are disjoint if $a[i] \neq b[i]$ for all $i = 1, 2, \ldots, d$.)*

▶ **Definition 12** (Exact k-Partial Cover). *Given a collection of subsets $S_1, ..., S_n \subseteq [N]$, decide whether there is a sub-collection T of these, in which all sets are pairwise-disjoint, and $\left|\biguplus_{S\in T} S\right| = k$.*

▶ **Theorem 13.** *There are efficient dynamic parameterized algorithms for the following problems. We write O^* here to hide factors that are polynomial in the parameters (m, k, t) and polylogarithmic in the size of the instance (n, N).*

- *For k-Partial Cover there are dynamic algorithms, with either randomized $O^*(2^k)$ or deterministic $O^*(4^k)$ update time.*
- *For m-Set k-Packing there are dynamic algorithms, with either randomized $O^*(2^{mk})$ or deterministic $O^*(4^{mk})$ update time.*
- *For t-Dominating Set there are dynamic algorithms, with either randomized $O^*(2^t)$ or deterministic $O^*(4^t)$ update time.*
- *For d-Dimensional k-Matching there are dynamic algorithms, with either randomized $O^*(2^{(d-1)k})$ or deterministic $O^*(4^{(d-1)k})$ update time.*
- *For Exact k-Partial Cover there are dynamic algorithms, with either randomized $O^*(2^k)$ or deterministic $O^*(4^k)$ update time.*

For the proof of Theorem 13 we refer the reader to the full version.

We also explain how the results can be extended to problems with additional constraints. As a simple example, we show how we can design an efficient sensitivity oracle for the problem of detecting whether a directed graph contains a walk of length k which visits at least $k-1$ vertices. As a second example, we discuss a different problem: given a directed graph G on n vertices, two (possibly intersecting) subsets $V_1, V_2 \subseteq V$, and two positive integers μ_1, μ_2,

decide whether G contains a k-path that contains at most μ_1 vertices of V_1 and at most μ_2 vertices of V_2. For this problem we give a static deterministic algorithm running in time $4^{k+\min\{k,|V_1\cap V_2|\}}\operatorname{poly}(k)\cdot n^2$, as well as a sensitivity oracle counterpart. We refer the reader to the full version for the full discussion.

1.2 Comparison with related work

Algorithms for k-Path

The static k-Path problem has a long history. The current best upper bounds for it are a randomized $1.66^k\operatorname{poly}(n)$ algorithm in undirected graphs due to Björklund, Husfeldt, Kaski and Koivisto [8], a randomized time $2^k\operatorname{poly}(n)$ in directed (and hence also undirected) graphs due to Williams [26], and deterministic time $2.554^k\operatorname{poly}(n)$ in directed graphs due to Tsur [25].

In comparison, our randomized sensitivity oracle for directed graphs in Theorem 3 has a dependency on k (both in preprocessing and query time) of $O^*(2^k)$, so one cannot hope to improve on it without improving the best static algorithm. For undirected graphs, we give in Theorem 6 a sensitivity oracle with a dependency of $O^*(1.66^k)$, which matches the best bound of [8]. Our deterministic sensitivity oracle has a dependency of $O^*(4^k)$; we leave open the question of whether the techniques of [25] can be used in a sensitivity oracle setting to achieve $O^*(2.554^k)$.

There has also been work on the dynamic version of k-Path. However, Alman, Mnich and Vassilevska [2] showed that under the aforementioned fine-grained conjectures, there is no dynamic algorithm for *directed* graphs. At the same time, they give a deterministic dynamic algorithm for the case of *undirected* graphs, with update time $k!\cdot 2^{O(k)}\cdot\operatorname{polylog}(n)$. A subsequent result due to Chen et al. [11] presents a deterministic dynamic algorithm with amortized update time $2^{O(k^2)}$.

In comparison, our sensitivity oracle works for directed graphs just as well as for undirected graphs, and with a lower dependence on k (only $2^{O(k)}$). However, we obtain a sensitivity oracle, rather than a dynamic algorithm.

Decremental Parameterized Sensitivity Oracles

A recent paper by Bilò et al. [6] constructs decremental sensitivity oracles ("fault tolerant") for the k-Path problem (i.e., where all updates are edge deletions). They give one construction with a randomized $k^\ell 2^k\operatorname{poly}(n)$ preprocessing time and $O(\ell\min\{\ell,k+\log\ell\})$ update time, and a second construction with a lower preprocessing time of $2^k\cdot\frac{(\ell+k)^{\ell+k}}{\ell^\ell k^k}\cdot\ell\operatorname{poly}(n)$ at the expense of an increased query time of $O\left(\frac{(\ell+k)^{\ell+k}}{\ell^\ell k^k}\cdot\ell\min\{\ell,k\}\log n\right)$. They also give a deterministic algorithm, requiring $k^\ell 2.554^k\operatorname{poly}(n)$ preprocessing time and a very low $O(\ell\min\{\ell,k+\log\ell\})$ update time. Our Theorem 3 improves on the dependency on ℓ, making it polynomial in both the preprocessing and update times, and allows for both increments and decrements, while the methods of [6] are specific to decrements. We also achieve an update time which is independent of n in the word-RAM model. In [6], preprocessing is made aware of the value of ℓ, due to the superpolynomial dependence on it.

The techniques of Bilò et al. [6] also achieve a very low memory footprint for their fault-tolerant oracle, attaining at most logarithmic space dependency on n in all variants, when k and ℓ are considered constants. In contrast, we prove a lower bound, showing that one cannot hope for such dependency if increments are allowed, and that any fully dynamic sensitivity oracle must store at least $\Omega(n^2)$ bits (see Theorem 4). This suggests

that the methods used in [6] cannot be used to devise a fully dynamic sensitivity oracle. Their algorithms are based on simple and elegant combinatorial arguments; for instance, in their algorithm with a faster preprocessing time, they precompute a collection of k-paths in the graph, in a way that ensures that with high probability, if there exists a path after ℓ edge deletions, one of the precomputed paths is still valid. We instead take a very different approach based on algebraic coding.

Cover, Matching, Dominating Set, and Packing problems

We are unaware of previous work on dynamic algorithms for the problems we study here (other than k-Path in undirected graphs, which we discussed above). Instead, we compare the dependence on k in the update times of our dynamic running times with the best known dependence on k for static algorithms. In the cases where we match, it is impossible to speed up the dependence on k in our dynamic algorithms without speeding up the fastest known static algorithms.

1. For both k-Partial Cover and t-Dominating Set, the fastest known static randomized running time is $2^k \operatorname{poly}(nk)$ by Koutis and Williams [22] (where for t-Dominating Set we replace k with t in the running time), which our dynamic algorithm matches, and the fastest known static deterministic running time is $2.554^k \operatorname{poly}(kn)$, stated by Tsur [25], whereas our deterministic dynamic algorithm achieves query time $O^*(4^k)$.
2. The fastest known static randomized algorithm for m-Set k-Packing by Björklund, Husfeldt, Kaski and Koivisto [8] has the intimidating running time of

$$\left(\frac{0.108157 \cdot 2^m (1 - 1.64074/m)^{1.64076-m} m^{0.679625}}{(m-1)^{0.679623}}\right)^k n^6 \operatorname{poly}(N)$$

   This is less than $2^{mk} \operatorname{poly}(n)$, and considerably so for smaller m. For example, when $m = 3$, the running time is bounded by $1.4953^{3k} n^6 \operatorname{poly}(N)$. Due to the super-quadratic dependence on n, these techniques are unlikely to be adaptable for the dynamic case.
   In contrast, Koutis [20] proposes a static randomized algorithm running in time

$$2^{mk} n \operatorname{poly}(mk) \operatorname{polylog}(n).$$

   Our dynamic result matches this running time.
3. The fastest known static randomized algorithm for d-Dimensional k-Matching, by Björklund, Husfeldt, Kaski, and Koivisto [8], runs in time $2^{(d-2)k} \operatorname{poly}(Nk)n$. In contrast, we match only an earlier algorithm by Koutis and Williams [22], which runs in time $2^{(d-1)k} \operatorname{poly}(kdn)$. The fastest known static deterministic algorithm runs in time $2.554^{(d-1)k} \operatorname{poly}(Nn)$, stated in Tsur [25], whereas we achieve deterministic dynamic update time $O^*(4^{(d-1)k})$.
4. Exact k-Partial Cover is a natural generalization of similar problems (such as m-Set k-Packing), though we are unaware of explicit prior work on it. A slight modification of the algorithm of Koutis [20] for m-Set k-Packing solves the static problem in randomized $O^*(2^k)$ time, matching the dependence on k in our dynamic algorithm.

Dynamic Parameterized Algorithms

Efficient dynamic algorithms have previously been proposed for a number of parameterized algorithms, often using dynamic versions of classic techniques from the design of fixed-parameter tractable algorithms. These techniques include dynamic kernels [2, 5, 12, 19],

color-coding [2], dynamic elimination forests [11], dynamic branching trees [2], sketching [12], and other methods [14, 15]. To our knowledge, no previous work on dynamic parameterized algorithms has used algebraic techniques.

2 Preliminaries

2.1 Notation

For a positive integer k, write $[k] := \{1, 2, \ldots, k\}$.

We will use the asymptotic notation $O^*(T)$; its meaning will depend on context, so we will define it each time it arises. Generally, in static algorithms and preprocessing times we hide factors that are polynomial in the input parameters (k, n, etc), but when talking about update times, we will only hide factors that are polylogarithmic in the input size.

Let ω be such that two $n \times n$ matrices can be multiplied using $O(n^\omega)$ field operations[3]; it is known that we can take $\omega < 2.373$ [3].

2.2 Sensitivity Oracles

▶ **Definition 14** (sensitivity oracle). *A* sensitivity oracle *is a data structure containing two functions:*

1. *Initialization with an input instance of size n.*
2. *Query with ℓ changes (with problem-specific definition) to the* original input*. This returns the desired output on the altered input.*

The time spent in the initialization step is called the preprocessing time*, and the time spent in the query step is called the* query time*.*

In random sensitivity oracles, we assume the queries are independent of the randomness of the oracle, and thus cannot be used to fool a sensitivity oracle by obtaining the randomness used at the preprocessing phase. We say that a random sensitivity oracle errs with probability at most p, if for any pair of an initial configuration and a query, the probability of error is at most p.

We say that a sensitivity oracle for a parameterized problem, with parameter k, is *efficient* if its preprocessing time is $f(k)\,\mathrm{poly}(n)$ for some computable function f, and its query time is $\mathrm{poly}(\ell)g(k)n^{o(1)}$ for some computable function g.

2.3 Exterior algebra

A key technical tool we will make use of in our algorithms is the exterior algebra. We give a brief introduction to its definition and properties which we will use.

Given a field $\mathbb{F}$ and a positive integer k, the exterior algebra $\Lambda(\mathbb{F}^k)$ is a non-commutative ring over $\mathbb{F}$. In this paper we will be interested in the cases where $\mathbb{F}$ is either the rational numbers $\mathbb{Q}$, or a finite field $\mathbb{F}_{2^d}$ of characteristic 2.

Following the terminology of Brand, Dell and Husfeldt [10], elements of the exterior algebra are called *extensors*. Multiplication in $\Lambda(\mathbb{F}^k)$ is denoted by the wedge sign; for $x, y \in \Lambda(\mathbb{F}^k)$, their product is $x \wedge y$. The generators of this ring are $e_1, e_2, ..., e_k$, the standard basis of $\mathbb{F}^k$, and we impose the relations $e_i \wedge e_j = -e_j \wedge e_i$ for all $i, j \in [k]$ and $e_i \wedge e_i = 0$

[3] This is a slight abuse of notation. The exponent of matrix multiplication ω is typically defined as the smallest constant such that $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ field operations, but we drop the "$o(1)$" in the exponent here for simplicity.

for all $i \in [k]$ (while the latter equation follows from the former for fields of characteristic different from 2, we still require it when $\operatorname{char}\mathbb{F} = 2$). We also require that the wedge product is bilinear[4]. Furthermore, it can be seen that the wedge product is associative[5].

Additionally, we consider any field element $a \in \mathbb{F}$ to be an element of $\Lambda(\mathbb{F}^k)$, with multiplication defined simply by $a \wedge e_i = e_i \wedge a = ae_i$.

For example, we have

$$\begin{aligned} e_1 \wedge (e_5 - e_2 + 3) \wedge e_3 &= e_1 \wedge e_5 \wedge e_3 - e_1 \wedge e_2 \wedge e_3 + 3e_1 \wedge e_3 \\ &= -e_1 \wedge e_3 \wedge e_5 - e_1 \wedge e_2 \wedge e_3 + 3e_1 \wedge e_3. \end{aligned}$$

To simplify the notation, for $I \subseteq [k]$ we also write $e_I := \bigwedge_{i \in I} e_i$, where the product is over the elements of I in ascending order, with the convention $e_\emptyset = 1$. We will sometimes also replace the wedge sign by a simple product sign, when it is clear from context that we mean the wedge product. For example, we can write $e_I := \prod_{i \in I} e_i$. Thus, the above example extensor can also be written as

$$e_1 \wedge (e_5 - e_2 + 3) \wedge e_3 = -e_{\{1,3,5\}} - e_{\{1,2,3\}} + 3e_{\{1,3\}}$$

We see that two products of the basis elements that differ only by the order of the terms can differ only in sign (e.g., $e_1 \wedge e_2 \wedge e_3 = -e_2 \wedge e_1 \wedge e_3$), and furthermore any product of e_is with a repeating factor vanishes (e.g., $e_1 \wedge e_2 \wedge e_2 = e_1 \wedge 0 = 0$). Thus, the set $\{e_I : I \subseteq [k]\}$ spans $\Lambda(\mathbb{F}^k)$ as a vector space over $\mathbb{F}$. It is in fact a basis of this vector space, and we have $\dim_{\mathbb{F}} \Lambda(\mathbb{F}^k) = 2^k$. That is, for any $x \in \Lambda(\mathbb{F}^k)$ there is a unique choice of the 2^k coefficients α_I so that $x = \sum_{I \subseteq [k]} \alpha_I e_I$. For any $x \in \Lambda(\mathbb{F}^k)$ and $T \subseteq [k]$ we denote by $[e_T]x$ the coefficient of e_T when x is represented in the basis $\{e_I : I \subseteq [k]\}$ (i.e., the value of α_T as above).

If there is a d such that all I for which $[e_I]x \neq 0$ have $|I| = d$, we say that x is a *degree-d* extensor. The space of degree-d extensors is denoted by $\Lambda^d(\mathbb{F}^k)$.

We identify any vector $v = (v[1], v[2], ..., v[k]) \in \mathbb{F}^k$ with the exterior algebra element $\sum_i v[i]e_i \in \Lambda^1(\mathbb{F}^k)$. A crucial property of the exterior algebra is that for any vector $v \in \mathbb{F}^k$ it holds that $v \wedge v = 0$. Indeed,

$$\begin{aligned} v \wedge v = \sum_{i,j} v[i]v[j]e_i \wedge e_j &= \sum_{i<j} v[i]v[j]e_i \wedge e_j + \sum_{i<j} v[i]v[j]e_j \wedge e_i \\ &= \sum_{i<j} v[i]v[j]e_i \wedge e_j - \sum_{i<j} v[i]v[j]e_i \wedge e_j = 0. \end{aligned}$$

Similarly, for any two vectors $u, v \in \mathbb{F}^k$, it holds that $u \wedge v = -v \wedge u$. It is important to notice that these properties do not hold for all elements in $\Lambda(\mathbb{F}^k)$. As an example, for $x = e_1 + e_2 \wedge e_3$ we have $x \wedge x = (e_1 + e_2 \wedge e_3) \wedge (e_1 + e_2 \wedge e_3) = e_1 \wedge e_2 \wedge e_3 + e_2 \wedge e_3 \wedge e_1 = 2e_1 \wedge e_2 \wedge e_3$. In fact, we can see that the wedge product of an even number of vectors commutes with any extensor.

Another very useful property of the exterior algebra is its connection to determinants. Let $v_1, v_2, ..., v_k \in \mathbb{F}^k$ be k vectors of dimension k, and consider their product $v_1 \wedge v_2 \wedge ... \wedge v_k$. Replace each v_i with its linear combination of the basis elements e_j and expand the product. We see that any monomial that repeats a basis element gets cancelled, and the others are all $e_{[k]}$ up to a sign. We get

$$v_1 \wedge v_2 \wedge ... \wedge v_k = \bigwedge_{i=1}^{k} \sum_{j=1}^{k} v_i[j]e_j = \sum_{\sigma \in S_k} \bigwedge_{i=1}^{k} v_i[\sigma(i)]e_{\sigma(i)} = \sum_{\sigma \in S_k} \operatorname{sgn}(\sigma) e_{[k]} \bigwedge_{i=1}^{k} v_i[\sigma(i)].$$

[4] $(a + b) \wedge c = (a \wedge c) + (b \wedge c)$ and $a \wedge (b + c) = (a \wedge b) + (a \wedge c)$.
[5] $(a \wedge b) \wedge c = a \wedge (b \wedge c)$

We recognize a determinant in the right hand side, thus showing that

$$v_1 \wedge v_2 \wedge ... \wedge v_k = \det(v_1|v_2|...|v_k)e_{[k]}. \tag{1}$$

2.4 Complexity of ring operations in the exterior algebra

We represent the elements of $\Lambda(\mathbb{F}^k)$ as the length-2^k vector of coefficients of the basis elements e_I. Addition is performed by coordinate-wise addition, hence requires $O(2^k)$ field operations. Multiplication is trickier, and there are a few important cases. All these cases were also noted and used in [10].

▶ **Proposition 15** (Skew product). *Computing $x \wedge v$ for a general extensor $x \in \Lambda(\mathbb{F}^k)$ and a vector $v \in \mathbb{F}^k$ can be done in $O(2^k \cdot k)$ field operations.*

Proof sketch. This is accomplished by simply expanding the product. ◀

▶ **Proposition 16** (General product over characteristic 2). *Computing $x \wedge y$ for any $x, y \in \Lambda(\mathbb{F}^k)$, when $\mathrm{char}(\mathbb{F}) = 2$, can be done in $O(2^k k^2)$ field operations.*

Proof sketch. Here, the ring $\Lambda(\mathbb{F}^k)$ is commutative. Then we can write $[e_I](x \wedge y) = \sum_{T \subseteq I}([e_T]x)([e_{I \setminus T}]y)$, which can be seen as a subset convolution operation. Using the fast subset convolution discovered by Björklund, Husfeldt, Kaski and Koivisto [7], we can compute this with $O(2^k k^2)$ field operations. ◀

▶ **Proposition 17** (General product over any characteristic). *Computing $x \wedge y$ for $x, y \in \Lambda(\mathbb{F}^k)$, for $\mathbb{F}$ of any characteristic, can be done in $O(2^{\omega k/2})$ field operations.*

Proposition 17 is a result of Włodarczyk [27], which reduces computing $x \wedge y$ over $\Lambda(\mathbb{F}^k)$ to k^2 multiplications in a Clifford algebra, which in turn can be embedded in matrices of size $2^{k/2} \times 2^{k/2}$.

2.5 Extensor-coding

In this subsection, we give a brief overview of the techniques recently used by Brand, Dell and Husfeldt [10] to design a randomized and a deterministic algorithm for the (static) k-Path problem; we heavily build off of these techniques in this paper.

Given a directed graph G, we denote its vertex set $V = \{v_1, ..., v_n\}$, and we denote by $W_s(G)$ the set of all walks in G of length s. Recall that walks may repeat vertices, whereas paths may not. For each edge $e \in E(G)$ we have an *edge variable* y_e, whose possible values will be in a field $\mathbb{F}$ we will pick later.

Furthermore, we define vectors $\chi : V \to \mathbb{F}^k$ by $\chi(v_i) = (1, j, j^2, \ldots, j^{k-1})$ where $j = f(i)$ for some injective function $f : [n] \to \mathbb{F}$. We call these *Vandermonde vectors*.

Given a walk in G of length s, $w = (w_1, w_2, ..., w_s) \in W_s(G)$, we define the corresponding *walk extensor* to be $\chi(w_1) \wedge y_{w_1 w_2} \wedge \chi(w_2) \wedge y_{w_2 w_3} \wedge \chi(w_3) \wedge ... \wedge y_{w_{s-1} w_s} \wedge \chi(w_s)$. We will sometimes denote the walk extensor of a walk w by $\chi(w)$.

Our goal is to compute the sum of all walk extensors for walks of length k,

$$\mathcal{Z} = \sum_{(w_1, w_2, ..., w_k) \in W_k(G)} \chi(w_1) \wedge y_{w_1 w_2} \wedge \chi(w_2) \wedge y_{w_2 w_3} \wedge \chi(w_3) \wedge ... \wedge y_{w_{k-1} w_k} \wedge \chi(w_k). \tag{2}$$

The result of this sum is seen to be proportional to the single basis element $e_{[k]}$, and by abuse of notation we will consider it as a scalar equal to that coefficient (or, more precisely, a polynomial in the y variables).

Any k-walk that is not a path does not contribute to Equation (2), because its walk extensor repeats a vector in the product. Thus, any nonzero term in the sum Equation (2) corresponds to a *k-path*.

The choice of $\chi(v_i)$ is such that for any walk $(w_1, w_2, ..., w_k)$ that is a k-path, the walk-extensor is a nonzero monomial in the y variables. This is seen using Equation (1), because $\chi(w_1) \wedge \chi(w_2) \wedge ... \wedge \chi(w_k) = e_{[k]} \det(\chi(w_1)|\chi(w_2)|...|\chi(w_k))$, which is the determinant of a Vandermonde matrix with unique columns, which is known to be nonzero.

Additionally, the monomial in the y variables given to each k-path is seen to uniquely identify the k-path (since there is a different y variable for each edge). Hence, the monomials of different k-paths cannot cancel, and we have proven:

▶ **Lemma 18.** *For a directed graph G, the value of $\mathcal{Z}$ given in Equation* (2) *is a nonzero polynomial in the y variables if and only if G contains a k-path.*

We now recall the classical DeMillo-Lipton-Schwartz-Zippel Lemma:

▶ **Lemma 19** ([13, 24, 28])**.** *Let $f \in \mathbb{F}[x_1, ..., x_n]$ be a nonzero polynomial in n variables with total degree at most d over some field $\mathbb{F}$, and let $S \subseteq \mathbb{F}$. For $a_1, a_2, ..., a_n \in S$ chosen uniformly at random, we have* $\Pr(f(a_1, a_2, ..., a_n) = 0) \leq \frac{d}{|S|}$.

Using the DeMillo-Lipton-Schwartz-Zippel Lemma, we now obtain a randomized algorithm for the k-Path detection problem, assuming we are able to efficiently compute $\mathcal{Z}$. Namely, for each $e \in E(G)$ we randomly select $y_e \in Y$ for some fixed $Y \subseteq \mathbb{F}$ of size $|Y| = 100k$, and compute $\mathcal{Z}$ as in Equation (2). If this value is nonzero, we know there is a k-path. Otherwise, there might still be a k-path, and we might have been unlucky and had the nonzero polynomial vanish for the specific choices of the y_e variables. We output that there is no k-path in this case. The probability of error is at most $\frac{k}{|Y|} = \frac{1}{100}$, and we can repeat this process to lower the probability of error as much as required.

It remains to show how to compute $\mathcal{Z}$ efficiently. We do this using dynamic programming. For any $1 \leq s \leq k$ and $1 \leq i \leq n$ we define $Q_s[i]$ as the sum of walk extensors of length s that end in v_i. Then $\mathcal{Z} = \sum_i Q_k[i]$, and we can compute the vector Q_{s+1} from Q_s by $Q_{s+1}[i] = \sum_{j:(v_j,v_i)\in E(G)} Q_s[j] \wedge y_{v_j,v_i} \wedge \chi(v_i)$. This requires kn^2 skew multiplications of extensors, each of which can be done in time $2^k \operatorname{poly}(k)$ (see Proposition 15). Thus, we can compute $\mathcal{Z}$ with $2^k \operatorname{poly}(k) n^2$ field operations, which is also the total running time of the randomized algorithm. We note that this works over any sufficiently large field $\mathbb{F}$ with $|\mathbb{F}| \geq 100k$.

Brand, Dell and Husfeldt [10] also give a deterministic variant of the algorithm, at the cost of increasing the time complexity. This is done with the beautiful idea of, for each vertex v, *"lifting"* the Vandermonde vector $\chi(v)$ to $\bar{\chi}(v) = \binom{\chi(v)}{0} \wedge \binom{0}{\chi(v)}$. By $\binom{\chi(v)}{0}$, we mean the vector of length $2k$ gotten by concatenating the vector $\chi(v)$ with k zeros. We then do the calculations in $\Lambda(\mathbb{F}^{2k})$ instead of $\Lambda(\mathbb{F}^k)$. Walk extensors of non-path walks still vanish. Now, as observed in [10], for any k-path we have

$$\bar{\chi}(w_1) \wedge \bar{\chi}(w_2) \wedge ... \wedge \bar{\chi}(w_k) = e_{[2k]} \det\left(\binom{\chi(w_1)}{0} \middle| \binom{0}{\chi(w_1)} ... \middle| \binom{\chi(w_k)}{0} \middle| \binom{0}{\chi(w_k)}\right)$$

By proper change of columns, the resulting determinant is equal to the determinant of a 2×2 block diagonal matrix, where the two diagonal blocks are identical. By basic properties of determinants, we then obtain

$$\bar{\chi}(w_1) \wedge \bar{\chi}(w_2) \wedge ... \wedge \bar{\chi}(w_k) = (-1)^{\binom{k}{2}} e_{[2k]} \det(\chi(w_1)|\chi(w_2)|...|\chi(w_k))^2. \tag{3}$$

Therefore, choosing $\mathbb{F} = \mathbb{Q}$, the walk extensors of different k-paths all have the same sign, and hence never cancel. The dynamic programming given above still works for this case, but extensor operations require $2^{2k}\,\mathrm{poly}(k)$ field operations (note we only need skew multiplications). Additionally, all numbers involved are of size at most $n^{O(k)} = 2^{O(k \log n)}$, and hence operations require only $\mathrm{poly}(k)\,\mathrm{polylog}(n)$ time, or $\mathrm{poly}(k)$ time in the word-RAM model which we assume. This produces a deterministic algorithm with running time $4^k\,\mathrm{poly}(k)n^2$.

3 Overview of our techniques

The key idea behind our algorithms is to identify appropriate sums of extensors which encode the answer to the problem (e.g., one where candidate solutions correspond to nonzero monomials), and which can be efficiently dynamically maintained. Similar to [10], we will frequently make use of the properties of exterior algebras to "nullify" repetitions (for example, of vertices in the k-Path problem, and of elements in the Exact k-Partial Cover problem). In many cases, the extensors or similar algebraic coding expressions used by the fastest known static algorithms seem difficult to efficiently maintain dynamically, and we will need to modify them, and identify useful precomputed values, to speed up our update time. While these techniques prove to be very well suited for designing sensitivity oracles and dynamic algorithms, we are unaware of previous work that uses the exterior algebra in such a way.

3.1 k-Path

For this problem, we aim to maintain $\mathcal{Z}$, the sum over walk extensors of length-k walks which we defined in Equation (2) above. Employing properties of the exterior algebra as discussed above, it is seen that any walk that is not a path (that is, a walk that repeats a vertex) does not contribute to the sum, and we thus indirectly compute a sum over only paths.

As we discussed in Section 2.5 above, Brand, Dell and Husfeldt [10] showed that computing this sum allows for extracting valuable information about the k-paths in a graph. In particular, by carefully selecting vectors or degree-2 extensors $\chi(v)$ for each vertex v, it allows for the design of randomized and deterministic algorithms for the k-path detection problem, as well as for approximately counting the number of k-paths with high probability.

Let us focus, first, on the case when our sensitivity oracle only allows for edge increments. We begin by precomputing, for each pair of vertices in the graph, a sum over the walk extensors between those vertices, so that we can "stitch" them together appropriately when new edges are inserted. Put precisely, let $W_s(u, v)$ denote the set of walks of length s from u to v, and let $Q_s[u, v]$ be the precomputed value of the sum of walk extensors for walks in $W_s(u, v)$ in the initial graph. Now suppose a new edge (v_1, v_2) is inserted. Then the additional walk extensors for walks of length s between any pair of vertices (t_1, t_2) can be computed as

$$\sum_{s'} \left(\sum_{w \in W_{s'}(t_1, v_1)} \chi(w) \right) \wedge y_{v_1, v_2} \wedge \left(\sum_{w \in W_{s-s'}(v_2, t_2)} \chi(w) \right) = \sum_{s'} Q_{s'}[t_1, v_1] \wedge y_{v_1, v_2} \wedge Q_{s-s'}[v_2, t_2].$$

At first, stitching might seem problematic, since we might need to iterate over the possible lengths of each stitched part. This is not be a problem for a single edge insertion, but leads to an exponential dependency on the number of edges inserted as we partition the total

length k between them. While this issue can be efficiently resolved with proper dynamic programming, we can circumvent this efficiently and more cleanly by instead working with sums over the walks of all possible lengths, while finally inspecting only the coefficient of $e_{[k]}$ in the resulting extensor, so that terms corresponding to paths of length other than k will either cancel out or have too low degree. Indeed, we note that only walks of length k contribute to this coefficient.

Another, more substantial problem occurs when one tries to "stitch" paths into a larger path that uses several new edges: there are $\ell!$ different orders to pass through the ℓ new edges. We combine properties of the exterior algebra with an algebraic trick (which we describe in more detail shortly) to allow computing the sum of the effects of all of these cases with only a polynomial dependence on ℓ.

In order to be able to work with only a small subset of the precomputed Q matrix, we additionally precompute the sum of walk extensors ending and beginning at each vertex (i.e., sums of rows and columns of Q), and the original sum over walk extensors in the original graph. This allows us to use only $O(\ell^2)$ precomputed values in the updating phase.

Finally, by further employing properties of the exterior algebra, we are able to combine these techniques with the inclusion-exclusion principle, to also allow edge removals, while keeping the same time and space complexities for preprocessing and updates.

We now describe the ideas in some more detail. We refer the reader to the full version for the full details. As a preprocessing, we compute an $n \times n$ matrix Q with extensor values, defined by

$$Q[i,j] = \sum_{\substack{(w_1,w_2,\ldots,w_s)\in W \\ w_1=v_i, w_s=v_j}} \chi(w_1) \wedge y_{w_1w_2} \wedge \chi(w_2) \wedge y_{w_2w_3} \wedge \chi(w_3) \wedge \ldots \wedge y_{w_{s-1}w_s} \wedge \chi(w_s)$$

where W is the set of *all walks* in the graph, which can have any length greater than 0. This is the sum of all walk extensors for walks from v_i to v_j. We then compute vectors S (resp. F) of length n, similarly computing in their ith entry the sum of walk extensors starting (resp. finishing) at each vertex v_i. That is, $S[i] = \sum_j Q[i,j]$ and $F[j] = \sum_i Q[i,j]$.

Finally, we compute $\mathcal{Z} = \sum_{i,j} Q[i,j]$, the sum of all walk extensors over all walks. $e_{[k]}\mathcal{Z}$ is exactly the value we aim to maintain after a query, as this is exactly the one used in the extensor coding technique (Section 2.5). We explain how with proper dynamic programming we can compute these values in preprocessing with $2^k \operatorname{poly}(k) n^2$ operations over $\mathbb{F}$, using only additions and skew multiplications.

Then, when prompted with a query, we aim to compute the sum over walk extensors in the updated graph, denoted $\mathcal{Z}_{\text{new}}$. Inspecting whether $e_{[k]}\mathcal{Z}_{\text{new}} \neq 0$ will let us test whether the updated graph contains a k-path.

We now begin handling a query. Given a list of ℓ updates that are each either an edge insertion or edge deletion, there are $n' \leq 2\ell$ vertices at the endpoints of the updated edges. We begin by extracting the $n' \times n'$ sub-matrix Q' of Q, and length-n' sub-vectors S', F' of S, F, gotten by taking entries corresponding to those endpoint vertices.

We next define an $n' \times n'$ matrix E_r^+ corresponding to the r-th edge insertion by setting each of its entries to 0 except for $E_r^+[i,j] = y_{i,j}$, where (v_i, v_j) is the r-th inserted edge. We similarly define E_r^- in the same way, to be the 0 matrix except for $E_r^-[i,j] = y_{i,j}$ where (v_i, v_j) is the r-th deleted edge. We then define $\Delta^+ = \sum_r E_r^+$, $\Delta^- = \sum_r E_r^-$, and $\Delta = \Delta^+ - \Delta^-$. We then compute that

$$\mathcal{Z}_{\text{new}} = \mathcal{Z} + \sum_{i=1}^{k} F'^T \Delta (Q'\Delta)^{i-1} S'.$$

This formula is crucial to our query algorithm, so we explain it in some detail here.

We first explain the correctness of this formula for the case of only increments. In this case, $\mathcal{Z}_{\text{new}} - \mathcal{Z}$ is exactly the sum of walk-extensors for walks that use the new edges. Here, upon expanding the expression when $\sum_r E_r^+$ is substituted for Δ^+, we note that $F'^T \Delta^+ S'$ counts the walks that use exactly one of the new edges. Indeed, for each r the term $F'^T E_r^+ S'$ accounts for walks passing through the r-th new edge exactly once. For counting walks that use exactly two of the newly inserted edges, we now need to compute $\sum_{r_1 \neq r_2} F'^T E_{r_1}^+ Q' E_{r_2}^+ S'$. Indeed, this expression correctly accounts for walks with nonzero walk extensors starting at any vertex, continuing through one new edge, travelling to a new edge then through it, then continuing to end at any vertex. We further note that any specific path is counted in this way exactly once.

We note that by linearity and the fact that only paths produce nonzero walk extensors, this is equal to $F'^T \Delta^+ Q' \Delta^+ S'$. That is, upon expanding the expression when $\sum_r E_r^+$ is substituted for Δ^+, we get exactly the terms we intended, plus terms accounting for walks passing through the new edges twice, which evaluate to zero in the exterior algebra. This considerably simplifies the calculations. Continuing in the same fashion, we argue we obtain

$$\mathcal{Z}_{\text{new}} - \mathcal{Z} = \sum_{i=1}^{k} F'^T \Delta^+ (Q' \Delta^+)^{i-1} S'$$

in total for the incremental case.

Consider now the general case, with both increments and decrements. Suppose w is any walk on the vertices which uses edges from the union of the original graph and the updated graph. We assume w is a path, since otherwise its walk extensor vanishes, and we need not worry about whether it appears in the sum $\mathcal{Z}_{\text{new}}$. Now suppose w has length at most k and uses a inserted edges and b removed edges.

Consider first when $a \geq 1$. In this case, w is not counted in the original sum $\mathcal{Z}$. When substituting $\Delta = \sum_j E_j^+ - \sum_j E_j^-$ into $\sum_{i=1}^{k} F'^T \Delta (Q' \Delta)^{i-1} S'$ and expanding, we see that w is counted only when all the chosen E_j^+ factors exactly correspond to the a new edges used by w, in the order they are used. It is also counted when choosing any $b' \leq b$ factors of type E_j^- that appear in w, but they must also appear in the right order, and they contribute the walk extensor of w with weight exactly $(-1)^{b'}$. In total, w is accounted for exactly $\sum_{b'=0}^{b} \binom{b}{b'} (-1)^{b'} = (1-1)^b = [b = 0]$ times[6] in $\mathcal{Z}_{\text{new}}$, which is what we want.

Now suppose $a = 0$. Then a similar argument shows that the number of times it is counted in $\sum_{i=1}^{k} F'^T \Delta (Q' \Delta)^{i-1} S'$ is exactly $\sum_{b'=1}^{b} \binom{b}{b'} (-1)^{b'} = (1-1)^b - 1 = [b = 0] - 1$. However, it is also counted in $\mathcal{Z}$ exactly once, so in total is counted $[b = 0]$ times, which is also exactly what we want.

Using this formula, we explain how we are generally able to compute $\mathcal{Z}_{\text{new}}$ in $O(\ell^2 2^{\omega k/2})$ field operations. However, we come back to this running time once we describe the specifics of the randomized and deterministic sensitivity oracles, each with its own details.

3.2 k-Partial Cover

We also make use of extensors to design fully dynamic algorithms for other parameterized problems. We focus here on one example: designing a deterministic dynamic algorithm for the k-Partial Cover problem. In this problem, we are given subsets $S_1, ..., S_n \subseteq [N]$, and wish to find the minimum number of such subsets whose union has size at least k (or report that no such collection exists).

[6] Here we use the notation $[b = 0] := \begin{cases} 1 & \text{if } b = 0, \\ 0 & \text{otherwise.} \end{cases}$

One could use a polynomial constructed by Koutis and Williams [22] for this problem, combined with similar techniques to those we used above for k-Path, to devise a deterministic dynamic algorithm with update time $O(2^{\omega k})$. However, we instead give a different polynomial which is easier to dynamically maintain, achieving a faster $O^*(4^k)$ update time.

As in the k-Path problem, we give each element in the universe $a \in [N]$ a Vandermonde vector $\chi(a) \in \mathbb{Q}^k$, then lift it to $\bar{\chi}(a) = \binom{\chi(a)}{0} \wedge \binom{0}{\chi(a)} \in \Lambda(\mathbb{Q}^{2k})$.

We introduce a single new variable z, and consider the polynomial

$$P(z) = \prod_S \left(1 + z\left[\prod_{a \in S}(1 + \bar{\chi}(a)) - 1\right]\right).$$

We argue that the solution we seek is the minimum t for which $[e_{[2k]}z^t]P \neq 0$ (that is, the coefficient of $e_{[2k]}$ in the coefficient of z^t in P). Our goal, then, will be to maintain the value of $P(z)$, and calculations will take place in polynomials over extensors, $\Lambda(\mathbb{Q}^{2k})[z]$. We also argue that $\deg P(z) \leq k$, and so is not too large to handle in update steps.

We first argue that the product in P should only be computed over sets S of cardinality less than k, as larger sets can be treated separately, knowing that the optimal solution is 1 if any such set exists in the collection. Since $P(z)$ is defined as a product over sets, to update it with a new set S of cardinality $|S| < k$, we can multiply $P(z)$ by the corresponding factor $1 + z\left[\prod_{a \in S}(1 + \bar{\chi}(a)) - 1\right]$. This seems too slow at first, since general extensor multiplication requires $O(2^{\omega k})$ time, which is more than our target $O^*(4^k)$. However, rather than computing the factor and then using general extensor multiplication, we argue that we can indirectly multiply by this factor using only additions and skew multiplications by rearranging the contributing terms, thus requiring only $O^*(4^k)$ field operations. Indeed, the product of $P(z)$ with the new factor is $(1 - z)P(z) + P(z)\prod_{a \in S}(1 + \bar{\chi}(a))$, which can be performed by separately computing $(1 - z)P(z)$ and $P(z)\prod_{a \in S}(1 + \bar{\chi}(a))$, where the latter can be computed by repeated skew-multiplications.

A new problem arises when a set S is removed. For this, we need to somehow cancel the factor in $P(z)$ corresponding to S. We show that in this case the corresponding factor has an inverse in $\Lambda(\mathbb{Q}^{2k})[z]$: we first note that it can be written as $1 + X$ for an element X that contains only extensors of degree ≥ 2, which implies that $X^{k+1} = 0$, and using this, we observe that

$$(1 + X)(1 - X + X^2 - \ldots + (-X)^k) = 1 - (-X)^{k+1} = 1,$$

so $(1 + X)^{-1} = 1 - X + X^2 - \ldots + (-X)^k$. Similar to the previous case, we argue that multiplying by this factor can also be done with $\text{poly}(k)$ additions and skew multiplications, and so can be done in $O^*(4^k)$ time.

This and all our other dynamic algorithms are described in detail in the full version.

4 Discussion on fixed-parameter complexity classes

We discuss the relationship between the following definitions.

▶ **Definition 20** (FPT). *A parameterized problem is in FPT if it is decidable in time $f(k)\,\text{poly}(n)$ for a computable function f.*

▶ **Definition 21** (FPD). *A parameterized problem is in FPD (Fixed-Parameter Dynamic) if there is a dynamic algorithm for it requiring $f(k)\,\text{poly}(n)$ preprocessing time and $g(k)n^{o(1)}$ update time, for computable functions f and g.*

▶ **Definition 22** (FPSO). *A parameterized problem is in FPSO (Fixed-Parameter Sensitivity Oracle) if there is a sensitivity oracle for it requiring $f(k)\,\mathrm{poly}(n)$ preprocessing time and $\mathrm{poly}(\ell)g(k)n^{o(1)}$ update time, for a computable function g.*

We note that FPD $\subseteq$ FPSO $\subseteq$ FPT. We can show that the inclusions are strict, at least under plausible hardness conjectures. For example, [2] shows that k-Path in directed graphs does not admit a dynamic algorithm under hardness conjectures (which are shown in this paper to be in FPSO). [18] shows that the #SSR problem does not have an efficient sensitivity oracle, assuming SETH, which in turn can be used to show that, assuming SETH, there exists an FPT problem that is not in FPSO.

We further note that many problems shown in [2] to not be in FPD can be shown to be in FPSO. As a few examples:

1. Triangle Detection, which is the problem of detecting whether a graph contains a triangle, can be solved efficiently by a sensitivity oracle by precomputing the square of the graph's adjacency matrix and computing the number of triangles in time $O(n^\omega)$, then updating the number of triangles in time $O(\ell^\omega)$ ($O(\ell)$ for triangles that use a single updated edge, $O(\ell^2)$ for triangles using two updated edges, and $O(\ell^\omega)$ for those using three updated edges).
2. Incremental st-Reachability, which is the problem of deciding whether two predetermined vertices s and t are connected in a directed graph, only allowing incremental updates, can be solved efficiently by precomputing reachability between any two vertices (for example by running BFS from all vertices in time $\mathrm{poly}(n)$), then using dynamic programming to answer a query in time $\mathrm{poly}(\ell)$ by updating the connectivity information only on the $\leq 2\ell + 2$ vertices that are either s, t or are part of any inserted edge
3. 3SUM, which is the problem of deciding whether there are 3 elements in a list that sum to 0. Here it is possible to precompute the sums of all pairs in time $O(n^2)$ (counting multiplicities), and the number of triples whose sum is 0. Then when adding or removing ℓ numbers it is possible to compute in $\mathrm{poly}(\ell)$ time the number of new solutions and the number of previous solutions that should be removed, and checking if the remaining number of solutions is nonzero.
4. k-Layered Reachability Oracle (k-LRO), the problem of deciding whether two vertices u, v are connected in a directed k-layered graph (that is, a graph whose vertices are partitioned into k parts, with edges only going from one part to the next), also has an efficient sensitivity oracle, and in fact can be seen to be equivalent to the directed k-Path problem. In particular, the same k-Path sensitivity oracle devised in this paper can be used here without any changes.

5 Open questions

We briefly discuss a few natural questions that arise.

1. Is there a fully-dynamic k-Path detection algorithm on undirected graphs with $f(k)n^{O(1)}$ preprocessing time for some computable function f, and update time $2^{O(k)}n^{o(1)}$?
   This would beat the dynamic algorithm proposed in [2] that is a simple adaptation of the original color-coding idea [4], while to the best of our knowledge, no improvement on this original color-coding idea is known to transfer to the dynamic setting. While our work shows a better dependency on k, we do not reach a truly dynamic algorithm, but rather only a sensitivity oracle.
   We note that it is unlikely that such an an algorithm exists for directed graphs, due to the conditional lower bound presented in [2].

2. Is there an efficient sensitivity oracle for the k-Tree problem?
   The k-Tree problem is as follows: given an undirected graph G on n vertices and a tree T on k vertices, determine whether T is a (not necessarily induced) subgraph of G. This problem is known to be FPT, and for example is shown in [22] to be solvable in time $2^k \operatorname{poly}(k) \operatorname{poly}(n)$, with techniques similar to those applied for the k-Path problem. However, it is unclear how to adapt the techniques here for an efficient sensitivity oracle. We conjecture that there is, in fact, no efficient sensitivity oracle for this problem. More specifically, we conjecture this for any k-Tree formed by connecting a single vertex to the beginning of $\Theta(\sqrt{k})$ paths of length $\Theta(\sqrt{k})$. We note that the techniques of [6] for a decremental sensitivity oracle work for any k-Tree just as well as they do for the k-Path problem.
3. Brand [9] noted that the algebra generated by the lifts of Vandermonde vectors, which is the algebra used in all the deterministic algorithms presented in this paper, has dimension $O(\varphi^{2k})$ where $\varphi = \frac{1+\sqrt{5}}{2}$ - much smaller than the anticipated $O(4^k)$. It is also commutative. It is an open problem whether, in light of this, multiplication in this algebra can be reduced to below $O(2^{\omega k})$. Such an algorithm will immediately improve the bounds discussed in this paper.
4. Can other techniques used to solve the static versions of the problems discussed in this paper, or other parameterized problems, be used to design faster dynamic algorithms and sensitivity oracles?

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443. IEEE, 2014.

2 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3395037.

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 522–539, USA, 2021. Society for Industrial and Applied Mathematics. URL: https://dl.acm.org/doi/10.5555/3458064.3458096.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

5 Max Bannach, Zacharias Heinrich, Rüdiger Reischuk, and Till Tantau. Dynamic Kernels for Hitting Sets and Set Packing. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2021.7.

6 Davide Bilò, Katrin Casel, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, J.A. Gregor Lagodzinski, Martin Schirneck, and Simon Wietheger. Fixed-Parameter Sensitivity Oracles. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:18, 2022. doi:10.4230/LIPIcs.ITCS.2022.23.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 67–74, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250801.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

9 Cornelius Brand. Patching Colors with Tensors. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:16, 2019. doi:10.4230/LIPIcs.ESA.2019.25.

10 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 151–164, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188902.

11 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, et al. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.

12 Rajesh Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1234–1251, 2015. doi:10.1137/1.9781611973730.82.

13 Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

14 Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for mso properties in graphs with bounded tree-depth. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014*, pages 334–345, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-44777-2_28.

15 Zdeněk Dvořák and Vojtěch Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *Proceedings of the 13th International Conference on Algorithms and Data Structures*, WADS'13, pages 304–315, Berlin, Heidelberg, 2013. Springer-Verlag. doi:10.1007/978-3-642-40104-6_27.

16 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms. *CoRR*, abs/2102.11169, 2021. arXiv:2102.11169.

17 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.

18 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. *CoRR*, 2017. arXiv:1703.01638.

19 Yoichi Iwata and Keigo Oka. Fast dynamic graph algorithms for parameterized problems. *ArXiv*, abs/1404.7307, 2014. arXiv:1404.7307.

20 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming*, pages 575–586, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-70575-8_47.

21 Ioannis Koutis. Constrained multilinear detection for faster functional motif discovery. *Information Processing Letters*, 112(22):889–892, November 2012. doi:10.1016/j.ipl.2012.08.008.

22 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3), May 2016. doi:10.1145/2885499.

23 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610, 2010.

24 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. doi:10.1145/322217.322225.

25 Dekel Tsur. Faster deterministic parameterized algorithm for k-path. *Theoretical Computer Science*, 790:96–104, 2019. doi:10.1016/j.tcs.2019.04.024.

26 Ryan Williams. Finding paths of length k in $\mathcal{O}^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, February 2009. doi:10.1016/j.ipl.2008.11.004.

27 Michał Włodarczyk. Clifford algebras meet tree decompositions. *Algorithmica*, 81(2):497–518, February 2019. doi:10.1007/s00453-018-0489-3.

28 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226. Springer-Verlag, 1979. doi:10.5555/646670.698972.

Low-Degree Polynomials Extract From Local Sources

Omar Alrabiah
EECS Department, University of California, Berkeley, CA, USA

Eshan Chattopadhyay
Computer Science Department, Cornell University, Ithaca, NY, USA

Jesse Goodman
Computer Science Department, Cornell University, Ithaca, NY, USA

Xin Li
Computer Science Department, Johns Hopkins University, Baltimore, MD, USA

João Ribeiro
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We continue a line of work on extracting random bits from weak sources that are generated by simple processes. We focus on the model of locally samplable sources, where each bit in the source depends on a small number of (hidden) uniformly random input bits. Also known as local sources, this model was introduced by De and Watson (TOCT 2012) and Viola (SICOMP 2014), and is closely related to sources generated by AC^0 circuits and bounded-width branching programs. In particular, extractors for local sources also work for sources generated by these classical computational models.

Despite being introduced a decade ago, little progress has been made on improving the entropy requirement for extracting from local sources. The current best explicit extractors require entropy $n^{1/2}$, and follow via a reduction to affine extractors. To start, we prove a barrier showing that one cannot hope to improve this entropy requirement via a black-box reduction of this form. In particular, new techniques are needed.

In our main result, we seek to answer whether low-degree polynomials (over $\mathbb{F}_2$) hold potential for breaking this barrier. We answer this question in the positive, and fully characterize the power of low-degree polynomials as extractors for local sources. More precisely, we show that a random degree r polynomial is a low-error extractor for n-bit local sources with min-entropy $\Omega(r(n \log n)^{1/r})$, and we show that this is tight.

Our result leverages several new ingredients, which may be of independent interest. Our existential result relies on a new reduction from local sources to a more structured family, known as local non-oblivious bit-fixing sources. To show its tightness, we prove a "local version" of a structural result by Cohen and Tal (RANDOM 2015), which relies on a new "low-weight" Chevalley-Warning theorem.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Randomness extractors, local sources, samplable sources, AC^0 circuits, branching programs, low-degree polynomials, Chevalley-Warning

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.10

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2022/082/`

Funding *Eshan Chattopadhyay*: Supported by NSF CAREER Award 2045576.
Jesse Goodman: Supported by NSF CAREER Award 2045576.
Xin Li: Supported by NSF CAREER Award CCF-1845349.
João Ribeiro: Research supported in part by the NSF grants CCF-1814603 and CCF-2107347 and by the following grants of Vipul Goyal: the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 10; pp. 10:1–10:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Randomness is a fundamental resource in many areas in computer science, such as algorithm design and cryptography. However, such tasks often assume access to a source of independent and uniform bits, while real-world physical processes (e.g., electromagnetic noise, timings of user keystrokes) generate randomness that is far from perfect. This state of affairs motivates the problem of *randomness extraction*. The goal is to design a deterministic function, called an *extractor*, that can distill a (nearly) uniform bit from any source belonging to a certain family.

▶ **Definition 1.1** (Extractor). *A function* $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}$ *is an* extractor *for a family of distributions* $\mathcal{X}$ *over* $\{0,1\}^n$ *with error* ε *if, for every* $\mathbf{X} \in \mathcal{X}$*,*

$$\left|\Pr[\mathsf{Ext}(\mathbf{X}) = 1] - \frac{1}{2}\right| \leq \varepsilon.$$

Besides their practical motivation, randomness extractors (and other related pseudorandom objects such as dispersers, condensers, and expander graphs) have deep connections to coding theory, combinatorics, and complexity theory.

In order to construct an extractor for a family $\mathcal{X}$ of sources, the most general assumption one can make about $\mathcal{X}$ is that each $\mathbf{X} \in \mathcal{X}$ has some "randomness." Here, it is typical to measure the randomness of a source $\mathbf{X}$ by its min-entropy $H_\infty(\mathbf{X}) := -\log \max_x \Pr[\mathbf{X} = x]$. However, even if we assume each $\mathbf{X} \in \mathcal{X}$ has a very high amount of this very strong notion of entropy, extraction is still impossible: indeed, one cannot hope to extract from $\mathcal{X}$ even if each source $\mathbf{X} \in \mathcal{X}$ is guaranteed to have min-entropy $k \geq n - 1$ [6]. To enable extraction, one must make additional assumptions on the structure of each $\mathbf{X} \in \mathcal{X}$.

Extractors for local sources, AC^0 sources, and small-space sources

In a seminal work, Trevisan and Vadhan [18] initiated the study of randomness extraction from sources that can be sampled by "simple" processes. In addition to the generality of such sources, it can be argued that they serve a reasonable model of randomness that might actually be found in nature.

More formally, Trevisan and Vadhan studied sources that can be sampled by polynomial size circuits that are given uniform bits as input. However, extracting randomness from this class of sources requires strong computational hardness assumptions. This motivated De and Watson [9] and Viola [21] to consider *unconditional* extraction from sources sampled by more restricted, but still natural, circuit families. To this end, they introduced the notion of *local sources*. Intuitively, a local source $\mathbf{X}$ is one that can be sampled by a low-depth circuit with bounded fan-in (a low-complexity process).

▶ **Definition 1.2** (Local source [9, 21]). *A distribution* $\mathbf{X} \sim \{0,1\}^n$ *is a* d-local source *if* $\mathbf{X} = g(\mathbf{U}_m)$*, where* $\mathbf{U}_m$ *is the uniform distribution over* m *bits (for some* m*), and* $g : \{0,1\}^m \to \{0,1\}^n$ *is a function where each output bit depends on at most* d *input bits.*

Local sources are closely connected to other models of sources sampled by simple processes. Viola [21] proved that every source generated by AC^0 circuits is (close to) a convex combination of local sources with small locality and slightly lower min-entropy. More recently, Chattopadhyay and Goodman [3] showed a similar result for sources generated by bounded-width branching programs [13]. Thus, extractors for local sources also work for sources generated by these classical computational models. In fact, the current state-of-the-art extractors for sources generated by AC^0 circuits and bounded-width branching programs are extractors for 1-local sources.

A barrier at $\sqrt{n}$ min-entropy

Despite the applications above and being introduced over a decade ago, little progress has been made on constructing extractors for local sources [9, 21, 16]. In particular, all known constructions require min-entropy at least $\sqrt{n}$, and follow via a reduction to extractors for affine sources (i.e., sources that are uniform over affine subspaces of $\mathbb{F}_2^n$). Thus, there appears to be a "barrier" at $\sqrt{n}$ min-entropy, at least when using affine extractors [23]. It is natural to ask how we might break this $\sqrt{n}$ barrier, which raises the question:

Can affine extractors be used to extract from local sources with min-entropy $k \ll \sqrt{n}$?

As motivation, we start by providing strong evidence that the answer to the above question is negative. In particular, we prove the following.

▶ **Theorem 0** (Barrier result). *It is not possible to extract randomness from 2-local sources with min-entropy $k \geq \sqrt{n}$ by applying an affine extractor in a black-box manner.*

Thus, if we would like to construct extractors for local sources with min-entropy significantly below $\sqrt{n}$, new techniques are needed.

Towards breaking the $\sqrt{n}$ barrier using low-degree polynomials

While explicit extractors that break the $\sqrt{n}$ min-entropy barrier for local sources are the end goal, these still seem beyond reach. We believe that the next best thing are non-explicit extractors that are of "low complexity". Our hope is that such extractors may help us eventually construct truly explicit extractors, as non-explicit extractors are more likely to be easier to derandomize if they belong to a low complexity class. At the same time, such non-explicit extractors may have applications in complexity theory (i.e., since the current state-of-the-art circuit lower bounds are against extractors [15]). There is a long line of work [20, 11, 5, 17, 19, 2, 5, 10, 12, 7] on the power of low-complexity computational models for extracting from various families of sources.

From the forefront of "low complexity" classes, we choose to study in this work the class of low-degree $\mathbb{F}_2$-polynomials. In particular, we ask whether (non-explicit) low-degree polynomials can help break the $\sqrt{n}$ barrier for extracting from local sources, and more generally we seek to answer the following question:

▶ **Question 1.** *How powerful are low-degree $\mathbb{F}_2$-polynomials as extractors for local sources?*

Beyond being a natural algebraic class, low-degree $\mathbb{F}_2$-polynomials have a natural combinatorial interpretation. We can represent a degree-2 $\mathbb{F}_2$-polynomial f as a graph G_f on n vertices, with edges representing monomials included in f. Then, f being a good extractor for local sources translates into a parity constraint on the number of edges in certain induced subgraphs of G_f. Likewise, a degree-3 $\mathbb{F}_2$-polynomial can be represented as a 3-hypergraph, and so on. Given the correspondence between low-degree polynomials and hypergraphs with small edge sizes, we hope that tools from combinatorics can be leveraged to make our constructions explicit and break the $\sqrt{n}$ min-entropy barrier for extracting from local sources (which would also give improved extractors for small-space sources).

Our motivation to study low-degree polynomials as our "low complexity" model also comes from the work of Cohen and Tal [7], which studied the same question in the context of *affine* sources. In their work, they showed that there exist degree-r $\mathbb{F}_2$-polynomials that extract from affine sources with min-entropy $O(rn^{\frac{1}{r-1}})$, and that this is tight. To answer Question 1, we aim to provide a local source analogue of this result.

1.1 Summary of our results

In this paper, we fully characterize the power of low-degree polynomials as extractors for local sources, answering Question 1 and proving a local-source analogue of Cohen-Tal. Along the way, we rely on several new ingredients which may be of independent interest. We present these results in Section 1.1.1 and Section 1.1.2, respectively.

1.1.1 Main result

Our main result gives a tight characterization of the power of low-degree polynomials as extractors for local sources. We state it formally, below.

▶ **Theorem 1** (Main result). *For every $d, r \in \mathbb{N}$ there exist constants $C, c > 0$ such that the following holds. For every $n \in \mathbb{N}$ there exists a (not necessarily explicit) degree r polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ that is an $2^{-\Omega(k)}$-extractor for d-local sources with min-entropy*

$$k \geq C(n \log n)^{1/r},$$

but for every degree r polynomial $g \in \mathbb{F}_2[x_1, \ldots, x_n]$ there exists a d-local source with min-entropy

$$k \geq c(n \log n)^{1/r}$$

on which it is constant.

Theorem 1 implies that degree-3 polynomials are already enough to extract from min-entropy $k = O((n \log n)^{1/3})$, which (non-explicitly) breaks the $\sqrt{n}$ min-entropy barrier on previous techniques (Theorem 0). Furthermore, given known reductions from AC^0 sources and small-space sources to local sources [21, 3], it also follows that low-degree polynomials can be used to break existing min-entropy barriers for extracting from these other models of weak sources. We refer the reader to the full version of this paper for a more in-depth discussion, where we also outline a different application of our result to sampling lower bounds against AC^0 (generalizing a result of Viola [22]).

While Theorem 1 is stated for constant locality d and constant degree r, we actually prove stronger results that hold for superconstant d, r. In particular, Theorem 1 follows immediately from the following two results, which provide upper and lower bounds on the entropy required to extract from d-local sources using a degree $\leq r$ polynomials (where d, r need not be constant).

▶ **Theorem 1.1** (Technical version of Theorem 1, Upper Bound). *There are universal constants $C, c > 0$ such that for all $n, d, r \in \mathbb{N}$, the following holds. With probability at least 0.99 over the choice of a random degree $\leq r$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$, it holds that f is an ε-extractor for d-local sources with min-entropy*

$$k = C2^d d^2 r \cdot (2^d n \log n)^{1/r},$$

where $\varepsilon = 2^{-\frac{ck}{r^3 2^d d^2}}$.

▶ Remark 1.3. If instead of extractors we aim to construct *dispersers*,[1] then we are able to improve the dependency on the locality d in Theorem 1.1 to hold for min-entropy $k = Cd^2 r \cdot (dn \log n)^{1/r}$.

[1] A function $\mathsf{Disp} : \{0,1\}^n \to \{0,1\}$ is a *disperser for a class of sources $\mathcal{C}$* if the support of $\mathsf{Disp}(\mathbf{X})$ is $\{0,1\}$ for all sources $\mathbf{X} \in \mathcal{C}$.

▶ **Theorem 1.2** (Technical version of Theorem 1, Lower Bound). *There are constants $C, c > 0$ such that for all $n, d, r \in \mathbb{N}$ with $r \leq c\log(n)$ and $d \leq 2^{\sqrt{\log n}}$, the following holds. For any degree $\leq r$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$, there is a d-local source $\mathbf{X} \sim \{0,1\}^n$ with min-entropy at least*

$$k \geq Cr(dn\log n)^{1/r}$$

such that $f(\mathbf{X})$ is constant.

1.1.2 Key new ingredients

Our main result follows from a collection of new ingredients, which may be of independent interest. In order to prove our upper bound on min-entropy (Theorem 1.1), we prove a new reduction from d-local sources to *d-local non-oblivious bit fixing (NOBF) sources*. Informally, a d-local NOBF source $\mathbf{X} \sim \{0,1\}^n$ of min-entropy k' is a source that has k' uniform independent bits, with all other bits depending on at most d of the k' bits.

▶ **Theorem 2** (Reduction from d-local sources to d-local NOBF sources). *There exists a universal constant $c > 0$ such that for any $n, k, d \in \mathbb{N}$, the following holds. Let $\mathbf{X} \sim \{0,1\}^n$ be a d-local source with min-entropy $\geq k$. Then $\mathbf{X}$ is ε-close to a convex combination of d-local NOBF sources with min-entropy $\geq k'$, where $\varepsilon = 2^{-ck'}$ and*

$$k' = \frac{ck}{2^d d^2}.$$

The family of d-local NOBF sources, introduced in [4], is a significant specialization of d-local sources. The above reduction shows that, at least for constant locality d, we can just focus on extracting from this simpler class - even in future explicit constructions.

To prove our lower bound on min-entropy (Theorem 1.2), we actually prove this lower bound for the special class of d-local sources known as *d-local affine sources*: such a source $\mathbf{X} \sim \mathbb{F}_2^n$ is uniform over a d-local affine subspace $X \subseteq \mathbb{F}_2^n$, which is a special type of affine subspace that admits a basis $v_1, \ldots, v_k \in \mathbb{F}_2^n$ where each coordinate $i \in [n]$ holds the value 1 in at most d of these vectors. For this special class of sources, our lower bound is actually tight, and can be viewed as a "local" version of a result by Cohen and Tal [7].

▶ **Theorem 3** (Local version of Cohen-Tal). *There exist universal constants $C, c > 0$ such that for every $n, r, d \in \mathbb{N}$ such that $r \leq c\log(n)$ and $d \leq 2^{\sqrt{\log n}}$, the following holds. For any degree r polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$, there exists a d-local affine subspace $X \subseteq \mathbb{F}_2^n$ of dimension*

$$k \geq Cr(dn\log n)^{1/r}$$

on which f is constant.

This is tight: there exists a degree r polynomial $g \in \mathbb{F}_2[x_1, \ldots, x_n]$ which is an extractor for d-local affine sources of dimension $k \geq Cr(dn\log n)^{1/r}$, which has error $\varepsilon = 2^{-ck/r}$.

We prove Theorem 3 by extending the techniques of Cohen and Tal [7], while leveraging a key new ingredient: a "low-weight" Chevalley-Warning theorem. This result, which may be of independent interest, shows that any small system of low-degree polynomials admits a (nontrivial) solution of low Hamming weight.

▶ **Theorem 4** (Low-weight Chevalley-Warning). *Let* $\{f_i\} \subseteq \mathbb{F}_2[x_1, \ldots, x_n]$ *be a set of polynomials with degree*[2] *at most* $D < n$ *and nonlinear degree*[3] *at most* Δ *such that* 0 *is a common solution. Then there is a common solution* $x \neq 0$ *of Hamming weight*

$$w \leq 24\Delta + 2D/\log(n/D).$$

1.2 Open problems

Our work leaves open several interesting avenues for future work. We highlight three of them:

- For any constant $r \geq 2$, does there exist an explicit $\mathbb{F}_2$-polynomial of degree r that extracts from 2-local sources of min-entropy $o(n)$?
- In Theorem 2, we showed a reduction from a d-local source of min-entropy k to a d-local non-oblivious bit fixing (NOBF) source of min-entropy $\Omega(k/2^d)$. It would be interesting to show a reduction to show a reduction from a d-local source of min-entropy k to a d-local NOBF source of min-entropy $\Omega(k/\text{poly}(d))$ (or show that such a reduction is impossible).
- Theorem 4 shows that if a collection of low-degree polynomials of total degree D and nonlinear degree Δ has the zero vector as a solution, then there exists a nonzero solution of weight at most $O(\Delta + D/\log(n/D))$. When $\Delta = 0$, this becomes asymptotically tight by the Hamming bound. Moreover, when $D = \Delta$, this will also be tight by picking the polynomial $f(x) = \sum_{1 \leq |S| \leq \Delta} \prod_{i \in S} x_i$. However, if we had $\Delta/2$ quadratic polynomials, will the upper bound of $O(\Delta)$ be tight?

2 Overview of our techniques

In this section, we provide an overview of the techniques that go into our three main results:

- An entropy *upper bound* for low-degree extraction from local sources (Theorem 1.1).
- An entropy *lower bound* for low-degree extraction from local sources (Theorem 1.2).
- A *barrier* for extracting from local sources using black-box affine extractors (Theorem 0).

Along the way, we will overview the several new key ingredients (Theorems 2, 3, and 4) that go into these main results.

2.1 Upper bounds

We begin by discussing our entropy upper bounds for low-degree extraction from local sources. By this, we mean that we upper bound the entropy *requirement* for extracting from d-local sources using degree $\leq r$ polynomials. In other words, we show that low-degree polynomials extract from local sources.

2.1.1 Low-degree extractors for local sources

We start by sketching the techniques behind our main upper bound (Theorem 1.1), which shows that most degree $\leq r$ polynomials are low-error extractors for d-local sources with min-entropy at least

$$k = O(2^d d^2 r \cdot (2^d n \log n)^{1/r}).$$

[2] The *degree* D is the sum of the degrees of the f_i's.
[3] The *nonlinear degree* is the sum of the degrees of the f_i's which have degree at least 2.

A strawman application of the probabilistic method

A natural first attempt at proving our result would use a standard application of the probabilistic method, which looks something like the following. First, let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a uniformly random polynomial of degree $\leq r$, meaning that each monomial of size $\leq r$ is included in f with probability $1/2$. Then, we let $\mathcal{X}$ be the family of d-local sources over $\{0,1\}^n$, each with min-entropy at least k. To prove that most of these polynomials are low-error extractors for this family, a standard application of the probabilistic method would suggest that we:

1. Prove that f is an extractor for a single $\mathbf{X} \in \mathcal{X}$ with extremely high probability.
2. Show that the family $\mathcal{X}$ does not contain too many sources.
3. Conclude, via the union bound, that f is an extractor for *every* $\mathbf{X} \in \mathcal{X}$ with high probability.

It is not too hard to complete Steps 2 and 3 in the above framework, but Step 1 turns out to be much more challenging. To see why, let us consider an arbitrary d-local source $\mathbf{X} \sim \{0,1\}^n$ with min-entropy at least k. By definition of d-local source, there exists some $m \in \mathbb{N}$ and functions $g_1, \ldots, g_n : \{0,1\}^m \to \{0,1\}$ such that each g_i depends on just d of its inputs, and such that given a uniform $\mathbf{Y} \sim \{0,1\}^m$, we have

$$\mathbf{X} = (g_1(\mathbf{Y}), g_2(\mathbf{Y}), \ldots, g_n(\mathbf{Y})).$$

Now, we want to argue that a random degree $\leq r$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ is a low-error extractor for $\mathbf{X}$. To do so, consider the function $F : \{0,1\}^m \to \{0,1\}$ defined as

$$F(y_1, \ldots, y_m) := (f \circ g)(y_1, \ldots, y_m) = f(g_1(y_1, \ldots, y_m), \ldots, g_n(y_1, \ldots, y_m)).$$

Notice that by the definition of $\mathbf{X}$ and by Definition 1.1 of extractor, we know that F is an extractor for $\mathbf{X}$ with error ε if

$$|\mathsf{bias}(F)| := \left| \Pr_{y \sim \mathbf{U}_m} [F(y) = 1] - \Pr[F(y) = 0] \right| \leq 2\varepsilon.$$

Thus, to argue that a random degree $\leq r$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ is a low-error extractor for $\mathbf{X}$, it suffices to argue that the function $F = f \circ g$ has low bias (with high probability over the selection of f).

Of course, the question now becomes: how can we ensure that F has low bias? We can start by noticing some properties of F. First, we know $F = f \circ g$, where f is a random degree $\leq r$ polynomial and g is a fixed function where each output bit depends on $\leq d$ input bits. Thus, it is not hard to argue that F will have degree $\leq rd$. Furthermore, since f is random and g is fixed, one may hope to argue that F is a *uniformly random* polynomial of degree $\leq rd$: in this case we would be done, since it is well-known that uniformly random low-degree polynomials have extremely low bias (with extremely high probability) [1].

Unfortunately, it is too much to hope that F is a uniformly random low-degree polynomial. Indeed, it is not hard to see that the distribution of F over degree $\leq rd$ polynomials depends heavily on the exact selection of g. Furthermore, for most selections of g, the random function F is *not* uniformly distributed over degree $\leq t$ polynomials for *any* t.

Thus, there is no obvious way to apply [1] in order to argue that F will have low bias. To proceed, it seems like we will somehow need to argue that the distribution of F over low-degree polynomials is guaranteed to have some specific *structure*, and then somehow argue that a random polynomial from any such structured distribution is guaranteed to have low-bias. Each of these steps seems quite challenging.

Reductions to the rescue

As it turns out, there is a simple trick we can use to greatly simplify the above approach. The key idea is to *reduce* local sources to a simpler class of sources. Given two familes $\mathcal{X}, \mathcal{Y}$ of distributions over $\{0,1\}^n$, we say that $\mathcal{X}$ *reduces* to $\mathcal{Y}$ if each $\mathbf{X} \in \mathcal{X}$ is (close to) a convex combination of $\mathbf{Y} \in \mathcal{Y}$.[4] Reductions are extremely useful, because of the following well-known fact: if $\mathcal{X}$ reduces to $\mathcal{Y}$, and $f : \{0,1\}^n \to \{0,1\}$ is an extractor for $\mathcal{Y}$, then f *is also an extractor for* $\mathcal{X}$.

Thus, in order to show that low-degree polynomials extract from d-local sources, a key new ingredient we use is a reduction from d-local sources to a simpler class of sources called *d-local non-oblivious bit-fixing (NOBF) sources* [4]. Using the above discussion, it then suffices to show that low-degree polynomials extract from d-local NOBF sources. Thus, we proceed by:

1. Defining local NOBF sources, and showing how we can appropriately tailor our previous attempt at the probabilistic method so that it works for local NOBF sources.
2. Providing a new reduction from local sources to local NOBF sources.

Low-degree extractors for local NOBF sources

A *d-local NOBF source* $\mathbf{X} \sim \{0,1\}^n$ is a natural specialization of a d-local source where the entropic bits of the source must show up "in plain sight" somewhere in the source.[5] More formally, a d-local NOBF source with min-entropy k is a random variable $\mathbf{X} \sim \{0,1\}^n$ for which there exist functions $g_1, \ldots, g_n : \{0,1\}^k \to \{0,1\}$ such that the following holds: each $g_i, i \in [n]$ depends on $\leq d$ input bits; for every $i \in [k]$ there is some $i' \in [n]$ such that $g_{i'}(y) = y_i$; and for uniform $\mathbf{Y} \sim \{0,1\}^k$ we have

$$\mathbf{X} = (g_1(\mathbf{Y}), g_2(\mathbf{Y}), \ldots, g_n(\mathbf{Y})).$$

In other words, some k "good" bits in $\mathbf{X}$ are uniform, and the remaining $n-k$ "bad" bits are d-local functions of the good bits.

We must now show that a random degree $\leq r$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ extracts from d-local NOBF sources with entropy k. As in our strawman application of the probabilistic method, consider an arbitary d-local NOBF source $\mathbf{X} = (g_1(\mathbf{Y}), \ldots, g_n(\mathbf{Y}))$ and let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a uniformly random degree $\leq r$ polynomial. To show that f extracts from all d-local NOBF sources, recall that we just need to show that the function $F : \{0,1\}^k \to \{0,1\}$ defined as

$$F(y) := (f \circ g)(y) = f(g_1(y), \ldots, g_n(y))$$

has extremely low bias with extremely high probability. Furthermore, recall that if we can show that F itself is a uniformly random low-degree polynomial, then we know via [1] that this is true.

It is still too much to hope that F is a uniform low-degree polynomial, but F is now "close enough in structure" to one so that we can make this work. To see why, we can first assume without loss of generality (by definition of local NOBF source) that $g_1(y) = y_1, \ldots, g_k(y) = y_k$. Thus, we can define $c_S \sim \{0,1\}$ as an independent uniform bit (for each $S \subseteq [n]$ of size $\leq r$) and write

[4] By this we mean that each $\mathbf{X} \in \mathcal{X}$ can be written in the form $\mathbf{X} = \sum_i p_i \mathbf{Y}_i$, where each $\mathbf{Y}_i \in \mathcal{Y}$, $\sum_i p_i = 1$, and $\mathbf{X}$ samples from $\mathbf{Y}_i$ with probability p_i.

[5] The relationship between local sources and local *NOBF* sources is not dissimilar to the relationship between error-correcting codes and *systematic* error-correcting codes.

$$F(y) = \sum_{S\subseteq[k]:|S|\leq r} c_S \prod_{i\in S} g_i(y) + \sum_{S\subseteq[n]:|S|\leq r, S\not\subseteq[k]} c_S \prod_{i\in S} g_i(y) = A(y) + B(y),$$

where $A \in \mathbb{F}_2[y_1, \ldots, y_k]$ is a uniformly random polynomial of degree $\leq r$, and $B \in \mathbb{F}_2[y_1, \ldots, y_k]$ is a random polynomial whose selection of monomials is *not uniformly random*, but is nevertheless *independent* of the selections made by A.

Thus, to show that F has extremely low bias, it suffices to show that $A+B$ has extremely low bias. And to show that $A+B$ has extremely low bias, it suffices to show that $A+B'$ has low bias for any fixed polynomial B' induced by fixing the random monomials selected by B.

To conclude, we actually show something stronger: recalling that $A \in \mathbb{F}_2[y_1, \ldots, y_k]$ is a uniformly random polynomial of degree $\leq r$, we show that: for *any* fixed function $B^* : \{0,1\}^k \to \{0,1\}$, it holds that $A + B^*$ has low bias. This does not follow immediately from the result [1] that a random low-degree polynomial has low bias: indeed, it is more general, since [1] is the special case where $B^* = 0$. However, it *does* follow immediately from known upper bounds on the list size of Reed-Muller code [14].

In the language we are using here, an upper bound on the list size of a Reed-Muller code is equivalent to saying that for any fixed function $D \in \mathbb{F}_2[x_1, \ldots, x_k]$, A will differ from D on many inputs, with very high probability. Thus, such bounds tell us that A differs from B^* on many inputs with very high probability, and A differs from $1 + B^*$ (or rather, *equals* B^*) on many inputs with very high probability. In other words, A is completely uncorrelated with B^*, meaning that $\mathsf{bias}(A + B^*)$ is extremely small, as desired.

Thus a random low-degree polynomial f extracts from the d-local NOBF source $\mathbf{X}$ with min-entropy k with very high probability. In other words, it fails to do so with some very small probability $\delta = \delta(k)$ which decreases rapidly as k grows. By applying the union bound, we get that f extracts from the entire family $\mathcal{X}$ of d-local NOBF sources, provided $\delta(k) \cdot |\mathcal{X}| \ll 1$. All that remains is to upper bound the size of $\mathcal{X}$, which can easily be done using the d-locality of the sources.

A reduction to local NOBF sources

Above, we saw that random low-degree polynomials extract from *local NOBF sources*. To complete the proof that they also extract from more general *local sources*, recall that we need to provide a reduction from local sources to local NOBF sources. In other words, we need to show that every d-local source with min-entropy k is (close to) a convex combination of d-local NOBF sources with min-entropy $k' \approx k$. This is the main key ingredient in our result that low-degree polynomials extract from local sources (Theorem 1.1).

Our reduction works as follows. First, pick an arbitrary d-local source $\mathbf{X} \sim \{0,1\}^n$ with min-entropy k. Let k' be a parameter which is slightly smaller than k, which will be picked later. We start by arguing that $\mathbf{X}$ is (close to) a convex combination of d-local NOBF sources where there are k' good bits, but the good bits may be biased (but not constant).

Towards this end, recall that $\mathbf{X} = (g_1(\mathbf{Y}), \ldots, g_n(\mathbf{Y}))$ for some d-local functions $g_1, \ldots, g_n : \{0,1\}^m \to \{0,1\}$ and uniform $\mathbf{Y} \sim \{0,1\}^m$. The key idea is to consider the *largest possible set* $T \subseteq [n]$ of "good bits," i.e., such that $\{\mathbf{X}_i\}_{i\in T}$ are independent (and none are constants). Then, we let $T' \subseteq [m]$ be the bits of $\mathbf{Y}$ on which $\{\mathbf{X}_i\}_{i\in T}$ depend. The key observation is that *every* bit in $\mathbf{X}$ depends on *some* bit in $\{\mathbf{Y}_i\}_{i\in T'}$, by the maximality of T. Using this observation, there are *two possible cases*, over which we perform a *win-win analysis*.

First, it is possible that T contains $\geq k'$ bits. In this case, we consider fixing all bits $\{\mathbf{Y}_i\}_{i \notin T'}$. It is then not too hard to show that $\mathbf{X}$ becomes a source which contains $\geq k'$ good bits (which are mutually independent and not constants), and the remaining bad bits in $\mathbf{X}$ are deterministic d-local functions of these good bits.[6] Thus in this case, we get that $\mathbf{X}$ is a convex combination of NOBF sources of the desired type.

Second, it is possible that T contains $< k'$ bits. In this case, we consider fixing all bits $\{\mathbf{Y}_i\}_{i \in T'}$. But since all bits in $\mathbf{X}$ depend on *some* bit in this set, this fixing *decrements* the locality $d \to d-1$. And furthermore, since this fixes $|T'| \leq d|T| < dk'$ bits, the entropy only decreases from $k \to k - dk'$ by the entropy chain rule. We then recurse until we hit the first case, or until we hit $d = 1$. If we eventually hit the first case, we already know that $\mathbf{X}$ is a convex combination of NOBF sources of the desired type. On the other hand, it is easy to show that a 1-local source is actually a 1-local NOBF source! Thus we will always arrive at a (biased) d'-local NOBF source with $d' \leq d$, proving that $\mathbf{X}$ is always convex combination of NOBF sources of the desired type. Depending on when this recursion stops, we will arrive at an NOBF source with the number of good bits equal to at least

$$\min\{k', k - dk', k - d(d-1)k', \ldots, k - k' \prod_{i \in [d]} i\} \geq \min\{k', k - d^2 k'\},$$

which is always at least k' provided $k' \leq \frac{k}{2d^2}$.

Thus we see that any d-local source with min-entropy k can be written as a convex combination of d-local NOBF source with $\Omega(k/d^2)$ good bits, where the good bits are mutually independent (and nonconstant), but they may be heavily biased. So all that remains is to show that such biased d-local NOBF sources can be written as a convex combination of *unbiased* d-local NOBF sources (as they were originally defined). This step is not difficult, by applying a standard Chernoff bound. However, since each good bit depends on up to d bits, each such good bit $\mathbf{X}_i$ may have $|\mathsf{bias}(\mathbf{X}_i)| = 1 - 2 \cdot 2^{-d}$. As a result, we end up with $\Omega(\frac{k}{d^2 2^d})$ unbiased good bits.

This completes the reduction from local to local NOBF sources. Given our earlier proof sketch that low-degree polynomials extract from local NOBF sources, we finally get that low-degree polynomials also extract from local sources, as desired.

2.1.2 Low-degree dispersers for local sources

We now proceed to sketch the proof of Remark 1.3, which shows that most degree $\leq r$ polynomials are dispersers for d-local sources with min-entropy at least

$$k = O(d^2 r \cdot (dn \log n)^{1/r}).$$

This improves our min-entropy requirement for extractors (which was $k = O(2^d d^2 r \cdot (2^d n \log n)^{1/r})$) by removing two terms of the form 2^d. We use a different key idea to remove each 2^d term. While the outer exponential term 2^d is the more dramatic one to remove, it turns out that it is also the easier one. To do this, we simply note that for dispersers, we can forego the last step in our local to local NOBF reduction, which incurs a factor of 2^d by making the biased local NOBF source into an unbiased one. This improves the entropy requirement for dispersers from

$$k = O(2^d d^2 r \cdot (2^d n \log n)^{1/r}) \to k = O(d^2 r \cdot (2^d n \log n)^{1/r}).$$

[6] Technically we need to fix a little more randomness to make this happen, but this can be done without much trouble by invoking some standard tricks from the extractor literature.

Removing the inner exponential term

Next, we focus on improving the entropy requirement for dispersers from

$$k = O(d^2 r \cdot (2^d n \log n)^{1/r}) \to k = O(d^2 r \cdot (dn \log n)^{1/r}),$$

turning the inner exponential term 2^d into d. This improvement is more challenging: while our first improvement relied on improving the local to local-NOBF reduction, this improvement relies on improving the entropy requirement for dispersing from local NOBF sources.

In order to show that low-degree polynomials extract from local NOBF sources, recall that we: (1) showed that a random low-degree polynomial extracts from an arbitrary local NOBF source with extremely high probability; and (2) used a union bound over the family $\mathcal{X}$ of local NOBF sources to conclude that it extracts from *all* local NOBF sources with high probability. To get our second improvement on the min-entropy requirement for dispersers, we get improved upper bounds on the size of $\mathcal{X}$, so that our union bound is over fewer terms.

Towards this end, the key new idea is to show that in order to disperse from the family of d-local NOBF sources $\mathcal{X}$, it actually suffices to disperse from the much smaller family $\mathcal{X}'$ of so-called *d-local, degree $\leq r$ NOBF sources.*[7] This source family is the exact same as d-local NOBF sources, except it has the added restriction that the bad bits (which still depend on $\leq d$ good bits each) can each be written as a degree $\leq r$ polynomials. However, this family is significantly smaller: natural estimates on the sizes of $\mathcal{X}, \mathcal{X}'$ give

$$|\mathcal{X}| \leq \binom{n}{k} \cdot \left(\binom{k}{d} \cdot 2^{2^d} \right)^{n-k},$$
$$|\mathcal{X}'| \leq \binom{n}{k} \cdot \left(\binom{k}{d} \cdot 2^{\binom{d}{\leq r}} \right)^{n-k}.$$

After plugging in these improved size bounds, it is straightforward calculation to see that the inner 2^d term from the entropy requirement drops out. So all that remains is to show the above claim that a disperser for d-local, degree $\leq r$ NOBF sources automatically works for the more general family of d-local NOBF sources.

The key ingredient that goes into this claim is a simple lemma on polynomial decomposition. We show the following: for any function $f : \{0,1\}^n \to \{0,1\}$, any degree $\leq r$ polynomials $a_1, \ldots, a_n : \{0,1\}^k \to \{0,1\}$, and any polynomials $b_1, \ldots, b_n : \{0,1\}^k \to \{0,1\}$ that have no monomials of size $\leq r$, the following holds. There exists a polynomial $h : \{0,1\}^k \to \{0,1\}$ with no monomials of size $\leq r$ such that

$$f(a_1(y) + b_1(y), \ldots, a_n(y) + b_n(y)) = f(a_1(y), \ldots, a_n(y)) + h(y).$$

Then, given an arbitrary d-local NOBF source $\mathbf{X} \sim \{0,1\}^n$, the idea is to write it in the form

$$\mathbf{X} = (a_1(\mathbf{Y}) + b_1(\mathbf{Y}), \ldots, a_n(\mathbf{Y}) + b_n(\mathbf{Y})),$$

where $\mathbf{Y} \sim \{0,1\}^k$ is uniform and a_i, b_i are as before. Using our polynomial decomposition lemma, it then (roughly) holds that f is a disperser for $\mathbf{X}$ if f is a disperser for the simpler class of d-local, degree $\leq r$ NOBF sources. More precisely, we actually require from f a property that is ever-so-slightly stronger than being a disperser: we require that for any d-local, degree $\leq r$ NOBF source $\mathbf{X}' = (a_1(\mathbf{Y}), \ldots, a_n(\mathbf{Y}))$, it holds that the polynomial

[7] Technically, we require something slightly stronger than dispersion from such sources, but this does not make a huge difference. We will go into more details below.

$f(a_1, \ldots, a_n)$ has a monomial of degree $\leq r$. Our polynomial decomposition lemma then guarantees that f will also have this property for the more general d-local source, since the polynomial h in our decomposition lemma does not have any monomials of degree $\leq r$. Intuitively, h is not able to "destroy" the monomial of degree $\leq r$ guaranteed to pop out of $f(a_1(y), \ldots, a_n(y))$.

Thus, it suffices to "disperse" from d-local, degree $\leq r$ NOBF sources in order to disperse from more general d-local NOBF sources, meaning that we can leverage our improved bound on the size of $\mathcal{X}'$ to get our claimed improvement on the disperser's entropy requirement.

2.2 Lower bounds

We now discuss our entropy lower bounds for low-degree extraction from local sources. By this, we mean that for every degree $\leq r$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ we can find a d-local source $\mathbf{X} \sim \{0,1\}^n$ with relatively high min-entropy k on which f is constant. In other words, we show that in order to disperse (and thus extract) from d-local sources, they must have min-entropy exceeding this value k.

We show that every degree $r \leq \Omega(\log n)$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ must admit a d-local source $\mathbf{X} \sim \{0,1\}^n$ of min-entropy at least

$$k = \Omega(r(dn \log n)^{1/r})$$

on which it is constant. That is, we sketch the proof of our lower bound (Theorem 1.2).

In order to prove this result, we actually prove a slightly stronger result: we show that we can find a d-local source $\mathbf{X} \sim \{0,1\}^n$ with the above parameters such that it is also *affine*.

Our starting point is a tight result of Cohen and Tal [7], which shows that any degree $\leq r$ polynomial $f : \mathbb{F}_2^n \to \mathbb{F}_2$ admits a subspace $V \subseteq \mathbb{F}_2^n$ of dimension $\Omega(rn^{1/(r-1)})$ on which it is constant. Here, we obtain a (tight) granular version of their result, and show that any degree $\leq r$ polynomial f admits a d-local subspace $X \subseteq \mathbb{F}_2^n$ of dimension $k = \Omega(r(dn \log n)^{1/r})$ on which it is constant. Here, we say that V is *d-local* if V has a basis $v_1, \ldots, v_k \in \mathbb{F}_2^n$ such that for any index $i \in [n]$, at most d of these basis vectors equal 1 at this index. It is straightforward to verify that the uniform distribution $\mathbf{X}$ over V is a d-local source with min-entropy k, so we focus now on proving the existence of such a V.

At a high level, the proof of Cohen and Tal proceeds by iteratively growing a subspace V on which f is constant. At each phase, they define a set $A \subseteq \mathbb{F}_2^n$ such that f is constant over $\mathsf{span}(V, x)$ for every $x \in A$. They note that if $|A|$ has size $> 2^{\mathsf{dim}(V)}$, then of course there is some $x \in A \setminus V$ and furthermore we already know that f is constant on $\mathsf{span}(V, x)$. Thus, they can grow their monochromatic subspace by one dimension.

In order to get a lower bound on $|A|$, they note that this set can be defined as the common solutions to a small collection of low-degree polynomials. A classical result known as the *Chevalley-Warning theorem* (Theorem 5.1) then shows that $|A| \geq 2^{n-t}$, where t is the sum of degrees across the collection of polynomials. To complete their proof, they grow their subspace V until they are no longer able to show $|A| > 2^{\mathsf{dim}(V)}$.

In our lower bound, we show that f is monochromatic on a *d-local subspace*. To prove this, we start with the same approach as Cohen and Tal. However, at each phase, we add extra constraints to A which guarantee the following: if we take any $x \in A$ and add it to our current subspace V (with basis, say, $v_1, \ldots, v_{|\dim(V)|}$), then the location of the 1s appearing in vectors $x, v_1, \ldots, v_{|\dim(V)|}$ satisfy the d-locality constraint defined above. Again, as long as A is large enough, we can find some $x \in A$ that grows the dimension of our d-local subspace.

In order to ensure that A remains large for as many iterations as possible, we would like to minimize the impact of the new "locality" constraints that we have added to A. Given the description of these constraints above, we observe that these constraints are minimized if we grow V by carefully selecting vectors that have the lowest possible Hamming weight. However, we now need an upper bound on the Hamming weight of the lightest (nontrivial) common solution to a system of polynomial equations. Thus, our key new ingredient will be a result of this type, which we call a "low-weight Chevalley-Warning theorem."

A low-weight Chevalley-Warning theorem

Above, we saw how the classical Chevalley-Warning theorem is critical in lower bounding the size of A, thereby showing that there is some (nontrivial) vector $v \in A$ by which we can grow our monochromatic subspace. Now, we need an additional guarantee that there is such a $v \in A$ that also has low Hamming weight. We prove such a result, and call it a *low-weight Chevalley-Warning theorem*. Our theorem roughly says the following. Given a collection of polynomials that have cumulative degree D (and a common solution 0), if most of these polynomials have degree ≤ 1 then they admit a nontrivial solution of Hamming weight

$$w \leq O(D/\log(n/D)).$$

In order to prove our result, we start by using the CLP lemma [8] (a result which was instrumental in the recent resolution of the cap set conjecture) in order to show that for any large enough set $A \subseteq \mathbb{F}_2^n$ of common solutions to a system of polynomial equations, it holds that $A + A$ also contains a (nontrivial) common solution to this system. We then combine this result with an argument that is similar in flavor to the classical proof of the Hamming bound, and thereby obtain our low-weight Chevalley-Warning theorem. Equipped with this new ingredient, our entropy lower bound for low-degree extraction from d-local affine spaces follows immediately via the proof sketch described above.

2.3 A barrier

To conclude our overview, we provide a proof sketch of our barrier result (Theorem 0), which shows that affine extractors (applied in a black-box manner) cannot extract from local sources with min-entropy $k = \Omega(\sqrt{n})$, even if the locality is 2. More formally, to show that an affine extractor also extracts from a different family $\mathcal{Q}$ of distributions with min-entropy k, the standard technique is to show that each source $\mathbf{Q} \in \mathcal{Q}$ with min-entropy k is (close to) a convex combination of affine sources with min-entropy slightly less than k. Here, we show that this is simply not possible for local sources with min-entropy $\sqrt{n}$. In particular, we show that there is a very simple 2-local source $\mathbf{Q} \sim \{0,1\}^n$ with min-entropy $\Omega(\sqrt{n})$ that has statistical distance *exponentially close to* 1 from any convex combination of affine sources with min-entropy k'.

In more detail, we consider the 2-local "clique" source $\mathbf{Q} \sim \{0,1\}^n$ defined as follows: first, pick any $\ell \in \mathbb{N}$ and set $n := \ell + \binom{\ell}{2}$. Then, pick uniform and independent bits $\mathbf{q}_1, \ldots, \mathbf{q}_\ell \sim \{0,1\}$ and set $\mathbf{Q}$ to be the concatenation of all $\mathbf{q}_i$ over $1 \leq i \leq n$ and $\mathbf{q}_i \cdot \mathbf{q}_j$ over $1 \leq i < j \leq \ell$. Now, let $\mathbf{X} \sim \{0,1\}^n$ be a convex combination of affine sources, each with min-entropy ℓ'. We argue that $|\mathbf{Q} - \mathbf{X}| \geq 1 - 2^{-\Omega(\ell')}$ by showing that for any ℓ'-dimensional affine $\mathbb{F}_2$-subspace $S \subseteq \{0,1\}^n$, it holds that $|\mathsf{supp}(\mathbf{Q}) \cap S|/|S| \leq 2^{-\Omega(\ell')}$. That is, we wish to show that $Q = \mathsf{supp}(\mathbf{Q})$ is *subspace-evasive*.

To show that cliques are subspace-evasive, we use the following key observation: For any nonempty set $Q' \subseteq Q$ of cliques, the set $Q' + Q' := \{u + v : u \in Q', v \in Q', u \neq v\}$ (where the sum is over $\mathbb{F}_2^n$) has a "Sidon property:" each element x in $Q' + Q'$ has a unique

pair $u, v \in Q'$ such that $x = u + v$. This observation is proven by noticing that by making another copy of each coordinate of the form $\mathbf{q}_i \cdot \mathbf{q}_j$, the set Q will correspond precisely to the symmetric rank-1 matrices of $\mathbb{F}_2^{\ell \times \ell}$. Thus all the elements in $Q' + Q'$ would correspond to symmetric $\mathbb{F}_2^{\ell \times \ell}$ matrices of rank at most 2. Hence by looking at the row space of $x \in Q' + Q'$, we can precisely find its symmetric rank-1 decomposition. That is, we can find $u, v \in Q'$ such that $x = u + v$. Now, pick $Q' = Q \cap S$. Since addition is closed in S, we see that $Q' + Q' \subseteq S$. Thus $|S| \geq |Q' + Q'| \geq \binom{|Q'|}{2} \geq \Omega(|Q \cap S|^2)$. Hence we find that $|Q \cap S|/|S| \leq O(\sqrt{|S|}/|S|) = O(1/\sqrt{|S|})$, which is $2^{-\Omega(\ell')}$ as $|S| = 2^{\ell'}$.

For more details, we refer the reader to the full version of this paper.

3 Preliminaries

We briefly outline some basic notation, definitions, and facts that will be used throughout the paper. We first discuss some notation. We use log to denote the base-2 logarithm, we define $[n] := \{1, 2, \ldots, n\}$, and we write $\binom{n}{\leq r} := \sum_{i=0}^{r} \binom{n}{i}$. Given a random variable $\mathbf{X}$, we let $\mathsf{supp}(\mathbf{X})$ denote its support and write $\mathbf{X} \sim S$ to denote that $\mathsf{supp}(\mathbf{X}) \subseteq S$. Finally, we use $\mathbf{U}_m$ to denote the uniform distribution over $\{0,1\}^m$.

We now discuss some basic notions from probability. First, the *statistical distance* between two random variables $\mathbf{X}, \mathbf{Y} \sim S$ is denoted by $\Delta(\mathbf{X}, \mathbf{Y})$ and defined as $\Delta(\mathbf{X}, \mathbf{Y}) := \max_{T \subseteq S} |\Pr[\mathbf{X} \in T] - \Pr[\mathbf{Y} \in T]| = \frac{1}{2} \sum_{s \in S} |\Pr[\mathbf{X} = s] - \Pr[\mathbf{Y} = s]|$. Moreover, we say $\mathbf{X}$ and $\mathbf{Y}$ are *ε-close*, denoted $\mathbf{X} \approx_\varepsilon \mathbf{Y}$, if $\Delta(\mathbf{X}, \mathbf{Y}) \leq \varepsilon$. Finally, recalling the definition of min-entropy from the introduction, we will use the following simple fact about this quantity. The proof is straightforward, and can be found in the full version.

▶ **Lemma 3.1.** *Suppose $\mathbf{X}$ and $\mathbf{Y}$ are arbitrary random variables such that $\mathbf{Y}$ is uniformly distributed over its support. Then, for every $y \in \mathsf{supp}(\mathbf{Y})$, it holds that*

$$H_\infty(\mathbf{X}|\mathbf{Y} = y) \geq H_\infty(\mathbf{X}) - \log |\mathsf{supp}(\mathbf{Y})|.$$

4 Entropy upper bounds

In this section, we obtain upper bounds on the entropy required to extract from d-local sources using degree $\leq r$ polynomials (Theorem 1.1). To do so, we combine two key ingredients. Our first key ingredient reduces d-local sources to d-local NOBF sources:

▶ **Theorem 4.1** (Theorem 2, restated)**.** *There exists a universal constant $c > 0$ such that for any $n, k, d \in \mathbb{N}$, the following holds. Let $\mathbf{X} \sim \{0,1\}^n$ be a d-local source with min-entropy $\geq k$. Then $\mathbf{X}$ is ε-close to a convex combination of d-local NOBF sources with min-entropy $\geq k'$, where $\varepsilon = 2^{-ck'}$ and*

$$k' = \frac{ck}{2^d d^2}.$$

Our second key ingredient gives upper bounds on the entropy required to extract from d-local NOBF sources using degree $\leq r$ polynomials.

▶ **Theorem 4.2** (Low-degree polynomials extract from d-local NOBF sources)**.** *There are universal constants $C, c > 0$ such that for all $n, d, r \in \mathbb{N}$, the following holds. With probability at least 0.99 over the choice of a random degree $\leq r$ polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$, it holds that f is an ε-extractor for d-local NOBF sources of min-entropy*

$$k = Cr \cdot (2^d \cdot n \log n)^{1/r}$$

with error $\varepsilon = 2^{-ck/r^3}$.

By combining the above two theorems, we immediately get Theorem 1.1. Moreover, if we only care about *dispersing* (instead of *extracting*), it turns out that we can streamline our arguments and remove the 2^d terms in Theorems 4.1 and 4.2, yielding Remark 1.3.

In the remainder of this section, we focus on proving our reduction, Theorem 4.1. But before doing so, we briefly note that to prove Theorem 4.2, the main idea is to show that a random low-degree polynomial has low correlation with any fixed function. This observation follows quite readily from known bounds [14] on the list-size of Reed-Muller codes, and Theorem 4.2 is not too hard to show given this observation. For a formal proof of Theorem 4.2, we refer the reader to Section 2 and the full version of this paper, where one can also find more details regarding Remark 1.3.

4.1 A reduction from d-local sources to d-local NOBF sources

We now turn towards proving our reduction, Theorem 4.1. In order to reduce d-local sources to d-local NOBF sources, we use an intermediate model called a *biased d-local NOBF source*.

▶ **Definition 4.3** (Biased local NOBF sources). *A random variable $\mathbf{X} \sim \{0,1\}^n$ is a (δ, k)-*biased d-local NOBF source *if there exists a set $S \subseteq [n]$ of size k such that both of the following hold:*

- *The bits in $\mathbf{X}_S$ are mutually independent (but need not be identically distributed), and each $\mathbf{X}_i, i \in S$ has bias $|\Pr[\mathbf{X}_i = 1] - \Pr[\mathbf{X}_i = 0]| \leq \delta$.*
- *Every other bit $\mathbf{X}_j, j \notin S$ is a deterministic function of at most d bits in $\mathbf{X}_S$.*

▶ Remark 4.4. A d-local NOBF source with entropy k is a $(0, k)$-biased d-local NOBF source.

Given this intermediate model, we prove Theorem 4.1 by combining two lemmas. The first lemma reduces d-local sources to biased d-local NOBF sources:

▶ **Lemma 4.5.** *Let $\mathbf{X} \sim \{0,1\}^n$ be a d-local source with min-entropy $\geq k$. Then $\mathbf{X}$ is a convex combination of (δ, k')-biased d-local NOBF sources, where $\delta \leq 1 - 2^{-d}$ and $k' \geq k/(2d^2)$.*

The second lemma reduces biased d-local NOBF sources to (unbiased) d-local NOBF sources.

▶ **Lemma 4.6.** *Let $\mathbf{X} \sim \{0,1\}^n$ be a (δ, k')-biased d-local NOBF source. Then $\mathbf{X}$ is ε-close to a convex combination of $(0, k'')$-biased d-local NOBF sources, where $k'' \geq (1-\delta)k'/4$ and $\varepsilon = 2^{-k''/4}$.*

By combining these two lemmas, Theorem 4.1 follows immediately. Lemma 4.6 is not too difficult to prove by simulating each biased bit with two consecutive independent coin flips (one with bias $|1 - 2\delta|$ and one with bias 0) and applying a standard Chernoff bound over the result of the first coin flip. We refer to the full version for more details, and conclude this section by proving Lemma 4.5.

Proof of Lemma 4.5. Let $\mathbf{X} \sim \{0,1\}^n$ be a d-local source with min-entropy $\geq k$. The key observation that we will prove is that for any t, one of the following *must* hold: either

- $\mathbf{X}$ is a convex combination of (δ, t)-biased d-local NOBF sources, for $\delta \leq 1 - 2^{-d}$; or
- $\mathbf{X}$ is a convex combination of $(d-1)$-local sources with min-entropy $> k - td$.

Before we prove this key observation, let us see how we can use it to prove the desired result. First, recall that convex combinations "stack" in the following sense: if a source $\mathbf{X}$ is a convex combination of convex combinations of sources from a family $\mathcal{X}$, then $\mathbf{X}$ is just a convex combination of sources from $\mathcal{X}$. Thus, by repeatedly applying the key observation until either the first item becomes true or we arrive at a 1-local source (the "base case"), we see that $\mathbf{X}$ is a convex combination of sources $\{\mathbf{Z}_i\}$, where each $\mathbf{Z}_i$ is either:

- A (δ, t)-biased d-local NOBF source, for $\delta \leq 1 - 2^{-d}$; or
- A 1-local source with min-entropy $> k - t \cdot (d + (d-1) + \cdots + 2) = k - t \cdot (d^2 + d - 2)$.

However, it is clear from the definitions that a 1-local source with min-entropy k' is a 1-local NOBF source with min-entropy k'. Furthermore, it is easy to see that a 1-local NOBF source with min-entropy $\geq k'$ is a convex combination of 1-local NOBF sources with min-entropy exactly k', by fixing any additional random "good" bits. Thus, for any $t \leq k'$, we know that a 1-local source with min-entropy $\geq k'$ is a convex combination of (δ, t)-biased d-local NOBF sources, for $\delta \leq 1 - 2^{-d}$.

By the above discussion, we see that for any $t \leq k - t \cdot (d^2 + d - 2)$, $\mathbf{X}$ is a convex combination of (δ, t)-biased d-local NOBF sources, where $\delta \leq 1 - 2^{-d}$. Setting $t = \frac{k}{2d^2}$ yields the result.

Thus, all that remains is to prove the key observation stated at the beginning of the proof. Towards this end, let $\mathbf{X} \sim \{0,1\}^n$ be a d-local source with min-entropy $\geq k$. By definition of d-local source, there exists some ℓ and $f : \{0,1\}^\ell \to \{0,1\}^n$ such that $\mathbf{X} = f(\mathbf{Y})$ for uniform $\mathbf{Y} \sim \mathbf{U}_\ell$, such that each bit $\mathbf{X}_i$ is a deterministic function of at most d bits in $\mathbf{Y}$. In other words, there exist sets $S_1, \ldots, S_n \subseteq [\ell]$ of size d and functions $f_1, \ldots, f_n : \{0,1\}^d \to \{0,1\}^n$ such that

$$\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n) = (f_1(\mathbf{Y}_{S_1}), f_2(\mathbf{Y}_{S_2}), \ldots, f_n(\mathbf{Y}_{S_n})).$$

Now, let $T \subseteq [n]$ be any set of coordinates of *maximal size* such that:

- $H_\infty(\mathbf{X}_i) > 0$ for all $i \in T$; and
- $S_i \cap S_j = \emptyset$ for any distinct $i, j \in T$.

Suppose T has size τ. Without loss of generality, assume $T = [\tau]$. We conclude with two cases.

Case (i): $\tau < t$. In this case, we fix the random variable $\mathbf{Y}_{S_1}, \ldots, \mathbf{Y}_{S_\tau}$. We know that with probability 1 over this fixing, all bits $\mathbf{X}_i, i \in [n]$ become deterministic functions of at most $d-1$ unfixed variables in $\mathbf{Y}$, by the maximality of T and its intersection property. In other words, $\mathbf{X}$ becomes a $(d-1)$-local source. Furthermore, by Lemma 3.1, we know that with probability 1 over this fixing, $\mathbf{X}$ loses $\sum_{i \in [\tau]} |S_i| = d\tau < dt$ bits of min-entropy. Thus in this case, $\mathbf{X}$ is a convex combination of $(d-1)$-local sources of min-entropy $> k - dt$.

Case (ii): $\tau \geq t$. In this case, define $\overline{S} := [n] - (\bigcup_{i \in [\tau]} S_i)$ and notice that $S_1, S_2, \ldots, S_\tau, \overline{S}$ partition the coordinates of $\mathbf{Y}$. Next, define the random variables $\mathbf{Z}_i := \mathbf{Y}_{S_i}$ for each $i \in [\tau]$, and define $\overline{\mathbf{Z}} := \mathbf{Y}_{\overline{S}}$. Notice that $\mathbf{X}_i = f_i(\mathbf{Z}_i)$ for each $i \in [\tau]$. Furthermore, it is straightforward to verify that for all $j > \tau$, there exists a set $Q_j \subseteq [\tau]$ of size at most d and a deterministic function f'_j such that $\mathbf{X}_j = f'_j(\mathbf{Z}_{Q_j}, \overline{\mathbf{Z}})$. In other words, we can rewrite $\mathbf{X}$ as

$$\begin{aligned}\mathbf{X} &= (\mathbf{X}_1, \ldots, \mathbf{X}_\tau, \mathbf{X}_{\tau+1}, \ldots, \mathbf{X}_n) \\ &= (f_1(\mathbf{Z}_1), \ldots, f_\tau(\mathbf{Z}_\tau), f'_{\tau+1}(\mathbf{Z}_{Q_{\tau+1}}, \overline{\mathbf{Z}}), \ldots, f'_n(\mathbf{Z}_{Q_n}, \overline{\mathbf{Z}})).\end{aligned}$$

Now, for each $i \in [\tau]$, define $\mathbf{A}_i := f_i(\mathbf{Z}_i)$. Furthermore, it is straightforward to show that we can define a new random variable $\mathbf{B}$ independent of $\mathbf{Y}$, and for each $i \in [\tau]$ a deterministic function g_i such that $g_i(\mathbf{A}_i, \mathbf{B}) = \mathbf{Z}_i$ for all $i \in [\tau]$. Thus, for any subset $Q \subseteq [\tau]$ we have $\mathbf{Z}_Q = g'_Q(\mathbf{A}_Q, \mathbf{B})$ for some deterministic function g'_Q. And finally, for each $j > \tau$ there must be some deterministic function ψ_j such that

$$f'_j(\mathbf{Z}_{Q_j}, \overline{\mathbf{Z}}) = \psi_j(\mathbf{A}_{Q_j}, \mathbf{B}, \overline{\mathbf{Z}}).$$

Thus we can rewrite $\mathbf{X}$ as:

$$\mathbf{X} = (\mathbf{A}_1, \ldots, \mathbf{A}_\tau, \psi_{\tau+1}(\mathbf{A}_{Q_{\tau+1}}, \mathbf{B}, \overline{\mathbf{Z}}), \ldots, \psi_n(\mathbf{A}_{Q_n}, \mathbf{B}, \overline{\mathbf{Z}})).$$

Notice that the collection $\{\mathbf{A}_i\}_{i\in[\tau]}$ are mutually independent, and each has bias at most $1 - 2^{-d}$ since it is a non-constant deterministic function of d uniform bits. Thus no matter how $\mathbf{B}, \overline{\mathbf{Z}}$ are fixed, $\mathbf{X}$ becomes a (δ, t)-biased d-local NOBF source, for $\delta \leq 1 - 2^{-d}$. ◀

5 Entropy lower bounds

In this section, we obtain lower bounds on the entropy required to extract from d-local sources using degree $\leq r$ polynomials (Theorem 1.2). To do so, we actually prove a stronger theorem, which can be viewed as a *local* version of a result by Cohen and Tal [7]. In particular, Cohen and Tal show that any low degree polynomial admits a large subspace on which it is constant. We show that this holds even for this special subclass of *local* subspaces.

▶ **Theorem 3** (Theorem 3, restated). *There exist universal constants $C, c > 0$ such that for every $n, r, d \in \mathbb{N}$ such that $r \leq c\log(n)$ and $d \leq 2^{\sqrt{\log n}}$, the following holds. For any degree r polynomial $f \in \mathbb{F}_2[x_1, \ldots, x_n]$, there exists a d-local subspace $X \subseteq \mathbb{F}_2^n$ of dimension*

$$k \geq Cr(dn\log n)^{1/r}$$

on which f is constant.

This is tight: there exists a degree r polynomial $g \in \mathbb{F}_2[x_1, \ldots, x_n]$ which is an extractor for d-local affine sources of dimension $k \geq Cr(dn\log n)^{1/r}$, which has error $\varepsilon = 2^{-ck/r}$.

Notice that this immediately implies Theorem 1.2, since (the uniform distribution over) a d-local subspace is not only an affine source, but it is also a d-local source.

The tightness claim in Theorem 3 is not too difficult to prove. To do so, one can simply use Gaussian elimination to observe that a d-local affine source is actually a d'-local *NOBF* source (for some d' that is not guaranteed to be equal to d). Then, the tightness claim follows via a standard application of the probabilistic method, using known bounds [1] on the bias of a random degree $\leq r$ polynomial.

The main part of Theorem 3 (preceding the tightness claim) is much more challenging to prove. The key new ingredient we rely on is a so-called "low-weight Chevalley-Warning theorem," which may be of independent interest. We present and prove this theorem in the following subsection. As discussed in Section 2, it is then not too difficult to use our low-weight Chevalley-Warning theorem to obtain Theorem 3, and we refer the reader to the full version for more details.

5.1 A low-weight Chevalley-Warning theorem

The classical *Chevalley-Warning theorem* guarantees that a small set of low-degree polynomials admits a common nontrivial solution:

▶ **Theorem 5.1** (Chevalley-Warning theorem [24]). *Let $\{f_i\} \subseteq \mathbb{F}_2[x_1, \ldots, x_n]$ be a set of polynomials with degree at most D such that 0 is a common solution. Then there are at least 2^{n-D} common solutions to $\{f_i\}$. In particular, if $D < n$, then there must be a nontrivial common solution.*

In this subsection, we prove a "low-weight" version of this theorem. In particular, it is natural to ask not only if $\{f_i\}$ contains a nontrivial common solution, but if $\{f_i\}$ contains a nontrivial common solution of *low Hamming weight* w. It is straightforward to use Theorem 5.1 to show that $w \leq D+1$, and we remark that this is tight in general. However, we show that if most of the polynomials in $\{f_i\}$ are linear, then we can improve this bound to roughly $w \leq O(D/\log(n/D))$.

▶ **Theorem 5.2.** *Let $\{f_i\} \subseteq \mathbb{F}_2[x_1, \ldots, x_n]$ be a set of polynomials with degree at most $D < n$ and nonlinear degree at most Δ such that 0 is a common solution. Then for any w satisfying*

$$\binom{n}{\leq \lfloor w/2 \rfloor} > 2^{D+1} \cdot \binom{n}{\leq \lfloor \Delta/2 \rfloor}, \tag{1}$$

there exists a nontrivial common solution with Hamming weight at most w.

It is straightforward to show that $w = 24\Delta + 2D/\log(n/D)$ satisfies inequality 1, yielding Theorem 4. The main ingredient that goes into the proof of Theorem 5.2 is the following lemma, which says that for any big enough set $A \subseteq \mathbb{F}_2^n$ of common solutions to a system of low-degree polynomials, it holds that $A + A$ also contains a (nontrivial) common solution.

▶ **Lemma 5.3.** *Let $\{f_i\} \subseteq \mathbb{F}_2[x_1, \ldots, x_n]$ be a set of polynomials with nonlinear degree at most Δ such that 0 is a common solution. Then for any set $A \subseteq \mathbb{F}_2^n$ of common solutions of size*

$$|A| > 2\binom{n}{\leq \lfloor \Delta/2 \rfloor}$$

it holds that $A + A$ contains a nontrivial common solution.

In order to prove this, we will make use of the CLP lemma, which was instrumental in the recent resolution of the cap set conjecture.

▶ **Lemma 5.4** (CLP lemma [8])**.** *Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ be a polynomial of degree at most r, and let M denote the $2^n \times 2^n$ matrix with entries $M_{x,y} = f(x+y)$ for $x, y \in \mathbb{F}_2^n$. Then*

$$\mathsf{rank}(M) \leq 2\binom{n}{\leq \lfloor r/2 \rfloor}.$$

Next, we show how to prove Lemma 5.3 using Lemma 5.4. Then, we conclude this section by showing how to obtain Theorem 5.2 from Lemma 5.3.

Proof of Lemma 5.3. First, let $\{g_i\} \subseteq \{f_i\}$ be the set of polynomials in $\{f_i\}$ that have degree > 1. Notice that if $A + A$ contains a nontrivial common solution to the system $\{g_i\}$, then it also contains a nontrivial common solution to $\{f_i\}$: this follows from the linearity of the polynomials of $\{f_i\} - \{g_i\}$ and the fact that every $a \in A$ is a common solution (by definition of A). Thus, it suffices to show the result for the set $\{g_i\}$.

Next, consider the polynomial $g \in \mathbb{F}_2[x_1, \ldots, x_n]$ defined as

$$g(x) := \prod_i (1 + g_i(x)).$$

It is straightforward to verify that g has degree at most Δ, and that $g(x) = 1$ if and only if x is a common solution to $\{g_i\}$. Now, suppose for contradiction that $A + A$ contains no nontrivial common solution to $\{g_i\}$: that is, for every distinct $x, y \in A$ it holds that $g(x+y) = 0$. Then, consider the $2^n \times 2^n$ matrix M with entries $M_{x,y} = g(x+y)$ for every

$x, y \in \mathbb{F}_2^n$. Define $k := |A|$, and let $M[A, A]$ denote the $k \times k$ submatrix of M obtained by taking the rows and columns of M indexed by A. Since 0 is a common solution to $\{g_i\}$, we get that $M[A, A] = I_k$ and thus

$$\mathsf{rank}(M) \geq \mathsf{rank}(M[A, A]) = \mathsf{rank}(I_k) = k > 2\binom{n}{\leq \lfloor \Delta/2 \rfloor}$$

which directly contradicts Lemma 5.4. ◀

Finally, we conclude this subsection with our proof of Theorem 5.2.

Proof of Theorem 5.2. Let $A \subseteq \mathbb{F}_2^n$ be the collection of common solutions to $\{f_i\}$. Fix any $w \in [1, n]$ such that all nonzero common solutions to $\{f_i\}$ have weight $> w$. Then for any Hamming ball $\mathcal{B} \subseteq \mathbb{F}_2^n$ of radius $\lfloor w/2 \rfloor$, it must hold that

$$|\mathcal{B} \cap A| \leq 2\binom{n}{\leq \lfloor \Delta/2 \rfloor}, \tag{2}$$

because otherwise Lemma 5.3 tells us there exist distinct $x, y \in \mathcal{B} \cap A$ such that $x + y \in A$, and $x + y$ is a nonzero vector with weight at most $2\lfloor w/2 \rfloor \leq w$ (by the triangle inequality).

Now, given any vector $v \in \mathbb{F}_2^n$, let $\mathcal{B}(v)$ denote the Hamming ball around v of radius $\lfloor w/2 \rfloor$. We consider the quantity $\sum_{v \in A} |\mathcal{B}(v)|$, and seek to sandwich it between two inequalities. By definition of Hamming ball we know that each $|\mathcal{B}(v)| = \binom{n}{\leq \lfloor w/2 \rfloor}$, and by Theorem 5.1 we know that $|A| \geq 2^{n-D}$. Combining these observations with inequality 2, we get that

$$2^{n-D}\binom{n}{\leq \lfloor w/2 \rfloor} \leq \sum_{v \in A} |\mathcal{B}(v)| \leq 2^n \cdot 2\binom{n}{\leq \lfloor \Delta/2 \rfloor},$$

since each $u \in \mathbb{F}_2^n$ is contained by at most $2\binom{n}{\leq \lfloor \Delta/2 \rfloor}$ balls (with radius $\lfloor w/2 \rfloor$) centered at a common solution $a \in A$, by inequality 2 (i.e., consider the set $\mathcal{B}(u) \cap A$). Thus

$$\binom{n}{\leq \lfloor w/2 \rfloor} \leq 2^{D+1} \cdot \binom{n}{\leq \lfloor \Delta/2 \rfloor}.$$

In summary, we have shown that any $w \in [1, n]$ for which all nonzero common solutions to $\{f_i\}$ have weight $> w$ must satisfy the above inequality. The result follows. ◀

References

1. Ido Ben-Eliezer, Rani Hod, and Shachar Lovett. Random low-degree polynomials are hard to approximate. *Comput. Complex.*, 21(1):63–81, 2012. doi:10.1007/s00037-011-0020-6.
2. Andrej Bogdanov and Siyao Guo. Sparse extractor families for all the entropy. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 553–560. Association for Computing Machinery, 2013. doi:10.1145/2422436.2422496.
3. Eshan Chattopadhyay and Jesse Goodman. Improved extractors for small-space sources. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 610–621, 2021. doi:10.1109/FOCS52979.2021.00066.
4. Eshan Chattopadhyay, Jesse Goodman, Vipul Goyal, and Xin Li. Extractors for adversarial sources via extremal hypergraphs. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1184–1197, 2020.
5. Kuan Cheng and Xin Li. Randomness extraction in AC^0 and with small locality. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, pages 37:1–37:20, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.37.

6 Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988. doi:10.1137/0217015.

7 Gil Cohen and Avishay Tal. Two structural results for low degree polynomials and applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, pages 680–709, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.680.

8 Ernie Croot, Vsevolod F Lev, and Péter Pál Pach. Progression-free sets in are exponentially small. *Annals of Mathematics*, pages 331–337, 2017.

9 Anindya De and Thomas Watson. Extractors and lower bounds for locally samplable sources. *ACM Trans. Comput. Theory*, 4(1), March 2012. doi:10.1145/2141938.2141941.

10 Yevgeniy Dodis and Kevin Yeo. Doubly-affine extractors, and their applications. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*, pages 13:1–13:23, 2021. doi:10.4230/LIPIcs.ITC.2021.13.

11 Oded Goldreich, Emanuele Viola, and Avi Wigderson. On randomness extraction in AC^0. In *30th Conference on Computational Complexity (CCC 2015)*, pages 601–668, 2015. doi:10.4230/LIPIcs.CCC.2015.601.

12 Xuangui Huang, Peter Ivanov, and Emanuele Viola. Affine extractors and AC^0-parity. *Electron. Colloquium Comput. Complex.*, page 137, 2021. URL: https://eccc.weizmann.ac.il/report/2021/137.

13 Jesse Kamp, Anup Rao, Salil Vadhan, and David Zuckerman. Deterministic extractors for small-space sources. *J. Comput. Syst. Sci.*, 77(1):191–220, 2011. doi:10.1016/j.jcss.2010.06.014.

14 Tali Kaufman, Shachar Lovett, and Ely Porat. Weight distribution and list-decoding size of Reed–Muller codes. *IEEE Transactions on Information Theory*, 58(5):2689–2696, 2012. doi:10.1109/TIT.2012.2184841.

15 Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *Electron. Colloquium Comput. Complex.*, volume 28, page 23, 2021.

16 Xin Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 168–177, 2016. doi:10.1109/FOCS.2016.26.

17 Chi-Jen Lu. Encryption against storage-bounded adversaries from on-line strong extractors. *J. Cryptol.*, 17(1):27–42, 2004. doi:10.1007/s00145-003-0217-1.

18 Luca Trevisan and Salil Vadhan. Extracting randomness from samplable distributions. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 32–42, 2000. doi:10.1109/SFCS.2000.892063.

19 Salil P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. Cryptol.*, 17(1):43–77, 2004. doi:10.1007/s00145-003-0237-x.

20 Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *Comput. Complex.*, 13(3-4):147–188, 2005. doi:10.1007/s00037-004-0187-1.

21 Emanuele Viola. Extractors for circuit sources. *SIAM Journal on Computing*, 43(2):655–672, 2014. doi:10.1137/11085983X.

22 Emanuele Viola. Quadratic maps are hard to sample. *ACM Trans. Comput. Theory*, 8(4), June 2016. doi:10.1145/2934308.

23 Emanuele Viola. Sampling lower bounds: Boolean average-case and permutations. *SIAM Journal on Computing*, 49(1):119–137, 2020. doi:10.1137/18M1198405.

24 Ewald Warning. Bemerkung zur vorstehenden arbeit von Herrn Chevalley. *Abh. Math. Sem. Univ. Hamburg*, 11:76–83, 1936.

Decremental Matching in General Graphs

Sepehr Assadi
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Aaron Bernstein
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Aditi Dudeja
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Abstract

We consider the problem of maintaining an approximate maximum integral matching in a dynamic graph G, while the adversary makes changes to the edges of the graph. The goal is to maintain a $(1+\varepsilon)$-approximate maximum matching for constant $\varepsilon > 0$, while minimizing the update time. In the fully dynamic setting, where both edge insertion and deletions are allowed, Gupta and Peng (see [29]) gave an algorithm for this problem with an update time of $O(\sqrt{m}/\varepsilon^2)$.

Motivated by the fact that the $O_\varepsilon(\sqrt{m})$ barrier is hard to overcome (see Henzinger, Krinninger, Nanongkai, and Saranurak [30]; Kopelowitz, Pettie, and Porat [34]), we study this problem in the *decremental* model, where the adversary is only allowed to delete edges. Recently, Bernstein, Probst-Gutenberg, and Saranurak (see [9]) gave an $O(\text{poly}(\log n/\varepsilon))$ update time decremental algorithm for this problem in *bipartite graphs*. However, beating $O(\sqrt{m})$ update time remained an open problem for *general graphs*.

In this paper, we bridge the gap between bipartite and general graphs, by giving an $O_\varepsilon(\text{poly}(\log n))$ update time algorithm that maintains a $(1+\varepsilon)$-approximate maximum integral matching under adversarial deletions. Our algorithm is randomized, but works against an adaptive adversary. Together with the work of Grandoni, Leonardi, Sankowski, Schwiegelshohn, and Solomon [26] who give an $O_\varepsilon(1)$ update time algorithm for general graphs in the *incremental* (insertion-only) model, our result essentially completes the picture for partially dynamic matching.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic algorithms, matching, primal-dual algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.11

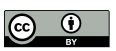
Category Track A: Algorithms, Complexity and Games

Funding *Sepehr Assadi*: Supported in part by an NSF CAREER Grant CCF-2047061, and a gift from Google Research.
Aaron Bernstein: Funded by NSF CAREER Grant 1942010.

1 Introduction

In dynamic graph algorithms, the main goal is to maintain a key property of the graph while an adversary makes changes to the edges of the graph. An algorithm is called *incremental* if it handles only insertions, *decremental* if it handles only deletions and *fully dynamic* if it handles both insertions as well as deletions. The goal is to minimize the update time of the algorithm, which is the time taken by the algorithm to adapt to a single adversarial edge insertion or deletion and output accordingly. For incremental/decremental algorithms, one typically seeks to minimize the *total update time*, which is the aggregate sum of update times over the *entire* sequence of edge insertions/deletions.

We consider the problem of maintaining a $(1+\varepsilon)$-approximation to the maximum matching in a dynamic graph. In the fully dynamic setting, the best known update time for this problem is $O(\sqrt{m}/\varepsilon^2)$ (see Gupta and Peng [29]), and the conditional lower bounds proved in the works



of Henzinger, Krinninger, Nanongkai, and Saranurak (see [30]) and Kopelowitz, Pettie, and Porat (see [34]) suggest that $O(\sqrt{m})$ is a hard barrier to break through. For this reason, several relaxations of this problem have been studied. For example, one line of research has shown that we can get considerably faster update times if we settle for large approximation factors (see for example [13, 14, 16, 11, 41, 37, 6, 33, 27, 5, 39, 15, 8, 12, 10, 11, 7]). Another research direction has been to consider the more relaxed incremental or decremental models. In the incremental (insertion-only) setting, there have been a series of upper and lower bound results (see [17, 19, 28]), culminating in the result of Grandoni, Leonardi, Sankowski, Schwiegelshohn, and Solomon (see [26]), who gave an optimal $O_\varepsilon(m)$ *total* update time (amortized $O_\varepsilon(1)$) for $(1+\varepsilon)$-approximate maximum matching.

The decremental (deletion-only) setting requires an entirely different set of techniques. In fact, even for the special case of bipartite graphs, $O_\varepsilon(m\sqrt{m})$ total time ($O_\varepsilon(\sqrt{m})$ update time) remained the best known until recently, when Bernstein, Probst-Gutenberg, and Saranurak (see [9]) gave an $\text{poly}(\log n/\varepsilon)$ amortized update time algorithm for the case of bipartite graphs. However, achieving a similar result for general graphs remained an open problem. Our main theorem essentially closes the gap between bipartite and general graphs.

▶ **Theorem 1.** *Let G be an unweighted graph and $\varepsilon \in (0,1)$. There is a decremental algorithm with total update time $O_\varepsilon(m \cdot \text{poly}(\log n))$ (amortized $O_\varepsilon(\text{poly}(\log n))$) that maintains a matching M of size at least $(1-\varepsilon)\cdot\mu(G)$ with high probability. Here G refers to the current version of the graph and $\mu(G)$ is the size of the maximum matching of G. The algorithm is randomized but works against an adaptive adversary. The dependence on ε is $2^{O(1/\varepsilon^2)}$.*

The guarantees of our algorithm hold against an *adaptive adversary*: which is allowed to choose an update sequence adaptively. This is in contrast to an *oblivious adversary* which cannot decide its updates based on the algorithm's output. Deterministic algorithms are more desirable because they are robust against such updates, which allows them to be used as a black-box in other static/dynamic algorithms. This property doesn't hold when we have the weaker oblivious adversary assumption. Thus, even though our algorithm is *randomized* (Monte Carlo), it has has the same power as a deterministic algorithm. We refer the reader to Section 1 of [41] and the references therein for a detailed discussion on this.

Our result largely completes the picture for partially dynamic matching by showing that in general graphs one can achieve $\text{poly}(\log n)$ update time in both incremental and decremental settings. But there are a few secondary considerations that remain. Firstly, our update time is $O_\varepsilon(\text{poly}(\log n))$, rather than the $O_\varepsilon(1)$ for the incremental setting (see [26]). Secondly, both the incremental result of [26] and our decremental result for general graphs have an exponential dependence on $1/\varepsilon$, whereas incremental/decremental algorithms for bipartite graphs have a polynomial dependence on $1/\varepsilon$ (see [26, 28]). Optimizing the dependence on ε and $\text{poly}(\log(n))$ factors thus remains an interesting open question for future work.

In algorithms literature, it has been the case that efficient matching algorithms for bipartite graphs do not easily extend to general graphs. Existence of blossoms (among other things), poses a technical challenge to obtaining analogous results for the general case. Consider the polynomial time algorithms for maximum matching for bipartite graphs, the most efficient algorithm, using alternating BFS, was discovered by Hopcroft and Karp; Karzanov (see [31, 32]) in 1973. However, several new structural facts and algorithmic insights were used by Micali and Vazirani to get the same runtime for general graphs (see [36]). This is also a feature of recent work in different models, such as the streaming (see [1, 22] and [24]), fully dynamic (see [10, 11] and [14, 7]), and parallel models (see [23, 40] and [35, 3]). We refer the reader to Section 1.3 of [2] for a detailed discussion of this phenomenon.

2 High-Level Overview

Our algorithm for Theorem 1 follows the high-level framework of *congestion balancing* introduced in [9]. They used it to solve approximate decremental matching in bipartite graphs, and also to solve more general flow problems. But the framework as they used it was entirely limited to cut/flow problems. As we discuss below, extending this framework to general graphs introduces significant technical challenges. Moreover, our result shows how the key subroutine of congestion balancing is naturally amenable to a primal-dual analysis, which we hope can pave the way for this technique to be applied to other decremental problems. Throughout this paper, we will use $\tilde{O}(\cdot)$ to hide $\text{poly}(\log n)$ factors in big-oh notations.

2.1 Previous Techniques

A fractional matching is a non-negative vector $\vec{x} \in \mathbb{R}_{\geqslant 0}$ satisfying fractional matching constraints: for all $v \in V$, $\sum_{e \ni v} x(e) \leqslant 1$. The starting point of [9] is that it is sufficient to develop an $\tilde{O}_\varepsilon(m)$ algorithm that does the following: it either maintains a *fractional matching* of size at least $(1-2\varepsilon) \cdot \mu(G)$ or certifies that $\mu(G)$ has dropped by a $(1-\varepsilon)$ factor because of adversarial deletions. Since a result of [41] enables us to round any bipartite fractional matching to an integral matching of almost the same value, such a fractional algorithm yields an algorithm to maintain an integral matching of size at least $(1-3\varepsilon) \cdot \mu(G)$ in $\tilde{O}_\varepsilon(m)$ total time under adversarial deletions. To motivate why [9] consider computing a fractional matching, consider the following "lazy" algorithm that works with an integral matching: compute an $(1+\varepsilon)$ approximate integral matching M of G using a static $O(m/\varepsilon)$ algorithm, wait for $\varepsilon \cdot \mu(G)$ edges of M to be deleted and then recompute the matching. Since we assume an adaptive adversary, the update time could be as large as $\Omega(m^2/\varepsilon \cdot n)$; this is because the adversary could proceed by only deleting edges of M. As a result, the goal should be to maintain a *robust* matching that can survive many deletions. Thus, the algorithm in [9] maintains a "balanced" fractional matching $\vec{x}$ that attempts to put a low value on every edge. In order to reduce the value of $\vec{x}$ by $\varepsilon \cdot \mu(G)$, the adversary will have to delete a lot of edges from every large matching.

Balanced Fractional Matching in Bipartite Graphs. In order to ensure that the fractional matching is spread out and robust, the algorithm in [9] imposes a capacity function κ on the edges of the graph (initially, all edges have low capacity) and compute a fractional matching obeying these capacities. The main ingredient of the algorithm is the subroutine M-OR-E*(G, ε, κ) which returns one of the following in $\tilde{O}_\varepsilon(m)$ time:

1. A fractional matching $\vec{x}$ with $\sum_{e \in E} x(e) \geqslant (1-\varepsilon) \cdot \mu(G)$, $x(e) \leqslant \kappa(e)$ for all $e \in E$, or,
2. A set of edges E^* with the following two properties.
   a. The total capacity through E^* must be small: $\kappa(E^*) = O(\mu(G) \log n)$ and,
   b. For all $|M| \geqslant (1-3\varepsilon) \cdot \mu(G)$, $|M \cap E^*| \geqslant \varepsilon \cdot \mu(G)$.

Property 2a ensures that the total capacity increase is small, while Property 2b ensures that we only increase capacity on important edges that are actually needed to form a large matching. The authors of [9] show that M-OR-E*() can be used as a black-box to solve decremental matching: at each step, M-OR-E*() is used to find a large fractional matching $\vec{x}$ (this matching is then rounded using [41] to get an integral matching), or to output the set E^* along which we increase capacities. They are able to show that because of Properties 2a and 2b, the edge capacities remain small on average.

The *congestion balancing* framework of [9] consists of an outer algorithm that uses M-OR-E*() as a subroutine. The outer algorithm for bipartite graphs, with some challenges, carries over to general graphs as well. But, M-OR-E*() is significantly more challenging to

implement for general graphs, so this subroutine will be our focus for the rest of the high level review. For bipartite graphs the algorithm M-OR-E*() is easier to implement because maximum fractional matchings correspond to max flows in bipartite graphs. Hence, existing algorithms for approximate maximum flows can be used to find the approximate maximum fractional matching obeying capacity κ. Moreover, if such a fractional matching is not large, then in bipartite graphs, the set of bottleneck edges E^* is exactly a minimum cut of the graph. For general graphs, due to the odd set constraints, max flow, which was the key analytic and algorithmic tool in [9], no longer corresponds to a maximum fractional matching that avoids the integrality gap.

2.2 Our Contribution: Implementing M-or-E*() in General Graphs

At a high-level, there are several structural and computational challenges to implementing M-OR-E*() in the case of general graphs. We explain what the potential impediments are, and detail how our techniques circumvent these.

Fractional Matchings in General Graphs. In general graphs, not all fractional matchings have a large integral matching in their support and therefore, cannot be rounded to give a large matching. While fractional matchings that obey odd set constraints do avoid the integrality gap, it seems hard to compute such a matching that also obeys capacity function κ. In order to get past this, we define a candidate fractional matching that is both easy to compute as well as contains a large integral matching in its support. More concretely, our fractional matching either puts flow one through an edge, or a flow of value at most ε (technically, we put flow much smaller than ε, but for this discussion, ε is sufficient). It is known that such a fractional matching has an integrality gap of at most $1+\varepsilon$, since it obeys all small odd set constraints. Our main contributions are two structural lemmas which show that we can find our candidate matching efficiently.

1. First, given a graph G with capacity κ, we want to determine if the *value* of the maximum fractional matching obeying κ and odd set constraints (denoted $\mu(G,\kappa)$) is at least $(1-\varepsilon)\cdot\mu(G)$. In general graphs, we do this by giving a sampling theorem: let G_s be the graph created by sampling edge e with probability proportional to $\kappa(e)$, then $\mu(G_s) \geqslant \mu(G,\kappa) - \varepsilon\cdot n$ with high probability. Thus, $\mu(G_s)$ is a good proxy for $\mu(G,\kappa)$ and it can be estimated efficiently by running any integral matching algorithm on G_s.
2. Suppose we have determined at some point that $\mu(G,\kappa)$ is large, we are still left with the task of finding a fractional matching. Our next contribution is a structural theorem that enables us to deploy existing flow algorithms to find such a matching. Let M be an approximate maximum matching of G_s. Let $M_L = \{e \in M \mid \kappa(e) \leqslant \beta\}$ and $M_H = \{e \in M \mid \kappa(e) > \beta\}$, where L and H are for low and high respectively and $\beta = O(1/\text{poly}\log n)$. Let $V_L = V(M_L)$ and $V_H = V(M_H)$. Intuitively, M breaks up our vertex set into two parts: vertices matched by low capacity edges (denoted V_L) and those that are matched by high capacity edges (denoted V_H). By adding some slack to our capacity constraints (we show that some slack can be incorporated in congestion balancing framework), we are able to treat the high-capacity edges as integral and compute a matching on $G[V_H]$ using a black box for integral matching in general graphs. Additionally, we show that the maximum fractional matching on low capacity edges of $G[V_L]$ has value at least as much as $|M_L|$ up to an additive error of $\varepsilon\cdot n$. To compute this fractional matching $\vec{f}$, we show that since we are only considering edges of small capacity, small odd set constraints are automatically satisfied, so we can transform $G[V_L]$ into a *bipartite graph* and then use an existing flow algorithm. We then output $M_H + \vec{f}$, which has the property that either the flow through an edge is 1 or at most ε.

The second obstacle is finding the set E^*. Recall, for bipartite graphs, the max flow-min cut theorem gives us an easy characterization of the bottleneck edges. However, for general graphs, this characterization is unclear. To overcome this, we consider the dual of the matching LP of G_s, and show that the bottleneck edges can be identified by considering the dual constraints associated with the edges. This generalizes the cut-or-matching approach of [9]. We compute an integral matching in G_s, and since the algorithm of [20] is primal-dual, we are also able to compute approximate duals for G_s.

Additionally, there are some secondary technical challenges as well. As mentioned before, our structural theorems only guarantee preservation of matching sizes up to an additive error of $\varepsilon \cdot n$. When $\mu(G) = o(n)$, then the results, applied directly are insufficient for us. To get around this, we use a vertex sparsification technique to get $O_\varepsilon(\log n)$ multigraphs which preserve all large matchings of G, but contain only $O(\mu(G)/\varepsilon)$ vertices (this reduction was first used in [4, 18]). However, we now have to show that all of our ideas work for multigraphs as well. In proving the second structural result, we need to introduce some slack in the capacities (see Definition 19). We show that congestion balancing extends to multigraphs, and is flexible enough to handle slack in capacities. Finally, the rounding scheme of [41] cannot be applied as a black-box to any fractional matching in a general graph. Thus, unlike in the bipartite case, we instead have to embed its techniques into the congestion balancing framework.

▶ Note 2. All our structural theorems (see Lemma 15 and Lemma 26) apply to graphs G with $\mu(G) \geqslant \Omega(\varepsilon \cdot n)$. A reduction used by the authors of [18, 4] allows us to construct $O_\varepsilon(\log n)$ multigraphs $H_1, \cdots, H_\lambda$ with $O(\mu(G)/\varepsilon)$ vertices, such that for every M of G with $|M| \geqslant (1-\varepsilon) \cdot \mu(G)$ at least $(1-\varepsilon) \cdot |M|$ edges of M are present in some H_i. Thus, at a high level, we have reduced our problem to the problem of decremental matching on multigraphs with large matchings. So, it is enough to prove our structural theorems for multigraphs with large matchings. We justify this assumption formally in the full version.

3 Preliminaries

We consider the problem of maintaining an approximate maximum integral matching in a graph G in the decremental setting. Throughout the paper, we will use G to refer to the current version of the graph, and let V and E be the vertex and edge sets of G respectively. Additionally, $\mu(G)$ denotes the size of the maximum integral matching of G. During the course of the algorithm, we will maintain a fractional matching $\vec{x}$. For a set $S \subseteq E$, we let $x(S) = \sum_{e \in S} x(e)$. Given a capacity function $\kappa(e)$, we say that x obeys κ if $x(e) \leqslant \kappa(e)$ for all $e \in E$. For a vector $\vec{x}$, we use $\text{supp}(\vec{x})$ to be set of edges that are in the support of $\vec{x}$. For a fractional matching $\vec{x}$, we say that $\vec{x}$ satisfies odd set constraints if for every odd-sized $B \subseteq V$, $\sum_{e \in G[B]} x(e) \leqslant \frac{|B|-1}{2}$. Given a capacity function κ on the edges of G, we use $\mu(G, \kappa)$ to denote the value of the maximum fractional matching of G that obeys the capacity function κ and the odd set constraints. Throughout this paper, for any $\varepsilon \in (0, 1)$, we will let $\alpha_\varepsilon = \log n \cdot 2^{60/\varepsilon^2}$ and $\rho_\varepsilon = \log n \cdot 2^{40/\varepsilon^2}$. We also need the following algorithm computing $(1+\varepsilon)$-approximate matching in general graphs.

▶ **Lemma 3** ([20, 25]). *There is an $O(m/\varepsilon)$ time algorithm Static-Match() that takes as input a graph G, and returns an integral matching M of G with $|M| \geqslant (1-\varepsilon) \cdot \mu(G)$.*

Roadmap for Extended Abstract. As mentioned in the overview, the main technical contribution of our paper is an algorithm for M-or-E*() in general graphs. The rest of the extended abstract focuses exclusively on this algorithm; the details of how M-or-E*() can be used a black box to attain Theorem 1 are left for the full version.

4 Main Contribution: M-or-E*() In General Graphs

As mentioned in the overview, we need our congestion balancing framework to act on multigraphs. This motivates our next few definitions.

▶ **Definition 4.** *Given a multigraph G, for a pair of vertices u and v, define $D(u,v)$ to be the set of edges between u and v. Similarly, if e is an edge between u, v, then $D(e) \coloneqq D(u,v)$.*

▶ **Definition 5.** *Let G be a multigraph with n vertices and m edges. Let κ be a capacity function on the edges. Suppose $\vec{x}$ is a fractional matching of G ($\vec{x}$ is a vector of length m). Then, we define $\vec{x}^C$ to be a vector of support size at most $\min\left\{m, \binom{n}{2}\right\}$, where for a pair of vertices, u and v, $x^C(u,v) \coloneqq \sum_{e \in D(u,v)} x(e)$. Essentially, if $\vec{x}$ is a fractional matching on a multigraph, then $\vec{x}^C$ is a fractional matching obtained by "collapsing" all edges together. Similarly, if $\vec{y}$ is a vector of support at most $\binom{n}{2}$, then we define $\vec{y}^D$ to be an m length vector such that for every $e \in E$ between a pair of vertices u and v, $y^D(e) \coloneqq \frac{y(u,v)\cdot\kappa(e)}{\kappa(D(e))}$ (D is for distributed, and we distribute the flow among the edges in proportion to their capacity).*

▶ Remark 6. Note that in doing the transformations in Definition 5, the support size of the transformed vector is always at most m. Thus, it doesn't negatively affect our runtime.

We now state our core ingredient, which either finds a balanced fractional matching of the multigraph G, or gives a set of edges E^* along which we can increase capacity.

▶ **Lemma 7.** *Let G be a multi-graph with $\mu(G) \geqslant \varepsilon \cdot n/16$. Let κ be a capacity function on the edges of G. There is an algorithm M-OR-E*(), that takes as input $G, \kappa, \varepsilon \in (0, 1/2)$ and $\mu \geqslant (1-\varepsilon)\cdot\mu(G)$ and in time $O(m \cdot \log n/\varepsilon)$ returns one of the following.*

(a) *A fractional matching $\vec{x}$ of value at least $(1-20\varepsilon)\cdot\mu$ with the following properties.*
- **(i)** *For any $e \in supp(\vec{x})$ with $\kappa(D(e)) > 1/\alpha_\varepsilon^2$, $x(e) = \frac{\kappa(e)}{\kappa(D(e))}$, and $x(D(e)) = 1$.*
- **(ii)** *For any $e \in supp(\vec{x})$ with $\kappa(D(e)) \leqslant 1/\alpha_\varepsilon^2$, $x(e) \leqslant \kappa(e)\cdot\alpha_\varepsilon$ and $x(D(e)) \leqslant \kappa(D(e))\cdot\alpha_\varepsilon$.*

(b) *A set E^* of edges such that $\kappa(E^*) = O(\mu \log n)$ such that for any integral matching M with $|M| \geqslant (1-3\varepsilon)\cdot\mu$, we have $|M \cap E^*| \geqslant \varepsilon\mu$. Moreover, $\kappa(e) < 1$ for all $e \in E^*$. Additionally, for every pair of vertices $u, v \in V$, either $D(u,v) \cap E^* = \emptyset$ or $D(u,v) \subseteq E^*$.*

We give some intuition for Lemma 7. Recall that we need a balanced fractional matching that contains a large integral matching in its support. However, as mentioned before, in the case of general graphs, finding such a balanced fractional matching is not straightforward. In order to get past this obstacle, we define a balanced fractional matching that is easy to find and also avoids the integrality gap. We will explain how to find it in the subsequent sections. For now, we explain at a high-level, why the fractional matching $\vec{x}$ found by Lemma 7 avoids the integrality gap. Consider $\vec{x}^C$. Observe from Lemma 7(a), that for any pair of vertices $u, v \in G$, either $x^C((u,v)) = 1$ or $x^C((u,v)) \leqslant 1/\alpha_\varepsilon \leqslant \varepsilon$. Thus, $\vec{x}$ satisfies odd-set constraints for all odd sets of size at most $1/\varepsilon$. By a folklore lemma, we can then argue that $\vec{x}$ contains an integral matching of size at least $(1+\varepsilon)^{-1} \cdot \sum_{u \neq v} x^C((u,v))$.

As mentioned before M-OR-E*() will be used as a subroutine in our **decremental algorithm**. The fractional matching output by M-OR-E*() will have certain properties, we state these properties now, since they will be helpful in visualizing the fractional matching. We give a proof of these in the full version.

▶ **Property 8.** *In our decremental algorithm, M-OR-E*$(G, \mu, \kappa, \varepsilon)$ outputs a fractional matching $\vec{x}$ with the following property: consider any $u, v \in V$ and let e, e' be edges between u, v (recall that G is a multigraph). Then, $\kappa(e') = \kappa(e)$ at all times during the algorithm.*

▶ **Definition 9.** *Let G be a multigraph, let $\varepsilon \in (0,1)$, let κ be a capacity function on the edges of G and let $\vec{x}$ be a fractional matching obeying κ. Then, we split $\vec{x}$ into two parts, $\vec{x}^f$ and $\vec{x}^i$, where $\vec{x} = \vec{x}^f + \vec{x}^i$, and $\text{supp}(\vec{x}^f) = \{e \in E \mid \kappa(D(e)) \leqslant 1/\alpha_\varepsilon^2\}$ and $\text{supp}(\vec{x}^i) = \{e \in E \mid \kappa(D(e)) > 1/\alpha_\varepsilon^2\}$ ($\vec{x}^f$ stands for fractional and $\vec{x}^i$ stands for integral. Though the edges in $\vec{x}^i$ do not have integral capacity, they are large enough to round them to 1).*

We briefly give the implications of this definition, since it is instructive to state them.

▶ **Property 10.** *Let G be any multigraph and $\vec{x}$ be the matching output by M-OR-E*$(G, \mu, \kappa, \varepsilon)$,*

(a) *For $\vec{x}$, we have $x(e) \leqslant \kappa(e) \cdot \alpha_\varepsilon^2$ for all $e \in E$. This follows immediately from Lemma 7(a).*

(b) *For any pair of vertices u, v, either $D(u,v) \subseteq supp(\vec{x}^i)$ and $D(u,v) \cap supp(\vec{x}^f) = \emptyset$ or, $D(u,v) \subseteq supp(\vec{x}^f)$ and $D(u,v) \cap supp(\vec{x}^i) = \emptyset$.*

(c) *Consider $\vec{z} = \vec{x}^i$. Then $supp(\vec{z}^C)$ is a matching. This is implied by Lemma 7(ai).*

5 Ingredients for Algorithm M-or-E*()

Recall we use $\mu(G, \kappa)$ to denote the value of the maximum fractional matching of G obeying capacity function κ and the odd set constraints. As in the congestion balancing set up of [9], we want to check if $\mu(G, \kappa) \geqslant (1 - \varepsilon) \cdot \mu(G)$. But, unlike in bipartite graphs, where we can use flows to find fractional matching, there is no simple way to check if $\mu(G, \kappa) \geqslant (1 - \varepsilon)\mu(G)$ in general graphs. Our first structural result circumvents this. Let G_s be obtained by sampling every edge e with probability $p(e) = \min\{1, \kappa(e) \cdot \rho_\varepsilon\}$. We show that $\mu(G_s) \geqslant \mu(G, \kappa) - \varepsilon\mu(G)$. Thus, Static-Match$(G_s, \varepsilon)$ is used to estimate $\mu(G, \kappa)$.

At a high level, M-OR-E*() proceeds in three phases. In Phase 1, it creates G_s and computes $\mu(G_s)$. If $\mu(G_s)$ is large, then by the above $\mu(G, \kappa)$ must also be large, so the algorithm proceeds to Phase 2, where it finds a fractional matching satisfying Lemma 7(a). On the other hand, if $\mu(G_s)$ is small, then it proceeds to Phase 3, where it finds the set of edges E^* satisfying Lemma 7(b), along which it increases capacity. In the subsequent sections, we will state the main structural properties we use in each of the phases. Finally, in Section 6, we put together these ingredients to give M-OR-E*(), and prove Lemma 7.

5.1 Phase 1 of M-or-E*()

Before we formally state the main guarantees of Phase 1, we will state some standard results in matching theory, that we will use in our main result for Phase 1.

5.1.1 Some Standard Ingredients For Phase 1

The first ingredient we use is the Tutte-Berge formula.

▶ **Definition 11.** *For any multigraph G and $U \subseteq V$, $odd_G(V \setminus U)$ denotes the number of odd components in $G[V \setminus U]$.*

▶ **Lemma 12** (Tutte-Berge Formula)**.** *[38] The size of a maximum matching in a graph $G = (V, E)$ is equal to $\frac{1}{2} \min_{U \subseteq V} (|U| + |V| - odd_G(V \setminus U))$.*

Additionally, we will use some properties of the matching polytope.

▶ **Lemma 13** ([38])**.** *Let G be any multigraph, let $\vec{x}$ be a fractional matching that in addition to the fractional matching constraints, also satisfies the following for all odd-sized $U \subseteq V$: $\sum_{e \in G[U]} x(e) \leqslant \frac{|U|-1}{2}$. Then, there is an integral matching $M \subseteq supp(x)$ with $|M| = \sum_{e \in E} x(e)$.*

▶ **Definition 14.** *Let G be any multigraph, and let $S, T \subseteq V$, then $\delta_G(S,T)$ is defined as the set of edges that have one endpoint in S and the other in T. Additionally, for $S \subseteq V$, we define $\delta_G(S)$ to be the set of edges that have one end point in S, and the other in $V \setminus S$.*

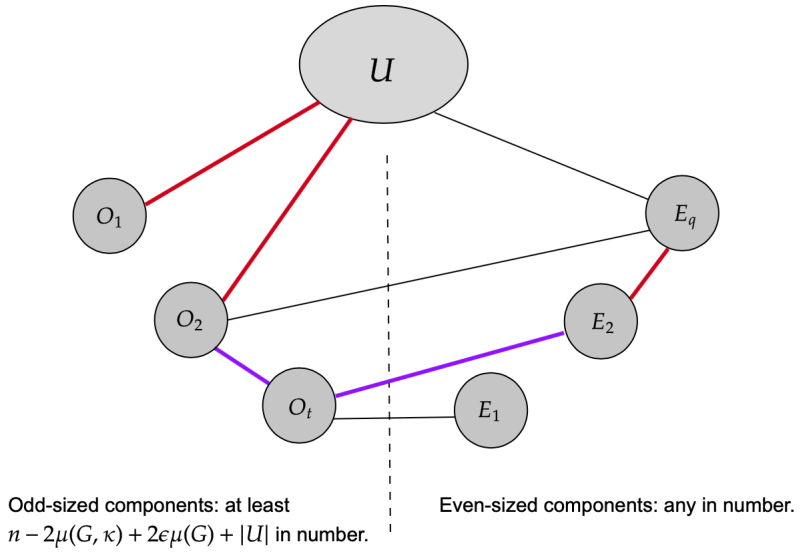


Figure 1 The figure shows the graph G, and a partition $\mathcal{P} = \{U, E_1, \cdots, E_q, O_1, \cdots, O_t\}$ satisfying properties (a) and (b) mentioned in proof of Lemma 15. The thick edges (in red and purple), are the edges in $\text{supp}(\vec{x})$, where $\vec{x}$ is the fractional matching realizing $\mu(G,\kappa)$. The purple edges (edges between the odd components, or between odd and even components) correspond to $E^{\mathcal{P}}_{\text{miss}}$, and $\sum_{e \in E^{\mathcal{P}}_{\text{miss}}} x(e) \geqslant 2 \cdot \varepsilon \cdot \mu(G)$.

5.1.2 Main Lemma for Phase 1

As mentioned earlier, in Phase 1 of M-OR-E*(), we first create a sampled graph G_s. In the following lemma, we show that $\mu(G_s)$ is a good estimate for $\mu(G,\kappa)$ with high probability.

▶ **Lemma 15.** *Let G be a multigraph with $\mu(G) \geqslant \varepsilon \cdot n/16$ where $\varepsilon \in (0, 1/2)$. Let κ be a capacity function on the edges of G, and let G_s be obtained by sampling every edge $e \in G$ with probability $p(e) = \min\{1, \kappa(e) \cdot \rho_\varepsilon\}$. Let $\mu(G,\kappa)$ be the value of the maximum fractional matching of G obeying the capacities κ, and the odd set constraints. Then, with high probability, $\mu(G_s) \geqslant \mu(G,\kappa) - \varepsilon \cdot \mu(G)$.*

Proof. We want to show that with high probability, $\mu(G_s) \geqslant \mu(G,\kappa) - \varepsilon \cdot \mu(G)$. In order to do this, by Lemma 12, it is sufficient to show that with high probability, $1 - 1/n^2$, $\frac{1}{2} \min_{U \subseteq V} (|U| + |V| - \text{odd}_{G_s}(V - U)) \geqslant \mu(G,\kappa) - \varepsilon \cdot \mu(G)$. Towards this, we consider a fixed partition $\mathcal{P}$ of V into sets $U, O_1, \cdots, O_t, E_1, \cdots, E_q$ with the following properties (see Figure 1).

(a) We have, $q \geqslant 0$ and $t > n - 2 \cdot \mu(G,\kappa) + 2\varepsilon \cdot \mu(G) + |U|$.

(b) Sets O_i for $i \in [t]$ are odd-sized sets and sets E_l for $l \in [q]$ are even-sized sets.

If $\mu(G_s) < \mu(G,\kappa) - \varepsilon \cdot \mu(G)$, then there is a partition $\mathcal{P} = \{U, O_1, \cdots, O_t, E_1, \cdots, E_q\}$ of G_s (from Lemma 12), satisfying (a) and (b) such that $G_s[V \setminus U]$ is the union of disconnected components $O_1, \cdots, O_t, E_1, \cdots, E_q$. If $G_s[V \setminus U]$ is the union of disconnected components $O_1, \cdots, O_t, E_1, \cdots, E_q$ then $\delta_{G_s}(O_i, O_l) = \emptyset$ for all $i \neq l$ and $\delta_{G_s}(O_i, E_l) = \emptyset$ for all $i \in [t]$,

$l \in [q]$. Thus, to upper bound the probability that $\mu(G_s) < \mu(G,\kappa) - \varepsilon \cdot \mu(G)$, it is sufficient to upper bound the probability that there exists a partition $\mathcal{P} = \{U, O_1, \cdots, O_t, E_1, \cdots, E_q\}$ satisfying (a) and (b), such that *none* of the edges $E^{\mathcal{P}}_{\mathsf{miss}} = \{e \mid e \in \delta_G(O_i, O_l) \text{ for } i \neq l\} \cup \{e \mid e \in \delta_G(O_i, E_l) \text{ for } i \in [t], l \in [q]\}$ are sampled in G_s. In order to bound this probability, we make the following claim.

▷ Claim 16. For a partition $\mathcal{P}$ satisfying (a) and (b), $\kappa(E^{\mathcal{P}}_{\mathsf{miss}}) \geqslant 2 \cdot \varepsilon \cdot \mu(G)$.

Proof. Let $\vec{x}$ be a fractional matching obeying odd set constraints and capacity function κ such that $\sum_{e \in E} x(e) = \mu(G,\kappa)$. We show that if $\kappa(E^{\mathcal{P}}_{\mathsf{miss}}) < 2 \cdot \varepsilon \cdot \mu(G)$, then, $x(E^{\mathcal{P}}_{\mathsf{miss}}) > \kappa(E^{\mathcal{P}}_{\mathsf{miss}})$, which will contradict the fact that $\vec{x}$ is a fractional matching obeying κ.

With this proof strategy in mind, for contradiction assume that $\kappa(E^{\mathcal{P}}_{\mathsf{miss}}) < 2 \cdot \varepsilon \cdot \mu(G) \leqslant n - 2 \cdot \mu(G,\kappa) + 2 \cdot \varepsilon \cdot \mu(G)$. The last inequality follows from the fact that $\mu(G,\kappa)$ corresponds to the value of the maximum fractional matching, so, $\mu(G,\kappa) \leqslant \frac{n}{2}$. Since $\vec{x}$ obeys odd set constraints, $\sum_{l \leqslant t} x(\delta_G(O_l)) \geqslant t$. Note that $\sum_{l \leqslant t} x(\delta_G(O_l, U)) \leqslant |U|$, otherwise for some $v \in U$, $\sum_{e \ni v} x(e) > 1$, violating the fact that $\vec{x}$ is a fractional matching. Next, we observe that $\sum_{l \leqslant t} x(\delta_G(O_l)) = x(E^{\mathcal{P}}_{\mathsf{miss}}) + \sum_{l \leqslant t} x(\delta_G(O_l, U))$. This follows from the fact that all edges emanating out of O_i in G, are incident on O_j for $j \neq i$, or E_k for some $k \in [q]$, or U. We have the following set of inequalities.

$$x(E^{\mathcal{P}}_{\mathsf{miss}}) = \sum_{l \leqslant t} x(\delta_G(O_l)) - \sum_{l \leqslant t} x(\delta_G(O_l, U)) \geqslant t - |U| > n - 2 \cdot \mu(G,\kappa) + 2 \cdot \varepsilon \cdot \mu(G)$$

The last inequality is because $\mathcal{P}$ satisfies (a) and (b). Thus, $x(E^{\mathcal{P}}_{\mathsf{miss}}) > \kappa(E^{\mathcal{P}}_{\mathsf{miss}})$. ◁

We also have a claim which allows us to only focus on $\mathcal{P}$ for which all $e \in E^{\mathcal{P}}_{\mathsf{miss}}$ have $\kappa(e) < 1/\rho_\varepsilon$. Let $\mathcal{H}^{\mathcal{P}}_{\mathsf{miss}}$ denote the event that none of the edges of $E^{\mathcal{P}}_{\mathsf{miss}}$ are sampled.

▶ **Observation 17.** *Suppose* $\kappa(e) \geqslant 1/\rho_\varepsilon$ *for any* $e \in E^{\mathcal{P}}_{miss}$, *then,* $\Pr\left(\mathcal{H}^{\mathcal{P}}_{miss}\right) = 0$.

The above observation follows from the fact that an edge e is sampled with probability $\min\{1, \kappa(e) \cdot \rho_\varepsilon\}$. From the above claim, it is sufficient to focus on $E^{\mathcal{P}}_{\mathsf{miss}}$ where all edges e have $\kappa(e) < 1/\rho_\varepsilon$, since these are the only $\mathcal{P}$ that contribute non-zero probability.

$$\begin{aligned}\Pr\left(\mathcal{H}^{\mathcal{P}}_{\mathsf{miss}}\right) &\leqslant \prod_{e \in E^{\mathcal{P}}_{\mathsf{miss}}} (1 - p(e)) \leqslant \exp\left(-\sum_{e \in E^{\mathcal{P}}_{\mathsf{miss}}} p(e)\right) = \exp\left(-\sum_{e \in E^{\mathcal{P}}_{\mathsf{miss}}} \kappa(e) \cdot \rho_\varepsilon\right) \\ &= \exp\left(-\varepsilon \cdot 2^{40/\varepsilon^2} \cdot \mu(G) \cdot \log n\right) \leqslant \exp\left(-2^{39/\varepsilon^2} \cdot \mu(G) \cdot \log n\right).\end{aligned}$$

Note that number of partitions $\mathcal{P}$ satisfying (a) and (b) are upper bounded by the number of ways of partitioning V, and it is known that a set of size n has $2^{n \cdot \log n}$ partitions. Since $n \leqslant 16 \cdot \mu(G)/\varepsilon$, the bound on the number of partitions is at most $2^{16\mu(G)/\varepsilon \cdot \log n}$ (by assumption of Lemma 15).

Thus, applying the equation above and taking a union bound over all the partitions, we know that the with probability $1 - \exp\left(-\mu(G) \cdot \log n/\varepsilon\right)$, in G_s, we have no partition $\mathcal{P} = \{U, O_1, \cdots, O_t, E_1, \cdots, E_q\}$ satisfying (a) and (b) such that $G_s[V \setminus U]$ is a union of disconnected components $O_1, \cdots, O_t, E_1, \cdots, E_q$. Thus, by Lemma 12, with high probability, $\mu(G_s) \geqslant \mu(G,\kappa) - \varepsilon \cdot \mu(G)$. ◀

5.2 Phase 2 of M-or-E*()

The algorithm proceeds to Phase 2 only if the integral matching M_s found in G_s is close to $\mu(G,\kappa)$. But note that although Phase 1 gives us a way to estimate the *value* of $\mu(G,\kappa)$ via $\mu(G_s)$ (by Lemma 15), it is unclear how to actually compute a corresponding fractional matching $\vec{x}$ that obeys capacities and odd set constraints. That is the goal of Phase 2: we show how to compute $\vec{x}$ and show that it is close in value to $\mu(G_s)$ with high probability, and therefore it is close to $\mu(G,\kappa)$ as well (by Lemma 15).

5.2.1 Preliminaries for Phase 2

Phase 2 starts by computing $M_s = \textsc{Static-Match}(G_s, \varepsilon)$ and then uses M_s to compute the desired fractional matching. We will split M_s into low capacity edges and high capacity edges, and as a result split V into vertices matched using high capacity edges, and low capacity edges. We begin by giving a formal definition of low capacity edges.

▶ **Definition 18.** *Let G be any multigraph, and let κ be a capacity function on the edges of G. Let $\varepsilon \in (0, 1/2)$. Define $E_L(G,\kappa) = \{e \in E \mid e \in D(u,v)$ and $\kappa(D(u,v)) \leqslant 1/\alpha_\varepsilon^2\}$. Intuitively, $E_L(G,\kappa)$ is the set of low total capacity edges.*

As mentioned in the high-level overview, in order to prove our probabilistic claims, we will give some slack to the capacities. This motivates our next definition.

▶ **Definition 19.** *Let G be a multigraph, and let κ be a capacity function on the edges of G. Let $\varepsilon \in (0, 1/2)$. We define the capacity function κ^+ as follows: for all $e \in E_L(G,\kappa)$, $\kappa^+(e) = \kappa(e) \cdot \alpha_\varepsilon$ and for all $e \in E \setminus E_L(G,\kappa)$, $\kappa^+(e) = \kappa(e)$.*

To make our analysis easier to follow, we need the following definition of a bipartite double cover of G.

▶ **Definition 20** (Bipartite Double Cover)**.** *Let G be a multigraph and κ be a capacity function on the edges of G. We define the $\textsc{bc}(G)$ to be the following bipartite graph with capacity function κ_{BC}.*

(a) *For every vertex $v \in V(G)$, make two copies v and v' in $V(\textsc{bc}(G))$.*

(b) *If e is an edge between $u, v \in V(G)$, then for each such e we add two edges e' and e'', one between u and v' and the other between v and u'. We let $\kappa_{BC}(e') = \kappa_{BC}(e'') = \kappa(e)$.*

We defer the proof of the following claim, relating $\mu(G)$ and $\mu(\textsc{bc}(G))$ to the full version.

▷ Claim 21. For any multigraph G, $\mu(\textsc{bc}(G)) \geqslant 2 \cdot \mu(G)$.

Next, we state the following lemma, which follows from standard techniques, and we give a formal proof of it in the full version. The lemma essentially states that a fractional matching which has low flow on all edges has a very small integrality gap.

▶ **Lemma 22.** *Let G be a multigraph, and let $\varepsilon \in (0,1)$. Let κ be a capacity function on the edges of G, with $\kappa(D(e)) \leqslant 1/\alpha_\varepsilon$ for all $e \in E(G)$. Then, $\mu(\textsc{bc}(G), \kappa_{BC}) \leqslant 2 \cdot (1+\varepsilon) \cdot \mu(G,\kappa)$, where $\mu(G,\kappa)$ is the maximum fractional matching of G obeying κ and the odd set constraints, and $\mu(\textsc{bc}(G), \kappa_{BC})$ is the maximum fractional matching of $\textsc{bc}(G)$ obeying κ_{BC}.*

Additionally, we will need the following lemma, which follows as a corollary of Lemma 12.

▶ **Proposition 23** (Extended Hall's Theorem)**.** *Let $G = (L \cup R, E)$ be a bipartite graph with $n = |L| = |R|$, then $\mu(G) = n - \max_{S \subseteq L}(|S| - |N_G(S)|)$, where $N_G(S)$ refers to the neighbourhood of S in G.*

In order to prove Lemma 26, we will use the following version of Chernoff bound.

▶ **Lemma 24** (Chernoff Bound). *[21] Let $X_1, \cdots, X_k$ be negatively correlated random variables, and let X denote their sum, and let $\mu = \mathbb{E}[X]$. Suppose $\mu_{min} \leqslant \mu \leqslant \mu_{max}$, then for all $\delta > 0$,* $\Pr(X \geqslant (1+\delta)\mu_{max}) \leqslant \left(\frac{e^{\delta}}{(1+\delta)^{\delta}}\right)^{\mu_{max}}$.

Additionally, we state the following observation. The proof follows from an application of the max-flow min-cut theorem (see [9]).

▶ **Observation 25.** *Let G be any bipartite multigraph, with vertex bipartitions S and T and capacity κ on the edges. Then, for any $C \subseteq S$, and $D \subseteq T$, we have, $|S| - |C| + |D| + \kappa(C, T \setminus D) \geqslant \mu(G, \kappa)$. Moreover, there are sets $C \subseteq S$ and $D \subseteq T$ such that equality holds.*

5.2.2 Main Result for Phase 2

We briefly give some intuition about the statement of next lemma. Recall in the high level review, we mentioned that M, the integral matching of G_s has two parts M_H, which is the high capacity part, and M_L, the low capacity part, and we defined $V_H = V(M_H)$ and $V_L = V(M_L)$. We said that congestion balancing allows us to give slack to capacities (κ^+ in Definition 19), and therefore, we can round up the capacities of M_H to 1. However, we still want to compute a fractional matching of $G[V_L]$. In order to do this, we observe that if the fractional matching in $G[V_L]$ is only on low capacity edges, then we can use flow algorithm on the low capacity edges of the bipartite graph $\text{BC}(G[V_L])$ to compute such a matching. Therefore, our main structural result for Phase 2 states that if $\vec{y}$ is a maximum fractional matching with support on the low capacity edges of $G[V_L]$, then with high probability $\sum_{e \in E} y(e) \geqslant |M_L| - \varepsilon \cdot \mu(G)$. We now state this result formally.

▶ **Lemma 26.** *Let G be a multigraph and let $\varepsilon \in (0, 1/2)$ and suppose $\mu(G) \geqslant \varepsilon \cdot n/16$. Let κ be a capacity function on the edges, and let G_s be the graph obtained from G by sampling each edge e with probability $p(e) = \kappa(e) \cdot \rho_{\varepsilon}$. Let $E_L \coloneqq E_L(G, \kappa)$. Then, with high probability, for all $W \subseteq V$, we have $\mu(\text{BC}(G_s[W] \cap E_L)) \leqslant \mu(\text{BC}(G[W] \cap E_L), \kappa^+_{BC}) + 8 \cdot \varepsilon\mu(G)$.*

▶ Remark 27. Note that by definition of E_L, all edges $e \in G[W] \cap E_L$ have $\kappa(e) \leqslant 1/\alpha_{\varepsilon}^2$. Thus, $\kappa^+_{\text{BC}}(e) = \kappa_{\text{BC}}(e) \cdot \alpha_{\varepsilon}$ for all $e \in \text{BC}(G[W] \cap E_L)$ (recall Definition 19 and Definition 18).

Before we prove it, we have the following statement as a corollary of Lemma 26.

▶ **Corollary 28.** *Let G be a multi-graph, and let $\varepsilon \in (0, 1)$. Let κ be a capacity function on the edges of G with $\kappa(e) \leqslant 1/\alpha_{\varepsilon}$ for all $e \in E(G)$. Then, with high probability, **for all** $W \subseteq V$, $\mu(G_s[W] \cap E_L) \leqslant \mu(G[W] \cap E_L, \kappa^+) + 5 \cdot \varepsilon\mu(G)$.*

Proof. The inequality in the statement follows due to the following line of reasoning.

$$
\begin{aligned}
2 \cdot \mu(G_s[W] \cap E_L) &\leqslant \mu(\text{BC}(G_s[W] \cap E_L)) \\
&\quad \text{(since } 2 \cdot \mu(H) \leqslant \mu(\text{BC}(H)), \text{ see Claim 21)} \\
&\leqslant \mu(\text{BC}(G[W] \cap E_L), \kappa^+_{\text{BC}}) + 8 \cdot \varepsilon\mu(G) \\
&\quad \text{(by Lemma 26)} \\
&\leqslant 2 \cdot (1+\varepsilon) \cdot \mu(G[W] \cap E_L, \kappa^+) + 8\varepsilon\mu(G) \\
&\quad \text{(by Lemma 22)} \\
&\leqslant 2 \cdot \mu(G[W] \cap E_L, \kappa^+) + 10 \cdot \varepsilon\mu(G).
\end{aligned}
$$

The second to last inequality follows from the fact that any fractional matching in G obeying κ^+ and odd set constraints is upper bounded by $\mu(G)$ (by Lemma 13). ◀

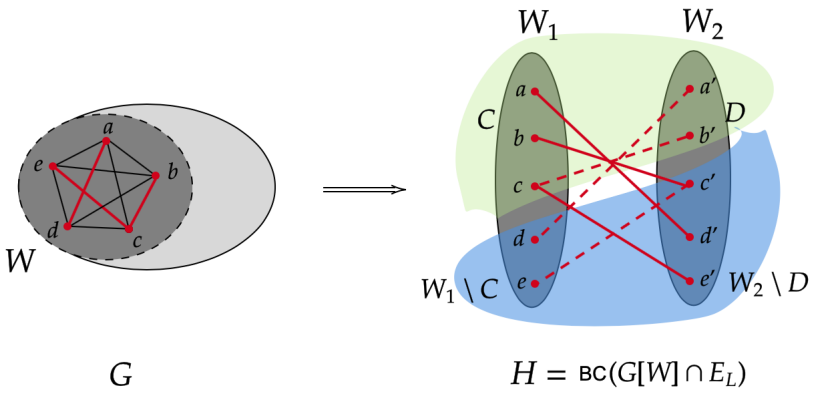


Figure 2 In the left panel, we consider the graph G, and $W = \{a, b, c, d, e\}$. The red edges correspond to the low capacity edges of $G[W]$, denoted $G[W] \cap E_L$. On the right panel, we have the bipartite graph $\text{BC}(G[W] \cap E_L)$, which has bipartitions W_1 and W_2. The solid red edges are the edges going between C and $W_2 \setminus D$, and these edges have capacity $\kappa(C, W_2 \setminus D)$.

Proof of Lemma 26. Throughout this proof we refer the reader to Figure 2. Consider a fixed $W \subseteq V$. From now, we will use H to denote $\text{BC}(G[W] \cap E_L)$ and let H_s denote $\text{BC}(G_s[W] \cap E_L)$. For the bipartite graph H, we will use W_1 and W_2 to denote the two bipartitions of H corresponding to W. We now want to prove that $\mu(H_s) \leqslant \mu(H, \kappa^+_{\text{BC}}) + 8\varepsilon\mu(G)$. To prove this, it is sufficient to show a set $C \subseteq W_1$ such that $|C| - |N_{H_s}(C)| \geqslant |W_1| - \mu(H, \kappa^+_{\text{BC}}) - 8\varepsilon\mu(G)$ with high probability. Then, by Proposition 23, we have the following inequality: $|W_1| - \mu(H, \kappa^+_{\text{BC}}) - 8\varepsilon\mu(G) \leqslant |C| - |N_{H_s}(C)| \leqslant |W_1| - \mu(H_s)$. This would prove our claim. Towards this, we consider the set $C \subseteq W$ satisfying the following equation (applying Observation 25 to H and κ^+_{BC}), and show that this is the required set.

$$|W_1| - |C| + |D| + \kappa^+_{\text{BC}}(C, W_2 \setminus D) = \mu(H, \kappa^+_{\text{BC}}). \tag{1}$$

We want to show that $|C| - |N_{H_s}(C)| \geqslant |W_1| - \mu(H, \kappa^+_{\text{BC}}) - 8\varepsilon\mu(G)$ with high probability. Let L be the set of vertices in $N_{H_s}(C) \cap W_2 \setminus D$. We know that $|N_{H_s}(C)| \leqslant |D| + |L|$. If we show that $|L| \leqslant \kappa^+_{\text{BC}}(C, W_2 \setminus D) + 8 \cdot \varepsilon\mu(G)$, then, $|N_{H_s}(C)| - 8\varepsilon \cdot \mu(G) \leqslant |D| + \kappa^+_{\text{BC}}(C, W_2 \setminus D)$. Substituting in Equation (1), we have $|W_1| - |C| + |N_{H_s}(C)| - 8\varepsilon \cdot \mu(G) \leqslant \mu(H, \kappa^+_{\text{BC}})$, which implies that $|C| - |N_{H_s}(C)| \geqslant |W_1| - 8 \cdot \varepsilon\mu(G) - \mu(H, \kappa^+_{\text{BC}})$. We now focus on bounding $|L|$.

Let $E_H(C, W_2 \setminus D)$ be the set of edges in H between C and $W_2 \setminus D$. Let X be the random variable that denotes the number of edges in $E_H(C, W_2 \setminus D)$ that are sampled in H_s. Note that X is not a sum of independent random variables. Recall that H is a subgraph of $\text{BC}(G)$, and suppose $e \in G[W] \cap E_L$ is included in G_s, then e' and e'' (recall e' and e'' are copies of e in $\text{BC}(G)$, Definition 20) are both included in H_s else both are excluded. Thus, the random variables associated with e' and e'' are correlated with each other. We instead consider an arbitrary subset of $E^*_H(C, W_2 \setminus D)$ of $E_H(C, W_2 \setminus D)$ that satisfies the following properties.

(a) For $e \in G[W]$, if $\{e', e''\} \subset E_H(C, W_2 \setminus D)$, then exactly one of e' or e'' is included in $E^*_H(C, W_2 \setminus D)$.

(b) For $e \in G[W]$, if $\{e', e''\} \cap E_H(C, W_2 \setminus D) = \{e'\}$, then only e' is included in $E^*_H(C, W_2 \setminus D)$ and if $\{e', e''\} \cap E_H(C, W_2 \setminus D) = \{e''\}$, then only e'' is included in $E^*_H(C, W_2 \setminus D)$.

We consider the random variable Y that denotes the number of edges in $E^*_H(C, W_2 \setminus D)$ that are included in H_s. Observe that Y is a sum of independent random variables satisfying the condition of Lemma 24. Moreover, $X \leqslant 2Y$. Thus, it is sufficient to upper bound the value

Y can take with high probability. Note that for any $e \in E_H(C, W_2 \setminus D)$, using the definition of H, $\kappa_{\text{BC}}(e) < 1/\alpha_\varepsilon^2$. Since $\rho_\varepsilon < \alpha_\varepsilon$, $p(e) = \rho_\varepsilon \cdot \kappa_{\text{BC}}(e) < 1$. So, we have,

$$\mathbb{E}[Y] \leqslant \sum_{e \in E^*_H(C, W_2 \setminus D)} p(e) \leqslant \rho_\varepsilon \cdot \kappa_{\text{BC}}(C, W_2 \setminus D) \leqslant \frac{\kappa^+_{\text{BC}}(C, W_2 \setminus D)}{2^{20/\varepsilon^2}}.$$

The last inequality follows from the fact that in H, $\kappa^+_{\text{BC}}(e) = \kappa_{\text{BC}}(e) \cdot \alpha_\varepsilon$ for all $e \in H$ (see Definition 19). This is because by definition of H and Definition 19, for all edges $e \in H$, the corresponding original edge in G is in $E_L(G, \kappa)$. We want to bound the $\Pr\left(Y \geqslant \frac{\kappa^+_{\text{BC}}(C, W_2 \setminus D)}{2^{20/\varepsilon^2}} + 4 \cdot \varepsilon\mu(G)\right)$. Applying Lemma 24 with $\delta = \frac{4 \cdot \varepsilon\mu(G) \cdot 2^{20/\varepsilon^2}}{\kappa^+_{\text{BC}}(C, W_2 \setminus D)}$, we have,

$$\begin{aligned}
\Pr\left(Y \geqslant \frac{\kappa^+_{\text{BC}}(C, W_2 \setminus D)}{2^{20/\varepsilon^2}} + 4 \cdot \varepsilon\mu(G)\right) &= \exp\left(\varepsilon\mu(G) - \varepsilon\mu(G) \log\left(1 + \frac{4 \cdot \varepsilon\mu(G) \cdot 2^{20/\varepsilon^2}}{\kappa^+_{\text{BC}}(C, W_2 \setminus D)}\right)\right) \\
&\leqslant \exp\left(\varepsilon\mu(G) - \varepsilon\mu(G) \log\left(1 + \varepsilon \cdot 2^{20/\varepsilon^2}\right)\right) \\
&\text{(Since } \kappa^+_{\text{BC}}(C, W_2 \setminus D) \leqslant 4 \cdot \mu(G) \text{ as proved below.)} \\
&\leqslant \exp\left(\varepsilon\mu(G) - \varepsilon\mu(G) \log\left(1 + 2^{19/\varepsilon^2}\right)\right) \\
&\text{(Using the fact that } 2^{1/\varepsilon^2} \geqslant 1/\varepsilon) \\
&= \exp\left(\varepsilon\mu(G) - 19\mu(G)/\varepsilon\right) \\
&\text{(Using the fact that } 2^{19/\varepsilon^2} \leqslant 2^{19/\varepsilon^2} + 1).
\end{aligned}$$

To see why $\kappa^+_{\text{BC}}(C, W_2 \setminus D) \leqslant 4 \cdot \mu(G)$, consider Equation (1), $\kappa^+_{\text{BC}}(C, W_2 \setminus D)$ is equal to

$$\begin{aligned}
\mu(H, \kappa^+_{\text{BC}}) - |W_1| + |C| - |D| &\leqslant 2 \cdot (1 + \varepsilon) \cdot \mu(G[W] \cap E_L(G, \kappa), \kappa^+) - |W_1| + |W_1| \\
&\leqslant 4 \cdot \mu(G).
\end{aligned}$$

The first inequality is due to Lemma 22, and the fact that $H = \text{BC}(G[W] \cap E_L(G, \kappa))$, and κ^+ satisfies the hypothesis. Finally, observe that $|L| \leqslant X \leqslant 2Y \leqslant \kappa^+_{\text{BC}}(C, W_2 \setminus D) + 8 \cdot \varepsilon\mu(G)$ with probability at least $\exp(-19\mu(G)/\varepsilon)$. Taking a union bound over all W, which are at most $2^{16 \cdot \mu(G)/\varepsilon}$ many (since by statement of the lemma, $\mu(G) \geqslant \varepsilon \cdot n/16$), we have our bound. ◀

5.3 Phase 3: Finding set E^*

The algorithm M-OR-E*() proceeds to Phase 3 if the matching found in G_s in Phase 1 is small, and hence $\mu(G, \kappa)$ is too small. In this case, we need to find a set E^* satisfying the properties of Lemma 7(b). In particular, we need to find a set E^* with $\kappa(E^*) = O(\mu(G) \log n)$ such that for every large matching M, there are a lot of edges going through E^*. In order to do this, we rely on the properties of the dual variables associated with the matching problem. The algorithm STATIC-MATCH(), luckily for us, solves both the primal as well as the dual solution. We first begin by stating the properties of the dual program, and then we state the properties of dual variables guaranteed by STATIC-MATCH().

▶ **Definition 29** ([38]). *We consider the dual of the matching linear program:*

1. *Every edge $e = (u, v)$ has a dual constraint $yz((u, v)) := y(u) + y(v) + \sum_{\substack{B \in V_{odd}, \\ e \in G[B]}} z(B)$.*
2. *We use $f(y, z) := \sum_{v \in V} y(u) + \sum_{B \subseteq V_{odd}} \frac{|B|-1}{2} z(B)$ to denote the dual objective function.*

We now describe some properties of the STATIC-MATCH(). All of the properties except Lemma 30(c) are given by the algorithm in [20]. We then show how to ensure property (c) as well by modifying the dual; see full version for details.

▶ **Lemma 30.** *There is an $O(m/\varepsilon)$ time algorithm* Static-Match() *that takes as input a graph G with m edges and a parameter $\varepsilon > 0$, and returns a matching M, and dual vectors $\vec{y}$ and $\vec{z}$ that have the following properties.*
(a) *It returns an integral matching M such that $|M| \geqslant (1-\varepsilon)\cdot\mu(G)$.*
(b) *A set Ω of laminar odd-sized sets, such that $\{B \mid z(B) > 0\} \subseteq \Omega$.*
(c) *For all odd-sized B with $|B| \geqslant 1/\varepsilon + 1$, $z(B) = 0$.*
(d) *Each $y(v)$ is a multiple of ε and $z(B)$ is a multiple of ε.*
(e) *For every edge $e \in E$, $yz(e) \geqslant 1-\varepsilon$. We say that e is* approximately covered *by $\vec{y}$ and $\vec{z}$.*
(f) *The value of the dual objective, $f(y,z)$ is at most $(1+\varepsilon)\cdot\mu(G)$.*

5.3.1 Main Guarantees of Phase 3

We now state the following helper lemma is instrumental in proving one of the two main properties of E^*, namely, that every large matching has a lot of edges passing through E^*.

▶ **Lemma 31.** *Suppose G is a graph, and let $H \subset G$ be a subgraph of G. Let $\vec{y}, \vec{z}$ be the dual variables returned on execution of* Static-Match(H, ε). *Let $E_H = \{e \in G \mid yz(e) \geqslant 1-\varepsilon\}$. For any matching M of G, then $|M \cap E \setminus E_H| \geqslant |M| - (1+\varepsilon)^2\cdot\mu(H)$.*

Proof. Suppose we scale up the dual variables $\vec{y}$ and $\vec{z}$ by a factor of $1+\varepsilon$. Then, $\vec{y}$ and $\vec{z}$ is a feasible solution for the dual matching program for the graph E_H. Thus, using weak duality, we have that $\mu(E_H) \leqslant (1+\varepsilon)f(y,z) \leqslant (1+\varepsilon)^2\cdot\mu(H)$ (this inequality follows from Lemma 30(f)). Suppose M is a matching of G with $|M\cap E\setminus E_H| < |M| - (1+\varepsilon)^2\cdot\mu(H)$ then, this implies that $|M \cap E \setminus E_H| < |M| - \mu(E_H)$. Thus, $|M| = |M\cap E_H| + |M\cap E\setminus E_H| < \mu(E_H) + |M| - \mu(E_H) = |M|$, which is a contradiction. ◀

We now define set E^*, and show that it has the property that $\kappa(E^*) = O(\mu(G)\log n)$.

▶ **Lemma 32.** *Let G be a graph multi-graph such that $\mu(G) \geqslant \varepsilon\cdot n/16$, and let κ be a capacity function on the edges of the graph. Suppose G_s is the graph obtained by sampling every edge $e \in E$ with probability $p(e) = \min\{1, \kappa(e)\cdot\rho_\varepsilon\}$. Let $\vec{y}, \vec{z}$ be the output of* Static-Match(G_s, ε). *Let $E^* = \{e \in E \mid yz(e) < 1-\varepsilon\}$. Then, for all $e \in E^*$, $\kappa(e) < 1$ and with high probability, $\kappa(E^*) = O(\mu(G)\log n)$.*

Proof. We consider the set $\mathcal{D}$ of assignments $\vec{y}, \vec{z}$ to the vertices and odd components that satisfy the following properties.
(a) For all $v \in V$, $y(v)$ is a multiple of ε, and for all $B \subset V$, $z(B)$ is a multiple of ε.
(b) Let $\Omega = \{B \subset V \mid z(B) > 0\}$, then Ω is laminar.
(c) If $z(B) > 0$ for some $B \subseteq V$, then $|B| \leqslant 1/\varepsilon$.
Observe that,

$$|\mathcal{D}| \leqslant \sum_{i=0}^{2n} \binom{n^{1/\varepsilon}}{i} \cdot \left(\frac{1}{\varepsilon}\right)^n \cdot \left(\frac{1}{\varepsilon}\right)^{2n} \leqslant n \cdot \binom{n^{1/\varepsilon}}{2n} \cdot \left(\frac{1}{\varepsilon}\right)^n \cdot \left(\frac{1}{\varepsilon}\right)^{2n} \leqslant 2^{(4/\varepsilon)\cdot n\log n}.$$

This number follows from the following argument. Since Ω is laminar, it can contain at most $2n$ sets. Moreover, from (c), we deduce that these $2n$ sets are chosen from among $n^{1/\varepsilon}$ sets. Further, from (a), we deduce that each $y(v)$ can be assigned at most $1/\varepsilon$ values, and for every $B \in \Omega$, $z(B)$ can be assigned $1/\varepsilon$ values. Therefore, for a given choice of Ω, there are at most $(1/\varepsilon)^n \cdot (1/\varepsilon)^{2n}$ choices for $\vec{y}$ and $\vec{z}$. Moreover, the above number also upper bounds the number of possible duals that can be a returned by the algorithm Static-Match(), since the duals in $\mathcal{D}$ satisfy a subset of the properties given in Lemma 30. Since $\mu(G) \geqslant \varepsilon\cdot n/16$, we can upper bound $|\mathcal{D}|$ by $2^{(32/\varepsilon^2)\cdot\mu(G)\log n}$.

Now consider a fixed $\vec{y}, \vec{z}$ that satisfies the above-mentioned properties, and let E^* be the set of edges that are not approximately covered by $\vec{y}, \vec{z}$. Note that for any $e \in E^*$, $\kappa(e) \leqslant 1/\rho_\varepsilon$. If not, then, $p(e) = 1$, and then it would be sampled in G_s, and be approximately covered by $\vec{y}, \vec{z}$ (by Lemma 30(e)). Thus, for any $e \in E^*$, $p(e) = \kappa(e) \cdot \rho_\varepsilon$. Now, suppose $\kappa(E^*) > \mu(G) \log n$. Then, observe that if $\vec{y}, \vec{z}$ is output by Static-Match(G_s, ε) (denote this event by $\mathcal{E}_{y,z}$) then none of the edges in E^* were sampled (denote this event by $\mathcal{E}_2$). Thus,

$$\begin{aligned}
\Pr(\mathcal{E}_{y,z}) &\leqslant \Pr(\mathcal{E}_2) \\
&\leqslant \prod_{e \in E^*} (1 - p(e)) \\
&\leqslant \exp\left(-\sum_{e \in E^*} \kappa(e) \cdot \rho_\varepsilon\right) \\
&\leqslant \exp\left(-2^{40/\varepsilon^2} \cdot \mu(G) \log n\right).
\end{aligned}$$

Our lemma follows by taking an upper bound over the set $\mathcal{D}$:

$$\begin{aligned}
\Pr\left(\bigcup_{\vec{y},\vec{z} \in \mathcal{D}} \mathcal{E}_{y,z}\right) &\leqslant \exp\left(-2^{40/\varepsilon^2} \cdot \mu(G) \log n\right) \cdot 2^{(32/\varepsilon^2) \cdot \mu(G) \cdot \log n} \\
&= \exp\left(-O(2^{1/\varepsilon^2} \cdot \mu(G) \cdot \log n)\right).
\end{aligned}$$

◀

6 Algorithm M-or-E*()

In this section, we give the main subroutine M-or-E*() (see Lemma 7). We first define a few terms, and then state algorithm Frac-Match(), which follows almost directly from known results, with some very minor modifications; see full version for details.

▶ **Definition 33.** *Recall Definition 18, and consider an integral matching M, define $E_L^M(G, \kappa) = E_L(G, \kappa) \cap M$, and let V_L^M be the endpoints of $E_L^M(G, \kappa)$.*

▶ **Lemma 34.** *Given a multigraph G (possibly non-bipartite), with edge capacities κ and $\varepsilon \in (0, 1)$, such that $\kappa(D(e)) \leqslant 1/\alpha_\varepsilon$ for all $e \in E$, then there is an algorithm Frac-Match() that takes as input G, κ and ε, and returns a fractional matching $\vec{x}$ such that $\sum_{e \in E} x(e) \geqslant (1 - \varepsilon) \cdot \mu(G, \kappa)$, obeying the capacities κ and the odd set constraints. The runtime of this algorithm is $O(m \cdot \log n/\varepsilon)$.*

For the purpose of the algorithm recall κ^+ in Definition 19. We now formalize M-or-E*() from Lemma 7 in Algorithm 1 below; recall that the input is a multigraph G with $\mu(G) \geqslant \varepsilon \cdot n/16$.

We now show Lemma 7 holds.

Proof of Lemma 7. We first show the runtime of the algorithm. Graph G_s can be computed in time $O(m)$. Using Lemma 30, we conclude that we can compute E^* in order $O(m/\varepsilon)$ time by running Static-Match(G_s, ε) and Lemma 34 implies that Algorithm 1 takes $O(m \cdot \log n/\varepsilon)$ time.

We show Lemma 7((a)). First observe that V_L^M and $V(M_I)$ are disjoint, and since $\vec{y}$ and $\vec{x}$ are fractional matchings, $\vec{z}$ is also a fractional matching. Note that $|M| \geqslant \mu - 7\varepsilon\mu$. Moreover, $\sum_{e \in E} x(e) \geqslant (1 - \varepsilon) \cdot \mu(G[V_L^M] \cap E_L, \kappa^+)$. This follows from applying Lemma 34

Algorithm 1 M-OR-E*$(G, \kappa, \varepsilon, \mu)$.

Include each $e \in E(G)$ independently with probability $p(e) = \min\{1, \kappa(e) \cdot \rho_\varepsilon\}$ into graph G_s.
Let M and $\vec{y}, \vec{z}$ be the output of STATIC-MATCH(G_s, ε). ▷ Phase 1
if $|M| < \mu - 7\varepsilon\mu$ **then** ▷ Phase 3
  Return $E^* = \{e \in E(G) \mid yz(e) < 1 - \varepsilon\}$.
else ▷ Phase 2
  $M_I \leftarrow M \setminus E_L(G, \kappa)$
  $\vec{y} \leftarrow M_I^D$ ▷ See Definition 5
  $\vec{x} \leftarrow$ FRAC-MATCH$(G[V_L^M] \cap E_L(G, \kappa), \kappa^+, \varepsilon)$
  Return $\vec{z} \leftarrow \vec{y} + \vec{x}$.

to $G[V_L^M] \cap E_L(G, \kappa)$ with capacity function κ^+. Recall Definition 19 and Definition 18 to see that $\kappa^+(D(e)) \leqslant 1/\alpha_\varepsilon$ for $e \in G[V_L^M] \cap E_L(G, \kappa)$, thus satisfying the requirements of Lemma 34. Next, applying Corollary 28, we have, $\mu(G_s[V_L^M] \cap E_L) \leqslant \mu(G[V_L^M] \cap E_L, \kappa^+) + 5 \cdot \varepsilon\mu(G)$. Thus, we have $\sum_{e \in E} x(e) \geqslant (1 - \varepsilon) \cdot \big(\mu(G_s[V_L^M] \cap E_L) - 5\varepsilon\mu\big) \geqslant (1 - \varepsilon) \cdot (|M \setminus M_I| - 5\varepsilon\mu)$. This is because $M \setminus M_I$ is a matching of $G_s[V_L^M] \cap E_L$. So, $\sum_{e \in E} z(e) \geqslant |M_I| + (1 - \varepsilon) \cdot (|M \setminus M_I| - 5\varepsilon\mu) \geqslant (1 - \varepsilon) \cdot (\mu - 12\varepsilon\mu) \geqslant \mu - 13\varepsilon\mu$.

We now show Lemma 7(ai) and (aii). Consider any edge $e \in \text{supp}(\vec{z})$ with $\kappa(D(e)) \leqslant 1/\alpha_\varepsilon^2$, $e \in G[V_L^M] \cap E_L(G, \kappa)$. Thus, $e \in \text{supp}(\vec{x})$, and therefore, from Lemma 34, $z(e) = x(e) \leqslant \kappa^+(e) \leqslant \kappa(e) \cdot \alpha_\varepsilon$ and $z(D(e)) = x(D(e)) \leqslant \kappa(D(e)) \cdot \alpha_\varepsilon$. Similarly, for any $e \in \text{supp}(\vec{z})$ with $\kappa(D(e)) > 1/\alpha_\varepsilon^2$, $e \in \text{supp}(\vec{y})$. By definition of $\vec{y}$, $z(e) = y(e) = \kappa(e)/\kappa(D(e))$ (recall Definition 5) and $z(D(e)) = 1$. This proves our claim.

Next, we show Lemma 7(b). First recall from the assumption of Lemma 7 that $\mu \geqslant (1-\varepsilon) \cdot \mu(G)$. From this fact and Lemma 32, we can conclude that $\kappa(E^*) = O(\mu \log n)$ and that for all $e \in E^*$, $\kappa(e) < 1$. Next, observe that $\mu(G_s) \leqslant (1+\varepsilon) \cdot |M| \leqslant (1-6\varepsilon) \cdot \mu$. Applying Lemma 31 with $H = G_s$, we have, that for any matching M' of G, $|E^* \cap M'| \geqslant |M'| - (1+\varepsilon)^2 \cdot (1-6\varepsilon) \cdot \mu$. If $|M'| \geqslant (1 - 3\varepsilon) \cdot \mu$, then we have $|E^* \cap M'| \geqslant \varepsilon \cdot \mu$. Finally, consider any pair of vertices u, v and let $e', e'' \in D(u, v)$. Then, either both e', e'' are both approximately covered by $\vec{y}, \vec{z}$ or neither of them are. This implies that either $D(u, v) \subseteq E^*$ or $D(u, v) \cap E^* = \emptyset$. ◀

References

1 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *Proceedings of the 38th International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'11, pages 526–538, Berlin, Heidelberg, 2011. Springer-Verlag.

2 Nima Anari and Vijay V. Vazirani. Matching is as easy as the decision problem, in the nc model. In *ITCS*, 2020.

3 Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in nc. *J. ACM*, 67(4), May 2020. doi:10.1145/3397504.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem with (very) few queries. *ACM Transactions on Economics and Computation (TEAC)*, 7(3):1–19, 2019.

5 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in o(log n) update time (corrected version). *SIAM J. Comput.*, 47(3):617–650, 2018. doi:10.1137/16M1106158.

6 Soheil Behnezhad and Sanjeev Khanna. *New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS*, pages 3529–3566. SIAM, 2022. doi:10.1137/1.9781611977073.140.

7 Soheil Behnezhad, Jakub Łącki, and Vahab Mirrokni. Fully dynamic matching: Beating 2-approximation in δ^{ϵ} update time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2492–2508. SIAM, 2020.

8 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. *ACM Trans. Algorithms*, 17(4):29:1–29:51, 2021. doi:10.1145/3469833.

9 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1123–1134. IEEE, 2020.

10 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. doi:10.1007/978-3-662-47672-7_14.

11 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711. SIAM, 2016. doi:10.1137/1.9781611974331.ch50.

12 Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic dynamic matching in O(1) update time. *Algorithmica*, 82(4):1057–1080, 2020. doi:10.1007/s00453-019-00630-4.

13 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 785–804, USA, 2015. Society for Industrial and Applied Mathematics.

14 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 398–411, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897568.

15 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 470–489. SIAM, 2017. doi:10.1137/1.9781611974782.30.

16 Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.27.

17 Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 384–393, 2014. doi:10.1109/FOCS.2014.48.

18 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1326–1344. SIAM, 2016.

19 Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. *CoRR*, abs/1602.06705, 2016. arXiv:1602.06705.

20 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

21 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

22 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63:490–508, 2011.

23 Stephen Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 754–763, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897564.

24 Manuela Fischer, Slobodan Mitrovic, and Jara Uitto. Deterministic $(1+\epsilon)$-approximate maximum matching with poly$(1/\epsilon)$ passes in the semi-streaming model. *CoRR*, abs/2106.04179, 2021. arXiv:2106.04179.

25 Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, October 1991. doi:10.1145/115234.115366.

26 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1 + \epsilon)$-approximate incremental matching in constant deterministic amortized time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1886–1898. SIAM, 2019. doi:10.1137/1.9781611975482.114.

27 Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. *Maintaining an EDCS in General Graphs: Simpler, Density-Sensitive and with Worst-Case Time Bounds*, pages 12–23. SIAM, 2022. doi:10.1137/1.9781611977066.2.

28 Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *FSTTCS*, 2014.

29 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 548–557, 2013.

30 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.

31 John E. Hopcroft and Richard M. Karp. An n5/2 algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

32 A. Karzanov. On finding a maximum flow in a network with special structure and some applications. *Matematicheskie Voprosy Upravleniya Proizvodstvom (L.A. Lyusternik, ed.), Moscow State Univ. Press, Moscow, 1973, Issue 5, pp. 81–94, in Russian.*, 1973.

33 Peter Kiss. Deterministic dynamic matching in worst-case update time. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 94:1–94:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.94.

34 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016.

35 Meena Mahajan and Kasturi R. Varadarajan. A new nc-algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In *STOC '00*, 2000.

36 Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

37 Mohammad Roghani, Amin Saberi, and David Wajc. Beating the Folklore Algorithm for Dynamic Matching. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 111:1–111:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2022.111.

38 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

39 Shay Solomon. Fully dynamic maximal matching in constant update time. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.43.

40 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-nc. *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707, 2017.

41 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 194–207, 2020.

Near-Optimal Algorithms for Stochastic Online Bin Packing

Nikhil Ayyadevara* ✉
Indian Institute of Technology Delhi, India

Rajni Dabas* ✉
Department of Computer Science, University of Delhi, India

Arindam Khan ✉
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

K. V. N. Sreenivas ✉
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Abstract

We study the online bin packing problem under two stochastic settings. In the bin packing problem, we are given n items with sizes in $(0, 1]$ and the goal is to pack them into the minimum number of unit-sized bins. First, we study bin packing under the i.i.d. model, where item sizes are sampled independently and identically from a distribution in $(0, 1]$. Both the distribution and the total number of items are unknown. The items arrive one by one and their sizes are revealed upon their arrival and they must be packed immediately and irrevocably in bins of size 1. We provide a simple meta-algorithm that takes an offline α-asymptotic proximation algorithm and provides a polynomial-time $(\alpha + \varepsilon)$-competitive algorithm for online bin packing under the i.i.d. model, where $\varepsilon > 0$ is a small constant. Using the AFPTAS for offline bin packing, we thus provide a linear time $(1 + \varepsilon)$-competitive algorithm for online bin packing under i.i.d. model, thus settling the problem.

We then study the random-order model, where an adversary specifies the items, but the order of arrival of items is drawn uniformly at random from the set of all permutations of the items. Kenyon's seminal result [SODA'96] showed that the Best-Fit algorithm has a competitive ratio of at most 3/2 in the random-order model, and conjectured the ratio to be ≈ 1.15. However, it has been a long-standing open problem to break the barrier of 3/2 even for special cases. Recently, Albers et al. [Algorithmica'21] showed an improvement to 5/4 competitive ratio in the special case when all the item sizes are greater than 1/3. For this special case, we settle the analysis by showing that Best-Fit has a competitive ratio of 1. We also make further progress by breaking the barrier of 3/2 for the *3-Partition* problem, a notoriously hard special case of bin packing, where all item sizes lie in $(1/4, 1/2]$.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Bin Packing, 3-Partition Problem, Online Algorithms, Random Order Arrival, IID model, Best-Fit Algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.12

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `http://arxiv.org/abs/2205.03622`

Funding *Arindam Khan*: Research partly supported by Pratiksha Trust Young Investigator Award, Google India Research Award, and Google ExploreCS Award.

Acknowledgements We thank Susanne Albers, Leon Ladewig, and Jirí Sgall for helpful initial discussions. We also thank three anonymous reviewers for their comments and suggestions.

* A part of this work was done when Nikhil and Rajni were summer interns at the Indian Institute of Science, Bengaluru. Rajni's internship was supported by ACM India Anveshan Setu Fellowship.

1 Introduction

Bin Packing (BP) is a fundamental NP-hard combinatorial optimization problem. In BP, we are given a set of items $I := (x_1, x_2, \ldots, x_n)$ with their associated weights (also called sizes) x_i's in $(0, 1]$ and the goal is to partition them into the minimum number of sets (bins) such that the total weight of each set is at most 1. The problem has numerous applications in logistics, scheduling, cutting stock, etc. [10]. Theoretically, bin packing has been the cornerstone for approximation and online algorithms and the study of the problem has led to the development of several interesting techniques [31, 15, 33].

Generally, the performance guarantee of an offline (resp. online) bin packing algorithm $\mathcal{A}$ is measured by asymptotic approximation ratio (AAR) (resp. competitive ratio (CR)). Let $\mathrm{Opt}(I)$ and $\mathcal{A}(I)$ are the objective values returned by the optimal (offline) algorithm and algorithm $\mathcal{A}$, respectively, on input I. Then AAR (resp. CR) is defined as $R^{\infty}_{\mathcal{A}} := \limsup_{m\to\infty}\left(\sup_{I:\mathrm{Opt}(I)=m} \frac{\mathcal{A}(I)}{\mathrm{Opt}(I)}\right)$. Note that $R^{\infty}_{\mathcal{A}}$ focuses on instances where $\mathrm{Opt}(I)$ is large and avoids pathological instances with large approximation ratios where $\mathrm{Opt}(I)$ is small.

Best-Fit (BF), First-Fit (FF), and Next-Fit (NF) are the three most commonly used algorithms for BP. Given x_i as the present item to be packed, they work as follows:

- BF: Pack x_i into the *fullest possible* bin; open a new bin if necessary.
- FF: Pack x_i into the *first possible* bin; open a new bin if necessary.
- NF: Pack x_i into the *most recently opened* bin; open a new bin if necessary.

Johnson et al. [28] studied several heuristics for bin packing such as Best-Fit (BF), First-Fit (FF), Best-Fit-Decreasing (BFD), First-Fit-Decreasing (FFD) and showed their (asymptotic) approximation guarantees to be 17/10, 17/10, 11/9, 11/9, resp. Bekesi et al. [7] gave an $O(n)$ time 5/4-asymptotic approximation algorithm. Another $O(n \log n)$ time algorithm is Modified-First-Fit-Decreasing (MFFD) [29] which attains an AAR of $71/60 \approx 1.1834$. Vega and Lueker [15] gave an asymptotic fully polynomial-time approximation scheme (AFPTAS) for BP: For any $1/2 > \varepsilon > 0$, it returns a solution with at most $(1+\varepsilon)\mathrm{Opt}(I) + O(1)$ [1] bins in time $C_\varepsilon + Cn \log 1/\varepsilon$, where C is an absolute constant and C_ε depends only on ε. Karmarkar and Karp [31] gave an algorithm that returns a solution using $\mathrm{Opt}(I) + O(\log^2 \mathrm{Opt}(I))$ bins. The present best approximation is due to Hoberg and Rothvoss [25] which returns a solution using $\mathrm{Opt}(I) + O(\log \mathrm{Opt}(I))$ bins.

3-Partition problem is a notoriously hard special case of BP where all item sizes are larger than 1/4. Eisenbrand et al. [17] mentioned that "much of the hardness of bin packing seems to appear already in the special case of 3-Partition when all item sizes are in $(1/4, 1/2]$". This problem has deep connections with Beck's conjecture in discrepancy theory [45, 38]. In fact, Rothvoss [25] conjectured that these 3-Partition instances are indeed the hardest instances for bin packing and the additive integrality gap of the bin packing configuration LP for these 3-Parition instances is already $\Theta(\log n)$.

In online BP, items appear one by one and are required to be packed immediately and irrevocably. Lee and Lee [33] presented the Harmonic algorithm with competitive ratio $T_\infty \approx 1.691$, which is optimal for $O(1)$ space algorithms. For general online BP, the present best upper and lower bounds for the CR are 1.57829 [4] and 1.54278 [5], respectively.

In this paper, we focus on online BP under a stochastic setting called the *i.i.d. model* [11] where the input items are sampled from a sequence of independent and identically distributed (i.i.d.) random variables. Here, the performance of an algorithm is measured by the

[1] In bin packing and related problems, the accuracy parameter ε is assumed to be a constant. Here, the term $O(1)$ hides some constants depending on ε.

expected competitive ratio (ECR) $ER_{\mathcal{A}} := \mathbb{E}[\mathcal{A}(I_n(F))]/\mathbb{E}[\mathrm{Opt}(I_n(F))]$, where $I_n(F) := (X_1, X_2, \ldots, X_n)$ is a list of n random variables drawn i.i.d. according to some unknown distribution F with support in $(0, 1]$. Mostly, bin packing has been studied under continuous uniform (denoted by $U[a, b], 0 \leq a < b \leq 1$, where item sizes are chosen uniformly from $[a, b]$) or discrete uniform distributions (denoted by $U\{j, k\}, 1 \leq j \leq k$, where item sizes are chosen uniformly from $\{1/k, 2/k, \ldots, j/k\}$). For $U[0, 1]$, Coffman et al. [13] showed that NF has an ECR of $4/3$ and Lee and Lee [34] showed that the Harmonic algorithm has an ECR of $\pi^2/3 - 2 \approx 1.2899$. Interestingly, Bentley et al. [8] showed that the ECR of FF as well as BF converges to 1 for $U[0, 1]$. It was later shown that the expected wasted space (i.e., the number of needed bins minus the total size of items) is $\Theta(n^{2/3})$ for First-Fit [44, 12] and $\Theta(\sqrt{n}\log^{3/4} n)$ for Best-Fit [44, 35]. Rhee and Talagrand [42] exhibited an algorithm that w.h.p. achieves a packing in $\mathrm{Opt} + O(\sqrt{n}\log^{3/4} n)$ bins for any distribution F on $(0, 1]$. However, note that their competitive ratio can be quite bad when $\mathrm{Opt} \ll n$. A distribution F is said to be *perfectly packable* if the expected wasted space in the optimal solution is $o(n)$ (i.e., nearly all bins in an optimal packing are almost fully packed). Csirik et al. [14] studied the Sum-of-Squares (SS) algorithm and showed that for any perfectly packable distribution, the expected wasted space is $O(\sqrt{n})$. However, for distributions that are not perfectly packable, the SS algorithm has an ECR of at most 3 and can have an ECR of $3/2$ in the worst-case [14]. For any discrete distribution, they gave an algorithm with an ECR of 1 that runs in pseudo-polynomial time in expectation. Gupta et al. [24] also obtained similar $o(n)$ expected wasted space guarantee by using an algorithm inspired by the interior-point (primal-dual) solution of the bin packing LP. However, it remains an open problem to obtain a polynomial-time $(1 + \varepsilon)$-competitive algorithm for online bin packing under the i.i.d. model for arbitrary general distributions. In fact, the present best polynomial-time algorithm for bin packing under the i.i.d. model is BF which has an ECR of at most $3/2$. However, Albers et al. [1] showed that BF has an ECR ≥ 1.1 even for a simple distribution: when each item has size $1/4$ with probability $3/5$ and size $1/3$ with probability $2/5$.

We also study the *random-order model*, where the adversary specifies the items, but the arrival order is permuted uniformly at random. The performance measure in this model is called asymptotic random order ratio (ARR): $RR^{\infty}_{\mathcal{A}} := \limsup_{m\to\infty} \left(\sup_{I:\mathrm{Opt}(I)=m}(\mathbb{E}[\mathcal{A}(I_\sigma)]/\mathrm{Opt}(I))\right)$. Here, σ is drawn uniformly at random from $\mathcal{S}_n$, the set of permutations of n elements, and $I_\sigma := (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$ is the permuted list. Random-order model generalizes the i.i.d. model [1], thus the lower bounds in the random-order model can be obtained from the i.i.d. model. Kenyon in her seminal paper [32] studied Best-Fit under random-order and showed that $1.08 \leq RR^{\infty}_{BF} \leq 3/2$. She conjectured that $RR^{\infty}_{BF} \leq 1.15$. The conjecture, if true, raises the possibility of a better alternate practical offline algorithm: first shuffle the items randomly, then apply Best-Fit. This then beats the AAR of $71/60$ of the present best practical algorithm MFFD. The conjecture has received a lot of attention in the past two decades and yet, no other polynomial-time algorithm is known with a better ARR than BF. Coffman et al. [30] showed that $RR^{\infty}_{NF} = 2$. Fischer and Röglin [21] achieved analogous results for Worst-Fit [27] and Smart-Next-Fit [39]. Recently, Fischer [9] presented an exponential-time algorithm, claiming an ARR of $(1 + \varepsilon)$.

Monotonicity is a natural property of BP algorithms, which holds if the algorithm never uses fewer bins to pack $\hat{I}$ when compared I, where $\hat{I}$ is obtained from I by increasing the item sizes. Murgolo [37] showed that while NF is monotone, BF and FF are not.

Several other problems have been studied under the i.i.d. model and the random-order model [16, 23, 18, 22, 19, 2, 20, 36, 24].

1.1 Our Contributions

Bin packing under the i.i.d. model. We achieve near-optimal performance guarantee for the bin packing problem under the i.i.d. model, thus settling the problem. For any arbitrary unknown distribution F on $(0, 1]$, we give a meta-algorithm (see Algorithm 1) that takes an α-asymptotic approximation algorithm as input and provides a polynomial-time $(\alpha + \varepsilon)$-competitive algorithm. Note that both the distribution F as well as the number of items n are unknown in this case.

▶ **Theorem 1.** *Let $\varepsilon \in (0, 1)$ be a constant parameter. For online bin packing under the i.i.d. model, where n items are sampled from an unknown distribution F, given an offline algorithm $\mathcal{A}_\alpha$ with an AAR of α and runtime $\beta(n)$, there exists a meta-algorithm (Algorithm 1) which returns a solution with an ECR of $(\alpha + \varepsilon)$ and runtime $O(\beta(n))$.* [2]

Using an AFPTAS for bin packing (e.g. [15]) as $\mathcal{A}_\alpha$, we obtain the following corollary.

▶ **Corollary 2.** *Using an AFPTAS for bin packing as $\mathcal{A}_\alpha$ in Theorem 1, we obtain an algorithm for online bin packing under the i.i.d. model with an ECR of $(1 + \varepsilon)$ for any $\varepsilon \in (0, 1/2)$.*

Most algorithms for bin packing under the i.i.d. model are based on the following idea. Consider a sequence of $2k$ items where each item is independently drawn from an unknown distribution F, and let $\mathcal{A}$ be a packing algorithm. Pack the first k items using $\mathcal{A}$; denote the packing by $\mathcal{P}'$. Similarly, let $\mathcal{P}''$ be the packing of the next k items using $\mathcal{A}$. Since each item is drawn independently from F, both $\mathcal{P}'$ and $\mathcal{P}''$ have the same properties in expectation; in particular, the expected number of bins used in $\mathcal{P}'$ and $\mathcal{P}''$ are the same. Thus, intuitively, we want to use the packing $\mathcal{P}'$ as a proxy for the packing $\mathcal{P}''$. However, there are two problems. First, we do not know n, which means that there is no way to know what a good sample size is. Second, we need to show the stronger statement that w.h.p. $\mathcal{P}' \approx \mathcal{P}''$. Note that the items in $\mathcal{P}'$ and $\mathcal{P}''$ are expected to be similar, but they may not be the same. So, it is not clear which item in $\mathcal{P}'$ is to be used as a proxy for a newly arrived item in the second half. Due to the online nature, erroneous choice of proxy items can be quite costly. Different algorithms handle this problem in different ways. Some algorithms exploit the properties of particular distributions, some use exponential or pseudo-polynomial time, etc.

Rhee and Talagrand [41, 42] used *upright matching* to decide which item can be considered as a proxy for a newly arrived item. They consider the model packing $\mathcal{P}_k$ of the first k items (let's call these the proxy items) using an offline algorithm. With the arrival of each of the next k items, they take a proxy item at random and pack it according to the model packing. Then, they try to fit in the real item using upright matching. They repeat this process until the last item is packed. However, they could only show a guarantee of $\mathrm{Opt} + O(\sqrt{n}\log^{3/4} n)$. The main drawback of [42] is that their ECR can be quite bad if $\mathrm{Opt} \ll n$ (say, $\mathrm{Opt} = \sqrt{n}$). One of the reasons for this drawback is that they don't distinguish between small and large items; when there are too many small items, the ECR blows up.

Using a similar approach, Fischer [9] obtained a $(1+\varepsilon)$-competitive randomized algorithm for the random-order model, but it takes exponential time, and the analysis is quite complicated. The exponential time was crucial in finding the optimal packing which was then used as a good proxy packing. However, prior to our work, no polynomial-time algorithm existed which achieves a $(1 + \varepsilon)$ competitive ratio.

[2] As mentioned in an earlier footnote, the $O(\cdot)$ notation hides some constants depending on ε here.

To circumvent these issues, we treat large and small items separately. However, a straightforward adaptation faces several technical obstacles. Thus our analysis required intricate applications of concentration inequalities and sophisticated use of upright matching. First, we consider the semi-random case when we know n. Our algorithm works in stages. For a small constant $\delta \in (0, 1]$, the first stage contains only $\delta^2 n$ items. These items give us an estimate of the distribution. If the packing does not contain too many large items, we show that the simple Next-Fit algorithm suffices for the entire input. Otherwise, we use a proxy packing of the set of first $\delta^2 n$ items to pack the next $\delta^2 n$ items. In the process, we pack the small and large items differently. Then we use the proxy packing of the first $2\delta^2 n$ items to pack the next $2\delta^2 n$ items and so on. In general, we use the proxy packing of the first $2^i\delta^2 n$ items to pack the next $2^i\delta^2 n$ items.

Finally, we get rid of the assumption that we know n using a doubling trick (we guess n first and keep doubling the guess if it turns out to be incorrect). However, there are several difficulties (e.g. input stopping midway of two consecutive guesses, large wasted space in the past stages). Yet, with involved technical adaptations, we can handle them (see Section 2.2).

Our algorithm is simple, polynomial-time (in fact, $O(n)$ time), and achieves essentially the best possible competitive ratio. It is relatively simpler to analyze when compared to Fischer's algorithm [9]. Also, unlike the algorithms of Rhee and Talagrand [42] as well as Fischer [9], our algorithm is deterministic. This is because, unlike their algorithms, instead of taking proxy items at random, we pack all the proxy items before the start of a stage and try to fit in the real items as they come. This makes our algorithm deterministic. Our algorithm is explained in detail in Section 2.1. The nature of the meta-algorithm provides flexibility and ease of application. See Table 1 for the performance guarantees obtained using different offline algorithms.

See Section 2 for the details of the proof and the description of our algorithm. In fact, our algorithm can easily be generalized to d-dimensional online vector packing [6], a multidimensional generalization of bin packing. See the full version for a $d(\alpha+\varepsilon)$ competitive algorithm for d-dimensional online vector packing where the i^{th} item X_i can be seen as a tuple $\left(X_i^{(1)}, X_i^{(2)}, \ldots, X_i^{(d)}\right)$ where each $X_i^{(j)}$ is independently sampled from an unknown distribution $\mathcal{D}^{(j)}$.

Bin packing under the random-order model. Next, we study BP under the random-order model. Recently, Albers et al. [1] showed that BF is monotone if all the item sizes are greater than $1/3$. Using this result, they showed that in this special case, BF has an ARR of at most $5/4$. We show that, somewhat surprisingly, in this case, BF actually has an ARR of 1 (see Section 3.1 for the detailed proof).

▶ **Theorem 3.** *For online bin packing under the random-order model, Best-Fit achieves an asymptotic random-order ratio of* 1 *when all the item sizes are in* $(1/3, 1]$.

Next, we study the 3-partition problem, a special case of bin packing when all the item sizes are in $(1/4, 1/2]$. This is known to be an extremely hard case [25]. Albers et al. [1] mentioned that "it is sufficient to have one item in $(1/4, 1/3]$ to force Best-Fit into anomalous behavior." E.g., BF is non-monotone in the presence of items of size less than $1/3$. Thus the techniques of [1] do not extend to the 3-Partition problem. We break the barrier of $3/2$ in this special case, by showing that BF attains an ARR of ≈ 1.4941.

▶ **Theorem 4.** *For online bin packing under the random-order model, Best-Fit achieves an asymptotic random-order ratio of* ≈ 1.4941 *when all the item sizes are in* $(1/4, 1/2]$.

Table 1 Analysis of Algorithm 1 depending on $\mathcal{A}_\alpha$. In the first row, C is an absolute constant and C_ε is a constant that depends on ε.

$\mathcal{A}_\alpha$	Time Complexity	Expected Competitive Ratio
AFPTAS [15]	$O(C_\varepsilon + Cn\log 1/\varepsilon)$	$(1+\varepsilon)$
Modified-First-Fit-Decreasing [29]	$O(n\log n)$	$(71/60+\varepsilon)$
Best-Fit-Decreasing [26]	$O(n\log n)$	$(11/9+\varepsilon)$
First-Fit-Decreasing [26]	$O(n\log n)$	$(11/9+\varepsilon)$
Next-Fit-Decreasing [3]	$O(n\log n)$	$(T_\infty+\varepsilon)$
Harmonic [33]	$O(n)$	$(T_\infty+\varepsilon)$
Next-Fit	$O(n)$	$(2+\varepsilon)$

We prove Theorem 4 in Section 3.2. As 3-partition instances are believed to be the hardest instances for bin packing, our result gives a strong indication that the ARR of BF might be strictly less than 3/2.

2 Online Bin Packing Problem under the i.i.d. Model

In this section, we provide the meta algorithm as described in Theorem 1. For the ease of presentation, we split this into two parts. In Section 2.1, we assume a semi-random model, i.e., we assume that the number of items n is known beforehand and design an algorithm. Later, in Section 2.2, we get rid of this assumption.

Let the underlying distribution be F. Without loss of generality, we assume that the support set of F is a subset of $(0,1]$. For any set of items J, we define $\mathcal{W}(J)$ as the sum of weights of all the items in J. For any $k \in \mathbb{N}_+$, we denote the set $\{1,2,\ldots,k\}$ by $[k]$. Let $\varepsilon \in (0,1)$ be a constant parameter and let $0 < \delta < \varepsilon/8$ be a constant. Let $\mathcal{A}_\alpha$ be an offline algorithm for bin packing with an AAR of $\alpha > 1$ and let Opt denote the optimal algorithm. For any $i \in [n]$, we call x_i to be a *large* item if $x_i \geq \delta$ and a *small* item otherwise. Let I_ℓ and I_s denote the set of large and small items in I, respectively.

2.1 Algorithm Assuming that the Value of n is Known

We now describe our algorithm which assumes the knowledge of the number of items. For simplicity, in this section we assume both n and $1/\delta^2$ to be powers of 2. Otherwise, we can round down $\delta \in [1/2^{i+1}, 1/2^i), i \in \mathbb{N}$ to $1/2^{i+1}$. Also, we will anyway get rid of the assumption on the knowledge of n in the next subsection, and there we do not need n to be a power of 2. We denote this algorithm by `Alg`. First, we give a high level idea of the algorithm. We divide the entire input into stages as follows: we partition the input set I into $m := \left(\log(1/\delta^2)+1\right)$ stages $T_0, T_1, \ldots, T_{m-1}$. The zeroth stage T_0, called the *sampling stage*, contains the first $\delta^2 n$ items, i.e., $x_1, x_2, \ldots, x_{\delta^2 n}$. For $j \in [m-1]$, T_j contains the items with index starting from $2^{j-1}\delta^2 n + 1$ till $2^j\delta^2 n$. In essence, T_0 contains the first $\delta^2 n$ items, T_1 contains the next $\delta^2 n$ items, T_2 contains the next $2\delta^2 n$ items, and so on. Note that the number of stages m is a constant. In any stage T_j, we denote the set of large items and small items by L_j and S_j, respectively. For $j \in [m-1]$, let $\overline{T_j}$ denote the set of items which have arrived before the stage T_j, i.e., $\overline{T_j} = T_0 \cup T_1 \cup \cdots \cup T_{j-1}$. Similarly, we define $\overline{L_j}$ and $\overline{S_j}$ as the set of large items and small items, respectively, in $\overline{T_j}$. Note that for any $j \in [m-1]$, $|T_j| = \left|\overline{T_j}\right|$ and since all the items are sampled independently from the same

distribution, we know that with high probability, the optimal solutions of T_j and $\overline{T_j}$ are quite similar. Since $\overline{T_j}$ would have arrived before T_j, we compute an almost optimal packing of $\overline{T_j}$ (in an offline manner) and use it as a blueprint to pack T_j.

The algorithm is as follows: first, we pack T_0, the sampling stage using Next-Fit. The sampling stage contains only a small but a constant fraction of the entire input set; hence it uses only a few number of bins when compared to the final packing but at the same time provides a good estimate of the underlying distribution. If the number of large items in the sampling stage is at most $\delta^3\mathcal{W}(T_0)$, then we continue using Next-Fit for the rest of the entire input too. Intuitively, NF performs well in this case as most of the items are small. Thus, from now on, let us assume otherwise. Now assume that we are at an intermediate point where $\overline{T_j}$ has arrived and T_j is about to arrive ($j \geq 1$). We create D_j, the set of *proxy* items, which is just a copy of $\overline{T_j}$. We pack D_j using $\mathcal{A}_\alpha$. Let this packing be denoted by $\mathcal{P}_j$. Let $B_j^{(k)}$ denote the k^{th} bin in the packing $\mathcal{P}_j$. We iterate over k and remove all the small items in the bin $B_j^{(k)}$ and create a slot in the free space of $B_j^{(k)}$. We call this slot to be an *S-slot*. When an item $x_i \in T_j$ arrives, we check if x_i is small or large

- If x_i is small, we pack it in one of the S-slots greedily, using Next-Fit. If it doesn't fit in any of the S-slots, then we create a new bin with only one S-slot spanning the entire bin (so, this bin will only be used to pack small items), and pack it there.
- If x_i is large, we remove the smallest proxy item with a size more than x_i in the packing $\mathcal{P}_j$ and pack it there. If no such proxy item exists, we open a new bin, pack x_i in there and close it, meaning that it will not be used to pack any further items.

After T_j is packed completely, we just discard the proxy items in the packing that haven't been replaced and move to the next stage. For more formal details and pseudocode for the algorithm, please refer to Algorithm 1.

We will proceed to analyze the algorithm. But first, we will discuss stochastic upright matching and a standard result on it. Using a standard probabilistic concentration lemma, we will formulate a few lemmas which are going to be very important for the analysis of the algorithm.

Stochastic Upright Matching. Rhee and Talagrand [43] studied stochastic upright matching problem in the context of analysis of bin packing algorithms. Consider a set $P = \{(x_i, y_i)\}_{i\in[2n]}$ of $2n$ points where each x_i is either $+1$ or -1 with equal probability and $y_1, y_2, \ldots, y_{2n}$ are sampled according to an i.i.d. distribution. We can define a bipartite graph $\mathcal{G}$ as follows: the vertex set is P, viewed as a set of points in $\mathbb{R}\times\mathbb{R}$. Two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ share an edge iff $x_1 = 1, x_2 = -1$ and $y_1 \geq y_2$.

The objective of the problem is to find a maximum matching in $\mathcal{G}$ or, in other words, minimize the number of unmatched points which we denote by $U(P)$. We denote this matching variant by $\mathcal{M}$. The following lemma shows that w.h.p., the number of unmatched points is $O(\sqrt{n}(\log n)^{3/4})$. The proof of the lemma follows from Lemma 3.1 in [43].

▶ **Lemma 5** ([43]). *Let P be an instance for $\mathcal{M}$. Then there exist constants $a, C, K > 0$ such that, $\mathbb{P}\left[U(P) \geq K\sqrt{n}(\log n)^{3/4}\right] \leq C\exp\left(-a(\log n)^{3/2}\right)$.*

Concentration Inequalities. Now we state the concentration inequalities.

▶ **Lemma 6** (Bernstein's Inequality). *Let $X_1, X_2, \ldots, X_n$ be independent random variables such that each $X_i \in [0,1]$. Then, for any $\lambda > 0$, the following inequality holds.*

$$\mathbb{P}\left[\left|\sum_{i=1}^{n} X_i - \sum_{i=1}^{n}\mathbb{E}\left[X_i\right]\right| \geq \lambda\right] \leq 2\exp\left(-\frac{\lambda^2}{2\left(\sum_{i=1}^{n}\mathbb{E}\left[X_i\right] + \lambda/3\right)}\right).$$

Algorithm 1 $\texttt{Alg}(x_1, x_2, \ldots, x_n)$: A nearly optimal algorithm for online bin packing assuming that the number of items n is known before-hand.

Input: $I_n(\mathcal{D}) = \{x_1, x_2, ..., x_n\}$.
for i= 1 to $\delta^2 n$ **do** ▷ Sampling Stage, T_0
  Pack x_i using NF.
end for
if $|L_0| \leq \delta^3 \mathcal{W}(T_0)$ **then** ▷ Very few large items
  Use NF for all remaining stages.
else
  for j= 1 to $m-1$ **do**
    $D_j \leftarrow \overline{T_j}$; $L(D_j) \leftarrow$ set of large items in D_j.
    Pack D_j using $\mathcal{A}_\alpha$.
    Let the packing be denoted by $\mathcal{P}_j$. ▷ Packing of proxy items
    $\mathcal{S}_j \leftarrow \phi$. ▷ the set of S-slots
    for bin B in $\mathcal{P}_j$ **do**
      Remove the small items in B.
      Create an S-slot H of size equal to (1−weight of all the large items in B).
      $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup H$.
    end for
    for $x_i \in T_j$ **do**
      if x_i is large **then**
        if $\exists d \in L(D_j)$ such that $d \geq x_i$ **then**
          Find smallest such d.
          $L(D_j) \leftarrow L(D_j) \setminus \{d\}$.
          Pack x_i in place of d in the packing $\mathcal{P}_j$.
        else
          Open a new bin and pack x_i and close the bin.
        end if
      else
        Try packing x_i in $\mathcal{S}_j$ using Next-Fit.
        if x_i couldn't be packed **then**
          Open a new bin B' with a single S-slot of unit capacity.
          $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup B'$.
          Pack x_i in B'.
        end if
      end if
    end for
  end for
end if

The following lemma is a direct implication of the results of [42, 40].

▶ **Lemma 7.** *For any $t \in [n]$, let $I(1,t)$ denote the first t items of the input set I. Then there exist constants $K, a > 0$ such that, $\mathbb{P}\left[\mathrm{Opt}(I(1,t)) \geq \frac{t}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + K\sqrt{n}(\log n)^{3/4}\right] \leq \exp\left(-a(\log n)^{3/2}\right)$.*

The following lemmas are about how a property of a part of the input (say, the total size) compares to that of the entire input.

▶ **Lemma 8.** *For an input set I of n items drawn from a distribution independently, for any arbitrary set $J \subseteq I$ we have,* $\mathbb{P}\left[\left|\mathcal{W}(J) - \frac{|J|}{n}\mathbb{E}\left[\mathcal{W}(I)\right]\right| \geq \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-\frac{1}{3}\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$.

▶ **Lemma 9.** *Let I be an input set of n items drawn from a distribution independently and let J be any subset of I. Suppose J_ℓ (resp. I_ℓ) denote the set of large items in J (resp. I). Then we have,* $\mathbb{P}\left[\left||J_\ell| - \frac{|J|}{n}\mathbb{E}\left[|I_\ell|\right]\right| \geq \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-\frac{\delta}{3}\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$.

▶ **Lemma 10.** *For any $t \in \{1, 2, \ldots, n\}$, Suppose $I(1,t)$ denote the first t items of the input set I. Then there exist constants $C, a > 0$ such that with probability at least $1 - \exp\left(-a\left(\log \mathbb{E}\left[\mathrm{Opt}(I)\right]\right)^{1/3}\right)$, we have* $\mathrm{Opt}(I(1,t)) \leq (1+2\delta)\frac{t}{n}\mathbb{E}\left[\mathrm{Opt}(I)\right] + C\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3}$.

Lemma 7 (resp. Lemmas 8 and 9) intuitively states that the optimal solution for (resp. the weight of the items of, and the number of large items in) a part of the input is almost a linear function of the length of the part. This makes sense because each item comes from the same distribution. Lemma 10 further tries to improve on Lemma 7 by bounding the lower order terms in terms of $\mathbb{E}\left[\mathrm{Opt}(I)\right]$ instead of n while losing only a small factor of $1 + 2\delta$. This is crucial since we need to bound the number of additional bins used by our algorithm in terms of $\mathbb{E}\left[\mathrm{Opt}(I)\right]$ and not in terms of n. We give the proofs of Lemmas 7–10 in the full version. As mentioned, Lemma 7 follows from [42, 40]. Lemmas 8 and 9 are simple applications of Bernstein's inequality. The high-level proof of Lemma 10 goes as follows: first we consider the optimal packing of the large items in $I(1,t)$, then we pack the small items in $I(1,t)$ greedily. If the small items open new bins then it means that all the bins (except one) are filled up to a level at least $1-\delta$. Otherwise, we don't use any extra bins. So, $\mathrm{Opt}(I(1,t)) \leq \max\{\mathrm{Opt}(I_\ell(1,t)), (1+2\delta)\mathcal{W}(I(1,t)) + 1\}$. Then we use Lemmas 7, 8 and 9 to prove Lemma 10.

With these helpful lemmas, we now proceed to analyze the algorithm. We split the analysis into the following two cases: when $|L_0| \leq \delta^3 \cdot \mathcal{W}(T_0)$ and when $|L_0| > \delta^3 \cdot \mathcal{W}(T_0)$.

2.1.1 Case 1: $|L_0| \leq \delta^3 \cdot \mathcal{W}(T_0)$

Recall that in this case, we just continue with Next-Fit for all the remaining items. To bound the Next-Fit solution, we first consider the number of bins that contain at least one large item. For this, we bound the value of $|I_\ell|$. Then we consider the bins that contain only small items and bound this value in terms of weight of all items $\mathcal{W}(I)$.

▷ Claim 11. Let $K := \mathbb{E}\left[\mathrm{Opt}(I)\right]$. For some positive constants C_1, C_2, a, we have that $\mathbb{P}\left[|I_\ell| \leq \delta \cdot \mathcal{W}(T_0) + C_1 K^{2/3}\right] \geq 1 - C_2\exp\left(-aK^{1/3}\right)$.

Proof. As the sampling stage contains $\delta^2 n$ items, $\mathbb{E}\left[|L_0|\right] = \delta^2\mathbb{E}\left[|I_\ell|\right]$. From Lemma 9, we have $\mathbb{P}\left[|L_0| \leq \delta^2\mathbb{E}\left[|I_\ell|\right] - \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-(\delta/3)\cdot\left(\mathbb{E}\left[\mathcal{W}(I)\right]\right)^{1/3}\right)$, and $\mathbb{P}\left[|I_\ell| \geq \mathbb{E}\left[|I_\ell|\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}\right] \leq 2\exp\left(-(\delta/3)\cdot\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$. From the above inequalities we have, $|I_\ell| \leq \frac{1}{\delta^2}|L_0| + \left(1+\frac{1}{\delta^2}\right)\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$ with probability at least $1 - 4\exp\left(-\frac{\delta}{3}\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$. We can use the inequalities $\mathbb{E}\left[\mathrm{Opt}(I)\right] \geq \mathbb{E}\left[\mathcal{W}(I)\right]$ and $|L_0| \leq \delta^3 \cdot \mathcal{W}(T_0)$ to conclude the proof of this claim. ◁

Now we bound the number of bins that are closed by small items. Note that Next-Fit fills each such bin up to a capacity at least $(1-\delta)$. So, the number of such bins is at most $\frac{1}{1-\delta}\mathcal{W}(I)$ when all items are packed by Next-Fit. Also, there can be at most one bin that

can be open. Thus combining all these results, (and using inequality $\frac{1}{1-\delta} \leq 1+2\delta$ for $\delta < \frac{1}{2}$) with high probability, $\mathrm{NF}(I) \leq |I_\ell| + (1+2\delta)\mathcal{W}(I) + 1 \leq \delta \cdot \mathcal{W}(T_0) + (1+2\delta)\mathcal{W}(I) + K\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3}$, for some constant K.

Using Lemma 8, we get that with high probability, $\mathcal{W}(I) \leq \mathbb{E}\left[\mathcal{W}(I)\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Using the facts $\mathcal{W}(I) \geq \mathcal{W}(T_0)$ and $\mathbb{E}\left[\mathrm{Opt}(I)\right]/2 \leq \mathbb{E}\left[\mathcal{W}(I)\right] \leq \mathbb{E}\left[\mathrm{Opt}(I)\right]$, we get,

$$\mathrm{NF}(I) \leq (1+3\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_3\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} \tag{1}$$

with probability of at least $1 - C_4 \exp\left(-a_1\mathbb{E}\left[\mathrm{Opt}(I)\right]^{1/3}\right)$ for some constants $C_3, C_4, a_1 > 0$. When the low probability event occurs, we can use the upper bound of $\mathrm{NF}(I) \leq 2\mathrm{Opt}(I) - 1$ to obtain the competitive ratio. Let $p = C_4 \exp\left(-a_1\mathbb{E}\left[\mathrm{Opt}(I)\right]^{1/3}\right)$.

$$\begin{aligned}\mathbb{E}\left[\mathrm{NF}(I)\right] &\leq (1-p)\left((1+3\delta)\mathbb{E}\left[\mathrm{Opt}(I)\right] + K\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + 1\right) + p(2\mathbb{E}\left[\mathrm{Opt}(I)\right] - 1) \\ &= (1+3\delta+2p)\mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right])\end{aligned}$$

Since $p = o(1)$ when $\mathbb{E}\left[\mathrm{Opt}(I)\right]$ tends to infinity, we obtain that the expected competitive ratio tends to at most $1 + 3\delta < 1 + \varepsilon$.

2.1.2 Case 2: $|L_0| > \delta^3 \cdot \mathcal{W}(T_0)$

We split our analysis in this case into two parts. We first analyze the number of bins used in the sampling stage T_0 and then analyze the number of bins used in the remaining stages.

Using Lemma 9, we obtain w.h.p. that $|L_0| \leq \delta^2\mathbb{E}\left[|I_\ell|\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Hence,

$$\begin{aligned}\mathbb{E}\left[|I_\ell|\right] &\geq \frac{1}{\delta^2}|L_0| - \frac{1}{\delta^2}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} \\ &\geq \delta\mathcal{W}(T_0) - \frac{1}{\delta^2}\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} && \text{(since } |L_0| > \delta^3 \cdot \mathcal{W}(T_0)\text{)} \\ &\geq \delta^3\mathbb{E}\left[\mathcal{W}(I)\right] - \left(\delta + \frac{1}{\delta^2}\right)\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} && (2)\end{aligned}$$

The last inequality follows from Lemma 8. For any $j \geq 1$, using $|T_j|/n \geq \delta^2$ and using Lemma 9, we get,

$$|L_j| \geq \delta^5\mathbb{E}\left[\mathcal{W}(I)\right] - (2+\delta^3)\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} \tag{3}$$

Each of the Eqs. (2),(3) holds with a probability of at least $1 - C\exp\left(-a\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$ for some constants $C, a > 0$.

Note that $\mathcal{W}(I) \geq \mathrm{Opt}(I)/2$. So from now on we assume that there exist constants $C_1, C_2 > 0$ which depend on δ such that w.h.p. both the following inequalities hold.

$$\mathbb{E}\left[|I_\ell|\right] \geq C_1 \cdot \mathbb{E}\left[\mathrm{Opt}(I)\right] \tag{4}$$

$$|L_j| \geq C_2 \cdot \mathbb{E}\left[\mathrm{Opt}(I)\right] \tag{5}$$

- **Analysis of the Sampling Stage:** Recall that the number of items considered in the sampling phase is $\delta^2 n$. We will bound the number of large items and the weight of items in this stage using Bernstein's inequality.
  1. Since sampling phase has $\delta^2 n$ items, $\mathbb{E}\left[|L_0|\right] = \delta^2\mathbb{E}\left[|I_\ell|\right]$. By applying Bernstein's inequality for $X_1, X_2, \ldots, X_{|T_0|}$ where X_i takes value 1 is x_i is large and 0 otherwise, we get,

$$\begin{aligned}\mathbb{P}\left[|L_0| \geq 2\delta^2\mathbb{E}\left[|I_\ell|\right]\right] &= \mathbb{P}\left[|L_0| \geq \mathbb{E}\left[|L_0|\right] + \delta^2\mathbb{E}\left[|I_\ell|\right]\right] \\ &\leq 2\exp\left(-\frac{\delta^4\mathbb{E}\left[|I_\ell|\right]^2}{2\mathbb{E}\left[|L_0|\right] + \frac{2}{3}\delta^2\mathbb{E}\left[|I_\ell|\right]}\right) \leq 2\exp\left(-\frac{1}{3}\delta^2\mathbb{E}\left[|I_\ell|\right]\right) \\ &\leq 2\exp\left(-a_1 \cdot \mathbb{E}\left[\mathrm{Opt}(I)\right]\right) \qquad \text{(from Equation (4))}\end{aligned}$$

for some constant $a_1 > 0$. So, with high probability, $|L_0| \leq 2\delta^2\mathbb{E}\left[|I_\ell|\right] \leq 2\delta\mathbb{E}\left[\mathrm{Opt}(I)\right]$.

2. Similarly, $\mathbb{E}\left[\mathcal{W}(T_0)\right] = \delta^2\mathbb{E}\left[\mathcal{W}(I)\right]$. By applying Bernstein's inequality for $X_1, X_2, \ldots, X_{|T_0|}$ where X_i takes value x_i, we get,

$$\begin{aligned}\mathbb{P}\left[\mathcal{W}(T_0) \geq 2\delta^2\mathbb{E}\left[\mathcal{W}(I)\right]\right] &= \mathbb{P}\left[\mathcal{W}(T_0) \geq \mathbb{E}\left[\mathcal{W}(S_0)\right] + \delta^2\mathbb{E}\left[\mathcal{W}(I)\right]\right] \\ &\leq 2\exp\left(\frac{-\delta^4\mathbb{E}\left[\mathcal{W}(I)\right]^2}{2\delta^2\mathbb{E}\left[\mathcal{W}(T_0)\right] + \frac{2}{3}\delta^2\mathbb{E}\left[\mathcal{W}(I)\right]}\right) \\ &\leq 2\exp\left(\frac{-\delta^2}{3}\mathbb{E}\left[\mathcal{W}(I)\right]\right) \leq 2\exp\left(\frac{-\delta^2\mathbb{E}\left[\mathrm{Opt}(I)\right]}{6}\right)\end{aligned}$$

So, with high probability we have, $\mathcal{W}(T_0) \leq 2\delta^2\mathbb{E}\left[\mathcal{W}(I)\right] \leq 2\delta^2\mathbb{E}\left[\mathrm{Opt}(I)\right]$.

Since the number of bins opened by Next-Fit $\mathrm{NF}(T_0)$ is at most $|L_0| + \frac{1}{1-\delta}\mathcal{W}(T_0) + 1$, using the bounds on the number of large items and weight of small items in sampling phase, w.h.p. we have,

$$\mathrm{NF}(T_0) \leq 2\delta\mathbb{E}\left[\mathrm{Opt}(I)\right] + \frac{2\delta^2}{1-\delta}\mathbb{E}\left[\mathrm{Opt}(I)\right] \leq 4\delta\mathbb{E}\left[\mathrm{Opt}(I)\right] \tag{6}$$

- **Analysis of the Remaining Stages:** Consider any stage T_j ($j > 0$) after the sampling stage. Note that $\left|\overline{T_j}\right| = |T_j|$. A bin can be opened in three different ways.
  1. When a new bin is opened while packing the set of proxy items D_j using $\mathcal{A}_\alpha$ (see the algorithm description in Section 2.1).
  2. When a large item can't replace a proxy item and hence a new bin is opened for it.
  3. When a small item can't fit in the set of S-slots and hence a new bin is opened with a single S-slot spanning the entire bin.

  In our analysis, first we bound the number of bins opened by $\mathcal{A}_\alpha$ for proxy items; we use Lemma 10 to obtain this bound. Then we show that the number of large items which cannot replace a proxy item would be very small by using upright matching arguments (Lemma 5). For small items, we bound the number of new bins opened by using the fact that $\mathcal{W}(S_j)$ and $\mathcal{W}(\overline{S_j})$ are very close which will be proved using Bernstein's inequality.

  Now we analyze the number of bins opened in the *first way* (say A^j_{proxy}). This is nothing but the number of bins opened by $\mathcal{A}_\alpha$ to pack $\overline{T_j}$. Since, $\mathcal{A}_\alpha$ has an AAR of α, by using Lemma 10, we have,

$$\begin{aligned}A^j_{\mathrm{proxy}} = \mathcal{A}_\alpha(\overline{T_j}) &\leq \alpha\mathrm{Opt}(\overline{T_j}) + o(\mathrm{Opt}(\overline{T_j})) \\ &\leq \alpha(1+2\delta)(\left|\overline{T_j}\right|/n)\mathbb{E}\left[\mathrm{Opt}(I)\right] + C_3\mathbb{E}\left[\mathrm{Opt}(I)\right]^{2/3} + o(\mathrm{Opt}(I)) \qquad (7)\end{aligned}$$

  with probability at least $1 - \exp\left(-a_2\log(\mathbb{E}\left[\mathrm{Opt}(I)\right])^{1/3}\right)$, for some constants $C_3, a_2 > 0$.

  We now bound the number of bins opened in the *second way*. Using Lemma 9, we get that w.h.p., $\left|\overline{L_j}\right| \geq \frac{\left|\overline{T_j}\right|}{n}\mathbb{E}\left[|I_\ell|\right] - \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$, and $|L_j| \leq \frac{|T_j|}{n}\mathbb{E}\left[|I_\ell|\right] + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Since $\left|\overline{T_j}\right| = |T_j|$ we have,

$$|L_j| \leq \left|\overline{L_j}\right| + 2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}. \tag{8}$$

So, w.h.p. the number of large items in T_j doesn't exceed that of those in $\overline{T_j}$ by a large number. Now consider the number of bins opened because there is no feasible proxy item that can be replaced. Let this number be A^j_{unmatch}. We can interpret this number as the number of unmatched items when we use the stochastic matching variant $\mathcal{M}$ from [43] as follows. We can interpret each item $\bar{t} \in \overline{L_j}$ as a point $P_{\bar{t}} := (+1, \bar{t})$ and each point $t \in L_j$ as a point $P_t := (-1, t)$. For simplicity, let's call the points with $+1$ as their first coordinate as *plus* points and the points with -1 as their first coordinate as *minus* points. We match a point $P_{\bar{t}}$ with P_t iff t replaced $\bar{t}$ in our algorithm. It is shown in [44] that such matching is always maximum. Hence the number of items that open new bins is at most the number of unmatched points in this maximum matching. There are two differences though. First, $\left|\overline{L_j}\right|$ may not be greater than $|L_j|$; but as we have shown, w.h.p, the difference can at most be $2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$. Secondly, in the matching variant $\mathcal{M}$, every point has equal chance to be a plus point or minus point. However, this is also inconsequential, since using concentration bounds for binomial random variables, we can show that the number of plus points/minus points lie in the range $\left(\mathbb{E}\left[\left|\overline{L_j}\right|\right] \pm \mathbb{E}\left[\left|\overline{L_j}\right|\right]^{2/3}\right)$ w.h.p. Hence by Lemma 5, we obtain that there exist constants a_3, C_4, K_1 s.t.

$$\mathbb{P}\left[A^j_{\text{unmatch}} \geq K_1\sqrt{\left|\overline{L_j}\right|}\left(\log\left|\overline{L_j}\right|\right)^{3/4} + 2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} + 2\mathbb{E}\left[\left|\overline{L_j}\right|\right]^{2/3}\right] \leq C_4\exp\left(-a_3(\log\left|\overline{L_j}\right|)^{3/2}\right)$$

We can simplify the above inequality using Equations (5) and (8) and the fact that $\text{Opt}(I) \leq 2\mathcal{W}(I)$ to obtain that there exist constants $a_4, C_4, K_2 > 0$ such that,

$$\mathbb{P}\left[A^j_{\text{unmatch}} \geq K_2\mathbb{E}\left[\text{Opt}(I)\right]^{2/3}\right] \leq C_4\exp\left(-a_4(\log\mathbb{E}\left[\text{Opt}(I)\right])^{3/2}\right) \tag{9}$$

The only part left is to bound the number of bins opened by small items in *third way*. Let this number be A^j_{small}. We will bound this by using the concentration of weights of small items in $\overline{T_j}$ and T_j. Consider the random variables $X_1, X_2 \ldots X_n$ where $X_i = 0$ if x_i is large, and $X_i = x_i$ otherwise. We have that $\mathcal{W}(S_j) = \sum_{X_i : x_i \in T_j} X_i$ and $\mathcal{W}(\overline{S_j}) = \sum_{X_i : x_i \in \overline{T_j}} X_i$. By applying Bernstein's inequality (similar to Lemma 8) we get, $\mathcal{W}(S_j) \leq \frac{|T_j|}{n}\mathcal{W}(I_s) + \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$, and $\mathcal{W}(\overline{S_j}) \geq \frac{|\overline{T_j}|}{n}\mathcal{W}(I_s) - \mathbb{E}\left[\mathcal{W}(I)\right]^{2/3}$ with a probability of at least $1 - C_5\exp\left(-a_5\mathbb{E}\left[\mathcal{W}(I)\right]^{1/3}\right)$ for some constants $C_5, a_5 > 0$. Combining both, we get,

$$\mathcal{W}(S_j) \leq \mathcal{W}(\overline{S_j}) + 2\mathbb{E}\left[\mathcal{W}(I)\right]^{2/3} \tag{10}$$

The initial allocated space for small items at the start of stage j in the packing $\mathcal{P}_j$ is $\mathcal{W}(\overline{S_j})$. Recall that $B^{(k)}_j$ denotes the k^{th} bin in the packing of $\mathcal{P}_j$. While packing the small items, if the S-slot in $B^{(k)}_j$ cannot accommodate a small item, this means that the remaining space in this S-slot is at most δ. So, the weight of small items which overflow the space allocated in packing $\mathcal{P}_j$ is at most $\mathcal{W}(S_j) - \mathcal{W}(\overline{S_j}) + \delta\left|\mathcal{P}_j\right|$ and this entire weight is packed in new bins opened exclusively for small items. Each of these bins (except possibly one) have an occupancy of at least $(1-\delta)$. Since $\mathcal{A}_\alpha$ is α-approximation algorithm, $|\mathcal{P}_j| = \mathcal{A}_\alpha(\overline{T_j}) \leq \alpha\text{Opt}(\overline{T_j}) + o(\text{Opt}(I))$. Using Equations (7) and (10) we get,

$$A^j_{\text{small}} \le \frac{1}{1-\delta}\left(\mathcal{W}(S_j) - \mathcal{W}(\overline{S_j}) + \delta \cdot \alpha \text{Opt}(\overline{T_j}) + o(\text{Opt}(I))\right) + 1$$
$$\le 2\delta \cdot \alpha \frac{|\overline{T_j}|}{n}\mathbb{E}\left[\text{Opt}(I)\right] + C_3\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I)) \tag{11}$$

with high probability. Combining Equations (7), (9), and (11), the number of bins, A_j, opened in the stage j is bounded as,

$$A_j = A^j_{\text{proxy}} + A^j_{\text{unmatch}} + A^j_{\text{small}}$$
$$\le \alpha(1+4\delta)\frac{|\overline{T_j}|}{n}\mathbb{E}\left[\text{Opt}(I)\right] + C_6\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I))$$
$$\le \alpha(1+4\delta)\frac{|T_j|}{n}\mathbb{E}\left[\text{Opt}(I)\right] + C_6\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I)) \tag{12}$$

w.h.p., for some constant C_6. To bound the sum of all A_js, first note that the number of "remaining stages" is $m-1$ which is a constant dependent on δ. Hence, with high probability,

$$\sum_{j=1}^{m-1} A_j \le \alpha(1+4\delta)\sum_{j=1}^{m-1}\frac{|\overline{T_j}|}{n}\mathbb{E}\left[\text{Opt}(I)\right] + (m-1)\cdot C_6\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I))$$
$$\le \alpha(1+4\delta)\mathbb{E}\left[\text{Opt}(I)\right] + C_7\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I)) \tag{13}$$

for some constant $C_7 > 0$ dependent on δ.

In the sampling phase, we have $\text{NF}(T_0) \le 4\delta\mathbb{E}\left[\text{Opt}(I)\right]$ with high probability and in all the remaining phases we have $\sum_{j=1}^{m-1} A_j \le \alpha(1+4\delta)\mathbb{E}\left[\text{Opt}(I)\right] + C_7\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I))$.

Combining both the results we get that w.h.p. the number of bins opened by `Alg` is,

$$\texttt{Alg}(I) \le \alpha(1+8\delta)\mathbb{E}\left[\text{Opt}(I)\right] + C_7\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I))$$
$$\le \alpha(1+\varepsilon)\mathbb{E}\left[\text{Opt}(I)\right] + C_7\mathbb{E}\left[\text{Opt}(I)\right]^{2/3} + o(\text{Opt}(I)) \tag{14}$$

In the low probability event when Equation (14) may not hold, we can bound $\texttt{Alg}(I)$ as follows. In the sampling stage, we have that $\texttt{Alg}(T_0) \le 2\text{Opt}(I) - 1$. For the remaining stages, we bound the number of bins containing at least one large item and the number of bins containing only small items. Because we create a proxy packing at the start of each stage, each large item is packed at most m times. So, the number of bins containing at least one large item is at most $m\,|I_\ell|$. In each stage, with one possible exception, every bin opened which has only small items has an occupancy of at least $(1-\delta)$. Combining over all the stages, the number of bins which contain only small items is at most $\frac{1}{1-\delta}\mathcal{W}(I_s) + m$. Thus, we can bound the total number of bins used by `Alg` to be at most $2\text{Opt}(I) + m\,|I_\ell| + \frac{1}{1-\delta}\mathcal{W}(I_s) + m$. On the other hand, we know that $\text{Opt}(I) \ge \mathcal{W}(I) \ge \delta\,|I_\ell| + \mathcal{W}(I_s)$. Hence, we obtain that $m\,|I_\ell| + \frac{1}{1-\delta}\mathcal{W}(I_s) \le \frac{m}{\delta(1-\delta)}\text{Opt}(I)$. Combining all these, we obtain that

$$\texttt{Alg}(I) \le \left(2 + \frac{m}{\delta(1-\delta)}\right)\text{Opt}(I) + m \tag{15}$$

Now, to obtain the competitive ratio, suppose Equation (14) holds with probability $p(= 1 - o(1))$. We combine Equations (14) and (15) similar to the case when $|L_0| \le \delta^3 \cdot \mathcal{W}(T_0)$.

$$\mathbb{E}\left[\texttt{Alg}(I)\right] \le p(\alpha(1+8\delta)\mathbb{E}\left[\text{Opt}(I)\right] + o(\mathbb{E}\left[\text{Opt}(I)\right]))$$
$$+ (1-p)\left(\left(2 + \frac{m}{\delta(1-\delta)}\right)\mathbb{E}\left[\text{Opt}(I)\right] + m\right)$$
$$\le \alpha(1+\varepsilon)\mathbb{E}\left[\text{Opt}(I)\right] + o(\mathbb{E}\left[\text{Opt}(I)\right]) \qquad \text{(since } 1-p = o(1)\text{)}$$

Scaling the initial value ε to ε/α before the start of the algorithm, we obtain a competitive ratio of $\alpha + \varepsilon$.

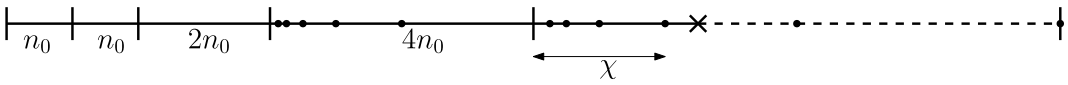


Figure 1 Doubling trick visualized. We start off with an estimate of n to be n_0 and keep doubling it. The first and second super-stages contain n_0 items each. The third super-stage contains $2n_0$ items and so on. Each super-stage is packed individually using `Alg`. The stages in each super-stage are marked by $\bullet$ (for simplicity, only the stages in the last two super-stages are marked). The $\times$ mark indicates the point where the input abruptly stopped. Notice how the fifth stage of the last super-stage is not full. Just before packing this stage, `Alg` constructs a proxy packing of χ by using $\mathcal{A}_\alpha$. But this is very lossy since χ is large compared to this last stage.

2.2 Getting Rid of the Assumption on the Knowledge of the Input Size

In this subsection, we will extend `Alg` for online bin packing with i.i.d. items to devise an algorithm that guarantees essentially the same competitive ratio without knowing the value of n. We denote this algorithm by `ImpAlg`.

We use a doubling trick as follows. We first guess the value of n to be a constant $n_0 := 1/\delta^3$. Then, we run `Alg` until $\min\{n, n_0\}$ items arrive (here, if $\min\{n, n_0\} = n$, then it means that the input stream has ended). If $n > n_0$, i.e., if there are more items to arrive, then we revise our estimate of n by doubling it, i.e., the new value of n is set as $n_1 := 2n_0$. We start `Alg` afresh on the next $\min\{n, n_1\} - n_0$ items. If $n > 2n_0$, then we set the new guess of n to be $n_2 := 2n_1 = 2^2 n_0$ and start `Alg` afresh on the next $\min\{n, n_2\} - n_1$ number of items. We continue this process of doubling our estimate of n until all the items arrive. See Figure 1 for an illustration. The pseudocode is provided in the full version.

We consider the following partition of the entire input into super-stages as follows: The first super-stage, Γ_0, contains the first n_0 items. The second super-stage, Γ_1, contains the next $n_1 - n_0$ items. In general, for $i > 0$, the $(i+1)^{\text{th}}$ super-stage, Γ_i, contains $\min\{n_i, n\} - n_{i-1}$ items which are given by $x_{n_{i-1}+1}, x_{n_{i-1}+2}, \ldots, x_{\min\{n,n_i\}}$. So, essentially, `ImpAlg` can be thought of running `Alg` on each super-stage separately. The number of super-stages is given by $\kappa := \lceil \log(n/n_0) \rceil$. The last super-stage might not be *full*, i.e., ideally, it should contain $n/2$ items but it may not. An even worse case would be when the last stage of the last super-stage is not full, i.e., when it contains fewer than $|\Gamma_{\tau-1}|/2$ items where $\Gamma_{\tau-1}$ is the last super-stage. To see why this is bad, note that since we had assumed the value of n in `Alg`, it was possible to make sure that the last stage had exactly $n/2$ items. This is crucial because our analysis heavily relied on the fact that $|T_j| = \left|\overline{T_j}\right|$ for any stage T_j. This meant that the item set T_j fit almost perfectly in the proxy packing of $\overline{T_j}$. But if the last stage T_{m-1} contains far fewer items than $\overline{T_{m-1}}$, then we might be opening far too many proxy bins than required (see the description of the last super-stage in Figure 1). Hence, we need a slight tweak in `Alg` in the way it packs a stage.

2.2.1 A Tweak to `Alg` in the Way in which a Stage is Packed

Recall that for any $j > 1$, we pack T_j as follows. Just before the stage T_j starts, we pack $\overline{T_j}$ using $\mathcal{A}_\alpha$ (the proxy packing). Instead of packing the entire set $\overline{T_j}$ at once, we do this in chunks. We divide the entire set $\overline{T_j}$ into $\eta := 1/\delta$ number of *equal* chunks. Let's denote these chunks by $\overline{T_j}^{(1)}, \overline{T_j}^{(2)}, \ldots, \overline{T_j}^{(\eta)}$. Suppose each chunk contains c_j number of items. We first pack $\overline{T_j}^{(1)}$ using $\mathcal{A}_\alpha$. Let this packing be denoted by $\mathcal{P}_j^{(1)}$. We pack the first c_j items of T_j by fitting them in the packing $\mathcal{P}_j^{(1)}$ just like we packed a stage in `Alg` by creating S-slots for small items and replacing a proxy item if we need to pack a large item. Then, we pack $\overline{T_j}^{(2)}$

using $\mathcal{A}_\alpha$ and fit the next c_j items of T_j in this packing just like before. We continue this process until all the items in T_j are packed. This way, if the size of T_j is very small compared to that of $\overline{T_j}$, then we only compute the proxy packing of only a few chunks and not of the entire set $\overline{T_j}$. On the other hand, if $|T_j|$ is significant when compared to $\left|\overline{T_j}\right|$, then we are anyway good since the number of chunks η is a constant. Intuitively, the optimal solution for $\overline{T_j}$ and the union of optimal solutions for η chunks $\overline{T_j}^{(1)}, \overline{T_j}^{(2)}, \ldots, \overline{T_j}^{(\eta)}$ are close enough when η is a constant. See the pseudocode of `PackStage` in the full version for details.

2.2.2 Analysis

The analyses of both `ImpAlg` and the tweak to `Alg` can be found in the full version. The intuition to why the tweak to `Alg` doesn't cause much problem has already been provided above. The analysis of `ImpAlg` is a bit complicated though. When n was known we had $O(1)$ stages. However, now we can have $\kappa := \lceil \log(n/n_0) \rceil$ number of super-stages. There can arise two problems:

- We can not analyze each super-stage individually and then sum up the performance guarantees, as we can not use union bound for a super-constant number of events. Moreover, we can't even use the analysis of `Alg` for the first few super-stages since they might only have a constant number of items. So, we consider the $\kappa_1 := \left\lceil \log(\delta^7 n) \right\rceil$ number of the initial super-stages at a time. We show that these initial super-stages contain only a small fraction of the entire input. Each of the final $(\kappa - \kappa_1)$ super-stages can be individually analyzed using the analysis of `Alg`.
- For each super-stage, we can have a constant number of S-bins (bins which contain only small items) with less occupancy. However, since the number of super-stages itself is a super-constant, this can result in a lot of wasted space. For this, we exploit the monotonicity of NF to ensure that we can pack small items from a super-stage into empty slots for small items from the previous stages.

See the full version for the details.

3 Best-Fit under the Random-Order Model

In this section, we will prove Theorems 3 and 4.

3.1 When Item Sizes are Larger than 1/3

First, let us recall upright matching and a related result that we will be using.

Upright Matching Problem. For a positive integer k, let S_k denote the set of all permutations of $[k]$. Consider a set of points P in the two-dimensional coordinate system. Suppose each item is marked as a *plus* point or a *minus* point. Let P^+ denote the set of plus points and let P^- denote the set of minus points. An edge exists between two points p^+, p^- iff $p^+ \in P^+, p^- \in P^-$ and iff p^+ lies above and to the right of p^-, i.e., both the coordinates of p^+ are greater than or equal to the corresponding coordinates of p^-. The objective is to find a maximum matching in this graph or, in other words, minimize the number of unmatched points. We denote the number of unmatched points by $U(P)$.

We will use the following variant of upright matching to prove the final result. Refer to [9] for the proof of the following lemma.

▶ **Lemma 12** ([9]). *Let $k \in \mathbb{N}$ and let $\mathcal{A} = \{a_1, a_2, \ldots, a_{2k}\}$ such that $a_i \geq a_{k+i}$ for all $i \in [k]$. Define a set of* plus *points $P^+ = \{(i, a_i) : i \in [k]\}$ and a set of* minus *points $P^- = \{(i, a_i) : k < i \leq 2k\}$. Suppose we randomly permute the x-coordinates of*

$P^+ \cup P^-$, i.e., for a uniform random permutation $\pi \in S_{2k}$, we redefine P^+ and P^- as $P^+ = \{(\pi(i), a_i) : i \in [k]\}$ and $P^- = \{(\pi(i), a_i) : k < i \leq 2k\}$. Let $P = P^+ \cup P^-$. Then, there exist universal constants $a, C, K > 0$ such that

$$\mathbb{P}\left[U(P) \geq K\sqrt{k}(\log k)^{3/4}\right] \leq C\exp(-a(\log k)^{3/2}) \tag{16}$$

▶ Remark 13. In the above lemma, if we change the definitions of P^+, P^- to be $P^+_{\text{new}} = \{(-\pi(i), a_i) : i \in [k]\}$, $P^-_{\text{new}} = \{(-\pi(i), a_i) : k < i \leq 2k\}$, the guarantee given by Equation (16) doesn't change since the new set $P^+_{\text{new}} \cup P^-_{\text{new}}$ can be constructed by taking a mirror image of the original set $P^+ \cup P^-$ with respect to the y-axis. Since we consider random permutations, the probability of a set and its mirror image is the same.

With the above lemma and remark at hand, we now proceed to prove Theorem 3. Albers et al. [1] showed that the asymptotic random order ratio of the Best-Fit algorithm is at most 1.25 when all the item sizes are more than 1/3. In this section, we improve it further and show that, Best-Fit for this special case under the random-order model is nearly optimal. We first show that the Modified Best-Fit algorithm [11] is nearly optimal and we analyze this using the above variant of stochastic upright matching. The Modified Best-Fit (MBF) algorithm is the same as BF except that it closes a bin if it receives an item of size less than 1/2. Shor[44] showed that MBF *dominates* BF, i.e., for any instance I, $\text{BF}(I) \leq \text{MBF}(I)$. MBF can be easily reduced to upright matching as follows. Given an instance $I = \{x_1, \ldots, x_n\}$, for any item $x_i \in I$, $x_i \in P^-$ if $x_i \leq 1/2$ with x-coordinate as $-i$ and y-coordinate as x_i, and $x_i \in P^+$ if $x_i > 1/2$ with x-coordinate as $-i$ and y-coordinate as $1 - x_i$. So, any item x_s of size $\leq 1/2$ can be matched with an item x_ℓ of size $> 1/2$ if and only if, x_ℓ arrives before x_s and the remaining space in the bin occupied by x_ℓ is more than the size of x_s.

Define an item x_i as a large item (L) if $x_i > 1/2$; otherwise, as a medium item (M) if $x_i \in (1/3, 1/2]$. We define a bin as LM-bin if it contains one large item and one medium item. We use the following lemma which was proved in [1] using the monotonicity property of BF when all item sizes are more than 1/3.

▶ **Lemma 14** ([1]). *Let I be any list that can be packed into $\text{Opt}(I)$ number of LM-bins. If Best-Fit has an AAR of α for I, then it has an AAR of α for any list of items larger than 1/3 as well.*

Consider an input instance which has an optimal packing containing only LM-bins. Consider the number of bins opened by MBF for such instances. Each large item definitely opens a new bin, and a medium item opens a new bin if and only if it can not be placed along with a large item, i.e., it is "unmatched". So, the number of bins opened by MBF equals (number of large items+number of unmatched medium items).

Now, we will prove our result.

▶ **Theorem 15.** *For any list I of items larger than 1/3, the asymptotic random order ratio $RR^{\infty}_{BF} = 1$.*

Proof. From Lemma 14, it is enough to prove the theorem for any list I that can be packed in $\text{Opt}(I)$ LM-bins. So, we can assume that I has k large items and k medium items where $\text{Opt}(I) = k$. Now consider the packing of MBF for a randomly permuted list I_σ. We have, $\text{MBF}(I_\sigma) = (k + \text{number of unmatched medium items})$. Since the optimal packing has all the items matched, we can reduce the following case into the matching variant in Lemma 12: Let ℓ_i, m_i denote the sizes of the large item and the medium item respectively in the i^{th} bin of the

optimal solution. For $i \in [k]$, we let $a_i = 1 - \ell_i$ and $a_{k+i} = m_i$ and let $\mathcal{A} = \{a_1, a_2, \ldots, a_{2k}\}$. Note that the required condition in Lemma 12, i.e., $a_i \geq a_{k+i}$ is satisfied. The arrival order is randomly sampled from S_{2k}. So, we have

$$\mathrm{MBF}(I_\sigma) = k + \frac{U(P)}{2} \leq k + K\sqrt{k}(\log k)^{3/4}$$

with probability of at least $1 - C\exp(-a(\log k)^{3/2})$ for some universal constants $a, C, K > 0$. Since MBF dominates BF we have

$$\mathbb{P}\left[\mathrm{BF}(I_\sigma) \leq k + K\sqrt{k}(\log k)^{3/4}\right] \geq 1 - C\exp(-a(\log k)^{3/2}).$$

In case the high probability event does not occur, we can use the bound of $\mathrm{BF}(I) \leq 1.7\mathrm{Opt}(I) + 2$. Let $p := C\exp(-a(\log k)^{3/2})$. Then

$$\begin{aligned}\mathbb{E}\left[\mathrm{BF}(I_\sigma)\right] &\leq p(1.7\mathbb{E}\left[k\right] + 2) + (1-p)(\mathbb{E}\left[k\right] + K\sqrt{\mathbb{E}\left[k\right]}(\log \mathbb{E}\left[k\right])^{3/4}) \\ &\leq \mathbb{E}\left[\mathrm{Opt}(I)\right] + o(\mathbb{E}\left[\mathrm{Opt}(I)\right]) \qquad \text{(since } p = o(1)\text{)}\end{aligned}$$

So, we get: $RR^{\infty}_{BF} = \limsup_{k\to\infty}\left(\sup_{I:\mathrm{Opt}(I)=k}(\mathbb{E}[\mathrm{BF}(I_\sigma)]/\mathrm{Opt}(I))\right) = 1$. This completes the proof. ◀

3.2 The 3-Partition Problem under Random-Order Model

In this section, we analyze the Best-Fit algorithm under the random-order model given that the item sizes lie in the range $(1/4, 1/2]$, and thus prove Theorem 4. We call an item *small* if its size lies in the range $(1/4, 1/3]$ and *medium* if its size lies in the range $(1/3, 1/2]$. Let I be the input list of items and let $n := |I|$. Recall that given σ, a uniform random permutation of $[n]$, I_σ denotes the list I permuted according to σ. We denote by $\mathrm{Opt}(I_\sigma)$, the number of bins used in the optimal packing of I_σ and by $\mathrm{BF}(I_\sigma)$, the number of bins used by Best-Fit to pack I_σ. Note that $\mathrm{Opt}(I_\sigma) = \mathrm{Opt}(I)$.

If there exists a set of three small items in I_σ such that they arrive as three consecutive items, we call that set to be an *S-triplet*. We call a bin to be a k-bin if it contains exactly k items, for $k \in \{1, 2, 3\}$. We sometimes refer to a bin by mentioning its contents more specifically as follows: An MS-bin is a 2-bin which contains a medium item and a small item. Similarly, an SSS-bin is a 3-bin which contains three small items. Likewise, we can define an M-bin, S-bin, MM-bin, SS-bin, MMS-bin, and MSS-bin.

Since the item sizes lie in $(1/4, 1/2]$, any bin in the optimal packing contains at most three items. For the same reason, in the packing by Best-Fit, every bin (with one possible exception) contains at least two items. This trivially shows that the ECR of Best-Fit is at most $3/2$. To break the barrier of $3/2$, we use the following observations.

- Any 3-bin must contain a small item.
- So, if the optimal solution contains a lot of 3-bins, then it means that the input set contains a lot of small items.

We will prove that if there exist many small items in the input, then with high probability, in a random permutation of the input, there exist many disjoint S-triplets.

▷ Claim 16. Let m be the number of small items in the input set I, and let X_σ denote the maximum number of mutually disjoint S-triplets in I_σ. Suppose $m \geq cn$ where $c = 0.00033$, then the following statements hold true:

1. $\mathbb{E}\left[X_\sigma\right] \geq m^3/(3n^2) \geq c^3 n/3$.
2. $X_\sigma \geq c^3 n/3 - o(n)$ with high probability.

Then, we prove that Best-Fit packs at least one small item from an S-triplet in a 3-bin or in an SS-bin.

▷ Claim 17. Let $\{S_1, S_2, S_3\}$ be an S-triplet in I such that S_3 follows S_2 which in turn follows S_1. Then, in the final packing of Best-Fit of I, at least one of S_1, S_2, S_3 is packed in a 3-bin or in an SS-bin.

But the number of SS-bins in the final packing of Best-Fit can be at most one. So, we obtain that the number of 3-bins in the Best-Fit packing is significant. With these arguments, the proof of Theorem 4 follows. The detailed proofs of the above two claims and the final theorem is given in the full version.

4 Conclusion

We studied online bin packing under two stochastic settings, namely the i.i.d. model, and the random-order model. For the first setting, we devised a meta-algorithm which takes any offline algorithm $\mathcal{A}_\alpha$ with an AAR of α (where α can be any constant ≥ 1), and produces an online algorithm with an ECR of $(\alpha + \varepsilon)$. This shows that online bin packing under the i.i.d. model and offline bin packing are almost equivalent. Using any AFPTAS as $\mathcal{A}_\alpha$ results in an online algorithm with an ECR of $(1 + \varepsilon)$ for any constant $\varepsilon > 0$. An interesting question of theoretical importance is to find whether achieving an ECR of 1 is possible or not. Another related open question is if we can settle online bin packing under the random-order model as well.

Then, we studied the analysis of the well-known Best-Fit algorithm under the random-order model. First, we proved that the ECR of Best-Fit is equal to one if all the item sizes are greater than $1/3$. Then, we improved the analysis of the Best-Fit from 1.5 to ≈ 1.4941, for the special case when the item sizes are in the range $(1/4, 1/2]$. An open question is to further improve this analysis since these instances are conjectured to be the hardest (offline) instances of bin packing. Another interesting problem would be to improve the lower bound on the ECR of Best-Fit in this model (which is currently 1.1 due to [2]).

References

1 Susanne Albers, Arindam Khan, and Leon Ladewig. Best fit bin packing with random order revisited. *Algorithmica*, 83(9):2833–2858, 2021.

2 Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and GAP in the random order model. *Algorithmica*, 83(6):1750–1785, 2021.

3 Brenda S Baker and Edward G Coffman, Jr. A tight asymptotic bound for next-fit-decreasing bin-packing. *SIAM Journal on Algebraic Discrete Methods*, 2(2):147–152, 1981.

4 János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new and improved algorithm for online bin packing. In *ESA*, volume 112, pages 5:1–5:14, 2018.

5 János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new lower bound for classic online bin packing. In *WAOA*, volume 11926, pages 18–28. Springer, 2019.

6 Nikhil Bansal, Marek Eliás, and Arindam Khan. Improved approximation for vector bin packing. In *SODA*, pages 1561–1579, 2016.

7 Jozsef Bekesi, Gabor Galambos, and Hans Kellerer. A 5/4 linear time bin packing algorithm. *Journal of Computer and System Sciences*, 60(1):145–160, 2000.

8 Jon Louis Bentley, David S Johnson, Frank Thomson Leighton, Catherine C McGeoch, and Lyle A McGeoch. Some unexpected expected behavior results for bin packing. In *STOC*, pages 279–288, 1984.

9 Carsten Oliver Fischer. *New Results on the Probabilistic Analysis of Online Bin Packing and its Variants*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, December 2019.

10 Edward G Coffman Jr, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of combinatorial optimization*, pages 455–531. Springer New York, 2013.

11 Edward G Coffman Jr, David S Johnson, George S Lueker, and Peter W Shor. Probabilistic analysis of packing and related partitioning problems. *Statistical Science*, 8(1):40–47, 1993.

12 Edward G Coffman Jr, David S Johnson, Peter W Shor, and Richard R Weber. Bin packing with discrete item sizes, part ii: Tight bounds on first fit. *Random Structures & Algorithms*, 10(1-2):69–101, 1997.

13 Edward G Coffman Jr, Kimming So, Micha Hofri, and AC Yao. A stochastic model of bin-packing. *Information and Control*, 44(2):105–115, 1980.

14 Janos Csirik, David S Johnson, Claire Kenyon, James B Orlin, Peter W Shor, and Richard R Weber. On the sum-of-squares algorithm for bin packing. *Journal of the ACM (JACM)*, 53(1):1–65, 2006.

15 W Fernandez de la Vega and George S Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349–355, 1981.

16 Brian C Dean, Michel X Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008.

17 Friedrich Eisenbrand, Dömötör Pálvölgyi, and Thomas Rothvoß. Bin packing via discrepancy of permutations. In *SODA*, pages 476–481, 2011.

18 Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and Shan Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *FOCS*, pages 117–126, 2009.

19 Thomas S Ferguson. Who solved the secretary problem? *Statistical science*, 4(3):282–289, 1989.

20 Carsten Fischer and Heiko Röglin. Probabilistic analysis of the dual next-fit algorithm for bin covering. In *LATIN*, pages 469–482, 2016.

21 Carsten Fischer and Heiko Röglin. Probabilistic analysis of online (class-constrained) bin packing and bin covering. In *LATIN*, volume 10807, pages 461–474. Springer, 2018.

22 Anupam Gupta, Ravishankar Krishnaswamy, and R Ravi. Online and stochastic survivable network design. *SIAM Journal on Computing*, 41(6):1649–1672, 2012.

23 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. *Mathematics of Operations Research*, 46(1):115–133, 2021.

24 Anupam Gupta and Sahil Singla. Random-order models. In *Beyond the Worst-Case Analysis of Algorithms*, pages 234–258. Cambridge University Press, 2020.

25 Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *SODA*, pages 2616–2625, 2017.

26 David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

27 David S Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, 1974.

28 David S Johnson, Alan Demers, Jeffrey D Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing*, 3(4):299–325, 1974.

29 David S Johnson and Michael R Garey. A 71/60 theorem for bin packing. *J. Complex.*, 1(1):65–106, 1985.

30 Edward G Coffman Jr, János Csirik, Lajos Rónyai, and Ambrus Zsbán. Random-order bin packing. *Discret. Appl. Math.*, 156(14):2810–2816, 2008.

31 Narendra Karmarkar and Richard M Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *FOCS*, pages 312–320, 1982.

32 Claire Kenyon. Best-fit bin-packing with random order. In *SODA*, pages 359–364, 1996.

33 Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, July 1985.

34 Chan C Lee and Der-Tsai Lee. Robust on-line bin packing algorithms. *Technical Report, Northwestern University*, 1987.

35 Tom Leighton and Peter Shor. Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. *Combinatorica*, 9(2):161–187, 1989.

36 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *STOC*, pages 597–606, 2011.

37 Frank D Murgolo. Anomalous behavior in bin packing algorithms. *Discret. Appl. Math.*, 21(3):229–243, 1988.

38 Alantha Newman, Ofer Neiman, and Aleksandar Nikolov. Beck's three permutations conjecture: A counterexample and some consequences. In *FOCS*, pages 253–262, 2012.

39 Prakash V Ramanan. Average-case analysis of the smart next fit algorithm. *Inf. Process. Lett.*, 31(5):221–225, 1989.

40 W. T. Rhee. Inequalities for bin packing-iii. *Optimization*, 29(4):381–385, 1994. doi:10.1080/02331939408843965.

41 Wansoo T Rhee and Michel Talagrand. Exact bounds for the stochastic upward matching problem. *Transactions of the American Mathematical Society*, 307(1):109–125, 1988.

42 Wansoo T Rhee and Michel Talagrand. On-line bin packing of items of random sizes, ii. *SIAM Journal on Computing*, 22(6):1251–1256, 1993.

43 Wansoo T Rhee and Michel Talagrand. On line bin packing with items of random size. *Mathematics of Operations Research*, 18(2):438–445, 1993.

44 Peter W Shor. The average-case analysis of some on-line algorithms for bin packing. *Combinatorica*, 6(2):179–200, 1986.

45 Joel Spencer. *Ten lectures on the probabilistic method*. SIAM, 1994.

Competitive Vertex Recoloring

Yossi Azar
School of Computer Science, Tel Aviv University, Israel

Chay Machluf
School of Electrical Engineering, Tel Aviv University, Israel

Boaz Patt-Shamir
School of Electrical Engineering, Tel Aviv University, Israel

Noam Touitou
School of Computer Science, Tel Aviv University, Israel

Abstract

Motivated by placement of jobs in physical machines, we introduce and analyze the problem of online recoloring, or online disengagement. In this problem, we are given a set of n weighted vertices and a k-coloring of the vertices (vertices represent jobs, and colors represent physical machines). Edges, representing conflicts between jobs, are inserted in an online fashion. After every edge insertion, the algorithm must output a proper k-coloring of the vertices. The cost of a recoloring is the sum of weights of vertices whose color changed. Our aim is to minimize the competitive ratio of the algorithm, i.e., the ratio between the cost paid by the online algorithm and the cost paid by an optimal, offline algorithm.

We consider a couple of polynomially-solvable coloring variants. Specifically, for 2-coloring bipartite graphs we present an $O(\log n)$-competitive deterministic algorithm and an $\Omega(\log n)$ lower bound on the competitive ratio of randomized algorithms. For $(\Delta + 1)$-coloring, we present tight bounds of $\Theta(\Delta)$ and $\Theta(\log \Delta)$ on the competitive ratios of deterministic and randomized algorithms, respectively (where Δ denotes the maximum degree). We also consider a dynamic case which allows edge deletions as well as insertions. All our algorithms are applicable to the case where vertices are weighted and the cost of recoloring a vertex is its weight. All our lower bounds hold even in the unweighted case.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Dynamic graph algorithms; Computer systems organization → Cloud computing

Keywords and phrases coloring with recourse, anti-affinity constraints

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.13

Category Track A: Algorithms, Complexity and Games

Funding *Yossi Azar*: Supported in part by the Israel Science Foundation (grant No. 2304/20).
Boaz Patt-Shamir: Supported in part by the Israel Science Foundation, grant No. 1948/21.

1 Introduction

The following situation is not uncommon in server farms, such as a data center of a cloud provider: Jobs (or virtual machines) are created and located at various physical machines, and then it turns out that some of the jobs cannot co-exist on the same machine. Such restrictions may be due to various reasons, e.g., conflicting resource requirements, security considerations etc. (cloud providers allow users to express these constraints using so-called "anti-affinity rules:" see, e.g., [1]). When such a conflict arises between co-located jobs, at least one of these jobs must migrate to another machine, at a cost. Motivated by such scenarios, in this paper we introduce and study the *Disengagement Problem*, in which disengagement (anti-affinity) requests arrive on-line, and the goal is to minimize the overall cost of job migrations.


More specifically, we abstract the problem as follows (more details in Section 2). We view the problem as an online graph coloring problem with recourse. Initially, we are given a vertex-weighted graph $G_0 = (V, \emptyset)$ of n isolated vertices, and a coloring $c_0 : V \to [k]$, where $k \in \mathbb{N}$ is a given parameter.[1] (Vertices correspond to jobs and colors correspond to machines.) The input is a sequence of edges $e_1, e_2, \ldots$ that arrive one at each step. After each step i, we are required to output a new coloring $c_i : V \to [k]$ such that under c_i, no edge in $e_1, \ldots, e_i$ is monochromatic. The cost of a given sequence of colorings $c_0, c_1, \ldots$ is the total weight of recolorings, i.e., the sum over all vertices of vertex weight times the number of times that vertex was recolored. Due to this formalization, in this paper we refer to the problem interchangeably as Vertex Recoloring or Disengagement.

Our results. We study online disengagement from the competitive analysis viewpoint [14]. (Note that an optimal offline solution never recolors a vertex more than once.) Since vertex coloring is a hard problem [16], we focus on two cases which are polynomially solvable, namely 2-colorable graphs and $(\Delta + 1)$-coloring, where Δ denotes the maximum vertex degree.

1. For 2-coloring, we give a deterministic, $O(\log n)$-competitive algorithm. We also show a matching lower bound of $\Omega(\log n)$ on the competitiveness of *randomized* algorithms.
2. For $(\Delta + 1)$-coloring, we present:
   - A deterministic, $O(\Delta)$-competitive algorithm, and a matching $\Omega(\Delta)$ lower bound.
   - A randomized, $O(\log \Delta)$-competitive algorithm, and a matching $\Omega(\log \Delta)$ lower bound.

In all the above cases, our algorithms work with weighted vertices, and our lower bounds apply even to unweighted instances.

We briefly consider the variant of *fully-dynamic* online disengagement, in which conflicts are temporary, as opposed to the semi-dynamic disengagement in which conflicts never disappear (see formalization in Section 2). It turns out that fully-dynamic disengagement is substantially harder than semi-dynamic disengagement. More specifically, even though in this case the offline cost is not bounded by n, all lower bounds of the semi-dynamic case apply as well. We also show that the special case of fully-dynamic disengagement with two colors on an odd cycle (which is of course not 2-colorable in the semi-dynamic case) is at least as hard as a certain metrical task system on an odd cycle, and hence admits a lower bound of $\Omega(n)$-competitiveness for deterministic algorithms (whereas the semi-dynamic problem on even cycles is solvable by an $O(\log n)$-competitive deterministic algorithm).

Our Techniques. For the bipartite (2-coloring) problem, our algorithm maintains a partition of the vertices into connected components. Since there are 2 colors, each connected component has exactly two legal colorings. When two components merge due to the arrival of a monochromatic edge, the algorithm must choose between the two legal colorings for that new component, where each coloring implies recoloring one of the joined components. There are two natural heuristics for choosing which component to recolor. The greedy approach is to recolor the lighter component; another approach, with an eye to the offline solution (which, knowing all future edges, recolors every vertex at most once), is to recolor so as to minimize the change from the initial coloring of the graph (as measured in weighted Hamming distance). Unfortunately, it can be shown that each of these algorithms has $\Omega(n)$ competitive ratio. However, perhaps surprisingly, by a balanced combination of these two noncompetitive approaches, we obtain an $O(\log n)$-competitive deterministic algorithm. The competitive ratio is bounded using amortized analysis.

[1] We use the notation $[k] \stackrel{\text{def}}{=} \{1, 2, \ldots, k\}$.

For $(\Delta+1)$-coloring, key observation is that vertices that are recolored by the optimal algorithm must be incident on all edges which are monochromatic by their initial coloring. Therefore, our algorithm maintains a vertex cover of the edges which are monochromatic by their initial coloring. In our algorithm, upon the arrival of an edge, only vertices in that vertex cover are allowed to change their color. This way, we limit the set of vertices that change their color. Coupling this with an upper bound on the total number of recolorings which a vertex undergoes (using Δ), we can prove a bound on the competitive ratio. Specifically, we maintain a 2-approximate weighted vertex cover using the primal-dual (local ratio) 2-competitive algorithm; when a monochromatic edge arrives, an appropriate vertex from the vertex cover is recolored by a color not taken by any of its neighbors. In the deterministic case, which achieves $O(\Delta)$-competitiveness, the new color is an arbitrary available color, and in the randomized case, the new color is a random (uniformly chosen) available color. In the latter case, some subtle probabilistic analysis shows $O(\log \Delta)$-competitiveness.

Related Work. We do not know of previous work on *competitive recoloring* of vertices: some considered recoloring, some considered competitive coloring, but this is the first work to consider both simultaneously. We review some relevant results below.

Competitive coloring. Let us start with competitive coloring. Work on online vertex coloring, starting with the paper of Lovász et al. [13], assume that vertices arrive one by one, each vertex with all its incident edges. When a vertex arrives it must be assigned a color irrevocably, maintaining a legal vertex coloring at all times. The goal is to minimize the number of colors used by the algorithm. It is known that the competitive ratio in this case is $\Theta(n/\log n)$ [9, 10].

Dynamic algorithms for recoloring. Recoloring has also been considered in the dynamic data structures setting, in which one seeks to maintain a proper coloring while minimizing the number of recolorings per vertex/edge update (arrival or departure); see, e.g., [8, 5, 15, 12]. This model differs from ours in that we do not measure costs w.r.t. the number of steps, but w.r.t. an optimal recoloring algorithm OPT for the input sequence. In particular, when $\text{OPT} \ll T$, where T is the length of the input sequence, we remain competitive against OPT (rather than T); however, in dynamic recoloring (or recoloring with recourse), a constant number of recolorings per edge arrival results in $\Omega(T)$ cost.

For general graphs, coloring a graph using a competitively-small palette (w.r.t. its chromatic number) is NP-hard; thus, restricting the set of graphs or their chromatic number is necessary for polynomial running time. This is the case for the work of Kashyop et al. [12] (who focus on bipartite graphs, max-degree bounded graphs, and graphs with bounded arboricity), and Bosek et al. [8] (who focus on interval graphs). Indeed, our work also focuses on coloring problems which can be tackled in polynomial time, namely 2-coloring and $(\Delta+1)$-coloring. However, one can also consider coloring in general graphs, using an (exponential time) oracle for graph coloring. This is the case for the works of Barba et al. [5] and Solomon and Wein [15]. The best current result, due to [15], is a deterministic algorithm that maintains a graph which is $O(\frac{\log^3 n}{d})$-competitive with respect to the number of colors (against the graph's chromatic number), while using $O(d)$ amortized recolorings per update, where d is a free parameter (allowing randomization improves this result slightly).

Dynamic graph partitioning. A problem closely related to online disengagement is "dynamic balanced graph partitioning," introduced by Avin et al. [3, 2]. In this problem vertices are located in finite-capacity clusters (servers), and communication requests arrive online.

Communication between vertices incurs cost unless the vertices are located in the same cluster. Vertices can migrate to other clusters, at a (larger) cost, and the algorithm is required to find a placement of the vertices in clusters at each step. The goal is to minimize cost, and the overall measure is the competitive ratio, possibly with resource augmentation (i.e., assuming that the capacity available to the algorithm is larger than the capacity available to the adversary). Intuitively, this problem is the flip side of disengagement: In partition, requests are to collocate vertices (subject to capacity constraints), whereas in disengagement, requests are to separate them. In [3], the fully-dynamic variant of the partition problem is considered, where collocation requests are temporary: a pair of vertices may be collocated by a request and later separated. It is shown in [3] that the competitive ratio of deterministic algorithms for the dynamic case is $O(k \log k)$ and $\Omega(k)$, where k is the capacity of a cluster (both bounds allow for resource augmentation). Recently, the semi-dynamic variant of the partition problem was considered, in which it is guaranteed that there exists a feasible placement of the vertices to clusters so that all communication is local (i.e., occurs within a cluster). In [11], tight bounds on the competitive ratios of deterministic and randomized algorithms for semi-dynamic partition are given: $\Theta(\ell \log W)$ and $\Theta(\log \ell + \log W)$, respectively, where ℓ is the number of servers and W is the server capacity .

Paper organization. The remainder of this paper is organized as follows. In Section 2 we formalize the model and introduce some notation. In Section 3 we study the bipartite case. In Section 4 we study $(\Delta + 1)$-disengagement. In Section 5 we consider fully-dynamic disengagement. We conclude in Section 6 with a few interesting open problems.

2 Model and Notation

Preliminaries. Given a natural number k, we use $[k]$ to denote $\{1, 2, \ldots, k\}$. In this paper we are concerned with colorings of vertices in graphs. Given an undirected simple graph $G = (V, E)$, a *proper k-coloring* of G is an assignment $c : V \to [k]$ such that $c(u) \neq c(v)$ for all $(u, v) \in E$.

We shall use the following definition extensively.

▶ **Definition 1.** *Let $G = (V, E)$ be a graph, and let $w : V \to \mathbb{R}^+$ be a vertex weight function. Let c, c' be two colorings of a graph G. The* weighted Hamming distance *between c and c', denoted $\delta_H(c, c')$, is defined as follows: $\delta_H(c, c') = \sum_{v\,:\,c(v)\neq c'(v)} w(v)$.*

Problem statement. In this paper, we study the following problem.

Online Disengagement (Vertex Recoloring)

Initial Input:

- A set V of n vertices with weights $w : V \to \mathbb{R}^+$
- The number of colors $k \in \mathbb{N}$
- A k-coloring c_0 of V

Online Input: In each step $i = 1, \ldots, M$, an edge e_i between two vertices of V.

Output: After each step i, a proper k-coloring of the vertices c_i w.r.t edges $\{e_1, \ldots, e_i\}$.

Goal: Minimize the total weight of vertex recolorings, i.e., $\sum_{i=1}^{M} \delta_H(c_{i-1}, c_i)$.

We use $E_i = \{e_1, e_2, ..., e_i\}$ to denote the set of edges that have arrived up to and including the ith step, and we use $G_i = (V, E_i)$ to denote the known graph after step i.

Note that in Online Disengagement, as is the case for any semi-dynamic graph problem, the input sequence length can be assumed to be finite without loss of generality, as there are only $\binom{n}{2}$ possible distinct edges connecting vertices of V.

Special cases. In this 1paperwe consider some special cases of online disengagement. The variants we consider are the following.

- In *Unweighted* Online Disengagement, all vertices have weight 1.
- In *Bipartite* Online Disengagement, we are guaranteed that the input graph is 2-colorable.
- In $(\Delta + 1)$ Online Disengagement, we are guaranteed that the maximum degree of the input graph does not exceed Δ, and the available number of colors is $\Delta + 1$.

Fully-dynamic disengagement. The problem, as stated above, is called *semi dynamic*, in the sense that edges only arrive and never leave. However, in some cases, the presence of edges restricting the solution may be temporary. In the *fully-dynamic* online disengagement variant, edges may leave too. The input and the output to the problem, as well as the cost measure, are the same, but the feasibility requirement is different: The coloring output by the algorithm after each step need be a proper coloring for $G_i = (V, E'_i)$, where $E'_i \subseteq E_i$. We discuss several different definitions of E'_i in Section 5.

3 Bipartite Disengagement

In this section we consider disengagement for bipartite graphs and $k = 2$. In other words, we are promised that the final graph G (and hence every intermediate graph) is 2-colorable, and the requirement is to find, at each step, a legal 2-coloring, while minimizing the overall cost.

Note that this case is stricter than $(\Delta + 1)$-disengagement, since the maximum degree Δ is typically large (the case $\Delta = 1$ is trivial). Indeed, we show in this section that the competitive ratio in this case is $\Theta(\log n)$. We start with an $O(\log n)$-competitive deterministic algorithm for the weighted case, and then prove an $\Omega(\log n)$ lower bound on the competitive ratio of any (randomized) algorithm that holds even in the unweighted case.

3.1 Simple approaches

Any algorithm for 2-coloring maintains, throughout its execution, the partition of the nodes into connected components. Since there are only 2 colors, each connected component has two possible proper colorings. Whenever two components are joined by a new edge, if their colorings agree (i.e. the two ends of the arriving edge are colored differently), the algorithm may output the previous coloring unaltered. Otherwise, when the edge is monochromatic, the algorithm must decide which of the two colorings is adopted by the newly created component: each coloring implies recoloring all nodes in one of the components being joined. There are two natural approaches one can consider. One approach we call "greedy" is to recolor the component of the smaller weight; another approach we call "conservative" is to choose the colorings which is closer to the initial coloring. Both approaches are not competitive, as we show below. For convenience, we refer to the two colors as red and blue.

The Greedy algorithm. First, consider the algorithm that always flips the color of the smaller component (weight-wise). Consider an input sequence of a connected component in the form of a double star:

1. Create an edge between two arbitrary red nodes. The node that stays red becomes the *red hub*, and the other one becomes the *blue hub*.

2. Do until there are no isolated nodes:
   a. Let v be some isolated node.
   b. Create a monochromatic edge between v and its matching hub.

Every isolated node is a component of size 1. When it is connected via an edge to the component of the hubs (which is at least of size 2) the algorithm flips its color. In fact, the online algorithm flips the color of every node in the graph except for the red hub (paying a cost of $n-1$), while the optimal algorithm only chooses the other coloring for the two hubs, paying a cost of 1. We conclude that the competitive ratio of the greedy algorithm is $\Omega(n)$.

The Conservative algorithm. Consider now the algorithm which always prefers the coloring that is closer (i.e, smaller Hamming distance) to the initial coloring. Assume w.l.o.g. that initially, at least half of the nodes are red. Now, we subject it to the following input sequence, which creates a connected component in the form of a chain:

1. Create an edge between two red nodes
2. Do until there are no isolated red nodes:
   a. Let v be some red, isolated node.
   b. If one of the chain end-nodes u is red, create an edge (u, v).
   c. Otherwise, create an edge between v and any of the chain end-nodes

The chain grows by one node each iteration; in half of the iterations is has an equal number of blue and red nodes. When that happens, the following holds:

1. One of the chain end-nodes is red (since the chain is of even length)
2. Flipping the colors of all of the nodes in the chain does not increase the distance from the initial coloring

Therefore, in every even iteration, a new edge connects two red nodes, to which the algorithm responds by flipping the color of every node in the chain. Overall, the algorithm pays a cost of $\Omega(n^2)$, while an optimal algorithm colors at most half of the nodes exactly once, or a cost of $O(n)$. It follows that the competitive ratio of the conservative algorithm is $\Omega(n)$ as well.

3.2 A Competitive Algorithm

While each approach discussed in Section 3.1 resulted in $\Omega(n)$ competitive ratio, the following combination of them is $O(\log n)$-competitive: If one of the two possible coloring is such that at least two-thirds of the weighted nodes will have their original color c_0 (and therefore that coloring has less than half the Hamming distance to the original coloring than the alternative), that alternative is chosen by the algorithm. Otherwise, no coloring has a significantly smaller Hamming distance, and in this case the algorithm prefers the "cheaper" coloring, i.e., the algorithm recolors the component with the smaller weight.

Pseudocode of the algorithm to be executed upon the arrival of a new edge e_i is given in Algorithm 1. For a subset of nodes V', we use $c{\restriction}_{V'}$ to denote the restriction of the coloring c to the set of nodes V'. For convenience, we write $\delta(c)$ as a shorthand for $\delta_H(c, c_0)$. We also use $\delta(c{\restriction}_{V'})$ to refer to $\delta_H(c{\restriction}_{V'}, c_0{\restriction}_{V'})$, i.e. the weighted Hamming distance between the coloring c and the initial coloring c_0 restricted to the set of nodes V'.

We consider the partition of V into connected components. Let $\mathcal{P}_i$ be the set of connected components after the arrival of e_i, and define $\mathcal{P}_0 = \{\{v\} \mid v \in V\}$. Note that in the bipartite case, the color of one node determines the coloring of all nodes in its connected component. Therefore, if e_i is monochromatic upon arrival, it necessarily connects two connected components $S_1, S_2 \in \mathcal{P}_{i-1}$ into a new component S (cf. line 5).

We now analyze the algorithm. Specifically, we prove the following theorem.

Algorithm 1 Recoloring for bipartite graphs, invoked upon arrival of edge $e_i = (u, v)$.

if $c_{i-1}(u) \neq c_{i-1}(v)$ **then**
  $c_i := c_{i-1}$
  return *// no recoloring*
end if *// otherwise, u and v belong in different components*
Let S_1, S_2 be the components of u and v in G_{i-1}, and let S be their common component in G_i
Let $w(S_i)$ be the sum of the weights of the nodes in the set S_i
Define colorings γ_1 and γ_2 by flipping the colors of components S_1 and S_2, respectively, i.e., for all $z \in V$:

$$\gamma_1(z) = \begin{cases} c_{i-1}(z), & \text{if } z \notin S_1 \\ 3 - c_{i-1}(z), & \text{if } z \in S_1 \end{cases} \quad \text{and} \quad \gamma_2(z) = \begin{cases} c_{i-1}(z), & \text{if } z \notin S_2 \\ 3 - c_{i-1}(z), & \text{if } z \in S_2 \end{cases}$$

if $\delta(\gamma_1 \restriction_S) \geq 2\delta(\gamma_2 \restriction_S)$ **then**
  $c_i := \gamma_2$
else if $\delta(\gamma_2 \restriction_S) \geq 2\delta(\gamma_1 \restriction_S)$ **then**
  $c_i \leftarrow \gamma_1$
else if $w(S_1) \geq w(S_2)$ **then**
  $c_i \leftarrow \gamma_2$
else
  $c_i \leftarrow \gamma_1$
end if

▶ **Theorem 2.** *Algorithm 1 is $O(\log n)$-competitive.*

Fix an input sequence $\{e_i\}_i$ and an optimal coloring c^*, i.e., $\delta(c^*)$ is minimal among all legal colorings of the final graph G. Denote $r := \delta(c^*)$, and let ALG denote the cost of Algorithm 1 on the given input sequence. We prove the theorem by showing that $\text{ALG} \leq O(r \log r)$.

We need some additional terminology.

- Given a connected component S and a coloring c, we say that S is *well colored* by c if $c \restriction_S = c^* \restriction_S$ is consistent with c^*. Otherwise, we say that S is *badly colored*.
- Let $R \subseteq V$ be the subset of nodes v such that $c^*(v) \neq c_0(v)$. By definitions, $w(R) = r$.

The following claim says that in every badly colored component, a significant fraction of the weight is contributed by nodes recolored by the optimum.

▶ **Proposition 3.** *For all i and all $S \in \mathcal{P}_i$, if S is badly colored then $w(S) < 3w(S \cap R)$.*

Proof. We prove the claim by induction on i, the number of edges inserted. For $i = 0$ we have that $\mathcal{P}_0$ is a collection of singletons. Let $S = \{v\} \in \mathcal{P}_0$. If S is badly colored then by definitions, $v \in R$, and hence $S \cap R = S$, proving the base case.

Assume now that the claim holds for all components in $\mathcal{P}_{i-1}$. It suffices to show that if a new component is created by the arrival of edge $e_i = (v_1, v_2)$, the claim holds for that component. So assume that S is created by merging $S_1, S_2 \in \mathcal{P}_{i-1}$, where $v_1 \in S_1$ and $v_2 \in S_2$.

If $c_{i-1}(v_1) \neq c_{i-1}(v_2)$, then $c_i = c_{i-1}$, and thus the component S is badly colored iff the components S_1, S_2 are both badly-colored (note that it cannot be that only one is badly colored, since that would imply the infeasibility of c^*). Using the induction hypothesis, if S is badly colored then

$$w(S) = w(S_1) + w(S_2) < 3(w(S_1 \cap R) + w(S_2 \cap R)) = 3w(S \cap R) \ ,$$

completing the proof for the case that $c_{i-1}(v_1) \neq c_{i-1}(v_2)$.

Assume henceforth that $c_{i-1}(v_1) = c_{i-1}(v_2)$. When the component S is formed, the algorithm considers the two colorings γ_1 and γ_2 for S, where γ_1 and γ_2 are formed from c_{i-1} by recoloring S_1 and S_2, respectively. Thus, the colorings of S under γ_1 and γ_2 are negations of one another. One of these colorings, wlog γ_1, is consistent with c^*, and thus $\delta(\gamma_1 \restriction_S) = w(S \cap R)$. Since γ_2 is the negation of γ_1, it holds that $\delta(\gamma_2 \restriction_S) = w(S) - w(S \cap R)$.

If the component S is badly colored, it must be that γ_2 is chosen. For this to happen, it must hold that $\delta(\gamma_2 \restriction_S) < 2\delta(\gamma_1 \restriction_S)$. Simplifying, we get $w(S) < 3w(S \cap R)$, completing the proof. ◀

▶ **Corollary 4.** *If a component has weight larger than $3r$, then the component is well-colored.*

Proof. By Proposition 3 and the fact that for any component S, $w(S \cap R) \leq r$. ◀

▶ **Lemma 5.** *The weight of nodes recolored by Algorithm 1 throughout the execution is at most $3r$.*

Proof. Clearly, the total weight of nodes in R which change color during the algorithm is at most $w(R) = r$. It remains to bound the weight of nodes that change color in $V \setminus R$; denote the set of those nodes by U.

Let $\mathcal{B} \subseteq 2^V$ be the collection of subsets of V which were badly-colored components in the algorithm at some point. We claim that $\mathcal{B}$ is a laminar collection – that is, for every two components $S_1, S_2 \in \mathcal{B}$ it holds that either $S_1 \subseteq S_2$, $S_2 \subseteq S_1$ or $S_1 \cap S_2 = \emptyset$. Indeed, $\mathcal{B}$ is laminar since it is contained in the collection of all connected components which are formed in the algorithm, which is itself laminar.

Since $\mathcal{B}$ is a laminar collection, there exists a subcollection $\mathcal{B}' \subseteq \mathcal{B}$ of disjoint components such that $\bigcup_{S \in \mathcal{B}} S = \bigcup_{S \in \mathcal{B}'} S$.

Consider any node $v \in U$. The coloring c^* is consistent with c_0 on this node v; thus, for this node to change color during the algorithm, it must be part of some badly-colored component at some point during the algorithm. This implies that $U \subseteq \bigcup_{S \in \mathcal{B}} S = \bigcup_{S \in \mathcal{B}'}$.

$$\begin{aligned} w(U) = \sum_{S \in \mathcal{B}'} w(U \cap S) &\leq \sum_{S \in \mathcal{B}'} w((V \setminus R) \cap S) \\ &= \sum_{S \in \mathcal{B}'} w(S \setminus R) \\ &\leq 2 \sum_{S \in \mathcal{B}'} w(S \cap R) && \text{by Proposition 3} \\ &\leq 2w(R) \;=\; 2r \;. \end{aligned}$$ ◀

Next, we define a potential function ϕ such that $\phi(i) = 6\delta(c_i)$. Note that $\phi(0) = 0$, and that ϕ is nonnegative. Following the conventional notation, denote $\Delta\phi_i = \phi(i) - \phi(i-1)$ for every $i > 0$.

For every $i \in \mathbb{N}$, let ALG_i be the cost incurred by the algorithm in step i, i.e., $\delta_H(c_i, c_{i-1})$. Define

$$I^+ \stackrel{\text{def}}{=} \{i \mid \text{ALG}_i + \Delta\phi_i > 0\} \;.$$

Then we have

$$\text{ALG} \;=\; \sum_i \text{ALG}_i \;\leq\; \sum_i (\text{ALG}_i + \Delta\phi_i) \;\leq\; \sum_{i \in I^+} (\text{ALG}_i + \Delta\phi_i) \;\leq\; \sum_{i \in I^+} 7\text{ALG}_i \;, \tag{1}$$

where the last inequality uses the fact that changing the coloring of nodes of total weight s increases the coloring's weighted Hamming distance from c_0 by at most s, and thus increases ϕ by at most $6s$.

▶ **Proposition 6.** *If $v \in V$ is recolored in step $i \in I^+$, then the weight of the connected component of v grows by a factor of at least $\frac{5}{4}$ in step i. That is, denoting by S, S' the old and new components of v respectively, it holds that $w(S') \geq \frac{5}{4} w(S)$.*

Proof. Node v changes color when some components $S_1, S_2 \in \mathcal{P}_{i-1}$ are merged to form $S \in \mathcal{P}_i$, and one of these components, wlog S_2, changes color, where $v \in S_2$. We now must show that $w(S) \geq \frac{5}{4} w(S_2)$. Denote by γ_1 and γ_2 the colorings formed from c_{i-1} by recoloring S_1 and S_2 respectively; since S_2 is recolored, the algorithm chose γ_2.

If γ_2 is chosen in line 13 of Algorithm 1, then trivially $w(S) \geq 2w(S_2) \geq \frac{5}{4} w(S_2)$ and the proof is complete.

Otherwise, the algorithm chose γ_2 in line 9 because $\delta(\gamma_1 \restriction_S) \geq 2\delta(\gamma_2 \restriction_S)$. Let us denote $s_1 = w(S_1)$, $s_2 = w(S_2)$, $x_1 = \delta(c_{i-1} \restriction_{S_1})$ and $x_2 = \delta(c_{i-1} \restriction_{S_2})$. It holds that $\delta(\gamma_1 \restriction_S) = s_1 - x_1 + x_2$ and $\delta(\gamma_2 \restriction_S) = s_2 - x_2 + x_1$. Thus,

$$s_1 + x_2 \geq s_1 - x_1 + x_2 \geq 2(s_2 - x_2 + x_1) \geq 2(s_2 - x_2) \ ,$$

and therefore $s_1 \geq 2s_2 - 3x_2$. Adding s_2 to both sides, we have that

$$s_1 + s_2 \geq 3(s_2 - x_2) \ . \tag{2}$$

Now, recall that $i \in I^+$, and thus

$$0 \ \leq \ \text{ALG}_i + \Delta\phi_i \ = \ s_2 + 6((s_2 - x_2) - x_2) \ = \ 7s_2 - 12x_2 \ . \tag{3}$$

Eq. (3) implies that $x_2 \leq \frac{7}{12} s_2$. Plugging into (2), we get $w(S) = s_1 + s_2 \geq 3s_2 - 3x_2 \geq 3s_2 - \frac{7}{4} s_2 = \frac{5}{4} s_2 = \frac{5}{4} w(S_2)$ which completes the proof of the proposition. ◀

Proof of Theorem 2. By Eq. (1), it is enough to bound $7 \sum_{i \in I^+} \text{ALG}_i$. Consider any step i in which some component $S \in \mathcal{P}_{i-1}$ is recolored, upon being connected to a second component in $\mathcal{P}_{i-1}$. Consider the subset $W \stackrel{\text{def}}{=} \left\{ u \in S \mid w(u) < \frac{w(S)}{2n} \right\}$. It holds that $w(W) \leq n \cdot \frac{w(S)}{2n} = \frac{w(S)}{2}$, and thus $w(S) \leq 2w(S \setminus W)$. Therefore,

$$\text{ALG}_i \ = \ w(S) \ \leq \ 2w\left(\left\{ u \in S \ \mid \ w(u) \geq \frac{w(S)}{2n} \right\} \right) \ . \tag{4}$$

Plugging Eq. (4) into Eq. (1), and denoting by $S_i \in \mathcal{P}_{i-1}$ the component that was recolored in step i, it thus remains only to bound the term

$$14 \sum_{i \in I^+} w\left(\left\{ u \in S_i \ \mid \ w(u) \geq \frac{w(S_i)}{2n} \right\} \right) \ . \tag{5}$$

Using Lemma 5, and denoting by U the set of nodes recolored by the algorithm, we know that $w(U) \leq 3r$. Consider a node $u \in U$: each time the node u is recolored in a step from I^+, the weight of the component containing u grows by a factor of at least $\frac{5}{4}$ (by Proposition 6). Thus, each node can be recolored in steps from I^+ at most $O(\log n)$ times before the weight of u's component exceeds $2n \cdot w(u)$. This implies that the term in Eq. (5) can be bounded by $O(\log n) \cdot w(U) = O(\log n) \cdot r$, and thus $\text{ALG} \leq O(\log n) \cdot r$, completing the proof. ◀

3.3 A Lower Bound

We present a general lower bound in the unweighted case. We note that a similar argument is used in [12] by Kashyop et al., where it is shown in the context of data structures that any deterministic algorithm that maintains a 2-coloring must perform $\Omega(n \log n)$ recolorings for the worst-case $O(n)$ edge insertions.

▶ **Theorem 7.** *The competitive ratio of any randomized algorithm for recoloring bipartite graphs is* $\Omega(\log(n))$.

Let $A(I)$ be the cost of algorithm A on an instance (input sequence) I. According to Yao's principle, it suffices to show an instance distribution D such that the expected cost of every deterministic algorithm A is $E[A(I)] = \Omega(\log n)$, where the expectation is for instances chosen by D. In the proof below we construct such a distribution.

Proof. Assume w.l.o.g. that n is a power of 2 (otherwise, we work only with $2^{\lfloor \log n \rfloor}$ nodes, leaving the others isolated). We construct an input sequence in $\log n$ phases, while maintaining the following invariant:

> After each phase h: (i) there are $n/2^h$ connected components, and (ii)each connected component is a chain of 2^h nodes.

Clearly, the invariant holds before the execution begins, when there are n isolated nodes. Note that we allow for any initial coloring.

In each phase h, for $h = 1, \ldots, \log n$, we connect segments in pairs by introducing $n/2^h$ edges: each edge connects *random endpoints* of two segments (this is the only randomization we use). This obviously maintains the invariant after phase h is completed.

Consider phase h for $h > 1$. The merging segments are of even length 2^{h-1} each, and hence each segment has exactly one endpoint of each color. It follows that by construction, each new edge is monochromatic with probability $1/2$. Therefore the expected cost incurred in phase $h > 1$ for any deterministic algorithm is

$$\frac{n}{2^h} \cdot \frac{1}{2} \cdot 2^{h-1} = \frac{n}{4} \ ,$$

because in phase h there are $n/2^h$ merges, and each merge has expected cost $\frac{1}{2} \cdot 2^{h-1}$. In summary, the cost of any deterministic algorithm on a random instance defined as above, over all phases, is at least $(\log n - 1) \cdot n/4 = \Theta(n \log n)$ (the cost of phase 1 depends on the initial coloring). On the other hand, the optimal cost for any n-node graph is never more than n: every node needs to be colored at most once, according to the final graph. The result follows. ◀

4 $\Delta + 1$ Disengagement

In this section we consider $(\Delta + 1)$-coloring, where Δ is an upper bound on the number of neighbors a node may have. We show that in this case, the competitive ratio of deterministic disengagement is $\Theta(\Delta)$ and that the randomized competitive ratio is $\Theta(\log \Delta)$.

4.1 Algorithms

We now present our algorithms for the $\Delta + 1$ disengagement problem. Algorithm 2 is used in both the deterministic and randomized versions: The only difference is the implementation of the "recolor" subroutine it invokes (Algorithm 3 and Algorithm 4). To see how the algorithm works, let us define an auxiliary graph $G'_i = (V, E'_i)$ as follows:

$$(u, v) \in E'_i \iff (u, v) \in E_i \text{ and } c_0(u) = c_0(v) \ ,$$

i.e., the auxiliary graph G'_i includes only edges that connect nodes with the same initial color. The underlying idea of Algorithm 2 is to maintain a small-weight vertex cover of G', denoted C, and apply recoloring only to nodes in C. (It turns out that there is no need to remember

Algorithm 2 $\Delta + 1$ disengagement: Upon arrival of edge (u, v) in step i.

1: **if** $c_{i-1}(u) \neq c_{i-1}(v)$ **then**
2: **return** // $c_i = c_{i-1}$, *no recoloring*
3: **else if** $|\{u, v\} \cap C| = 1$ **then**
4: let $x \in \{u, v\}$ be the node in C; recolor(x)
5: **else if** $|\{u, v\} \cap C| = 2$ **then**
6: let $x \in \{u, v\}$ be the node with the higher degree; recolor(x) // *break ties arbitrarily*
7: **else if** $\{u, v\} \cap C = \emptyset$ **then**
8: $w_i(u) \leftarrow w_{i-1}(u) - \min(w_{i-1}(u), w_{i-1}(v))$ // $w_0(v) = w(v)$ *for all* $v \in V$
9: $w_i(v) \leftarrow w_{i-1}(u) - \min(w_{i-1}(u), w_{i-1}(v))$
10: **if** $w_i(u) = 0$ **then**
11: $C \leftarrow C \cup \{u\}$; $x \leftarrow u$
12: **end if**
13: **if** $w_i(v) = 0$ **then**
14: $C \leftarrow C \cup \{v\}$; $x \leftarrow v$
15: **end if**
16: recolor(x)
17: **end if**

the initial coloring.) For a deterministic algorithm, we recolor using the procedure given in Algorithm 3: change the node's color to the first available color, where a color is said to be available if it is not taken by any neighbor. For a randomized algorithm, we choose uniformly at random among the available colors (Algorithm 4).

To maintain a light vertex cover, we use the classical 2-approximation algorithm of Bar-Yehuda and Even [4] (we can use any other algorithm which processes edges one at a time). Whenever an uncovered edge is considered, the residual weights of its endpoints are reduced by the same amount so that one of them reaches zero. The node with residual weight 0 (possibly both) is then added to C.

▶ **Theorem 8.** *Algorithm 2 with Algorithm 3 is a deterministic algorithm for* $\Delta + 1$ *disengagement with competitive ratio* $O(\Delta)$*. Algorithm 2 with Algorithm 4 is a randomized algorithm for* $\Delta + 1$ *disengagement with expected competitive ratio* $O(\log \Delta)$*.*

We first analyze the general framework of Algorithm 2.

▶ **Lemma 9.** *After every step,* C *is a vertex cover of* G'*. Moreover,* $w(C)$ *is at most twice the weight of any vertex cover of* G'*.*

Proof. We first argue that C is a vertex cover of G'. Let $e_i = (u, v) \in E'$. We consider two cases. If e_i is monochromatic upon arrival, then Algorithm 2 makes sure that if none of its endpoints are in C, then at least one of them enters C (lines 7-17), so e_i is covered by C in this case. If $c_i(u) \neq c_0(u)$, then node u was necessarily recolored at some point in the past. However, since Algorithm 2 recolors only nodes in C, we must have $u \in C$, so e_i is covered by C in this case too. Finally, the approximation bound of $w(C)$ follows from [4]. ◀

Algorithm 3 recolor(u): deterministically recolor node u with the first available color.

1: Let $S = \{c(v) \mid (u, v) \in E\}$
2: Let $c \in \{1, \ldots, \Delta + 1\} \setminus S$ // *choose arbitrarily*
3: $c(u) \leftarrow c$

Algorithm 4 recolor(u): randomly recolor node u with an available color.

1: Let $S = \{c(v) \mid (u, v) \in E\}$
2: Let $c \in_R \{1, \ldots, \Delta + 1\} \setminus S$ // *choose randomly*
3: $c(u) \leftarrow c$

▶ **Lemma 10.** *Every algorithm pays a cost of at least $w(C)/2$.*

Proof. Consider any solution to the given input. The edges in E' are monochromatic by the initial coloring, so every disengagement algorithm has to recolor at least one endpoint of every edge in E' at least once. Therefore, the set of nodes recolored by any solution is a vertex cover of G'. The result follows now from Lemma 9. ◀

We now consider the way recoloring is done. First, the deterministic version.

▶ **Lemma 11.** *Algorithm 2 with Algorithm 3 pays at most $\Delta \cdot w(C)$.*

Proof. By the code, Algorithm 2 recolors only nodes in C. Observe that a node may be recolored only when a new incident edge is introduced. Since the maximum degree of a node is Δ, the result follows. ◀

If we use randomized recoloring, we have the following. Let $H_\Delta = 1 + \frac{1}{2} + \cdots + \frac{1}{\Delta}$.

▶ **Lemma 12.** *The expected cost of Algorithm 2 with Algorithm 4 is at most $H_\Delta \cdot w(C)$*

▶ **Proposition 13.** *Consider any two distinct nonadjacent nodes $u, v \in C$. At any iteration, it holds that*

$$\Pr(c(u) = c(v)) \;\leq\; \max\left\{ \frac{1}{\Delta + 1 - \deg(u)}, \frac{1}{\Delta + 1 - \deg(v)} \right\} .$$

Proof. Let $w \in \{u, v\}$ be the node that changed its color last (since both $u, v \in C$, they both changed colors so w is always defined). It holds that:

$$\Pr(c(u)=c(v)) \;=\; \Pr(w=v)\cdot\Pr(c(u)=c(v) \mid w=v) + \Pr(w=u)\cdot\Pr(c(u)=c(v) \mid w=u) .$$

Let A_v denote the event that $w = v$, and assume that A_v holds. Let t' denote the last iteration in which v changed its color, and let $\deg'(v)$ be its degree at time t'. Then v chose its color uniformly at random from a set of available colors S whose size is at least

$$(\Delta + 1 - \deg'(v)) \;\geq\; (\Delta + 1 - \deg(v)) \;\geq\; \min\{\Delta + 1 - \deg(u), \Delta + 1 - \deg(v)\} .$$

Thus:

$$\begin{aligned}
\Pr(c_u = c_v \mid A_v) &= \sum_{S' \subseteq [\Delta+1]} \sum_{s \in S'} \Pr\left(S = S' \mid A_v\right) \cdot \Pr\left(c_u = c_v = s \mid S = S', A_v\right) \\
&\leq \sum_{S' \subseteq [\Delta+1]} \sum_{s \in S'} \Pr\left(S = S' \mid A_v\right) \cdot \Pr\left(c_u = s \mid S = S', A_v\right) \cdot \underbrace{\Pr\left(c_v = s \mid c_u = s, S = S', A_v\right)}_{=\frac{1}{|S'|}} \\
&= \sum_{S' \subseteq [\Delta+1]} \Pr\left(S = S' \mid A_v\right) \cdot \frac{1}{|S'|} \cdot \underbrace{\left(\sum_{s \in S'} \Pr(c_u = s \mid S = S', A_v) \right)}_{\leq 1} \\
&\leq \sum_{S' \subseteq [\Delta+1]} \Pr\left(S = S' \mid A_v\right) \cdot \max\left\{ \frac{1}{\Delta + 1 - \deg(u)} \;,\; \frac{1}{\Delta + 1 - \deg(v)} \right\} \\
&= \max\left\{ \frac{1}{\Delta + 1 - \deg(u)} \;,\; \frac{1}{\Delta + 1 - \deg(v)} \right\} .
\end{aligned}$$

◀

▶ **Proposition 14.** *Consider any two nonadjacent nodes $v \in C$ and $u \notin C$. At any iteration, it holds that*

$$\Pr(c(u) = c(v)) \leq \frac{1}{\Delta + 1 - \deg(v)} .$$

Proof. The proof is similar to that of Proposition 13. Observe the last iteration in which v had changed its color, and denote its degree at that iteration as $\deg'(v)$. At that time, v chose a random color from a set S of available colors, such that $|S| \geq \Delta+1-\deg'(v) \geq \Delta+1-\deg(v)$. Additionally, since $u \notin C$, it has its initial coloring $c(u) = c_0(u)$. Therefore:

$$\begin{aligned}\Pr(c_u = c_v) &\leq \sum_{S' \subseteq [\Delta+1]} \Pr(S = S') \cdot \Pr(c_v = c_u \mid S = S') \\ &\leq \sum_{S' \subseteq [\Delta+1]} \Pr(S = S') \cdot \frac{1}{|S'|} \\ &\leq \frac{1}{\Delta + 1 - \deg(v)} \sum_{S' \subseteq [\Delta+1]} \Pr(S = S') \;=\; \frac{1}{\Delta + 1 - \deg(v)} \;,\end{aligned}$$

where the second inequality is due to the fact that $\Pr(c_v = c_u \mid S = S')$ equals 0 if $c(u) \notin S'$ and equals $\frac{1}{|S'|}$ otherwise. ◀

Proof of Lemma 12. Fix the input sequence. Let X be a random variable representing the overall cost of running the algorithm, and let X^v be a random variable representing the cost incurred by recoloring a given node $v \in V$. We bound $E[X^v]$.

Since Algorithm 2 recolors only the nodes in C, consider $v \in C$. Let $e^v_1, \ldots, e^v_{\deg(v)}$ be the subsequence of input edges incident on v. Let X^v_j denote the expected cost of recoloring v due to edge e^v_j. We bound $E[X^v_j]$ as follows. Suppose that v is recolored for the first time when e_{j_0} is input (if none exists, we are trivially done). Then for $j = j_0$ we have $E[X^v_{j_0}] = w(v)$. For $j > j_0$, consider the arrival of $e^v_j = (v, u)$.

- If $u \notin C$, then v changes its color w.p $\leq \frac{1}{\Delta+1-j}$ (according to Proposition 14)
- If $u \in C$ and $j < \deg(u)$, then v does not change its color
- If $u \in C$ and $j \geq \deg(u)$, then v changes its color w.p $\leq \frac{1}{\Delta+1-j}$ (according to Proposition 13)

Therefore, the expected cost incurred by v due to e^v_j is at most $\frac{w(v)}{\Delta+1-j}$.

It follows that the expected cost incurred by the recoloring of node $v \in C$ is at most

$$E[X^v] = E\left[\sum_{j=1}^{\deg(v)} X^v_j\right] \;\leq\; w(v) + \sum_{j=j_0+1}^{\deg(v)} E\left[X^v_j\right] \;\leq\; w(v) + \sum_{j=2}^{\deg(v)} \frac{w(v)}{\Delta+1-j} \;\leq\; w(v) \cdot H_{\deg(v)} \;.$$

We can therefore summarize that the expected cost due to all nodes is at most

$$E[X] \;=\; E\left[\sum_{v \in V} X^v\right] \;=\; E\left[\sum_{v \in C} X^v\right] \;=\; \sum_{v \in C} E\left[X^v\right] \;\leq\; \sum_{v \in C} w(v) \cdot H_{\deg(v)} \;\leq\; w(C) \cdot H_\Delta \;. \blacktriangleleft$$

Proof of Theorem 8. Follows from the lower bound on the optimum cost of Lemma 10, and from the upper bounds on the cost of the deterministic algorithm (Lemma 11), and on the expected cost of the randomized algorithm (Lemma 12). ◀

We note that our deterministic algorithm does not require knowledge of Δ, but the randomized one does.

4.2 Lower Bounds for $(\Delta + 1)$-Recoloring

We now state lower bounds on the competitive ratios of deterministic and randomized algorithms for $(\Delta + 1)$-recoloring. We prove the lower bounds in unweighted instances, i.e., the weight of each node is one. We start with a deterministic lower bound.

▶ **Theorem 15.** *The competitive ratio of any deterministic algorithm solving* $(\Delta+1)$*-recoloring is* $\Omega(\Delta)$*, for any initial coloring.*

We need the following lemma.

▶ **Lemma 16.** *The optimal algorithm's cost after the* i*-th step is at most the number of non-isolated nodes in* $G_i = (V, E_i)$*.*

Proof. Since edges are constantly added to the graph G_i, any coloring that is proper for G_i is also proper for G_j, where $j \leq i$. The optimal offline algorithm can color G_i greedily using $\Delta + 1$ colors, and use this coloring until the i-th step. Since every node is recolored at most one time, and the optimal algorithm might only recolor the non-isolated nodes, its cost is at most the number of such nodes. ◀

Proof of Theorem 15. Let a deterministic algorithm be given. We construct an input sequence in phases as follows. First we choose a set of some $\Delta + 2$ nodes, denoted V'. Note that since $|V'| > \Delta + 1$, according to the pigeonhole principle, there must be at least two nodes $u, v \in V'$ with the same color. We then proceed in phases, where each phase is described as follows.

1. Do until $\exists v \in V'$ with $\deg(v) = \Delta$:
   a. Select two nodes $u_1, u_2 \in V'$ s.t. $c(u_1) = c(u_2)$
   b. Add edge (u_1, u_2)
2. Let $V_\Delta = \{v \in V' \mid \deg(v) = \Delta\}$ *(note that* $1 \leq |V_\Delta| \leq 2$*)*
3. Let $V_0 \subseteq V$ be a set of $|V_\Delta|$ isolated nodes
4. $V' \leftarrow V_0 \cup V' \setminus V_\Delta$

Note that the number of phases can be as large as we wish, since it is bounded only by the number of nodes, regardless of k and Δ. Consider the cost of the online algorithm. Every new edge is monochromatic, so it pays a cost of 1 for each new edge. After s phases, at least s nodes have left V', each with a degree of Δ. It follows that the cost of the online algorithm for s phases is at least $s \cdot \Delta/2$ (since every edge could be counted twice).

On the other hand, by Lemma 16, the optimal cost is no more than the number of non-isolated nodes. This number is bounded by the number of nodes that ever were in V', and hence, in s phases, it is at most $2s + \Delta + 2$: at most $2s$ nodes were removed from V', and at most $\Delta + 2$ nodes are in V' after s phases. It follows that for $s \gg \Delta$ phases, the competitive ratio of any deterministic algorithm is $\Omega(\Delta)$. ◀

Intuitively, in the $\Omega(\Delta)$ lower-bound proof, both Online and Offline always have to pay (a cost of at least 1) when the new edge creates a new connected component (of size 2), since it involves two nodes that still have their initial coloring. However, the Offline algorithm does not have to pay when the edge is incident on a node that Offline had already changed. We now present a lower bound for randomized algorithms.

▶ **Theorem 17.** *The competitive ratio of any randomized algorithm for* $(\Delta + 1)$*-coloring is* $\Omega(\log(\Delta))$*, assuming that in the initial coloring there are at least two nodes of each color.*

Proof of Theorem 17. Let $A(I)$ be the cost of algorithm A on an instance (input sequence) I. According to Yao's principle, if there is an instance distribution, for which the expected cost of every deterministic algorithm A is $E[A(I)] \geq L$, then for every randomized algorithm there exists an instance I such that the algorithm's expected cost on I is at least L. We construct a distribution over input sequences as follows. First we pick a random permutation $\sigma : [\Delta + 1] \to [\Delta + 1]$ of the colors. We then introduce the edges in phases as follows. In the

first phase we pick two nodes $v_0, v_1 \in V$ of color $\sigma(1)$ and connect them with an edge. In each subsequent phase i, $i \geq 2$, we pick a node v_i of color $\sigma(i)$ and connect it to all nodes $v_0, \ldots, v_{i-1}$. Clearly, after phase i, the input edges constitute a clique over $v_0, v_1, \ldots, v_i$. After phase Δ we stop (we can repeat the construction with a fresh set of $\Delta + 1$ isolated nodes).

For the cost analysis, let C_i denote the set of colors used by nodes $v_0, \ldots, v_i$ after phase i. Clearly $|C_i| = i + 1$. In particular, for any deterministic algorithm, there exists at least one color $c_i^* \in C_i \setminus \{\sigma(1), \ldots, \sigma(i)\}$. Consider now the node v_{i+1}, added in phase $i + 1$. Its color is $\sigma(i + 1)$, which is uniformly distributed over $[\Delta + 1] \setminus \{\sigma(1), \ldots, \sigma(i)\}$, and hence $\Pr[\sigma(i + 1) = c_i^*] = \frac{1}{\Delta+1-i}$. It follows that the expected number of monochromatic edges (which is the expected cost of the deterministic algorithm) in phase $i + 1$ is at least $\frac{1}{\Delta+1-i}$. Also by construction, the number of monochromatic edges in the first phase is 1. Therefore, the total expected cost across all phases is

$$E[\text{cost}] \;\geq\; 1 + \sum_{i=1}^{\Delta-1} \frac{1}{\Delta + 1 - i} \;=\; H_\Delta \;=\; O(\log \Delta) \;,$$

where H_n denotes the nth harmonic sum. On the other hand, an optimal algorithm would pay a cost of 1 by recoloring only a single node in the first iteration with the single color that is never used, namely color $\sigma(\Delta + 1)$. ◀

5 Fully-Dynamic Disengagement

In previous sections, we considered the semi-dynamic variant of the disengagement problem, in which every new edge is an additional, permanent constraint. In this section,we consider the fully-dynamic variant of the online disengagement problem (abbreviated FD below), where at every iteration i, the edge constraints may also be deleted (in FD, input sequences may be of unbounded length). It turns out that FD is more difficult than semi-dynamic disengagement. In this section we first discuss various plausible models of FD and show their equivalence, and then show that even a very simple dynamic instance forces an $\Omega(n)$ lower bound on the competitiveness of deterministic algorithms. The latter result is obtained by reduction from Metrical Task Systems.

5.1 Dynamic Models

There are several (yet equivalent) ways to define the fully-dynamic Disengagement problem:

The single edge model (SE): At iteration i upon the arrival of edge $e_i = (u, v)$, the algorithm must make sure that $c_i(u) \neq c_i(v)$, i.e, only the coloring of the last edge must be maintained.

Expiration time model (ET): Every edge arrives with a prescribed expiration time - after which it is deleted.

Edge insertions and deletions model (InD): In this model, at every iteration either an existing edge in the graph is deleted, or an non-existing one appears. The algorithm is required at every iteration to maintain a proper coloring of the graph.

We first argue that the models are essentially equivalent.

▶ **Theorem 18.** *A lower bound on the competitive ratio of (randomized) algorithms under one of the three models of dynamic disengagement, holds under the other two models as well.*

Theorem 18 is proven using the following three lemmas.

▶ **Lemma 19.** *If ρ_1 is a lower bound on the deterministic (or randomized) competitive ratio for the "SE" model, then ρ_1 is a lower bound on the deterministic (respectively, randomized) competitive ratio for the "ET" model.*

Proof. It can be easily shown that any input sequence of the "SE" model is also an input sequence of the "ET" model, with the expiration time of every edge set to 1. Since the lower bound for the special case also holds for the general case, the claim holds. ◀

▶ **Lemma 20.** *If ρ_2 is a lower bound on the deterministic (or randomized) competitive ratio for the "ET" model, then ρ_2 is a lower bound on the deterministic (respectively, randomized) competitive ratio for the "InD" model.*

Proof. It can also be demonstrated that any input sequence of the "ET" model can be represented using the "InD" model. Each time an edge is presented (or expires) in the former model, it is inserted (or deleted) in the latter model. ◀

▶ **Lemma 21.** *If ρ_3 is a lower bound on the deterministic (or randomized) competitive ratio for the "InD" model, then ρ_3 is a lower bound of the deterministic (respectively, randomized) competitive ratio for the "SE" model.*

To prove Lemma 21, we first prove the following lemma:

▶ **Lemma 22.** *Let A be a (possibly randomized) disengagement algorithm in the "SE" model, with a bounded competitive ratio. Then for any input sequence I in the "InD" model, there exists an input sequence I_D in the "SE" model such that with probability 1, A, when run on I_D, outputs a proper coloring of G_i, where $G_i = (V, E_i)$ is the graph in the "InD" model, after iteration i.*

Proof. We construct I_D by repeating the edges of E_i cyclically. Specifically, for each i, I_D contains a sequence of *phases*, where in each phase, all edges of E_i are introduced (in any order). The number of phases depends on A: we claim that for any $\varepsilon > 0$ there exists M_ε such that the probability that A outputs a proper coloring of G_i after M_ε phases is least $1 - \varepsilon$. To prove the claim, suppose, for contradiction, that for some $\varepsilon > 0$, the probability of A outputting a proper coloring of G_i after any number of phases is never more $1 - \varepsilon$. Then, with probability of at least ε, A pays at least 1 in each phase (when a monochromatic edge is inserted). The total expected cost of A is therefore unbounded. On the other hand, the optimal cost is at most $|V|$, contradicting the assumption that A has bounded competitive ratio. ◀

Proof of Lemma 21. Assume, for contradiction, that there exists an algorithm A_d for the "SE" model with a competitive ratio strictly smaller than ρ_3. We construct an algorithm A_s for the "InD" model as follows. Given input $I = \{G_1, G_2, \ldots\}$ (in the "InD" model), A_s runs A_d on input I_D (in the "SE" model) as defined in Lemma 22. According to Lemma 22, since A_d has a bounded competitive ratio, then A_d outputs a solution to E_i w.p. 1. Hence, A_s is well defined.

To analyze the cost, consider iteration i of A_s. Let c_i^m denote the output of A_d after E_i was input to it for m-th time, and let c_i denote the output of A_s in the i-th step.

Let OPT_s denote an optimal solution to I and let OPT_d denote an optimal solution to I_d. Then we have

$$\begin{aligned}
\text{cost}(A_s(I)) &= \sum_{i=1} \delta_H(c_i, c_{i-1}) \\
&\le \sum_{i=1} \sum_{m \ge 1}^{M} \delta_H\left(c_i^m, c_i^{m-1}\right) && \text{by construction} \\
&= \text{cost}(A_d(I_d)) \\
&< \rho_3 \cdot \text{cost}(\text{OPT}_d(I_d)) && \text{by assumption on competitiveness of } A_d \\
&\le \rho_3 \cdot \text{cost}(\text{OPT}_s(I)) && \text{because } \text{OPT}_s(I) \text{ is also a solution to } I_d,
\end{aligned}$$

contradicting the assumption that ρ_3 is a lower bound on the competitiveness of deterministic algorithms for the "InD" model. ◀

We note that FD is at least as hard as the semi-dynamic model.

▶ **Corollary 23.** *If ρ is a lower bound on the deterministic (or randomized) competitive ratio for semi-dynamic disengagement, then ρ is a lower bound of the deterministic (respectively, randomized) competitive ratio for fully-dynamic disengagement.*

Proof. The SD disengagement model can be directly represented by the FD "ET" model, with the expiration time of every edge set to ∞. Hence, the lower bound for the SD case also holds for the "ET" model, and according to Theorem 18, such bound holds for any of the other FD models. ◀

5.2 Full-Dynamic Disengagement and Metrical Task Systems

We now give evidence to the hardness of FD disengagement using a reduction from metrical task systems (MTS). Let us first define the terms.

In the MTS problem [7], a metric space M of n points is given. A server is always located at some point in the metric space. Requests arrive in an online fashion, where the i'th request is a cost function $f_i : M \to \mathbb{R}^+$. The algorithm responds to a request f_i by moving the server from its current location $u \in M$ to a new location $v \in M$, paying the distance between the locations in the metric space (or not moving the server, and paying nothing). Then, the algorithm must pay the cost of the current location of the server, which is $f_i(v)$. We show that FD is as hard as a particular type of an MTS.

▶ **Theorem 24.** *If there exists an $O(f(n))$-competitive algorithm for FD disengagement for every graph of n nodes with 2 colors, then there exists an $O(f(n))$-competitive algorithm for metrical task system on an odd cycle on length n.*

In [7], a lower bound of $\Omega(n)$-competitive was presented for every deterministic algorithm for MTS on every set of n points. Hence we have the following.

▶ **Corollary 25.** *Any deterministic algorithm for FD disengagement with n nodes is $\Omega(n)$-competitive.*

A lower bound of $\Omega(\log n / \log\log n)$-competitiveness for every randomized algorithm for MTS on every set of n points was given in [6]. This implies the following corollary.

▶ **Corollary 26.** *Any randomized algorithm for FD disengagement with n nodes is $\Omega\left(\frac{\log n}{\log\log n}\right)$-competitive.*

Proof of Theorem 24. We assume without loss of generality that the functions in the MTS input are in fact Kronecker delta functions: they are supported on only one point, and the cost of that point is 1. We also assume w.l.o.g. that the FD disengagement algorithm is lazy, i.e., it only recolors endpoints of the current request.

We now describe the reduction. Suppose we are given an instance $\mathcal{M}$ of MTS with an odd-cycle metric space C_1 of n points. We construct an instance $\mathcal{D}$ of FD with n points arranged in a cycle, such that the nodes of C_1 correspond to the n edges of C_2 in the natural way. The intended interpretation of nodes in C_1 is colorings in $\mathcal{D}$, where a node x of C_1 means in C_2 that only the edge corresponding to x is monochromatic (and all other edges in C_2 are bichromatic). The initial coloring in $\mathcal{D}$ is such that the only monochromatic edge in $\mathcal{D}$ is the edge corresponding to the initial position of the server in $\mathcal{M}$. Thereafter, whenever a point $x \in C_1$ is requested in $\mathcal{M}$, the corresponding edge in C_2 is requested in $\mathcal{D}$. To see that the intended interpretation is maintained, consider any algorithm ALG_D for FD. Easy induction shows that after every step, there is exactly one monochromatic edge in the odd cycle C_2: The base case holds by the initial coloring; for the induction step, note that if a non-monochromatic edge is requested, no recoloring takes place. Otherwise, the currently monochromatic edge e is requested, and ALG_D must recolor exactly one endpoint of e. If the clockwise (say) endpoint of e is recolored, then e is no longer monochromatic, and the next edge clockwise from e becomes monochromatic (and similarly for counter-clockwise), and the induction is proved.

Given the reduction, algorithm ALG_M for the MTS problem is obtained from a given algorithm ALG_D for FD by taking the MTS instance $\mathcal{M}$, reducing it online to the FD instance $\mathcal{D}$ as desribed above, running ALG_D on $\mathcal{D}$, and interpreting the responses of ALG_D back in $\mathcal{M}$. That is, whenever ALG_D moves the monochromatic edge clockwise, ALG_M moves the server clockwise, and the same for counter-clockwise. Clearly, we have that ALG_D has the same cost on $\mathcal{D}$ as ALG_M has on $\mathcal{M}$.

Finally, denoting by OPT_M and OPT_D the optimal solutions to $\mathcal{M}$ and $\mathcal{D}$ respectively, note that $\mathrm{OPT}_D \leq 2\,\mathrm{OPT}_M$: OPT_D can move the monochromatic edge whenever OPT_D moves the server, and can move the monochromatic edge clockwise and then immediately counter-clockwise to simulate paying a penalty (at a cost of 2). This completes the proof of Theorem 24. ◀

6 Conclusion

In this paper we have introduced the problem of online disengagement and determined its competitive ratio in the case that conflicts are permanent and either that the final graph is bipartite or that the number of colors is larger than the maximum degree. Many problems remain open.

A major problem we leave open is the competitive ratio in the fully-dynamic case, where conflicts are temporary.

Other natural variants that we leave for future research are the following.

- Explore further the *capacitated disengagement* case. Using the coloring formalism, in this case we assume that each color l comes with prescribed maximal capacity W_l such that in any coloring output by the algorithm, the total number (or, more generally, weight) of nodes assigned to color l does not exceed W_l.
- The *list recoloring* variant represents the case where jobs have both affinities and anti-affinities, i.e., each job is given both a subset of machines on which it may run, and disengagement requests express job separation constraints. Formally, this is modeled by requiring the colorings output by the algorithm to be list colorings. The lists may be fixed or change on-line.

- In *multiple disengagement*, disengagement requests are arbitrary sets of jobs. If the requirement is that at least one of the jobs in a conflict set is not collocated with all others, we have a hypergraph recoloring problem.
- Both Kashyop et al. [12] and Solomon and Wein [15] provide dynamic-coloring algorithms in the case of bounded (or constant) arboricity. Their methods might prove useful for the competitive recoloring case as well.

Note that all versions listed above make sense in either the static or the dynamic variants of the disengagement problem.

References

1. Zaid Allybokus, Nancy Perrot, Jérémie Leguay, Lorenzo Maggi, and Eric Gourdin. Virtual function placement for service chaining with partial orders and anti-affinity rules. *Networks*, 71(2):97–106, 2018. doi:10.1002/net.21768.
2. Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM J. Discret. Math.*, 34(3):1791–1812, 2020. doi:10.1137/17M1158513.
3. Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Online balanced repartitioning. In *30th International Symposium on Distributed Computing (DISC)*, 2016. URL: http://eprints.cs.univie.ac.at/5540/.
4. Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981. doi:10.1016/0196-6774(81)90020-1.
5. Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. *Algorithmica*, 81(4):1319–1341, 2019. doi:10.1007/s00453-018-0473-y.
6. Y. Bartal, B. Bollobas, and M. Mendel. A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 396–405, 2001. doi:10.1109/SFCS.2001.959914.
7. Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, October 1992. doi:10.1145/146585.146588.
8. Bartłomiej Bosek, Yann Disser, Andreas Emil Feldmann, Jakub Pawlewicz, and Anna Zych-Pawlewicz. Recoloring Interval Graphs with Limited Recourse Budget. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, volume 162 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2020.17.
9. Magnus M Halldórsson. Parallel and on-line graph coloring. *Journal of Algorithms*, 23(2):265–280, 1997. doi:10.1006/jagm.1996.0836.
10. Magnus M. Halldórsson and Mario Szegedy. Lower bounds for on-line graph coloring. *Theoretical Computer Science*, 130(1):163–174, 1994. doi:10.1016/0304-3975(94)90157-0.
11. Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Tight bounds for online graph partitioning. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2799–2818. SIAM, 2021. doi:10.1137/1.9781611976465.166.
12. Manas Jyoti Kashyop, N. S. Narayanaswamy, Meghana Nasre, and Sai Mohith Potluri. Dynamic coloring for bipartite and general graphs. *CoRR*, abs/1909.07854, 2019. arXiv:1909.07854.
13. L. Lovász, M. Saks, and W.T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1-3):319–325, 1989. doi:10.1016/0012-365X(89)90096-4.

14 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

15 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3), June 2020. doi:10.1145/3392724.

16 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007. doi:10.4086/toc.2007.v003a006.

Smoothed Analysis of the Komlós Conjecture

Nikhil Bansal
University of Michigan, Ann Arbor, MI, USA

Haotian Jiang
University of Washington, Seattle, WA, USA

Raghu Meka
University of California, Los Angeles, CA, USA

Sahil Singla
Georgia Institute of Technology, Atlanta, GA, USA

Makrand Sinha
Simons Institute and University of California, Berkeley, CA, USA

Abstract

The well-known Komlós conjecture states that given n vectors in $\mathbb{R}^d$ with Euclidean norm at most one, there always exists a ± 1 coloring such that the ℓ_∞ norm of the signed-sum vector is a constant independent of n and d. We prove this conjecture in a smoothed analysis setting where the vectors are perturbed by adding a small Gaussian noise and when the number of vectors $n = \omega(d \log d)$. The dependence of n on d is the best possible even in a completely random setting.

Our proof relies on a weighted second moment method, where instead of considering uniformly randomly colorings we apply the second moment method on an implicit distribution on colorings obtained by applying the Gram-Schmidt walk algorithm to a suitable set of vectors. The main technical idea is to use various properties of these colorings, including subgaussianity, to control the second moment.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Komlós conjecture, smoothed analysis, weighted second moment method, subgaussian coloring

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.14

Category Track A: Algorithms, Complexity and Games

Funding *Nikhil Bansal*: Supported in part by the NWO VICI grant 639.023.812.
Haotian Jiang: Supported by NSF awards CCF-1749609, DMS-1839116, DMS-2023166, CCF-2105772.
Makrand Sinha: Supported by a Simons-Berkeley postdoctoral fellowship.

1 Introduction

A central question in discrepancy theory is the following Komlós problem: given vectors $v_1, \ldots, v_n \in \mathbb{R}^d$ with Euclidean length at most 1, i.e., $\|v_i\|_2 \leq 1$ for all $i \in [n]$, find signs $x_i \in \{-1, 1\}$ for $i \in [n]$ to minimize the discrepancy $\|\sum_{i=1}^n x_i v_i\|_\infty$. The long-standing Komlós conjecture says that the discrepancy of any collection of such vectors is $O(1)$, independent of n and d. An important special case (up to scaling by $t^{1/2}$) is the Beck-Fiala problem, where the vectors $v_1, \ldots, v_n \in \{0, 1\}^d$ and each v_i has at most t ones, so $\|v_i\|_2 \leq t^{1/2}$. Here, the Komlós conjecture reduces to the Beck-Fiala conjecture [8], which says that the discrepancy is $O(t^{1/2})$. The question of either proving or disproving these conjectures has received a lot of attention, and after a long line of work, the current best bounds for the Komlos and the Beck-Fiala problem are $O((\log n)^{1/2})$ and $O((t \log n)^{1/2})$ respectively, due to Banaszczyk [4].

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 14; pp. 14:1–14:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Motivated by the lack of progress for general worst-case instances, there has been a lot of recent work on these problems for random instances, with several interesting results and techniques, see e.g., [9, 7, 13, 10, 16, 18, 3, 14]. In this work, we consider the Komlós problem in the more general setting of smoothed analysis, where the input is generated by taking an arbitrary worst case Komlós instance and perturbing it randomly. The smoothed analysis model was first introduced by Spielman and Teng [17], and it interpolates nicely between worst case and average case analysis, and has been used extensively since then to study various problems. Recently smoothed analysis models have also been considered in discrepancy theory in a few other works [6, 11] – however the setting and focus of these results is quite different, and in particular they are not directly related to the Komlós or Beck-Fiala conjectures.

Random instances. To put our results in the proper context, we first describe the results on random instances. In general, these results depend on the different regimes of the parameters d, n and t, and we focus here on the more interesting case of $n \gg d$.

A natural model for random Beck-Fiala instances is where each entry is 1 with probability $p = t/d$, so that each column has t ones in expectation. In a surprising result, Hoberg and Rothvoss [13] showed that $\mathrm{disc}(A) \leq 1$ w.h.p.[1] if $n = \Omega(d^2 \log d)$. Independently, Franks and Saks [10] showed that $\mathrm{disc}(A) \leq 2$ w.h.p. if $n = \Omega(d^3 \log^2 d)$, for a more general class of instances. Both these results use interesting Fourier analysis based techniques.

It is not hard to see[2] that $n = \Omega(d \log d)$ is necessary for $O(1)$ discrepancy (provided p is not too small). An important step towards obtaining this optimal dependence was made by Potukuchi [15], who showed that $\mathrm{disc}(A) \leq 1$ if $n = \Omega(d \log d)$ for the dense case of $p = 1/2$, using the second moment method. However, the sparse setting with $p \ll 1$ turns out to be more subtle, and was only recently resolved by Altschuler and Weed [3] using a more sophisticated conditional second moment method together with Stein's method of exchangeable pairs. They show that $\mathrm{disc}(A) \leq 1$ w.h.p. for $n \geq \Omega(d \log d)$, for every p.

The case of Gaussian matrices with i.i.d. $\mathcal{N}(0, 1)$ entries has also been considered, where Turner, Meka and Rigollet [18] give almost tight bounds for the entire regime, and in particular show that for $n = \Omega(d \log d)$ a discrepancy bound of $1/\mathrm{poly}(d)$ holds.

The smoothed Komlós model. We now define our model formally. The input matrix is of the form $A = M + R$, where $M \in \mathbb{R}^{d \times n}$ is some worst-case matrix with columns of ℓ_2-norm at most 1 and $R \in \mathbb{R}^{d \times n}$ is a random matrix with i.i.d. Gaussian entries distributed as $\mathcal{N}(0, \sigma^2/d)$, where $\sigma \leq 1$. The σ^2/d variance ensures that each column of R has ℓ_2-norm roughly σ (and hence much less than that of M). Our goal is to understand the discrepancy of A. We will only be interested in showing the existence of a low discrepancy coloring for A, and not in algorithmically finding it (this seems far beyond the current techniques).

1.1 Results and Techniques

Our main result is the following.

▶ **Theorem 1** (Smoothed Komlós). *Let $\sigma > 0$ and $n = \frac{\omega(d \log d)}{\sigma^{4/3}}$. Then with probability $1 - o_d(1)$, the discrepancy of $M + R$ is at most $1/\mathrm{poly}(d)$, where $M \in \mathbb{R}^{d \times n}$ is an arbitrary Komlós instance and $R \in \mathbb{R}^{d \times n}$ has i.i.d. $\mathcal{N}(0, \sigma^2/d)$ entries.*

[1] This is much better than the $O(t^{1/2})$ bound in the Beck-Fiala conjecture.

[2] If we fix any coloring x and consider a random instance, a fixed row has discrepancy $O(1)$ with probability $\approx (pn)^{-1/2}$, so the probability that each row has discrepancy $O(1)$ is $(pn)^{-\Omega(d)}$. As there are (only) 2^n possible colorings, a first moment argument already requires that $2^n (pn)^{-d} = \Omega(1)$.

An interpretation of Theorem 1 is that any counter-example to the Komlós conjecture (if it exists) will be rigid, or have $n \approx d$. Also, notice that dependence of n on d in Theorem 1 is essentially the best possible, as already evident in the very special case of $M = \mathbf{0}$ and $\sigma = 1$, i.e., a random matrix with i.i.d. $\mathcal{N}(0,1)$ entries, where $n = \Omega(d \log d)$ is necessary to achieve $1/\mathrm{poly}(d)$ discrepancy, as discussed earlier.

▶ Remark 2. Our proof techniques also give a high probability bound when $n = \Omega_\sigma(d^{1+\epsilon})$ for any constant $\epsilon > 0$. However, we do not explore this direction here. It would be interesting to know if the result also holds with high probability when $n = \omega(d \log d)$, as in the (fully) random setting.

▶ Remark 3. The dependence on the noise parameter σ in $n = \Omega_d(1/\sigma)$ in Theorem 1 is necessary, otherwise this would imply an $O(1)$ bound for the worst case Komlós problem. In particular, each row of the random part R must at least have enough ℓ_1 norm to offset the discrepancy from the worst case part M (which can be $O((\log n)^{1/2})$ given the currently known results). As each entry of R has magnitude about $\sigma d^{-1/2}$, we thus require $n = \Omega_d(1/\sigma)$ for each row of $M + R$ to have discrepancy $O(1)$.

The proof of Theorem 1 is based on the classical second moment method, however, it requires several additional ideas beyond those used for random instances, to handle the effect of the worst case part and its interplay with the random part. We describe these briefly next, and discuss them in more detail in Section 1.2.

- **Weighted second moment method.** Instead of applying the second moment method to the uniform distribution on the 2^n colorings, we consider a distribution on low-discrepancy colorings for M. This is necessary as for a random coloring $x \in \{-1,1\}^n$, a typical entry of Mx will scale as $\sqrt{n/d}$, which is very unlikely to be *cancelled* by the discrepancy of the random part Rx, which typically scales as $\sigma\sqrt{n/d}$ (note that we want to show the existence of some x such that $(Rx)_i \approx -(Mx)_i$ for each coordinate $i \in [d]$).
- **Subgaussianity of colorings.** To ensure that $\|Mx\|_\infty$ is typically small, we consider the (implicit) distribution on colorings produced by the Gram-Schmidt (GS) algorithm [5] applied to M, which ensures that Mx is a 1-subgaussian vector [12] (details in Section 1.2). However, apriori the GS algorithm only guarantees that Mx is subgaussian, and says nothing about the distribution on the colorings x. For instance, it could be that any two colorings in the support have the first $9n/10$ coordinates identical, and thus look very non-random. This makes the second moment bounds much worse and harder to control. To handle this, we use a simple but useful trick to ensure that the distribution on the colorings x produced by the GS algorithm is also $O(1)$-subgaussian. Roughly, this allows us to pretend that colorings x in the GS distribution behave randomly.
- **Exploiting subgaussianity to get cancellations across rows.** Most importantly, due to the worst case part M, doing a row by row analysis as is typically done in second moment computations for random instances, only works when $n = \Omega(d^2/\sigma^2)$ (details in Section 1.2). Roughly, the problem is that considering each coordinate of Mx separately completely ignores the global properties across the different coordinates that subgaussianity of Mx implies.
  To get the optimal dependence of n on d, a key conceptual idea is to analyze *all* the rows together and use the subgaussianity of Mx and x carefully to get various cancellations across the different rows in the second moment computation. Exploiting subgaussianity also leads to various technical difficulties, as subgaussian vectors can differ from fully random Gaussian vectors in various non-trivial ways.

Notation. Throughout this paper, log denotes the natural logarithm. We use the asymptotic notation $\omega(\cdot)$ or $o(\cdot)$ where the growth is always with respect to d – sometimes to emphasize this dependence we will also write $\omega_d(\cdot)$ or $o_d(\cdot)$. We write $\mathbb{E}_{x\sim\mathcal{G}}[f(x)]$ to denote the expectation of a function f where x is sampled from the distribution $\mathcal{G}$ and we abbreviate this to $\mathbb{E}[f(x)]$ when the distribution is clear from the context. For reals $a, b \in \mathbb{R}$, the notation $[a \pm b]$ is used as a shorthand to denote the interval $[a-b, a+b]$. For a set $S \in \mathbb{R}^d$, we write $\delta S = \{\delta x \mid x \in S\}$ to denote the δ scaling of S.

1.2 Overview and Preliminaries

We now give a more detailed overview of the proof and the ideas. We also briefly describe the second moment method and some concepts we need such as subgaussianity and properties of the Gram-Schmidt algorithm.

Second moment method. The second moment method (e.g. [2]) is based on the following Paley-Zygmund inequality. For any non-negative random variable Z, we have that

$$\mathbb{P}[Z > 0] \geq \frac{(\mathbb{E}[Z])^2}{\mathbb{E}[Z^2]}.$$

So, if $\mathbb{E}[Z^2] = (1+o(1))(\mathbb{E}[Z])^2$, then this implies that $\mathbb{P}[Z > 0] \geq 1 - o(1)$.

For constraint satisfaction problems, a standard way to use this to show that most random instances are feasible is by defining $S = S(R)$ as the number of solutions to an instance R, and showing that

$$\mathbb{E}_R[S^2] = (1+o(1))(\mathbb{E}_R[S])^2, \tag{1}$$

which gives that $\mathbb{P}_R[S(R) > 0] = \mathbb{P}_R[S(R) \geq 1] \geq 1 - o(1)$.

Let us consider (1) more closely and define $S(R, x) = 1$ if x is a valid solution for instance R, and 0 otherwise. Then $S(R) = \sum_x S(R, x)$ and (1) can be written as

$$\mathbb{E}_R\left[\mathbb{P}_{x,y\sim\mathcal{U}}\left[\, S(R,x) = 1\,,\; S(R,y) = 1\,\right]\right] = (1+o(1))\left(\mathbb{E}_R\left[\mathbb{P}_{x\sim\mathcal{U}}\left[S(R,x) = 1\right]\right]\right)^2, \tag{2}$$

where $\mathcal{U}$ is the uniform distribution over all inputs.

Second moment method for smoothed Komlós. Let Δ denote the desired discrepancy bound. In our setting, denote by $S(R, x) = 1$ that $x \in \{\pm 1\}^n$ is a feasible coloring for the smoothed Komlós instance $M + R$, that is, if $\|(M+R)x\|_\infty \leq \Delta$. Roughly, this condition means that $Rx = -Mx$ and hence the discrepancy of the random part R cancels that of the worst case part M.

However, if x is chosen uniformly from $\{\pm 1\}^n$, it is not hard to see that this cannot work. The entries $(Mx)_i$ will be distributed roughly as $\mathcal{N}(0, m_i^2)$ where $m_i = (\sum_j M_{ij}^2)^{1/2}$ is the ℓ_2-norm of row i of M, and in general will be much larger (around $1/\sigma \gg 1$ times) than the entries $(Rx)_i$.

Weighted second moment. To allow a reasonable probability of Rx cancelling Mx, a natural idea is to consider a distribution that is mostly supported over colorings x with low discrepancy on M. So, we will show (2) where x, y are sampled from some another suitable distribution $\mathcal{G}$ instead of the uniform distribution $\mathcal{U}$. Similar ideas have also been used in other contexts such as [1]. Notice that this does not affect Rx, as for any fixed x, the contribution of the random part $(Rx)_i$ is still distributed as $N(0, n\sigma^2/d)$ (over the randomness of R).

A natural candidate is the distribution on colorings produced by the Gram-Schmidt (GS) walk algorithm [5]. In particular, we use the following result.

▶ **Theorem 4** ([12]). *Given vectors $v_1, \ldots, v_n \in \mathbb{R}^m$ with $\|v_j\|_2 \le 1$, the Gram-Schmidt walk algorithm outputs a random coloring $x \in \{-1,1\}^n$ such that $\sum_{j=1}^n x_j v_j$ is 1-subgaussian.*

Recall that a random vector $Y \in \mathbb{R}^m$ is α-subgaussian if for all test vectors $\theta \in \mathbb{R}^m$,

$$\mathbb{E}\big[\exp(\langle \theta, Y\rangle)\big] \le \exp\left(\frac{\alpha^2\|\theta\|_2^2}{2}\right).$$

Roughly, this means that Y looks like a Gaussian with variance at most α^2 in every direction.

Let $\mathcal{G}$ denote the (implicit) distribution over the colorings output by the GS walk algorithm. For a coloring x, let us denote $P_x := \mathbb{P}_R[S(R,x) = 1]$ and for two colorings x and y, let

$$P_{x,y} := \mathbb{P}_R\big[S(R,x) = 1\,,\, S(R,y) = 1\big].$$

Then changing the order of expectation in (2) and substituting, our goal is to show that

$$\mathbb{E}_{x,y\sim\mathcal{G}}[P_{xy}] = (1 + o(1)) \cdot \mathbb{E}_{x,y\sim\mathcal{G}}[P_x P_y]. \tag{3}$$

However, the set of low discrepancy colorings for M and the distribution $\mathcal{G}$ can be quite complicated and hard to work with. Later, we will ensure that $\mathcal{G}$ is also $O(1)$-subgaussian, which will suffice for our purposes. Let us first consider (3) more closely.

The key computation. As R has i.i.d. Gaussian entries, the quantities P_x and P_{xy} can be written in a very clean way. In particular, as $(Rx)_i \sim \mathcal{N}(0, \sigma^2 n/d)$ for any coloring x, and the $(Rx)_i$ are independent for $i \in [d]$, we can write

$$P_x \;=\; \mathbb{P}_R\left[\bigcap_{i=1}^d ((Rx)_i \in (Mx)_i \pm \Delta)\right] \;=\; \prod_{i=1}^d \mathbb{P}_R\big[g_i \in (Mx)_i \pm \Delta\big],$$

where $g_i \sim N(0, \sigma^2 n/d)$ and g_i's are independent.

Similarly, for any fixed colorings x and y, writing $g_i = (Rx)_i$ and $g_i' = (Ry)_i$ we have

$$P_{xy} = \prod_{i=1}^d \mathbb{P}_R\big[g_i \in (Mx)_i \pm \Delta\,,\, g_i' \in (My)_i \pm \Delta\big],$$

where g_i and g_i' are correlated with $\mathbb{E}[g_i g_i'] = \langle x, y\rangle \cdot \sigma^2/d$.

A standard computation of 2-dimensional gaussian probabilities over rectangles (and ignoring some less crucial terms for the discussion here) gives

$$\frac{\mathbb{P}\big[g_i \in (Mx)_i \pm \Delta\,,\, g_i' \in (My)_i \pm \Delta\big]}{\mathbb{P}\big[g_i \in (Mx)_i \pm \Delta\big] \cdot \mathbb{P}\big[g_i' \in (My)_i \pm \Delta\big]} \approx \exp\left(\frac{d\langle x, y\rangle (Mx)_i (My)_i}{\sigma^2 n^2}\right). \tag{4}$$

So to prove (3), we could try to show that for each $i \in [d]$,

$$\mathbb{E}_{x,y\sim\mathcal{G}}\left[\frac{d\langle x, y\rangle (Mx)_i (My)_i}{\sigma^2 n^2}\right] = o\left(\frac{1}{d}\right). \tag{5}$$

Indeed, as $|\langle x, y\rangle| \le n$ and $(Mx)_i, (My)_i$ are typically $O(1)$ (as Mx and My are subgaussian), setting $n = \omega(d^2/\sigma^2)$ would suffice to complete the second moment proof. However, this does not give us the optimal $d \log d$ dependence.

Next, we sketch the two ideas to obtain the optimal dependence.

Subgaussianity of the distribution $\mathcal{G}$. If x and y were random colorings, we would typically expect that $|\langle x, y\rangle| \approx \sqrt{n}$ instead of n above. To achieve this, we apply the GS walk algorithm to the $(d+n) \times n$ matrix with M in top d rows and I_n in the bottom n rows. (Note that each column still has $O(1)$ length.) This ensures that the resulting distribution $\mathcal{G}$ on the colorings x is $O(1)$-subgaussian, while ensuring that Mx is also $O(1)$-subgaussian.

Handling the rows together. Next, to exploit the subgaussianity of Mx and My, we look at all the rows together in (5) and consider

$$\sum_i \mathbb{E}_{x,y\sim\mathcal{G}}\left[\frac{d\langle x,y\rangle (Mx)_i (My)_i}{\sigma^2 n^2}\right] = \mathbb{E}_{x,y\sim\mathcal{G}}\left[\frac{d\langle x,y\rangle\langle Mx, My\rangle}{\sigma^2 n^2}\right]. \tag{6}$$

By the subgaussianity of the colorings x, y and discrepancy vectors Mx, My, we expect that $\mathbb{E}_{x,y\sim\mathcal{G}}|\langle x, y\rangle| \approx \sqrt{n}$ and $\mathbb{E}_{x,y}|\langle Mx, My\rangle| \approx \sqrt{d}$. Roughly speaking, this implies that the right side of (6) is typically $d^{3/2}/(\sigma^2 n^{3/2})$, and hence $n \gg d/\sigma^{4/3}$ suffices.

The formal argument needs some more care as $\langle x, y\rangle$ and $\langle Mx, My\rangle$ are correlated, and as we need to bound the exponential moment of $d\langle x, y\rangle\langle Mx, My\rangle/(\sigma^2 n^2)$ in (4), instead of the expectation, which gives the additional (necessary) logarithmic factor of $\log d$.

2 Proof of the Smoothed Komlós Conjecture

We use a weighted version of the second moment method as mentioned in the proof overview. Let $\mathcal{G}$ be a distribution over coloring that will be specified later. We define the following random variable S which depends only on the randomness of R,

$$S = S(R) := \mathbb{E}_{x\sim\mathcal{G}}[\mathbf{1}\{\|(M+R)x\|_\infty \le \Delta\}],$$

for some parameter $\Delta = 1/\mathrm{poly}(d)$ to be chosen later. The purpose of this variable is that the event $\{S > 0\}$ implies there exists a coloring $x \in \mathsf{supp}(\mathcal{G})$ with discrepancy at most Δ. Our goal is to show that $\mathbb{P}(S > 0) = 1 - o(1)$. As explained in the proof overview, this would follow from the Paley-Zygmund inequality if we can establish that the first moment $\mathbb{E}_R[S]$ is always positive, and the second moment satisfies $\mathbb{E}_R[S^2] = (1 + o(1)) \cdot (\mathbb{E}_R[S])^2$. We next compute the moments.

First moment computation. We can compute

$$\mathbb{E}_R[S] \;=\; \mathbb{E}_{x\sim\mathcal{G}}\mathbb{E}_R[\mathbf{1}\{\|(M+R)x\|_\infty \le \Delta\}] \;>\; 0,$$

where the strict inequality follows because fixing any outcome $x \sim \mathcal{G}$, the event $\{\|(M + R)x\|_\infty \le \Delta\}$ happens with positive probability (recall that R is a Gaussian random matrix with each entry $\mathcal{N}(0, \sigma^2/d)$).

Second moment computation. For any $i \in [d]$, denote by m_i and r_i the i^{th} row of the matrices M and R respectively. The second moment is given by

$$\begin{aligned}
\mathbb{E}_R[S^2] &= \mathbb{E}_R\left[\mathbb{E}_x[\mathbf{1}\{\|(M+R)x\|_\infty \le \Delta\}] \cdot \mathbb{E}_y[\mathbf{1}\{\|(M+R)y\|_\infty \le \Delta\}]\right] \\
&= \mathbb{E}_R\mathbb{E}_{x,y}\left[\mathbf{1}\left\{\|(M+R)x\|_\infty \le \Delta, \|(M+R)y\|_\infty \le \Delta\right\}\right] \\
&= \mathbb{E}_{x,y}\left[\mathbb{P}_R\left(\|(M+R)x\|_\infty \le \Delta, \|(M+R)y\|_\infty \le \Delta\right)\right] = \mathbb{E}_{xy}[P_{xy}],
\end{aligned}$$

where we define

$$P_{x,y} := \mathbb{P}_R(\|(M+R)x\|_\infty \le \Delta, \|(M+R)y\|_\infty \le \Delta).$$

Similarly, denoting

$$P_x := \mathbb{P}_R(\|(M+R)x\|_\infty \le \Delta),$$

we also have

$$\begin{aligned}(\mathbb{E}_R[S])^2 &= (\mathbb{E}_x \mathbb{P}_R(\|(M+R)x\|_\infty \le \Delta)) \cdot (\mathbb{E}_y \mathbb{P}_R(\|(M+R)y\|_\infty \le \Delta)) \\ &= \mathbb{E}_{x,y}\left[\mathbb{P}_R(\|(M+R)x\|_\infty \le \Delta) \cdot \mathbb{P}_R(\|(M+R)y\|_\infty \le \Delta)\right] \\ &= \mathbb{E}_{xy}[P_x \cdot P_y].\end{aligned}$$

To compare the quantities $\mathbb{E}_R[S^2]$ and $(\mathbb{E}_R[S])^2$, we first consider a distribution over colorings. A natural distribution to consider is the distribution on colorings derived from the Gram-Schmidt walk which ensures that the discrepancy vector Mx is 1-subgaussian if x is sampled from this distribution. However, we shall also need that the colorings x themselves have a subgaussian tail as well as some additional nice properties that will be useful to compute the second moment. In particular, we prove the following lemma in Section 2.1.

▶ **Lemma 5** (Truncated Gram-Schmidt Distribution). *Let $M \in \mathbb{R}^{d\times n}$ be a worst-case Komlós instance. Then, for any constant $C' > 1$ there exists a distribution $\mathcal{G}$ over colorings $x \in \{\pm 1\}^n$ satisfying the following properties:*

- *Almost Constant Euclidean Norm for the discrepancy vectors: for every $x \in \mathsf{supp}(\mathcal{G})$, we have $\|Mx\|_2 \in [r \pm \Delta]$ where $r = O(d^{1/2})$ and $\Delta = d^{-C'}$.*
- *Almost subgaussian tails for the colorings and discrepancy vectors: there exists a constant C depending on C', such that for every $u \in \mathbb{S}^{n-1}$,*

$$\mathbb{P}_{x\sim\mathcal{G}}\left[|\langle x, u\rangle| \ge t\right] \le 2d^C \cdot e^{-t^2/8} \text{ and } \mathbb{P}_{x\sim\mathcal{G}}\left[|\langle Mx, u\rangle| \ge t\right] \le 2d^C \cdot e^{-t^2/8}.$$

Since the colorings sampled from the above distribution are subgaussian, $|\langle x, y\rangle| \le n/2$ holds with high probability. To compute the second moment to a good precision, we need a careful comparison of the ratio $P_{xy}/(P_x \cdot P_y)$ for any two colorings x and y where this event occurs. We show the following bound in this case (proof in Section 2.2).

▷ Claim 6 (Strong bound). For any two colorings $x, y \in \mathsf{supp}(\mathcal{G})$, denote $\epsilon = \epsilon(x,y) = \langle x, y\rangle/n$. If $|\epsilon| \le 1/2$, then we have

$$P_{x,y} \le P_x P_y \cdot \beta(x,y) \text{ where } \beta(x,y) = \exp\left(\delta_1 + d\epsilon^2 + d\delta^2\epsilon^2 + \delta^2\epsilon \cdot \langle Mx, My\rangle\right),$$

where the scaling factor $\delta := \frac{\sqrt{d}}{\sigma\sqrt{n}}$ and the error parameter $\delta_1 \le 1/\mathrm{poly}(d)$.

When the low probability event $|\epsilon| \ge 1/2$ occurs, we use the weak bound $P_{xy} \le \min\{P_x, P_y\}$.

As x is sampled from the truncated Gram-Schmidt distribution, the probabilities P_x turn out to be almost constant for all colorings $x \in \mathsf{supp}(\mathcal{G})$ as the following claim shows.

▷ Claim 7. For any coloring $x \in \mathsf{supp}(\mathcal{G})$,

$$P_x = \exp(\delta_x) \cdot p \text{ where } p := \left(\frac{\delta\Delta}{\sqrt{2\pi}}\right)^d \exp\left(-\frac{\delta^2 r^2}{2}\right), \tag{7}$$

with the scaling factor $\delta := \frac{\sqrt{d}}{\sigma\sqrt{n}}$ and the error parameter δ_x satisfying $|\delta_x| \le \delta_1 \le 1/\mathrm{poly}(d)$.

The proof of this claim is in Section 2.2.

We now focus on the case when $|\epsilon| \leq 1/2$. When we take $x, y \sim \mathcal{G}$, as P_x and P_y are essentially constant, by Claim 6, applying the second moment method reduces to bounding $\beta(x, y)$ as defined in Claim 6. To do this, we will use the properties, as described in Lemma 5, of the underlying random variables x and Mx. The following technical lemma gives a bound on the exponential moment for such random variables.

▶ **Lemma 8.** *Let X be a non-negative random variable X that satisfies*

$$\mathbb{P}(X \geq t) \leq d^{C_1} \cdot e^{-t^2/8} \text{ for any } t > 0,$$

for some fixed constant $C_1 > 0$. Then for any $\lambda = c_2\sqrt{\log d}$ with $c_2 \geq \sqrt{32C_1}$,

$$\mathbb{E}[\exp(X^2/\lambda^2)] \leq 1 + 32C_1/c_2^2 + o_d(1).$$

We shall prove this lemma in Section 2.1.

We can now complete the proof of Theorem 1 by comparing $\mathbb{E}_R[S^2]$ and $(\mathbb{E}_R[S])^2$. We show that

▶ **Lemma 9.** *For $n = \omega(d \log d)\sigma^{-4/3}$, we have*

$$(\mathbb{E}_R[S])^2 = p^2(1 - o_d(1)) \text{ and } \mathbb{E}_R[S^2] = p^2(1 + o_d(1)).$$

The above implies that $\mathbb{E}_R[S^2] = (1 + o(1))(\mathbb{E}_R[S])^2$, and thus the Paley-Zygmund inequality implies Theorem 1 as discussed in the proof overview.

Proof of Lemma 9. For the first moment, Claim 7 implies that

$$(\mathbb{E}_R[S])^2 = \mathbb{E}_{x,y\sim\mathcal{G}}[P_x P_y] = p^2\mathbb{E}[\exp(\delta_x + \delta_y)] \geq p^2 \exp(-2\delta_1).$$

Since $0 < \delta_1 \leq 1/\text{poly}(d)$, the bound follows.

To compute the second moment, $\mathbb{E}_R[S^2] = \mathbb{E}_{x,y\sim\mathcal{G}}[P_{xy}]$, we define $\mathcal{E}$ to be the event that the colorings $x, y \sim \mathcal{G}$ satisfy $|\langle x, y\rangle| > n/2$ and compute the contribution to the expectation under $\mathcal{E}$ and its complement separately. In particular, using Claim 7 and Claim 6, we have

$$\mathbb{E}_R[S^2] = \mathbb{E}_{x,y\sim\mathcal{G}}[P_{x,y}] \leq \mathbb{P}_{x,y\sim\mathcal{G}}[\mathcal{E}] \cdot p + \mathbb{E}_{x,y\sim\mathcal{G}}\left[P_x P_y \beta(x, y) \cdot \mathbf{1}[\overline{\mathcal{E}}]\right] \tag{8}$$

For the first term in (8), since $n \geq d$, Lemma 5 implies that

$$\mathbb{P}_{x,y\sim\mathcal{G}}[\mathcal{E}] \leq \text{poly}(d) \cdot e^{-n/4} \leq e^{-n/8}.$$

Thus, using the exact bound for p from Claim 7 and that $\delta\Delta \leq \text{poly}(\sigma/d)$, the first term

$$\begin{aligned}
\mathbb{P}_{x,y\sim\mathcal{G}}[\mathcal{E}] \cdot p = p^2 \cdot \mathbb{P}[\mathcal{E}] \cdot p^{-1} &\leq p^2 \cdot e^{-n/8} \cdot \left(\frac{\sqrt{2\pi}}{\delta\Delta}\right)^d \exp\left(\frac{\delta^2 r}{2}\right) \\
&\leq p^2 \cdot e^{-n/8} \cdot \exp\left(O\left(d\log(dn/\sigma) + d^2/(\sigma^2 n)\right)\right) \\
&= p^2 \cdot o_d(1),
\end{aligned} \tag{9}$$

when $n = \omega(d \log d)\sigma^{-4/3}$. In particular, as $\sigma \leq 1$, we have $n/8 \gg d\log(dn/\sigma) + d^2/(\sigma^2 n)$.

For the second term in (8), using Claim 7, we have that $P_x = p \cdot \exp(\delta_x)$ where $|\delta_x| \leq |\delta_1| \leq 1/\text{poly}(d)$. Thus,

$$\begin{aligned}
\mathbb{E}_{x,y\sim\mathcal{G}}\left[P_x P_y \beta(x, y) \cdot \mathbf{1}[\overline{\mathcal{E}}]\right] &\leq p^2 \cdot \exp(2|\delta_1|) \cdot \mathbb{E}\left[\beta(x, y) \cdot \mathbf{1}[\overline{\mathcal{E}}]\right] \\
&\leq \exp(2|\delta_1|) \cdot \mathbb{E}\left[\beta(x, y)\right],
\end{aligned} \tag{10}$$

since $\beta(x,y)$ is a non-negative random variable. Recall from Claim 6 that $\beta(x,y) \le \exp(\delta_1) \cdot \exp(Z)$ where $\delta_1 \le 1/\mathrm{poly}(d)$ and

$$Z = d\epsilon^2 + 2\delta^2\epsilon^2 r^2 + 2\delta^2|\epsilon\langle Mx, My\rangle|.$$

Renormalizing $\overline{\epsilon} = \langle x, n^{-1/2}y\rangle$ and $\overline{\theta} = \langle Mx, r^{-1}My\rangle$ and using that $\delta = \frac{\sqrt{d}}{\sigma\sqrt{n}}$ and $r \le \sqrt{d}$, we have

$$Z \;\le\; \left(\frac{d}{n} + \frac{2d^2}{\sigma^2 n^2}\right)\cdot\overline{\epsilon}^2 + \frac{2d\sqrt{d}}{\sigma^2 n\sqrt{n}}\cdot|\overline{\epsilon}\cdot\overline{\theta}| \;\le\; (|\overline{\epsilon}| + |\overline{\theta}|)^2/\lambda_{\min}^2,$$

where we denote

$$\lambda_{\min} = \frac{1}{3}\sqrt{\min\left\{\frac{n}{d}, \frac{\sigma^2 n^2}{2d^2}, \frac{\sigma^2 n^{1.5}}{2d^{1.5}}\right\}}.$$

Note that $\lambda_{\min} = \omega_d(1)\cdot\sqrt{\log d}$ when $n = \omega(d\log d)\sigma^{-4/3}$.

We now bound the tails of $\overline{\epsilon}$ and $\overline{\theta}$, which will allow us to bound $\mathbb{E}[\exp(Z)]$. Conditioned on any outcome of $y \sim \mathcal{G}$, and as $\|y\|n^{-1/2} = 1$, the second property in Lemma 5 gives that $\mathbb{P}_{x\sim\mathcal{G}}[|\langle x, n^{-1/2}y\rangle| \ge t] \le 2d^C\exp(-t^2/8)$. Averaging over y thus gives that

$$\mathbb{P}\left[|\overline{\epsilon}| \ge t\right] \le 2d^C\cdot e^{-t^2/8}.$$

Similarly, as $\|My\|r^{-1} \le 1$ for any y in the support of $\mathcal{G}$, we have that $\mathbb{P}_{x\sim\mathcal{G}}[|\langle Mx, r^{-1}My\rangle| \ge t] \le 2d^C\exp(-t^2/8)$, and averaging over y gives that

$$\mathbb{P}\left[|\overline{\theta}| \ge t\right] \le 2d^C\cdot e^{-t^2/8}.$$

By a union bound, it follows that the random variable $X := |\overline{\epsilon}| + |\overline{\theta}|$ satisfies the tail condition of Lemma 8 with constant $C_1 = 2C$. So when $\lambda_{\min} = \omega_d(1)\cdot\sqrt{\log d}$, the parameter $c_2 := \lambda_{\min}/\sqrt{\log d}$ in Lemma 8 satisfies $32C_1/c_2^2 = o_d(1)$. Therefore, Lemma 8 implies that

$$\mathbb{E}[\beta(x,y)] \le \exp(\delta_1)\cdot\mathbb{E}[\exp(Z)] \le (1+o_d(1))(1+o_d(1)) = 1 + o_d(1).$$

Plugging the above in (10), it follows that the second term

$$\mathbb{E}_{x,y\sim\mathcal{G}}\left[P_x P_y\beta(x,y)\cdot\mathbf{1}[\overline{\mathcal{E}}]\right] \le p^2(1+o_d(1)).$$

Combining this with (8) and (9), we get that $\mathbb{E}_R[S^2] \le p^2(1+o_d(1))$. ◀

We now prove the lemmas and claims used in the proof of Theorem 1 above.

2.1 Truncated Gram-Schmidt Distribution and Exponential Moments

Proof of Lemma 5. Consider running the Gram-Schmidt walk algorithm on the matrix M stacked with the identity matrix, i.e. $\begin{pmatrix} M \\ I_n \end{pmatrix}$, and let $\mathcal{G}_0$ be the distribution over colorings obtained as an output of the algorithm.

Since each column of the stacked matrix has Euclidean norm at most 2, the properties of the Gram-Schmidt walk (Theorem 4) guarantees that $(x, Mx) \in \mathbb{R}^{n+d}$ where $x \in \{\pm 1\}^n$ and $Mx \in \mathbb{R}^d$ is 2-subgaussian. It follows that both x and Mx are 2-subgaussian as well when $(x, Mx) \sim \mathcal{G}_0$.

To obtain a distribution $\mathcal{G}$ where $\|Mx\|_2$ is almost constant for each coloring $x \in \mathsf{supp}(\mathcal{G})$, we will truncate the distribution $\mathcal{G}_0$ in such a way that the tails are also preserved up to poly(d) factors. Towards this end, we first note that with probability $1 - e^{-cd}$, we have that $\|Mx\|_2 \le c'\sqrt{d}$ for constants c and c'. This is because for any σ-subgaussian mean-zero random vector X, the Euclidean norm of $\|X\|$ has a subgaussian tail (e.g. Exercise 6.3.5 in [19]). In particular, $\mathbb{P}[\|X\|_2 \ge c_1\sigma\sqrt{d} + t] \le e^{-c_2 t^2/\sigma^2}$ for some universal constants $c_1, c_2 > 0$. Now, by a pigeonhole argument, for a large enough constant C' there exist an annulus W with width $\Delta = d^{-C'}$ and inner radius $r \le c'\sqrt{d}$ such that $\mathbb{P}_{x\sim\mathcal{G}_0}(x \in W) \ge d^{-C}$ for a constant C depending on C'.

We take the distribution $\mathcal{G}$ to be the probability measure of $\mathcal{G}_0$ conditioned on the event that $Mx \in W$. It then follows that for any coloring $x \sim \mathcal{G}$, we have $|\|Mx\|_2 - r| \le \Delta$. Moreover, since x and Mx were 2-subgaussian prior to conditioning, and the probability mass of the annulus is at least d^{-C}, conditioning can only increase the probability of any event by a factor of d^C. Thus, the tail bounds as stated in the statement of the lemma also follow. ◀

Proof of Lemma 8. The assumption on X implies that for any $t \ge 4 \cdot \sqrt{C_1 \log d}$, we have

$$\mathbb{P}(X \ge t) \le \exp(-t^2/16). \tag{11}$$

We express the expectation as an integration

$$\begin{aligned}\mathbb{E}[\exp(X^2/\lambda^2)] &= \int_0^\infty \mathbb{P}[\exp(X^2/\lambda^2) > s]ds = \int_0^\infty \mathbb{P}(X \ge \lambda\sqrt{\log s})ds \\ &\le 1 + c_3 + \int_{1+c_3}^\infty \mathbb{P}(X \ge \lambda\sqrt{\log s})ds.\end{aligned}$$

Let us set $c_3 = 32C_1/c_2^2$, so that $c_3 \le$ as $c_2 \ge \sqrt{32C_1}$. For $s \ge 1 + c_3$, we have

$$\lambda\sqrt{\log s} \ge c_2\sqrt{\log d} \cdot \sqrt{c_3/2} \ge 4 \cdot \sqrt{C_1 \log d}$$

using that $\sqrt{\log(1+x)} \ge \sqrt{x/2}$ for $x \in [0,1]$ and as $c_3 \le 1$. So the condition $t \ge 4\cdot\sqrt{C_1 \log d}$ for (11) is satisfied whenever $s \ge 1 + c_3$, and applying (11) to the above integration gives

$$\begin{aligned}\int_{1+c_3}^\infty \mathbb{P}(X \ge \lambda\sqrt{\log s})ds &\le \int_{1+c_3}^\infty \exp(-\lambda^2 \log s/16)ds \\ &= \left(\frac{\lambda^2}{16} - 1\right)^{-1} \cdot (1+c_3)^{-\lambda^2/16+1} \le \exp(-c_2^2 c_3 \log d/16).\end{aligned}$$

By our choice of c_3, the above is at most d^{-2C_1}. Thus, it follows that $\mathbb{E}[\exp(X^2/\lambda^2)] \le 1 + 32C_1/c_2^2 + d^{-2C_1}$. This proves the lemma. ◀

2.2 Proof of Claims from Section 2

Proof of Claim 6. Since the rows of R are independent, to compute the above ratio, it suffices to compute the ratio for a single row of $M + R$. Fix $i \in [d]$, and let $m = m_i$ and $r = r_i$ denote the i^{th} row and define $a = a_i(x) := -m^\top x$ and $b = b_i(y) := -m^\top y$. We want to compare the ratio of $\mathbb{P}(r^\top x \in [a \pm \Delta], r^\top y \in [b \pm \Delta])$ to $\mathbb{P}(r^\top x \in [a \pm \Delta]) \cdot \mathbb{P}(r^\top y \in [b \pm \Delta])$.

Notice that $r^\top x$ and $r^\top y$ are Gaussian random variables with mean 0, variance $1/\delta^2$, and covariance $\mathbb{E}_r[r^\top x r^\top y] = \mathbb{E}_r[x^\top r r^\top y] = \epsilon/\delta^2$. Denoting the square $K := [a \pm \Delta] \times [b \pm \Delta]$, we have that

$$\mathbb{P}(r^\top x \in [a \pm \Delta], r^\top y \in [b \pm \Delta]) = \mu_\epsilon(\delta K),$$

where μ_ϵ is the 2-dimensional centered Gaussian measure with covariance matrix $\begin{pmatrix} 1 & \epsilon \\ \epsilon & 1 \end{pmatrix}$ and δK denotes the δ scaling of K. Similarly, we can write $\mathbb{P}(r^\top x \in [a \pm \Delta]) \cdot \mathbb{P}(r^\top y \in [b \pm \Delta]) = \mu(\delta K)$, where μ is the *standard* 2-dimensional Gaussian measure.

We will compare the ratio $\mu_\epsilon(\delta K))/\mu(\delta K)$ by approximating the Gaussian measure over δK with the density at the center and show the following bound

$$\mu_\epsilon(\delta K))/\mu(\delta K) \leq \exp\left(3\alpha + \epsilon^2 + \delta^2\epsilon^2(a^2+b^2) + 2\delta^2\epsilon ab\right). \tag{12}$$

Since $(a_1(x), \ldots, a_d(x)) = Mx$ and $(b_1(x), \ldots, b_d(x)) = My$, using the above bound for all the rows $i \in [d]$, we have

$$\frac{P_{x,y}}{P_x P_y} \leq \exp\left(3d\alpha + d\epsilon^2 + \delta^2\epsilon^2 \cdot (\|Mx\|_2^2 + \|My\|_2^2) + \delta^2\epsilon \cdot \langle Mx, My\rangle\right).$$

Since $\|Mx\|_2^2 \leq d + \mathrm{poly}(1/d)$ for every $x \in \mathsf{supp}(\mathcal{G})$, taking $\delta_1 = 4d\alpha$ gives the statement of the claim. To finish the proof we prove (12) now.

Abusing notation and denoting by $\mu(s,t)$ and $\mu_\epsilon(s,t)$ the corresponding densities at $(s,t) \in \mathbb{R}^2$, we have the following explicit formula for the density μ_ϵ:

$$\mu_\epsilon(s,t) = \frac{1}{2\pi\sqrt{1-\epsilon^2}} \cdot \exp\left(-\frac{s^2+t^2-2\epsilon st}{2(1-\epsilon^2)}\right).$$

Since the edge length of the square δK is $2\delta\Delta$, whenever $|\epsilon| \leq 1/2$, a direct calculation with the densities shows that

$$\frac{\sup_{(s,t)\in\delta K}\mu(s,t)}{\inf_{(s,t)\in\delta K}\mu(s,t)} = \exp\left(2\delta^2\Delta(|a|+|b|)\right) \leq \exp\left(2\delta^2\Delta(|a|+|b|+2\Delta)\right) \leq \exp(\delta_1),$$

and that

$$\frac{\sup_{(s,t)\in\delta K}\mu_\epsilon(s,t)}{\inf_{(s,t)\in\delta K}\mu_\epsilon(s,t)} \leq \exp(4\delta^2\Delta(|a|+|b|+2\Delta)) \leq \exp(2\delta_1),$$

where δ_1 is as defined in the claim. It follows that whenever $|\epsilon| \leq 1/2$, we can use the density at the center of K to obtain

$$\begin{aligned}\frac{\mu_\epsilon(K)}{\mu(K)} &\leq \exp(3\delta_1) \cdot \frac{\mu_\epsilon(\delta a, \delta b)}{\mu(\delta a, \delta b)} = \frac{1}{\sqrt{1-\epsilon^2}} \cdot \exp\left(3\delta_1 + \frac{\delta^2\epsilon^2(a^2+b^2)}{2(1-\epsilon^2)} + \frac{\delta^2\epsilon ab}{1-\epsilon^2}\right) \\ &\leq \exp\left(3\delta_1 + \epsilon^2 + \delta^2\epsilon^2(a^2+b^2) + 2\delta^2\epsilon ab\right),\end{aligned}$$

thus proving (12). ◁

Proof of Claim 7. We have $P_x = \prod_{i\in[d]} \mathbb{P}[r_i^\top x \in [a_i \pm \Delta]]$. For any fixed $i \in [d]$, $r_i^\top x$ is distributed as $\mathcal{N}(0, 1/\delta^2)$, so after scaling the quantity $\mathbb{P}[r_i^\top x \in [a_i \pm \Delta]] = \mu(\delta \cdot I)$ where $I = [a_i \pm \Delta]$ and μ is the standard Gaussian measure in $\mathbb{R}$. Analogous to the proof of Claim 6, one can approximate the Gaussian density at any at the point in I by the center point a, and compute similarly to the proof of Claim 6 that

$$P_x = \prod_{i\in[d]} \mathbb{P}[r_i^\top x \in [a_i \pm \Delta]] = \left(\frac{\delta\Delta}{\sqrt{2\pi}}\right)^d \exp\left(\alpha_x - \frac{\delta^2\|Mx\|_2^2}{2}\right),$$

for some small error $|\alpha_x| \leq 2\delta^2\Delta(\|Mx\|_1 + d\Delta)$. As $\|Mx\|_2 \in [r \pm \Delta]$ and $r = O(\sqrt{d})$, we have that $\|Mx\|_1 = O(d)$ and the statement of the claim follows for some $\delta_x \leq |\alpha_x| + 1/\mathrm{poly}(d) \leq 1/\mathrm{poly}(d)$. ◁

3 Conclusion

For the Komlós problem, as studied in this paper, Gaussian noise is a natural way to model a smoothed analysis setting since the input vectors have Euclidean norm at most one. One can wonder whether similar results can be obtained with more general noise models, for instance, Bernoulli or other discrete noise models. Such noise models are also more conducive for smoothed analysis in other discrepancy settings, such as for the Beck-Fiala problem. The weighted second moment approach used here can also handle Bernoulli noise when the number of vectors $n \gg d^2$ but the second moment becomes difficult to control when n is smaller. It remains an interesting open problem to see if Bernoulli or other discrete noise models can be handled for the regime $n \gg d \log d$.

References

1 Dimitris Achlioptas and Yuval Peres. The threshold for random k-sat is $2^k \log 2 - o(k)$. *J. Amer. Math. Soc.*, 4:947–973, 2004.
2 Noga Alon and Joel H. Spencer. *The Probabilistic Method*. John Wiley & Sons, 2016.
3 Dylan J. Altschuler and Jonathan Niles-Weed. The discrepancy of random rectangular matrices. *CoRR*, abs/2101.04036, 2021. `arXiv:2101.04036`.
4 Wojciech Banaszczyk. Balancing vectors and Gaussian measures of n-dimensional convex bodies. *Random Struct. Algorithms*, 12(4):351–360, 1998.
5 Nikhil Bansal, Daniel Dadush, Shashwat Garg, and Shachar Lovett. The Gram-Schmidt walk: A cure for the Banaszczyk blues. *Theory Comput.*, 15:1–27, 2019.
6 Nikhil Bansal, Haotian Jiang, Raghu Meka, Sahil Singla, and Makrand Sinha. Prefix discrepancy, smoothed analysis, and combinatorial vector balancing. In *Proceedings of ITCS*, pages 13:1–13:22, 2022.
7 Nikhil Bansal and Raghu Meka. On the discrepancy of random low degree set systems. In *Proceedings of SODA 2019*, pages 2557–2564, 2019.
8 József Beck and Tibor Fiala. "Integer-making" theorems. *Discrete Appl. Math.*, 3(1):1–8, 1981.
9 Esther Ezra and Shachar Lovett. On the Beck-Fiala conjecture for random set systems. *Random Struct. Algorithms*, 54(4):665–675, 2019.
10 Cole Franks and Michael Saks. On the discrepancy of random matrices with many columns. *Random Struct. Algorithms*, 57(1):64–96, 2020.
11 Nika Haghtalab, Tim Roughgarden, and Abhishek Shetty. Smoothed analysis with adaptive adversaries. In *Proceedings of FOCS*, 2021.
12 Christopher Harshaw, Fredrik Sävje, Daniel Spielman, and Peng Zhang. Balancing covariates in randomized experiments using the gram-schmidt walk. *arXiv e-prints*, 2019. `arXiv:1911.03071`.
13 Rebecca Hoberg and Thomas Rothvoss. A Fourier-Analytic Approach for the Discrepancy of Random Set Systems. In *Proceedings of SODA*, pages 2547–2556, 2019.
14 Calum MacRury, Tomáš Masařík, Leilani Pai, and Xavier Pérez-Giménez. The phase transition of discrepancy in random hypergraphs, 2021. `arXiv:2102.07342`.
15 Aditya Potukuchi. Discrepancy in random hypergraph models. *CoRR*, abs/1811.01491, 2018.
16 Aditya Potukuchi. A spectral bound on hypergraph discrepancy. In *Proceedings of ICALP*, pages 93:1–93:14, 2020.
17 Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004.
18 Paxton Turner, Raghu Meka, and Philippe Rigollet. Balancing gaussian vectors in high dimension. In *Proceedings of COLT*, pages 3455–3486, 2020.
19 Roman Vershynin. *High-Dimensional Probability*, volume 1. Cambridge University Press, 2018.

Minimum+1 (s,t)-cuts and Dual Edge Sensitivity Oracle

Surender Baswana
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Koustav Bhanja
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Abhyuday Pandey
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Abstract

Let G be a directed multi-graph on n vertices and m edges with a designated source vertex s and a designated sink vertex t. We study the (s,t)-cuts of capacity minimum+1 and as an important application of them, we give a solution to the dual edge sensitivity for (s,t)-mincuts – reporting the (s,t)-mincut upon failure or addition of any pair of edges.

Picard and Queyranne [Mathematical Programming Studies, 13(1):8-16, 1980] showed that there exists a directed acyclic graph (DAG) that compactly stores all minimum (s,t)-cuts of G. This structure also acts as an oracle for the single edge sensitivity of minimum (s,t)-cut. Dinitz and Nutov [STOC, pages 509-518, 1995] showed that there exists an $\mathcal{O}(n)$ size 2-level cactus model that stores all global cuts of capacity minimum+1. However, for minimum+1 (s,t)-cuts, no such compact structures exist till date. We present the following structural and algorithmic results on minimum+1 (s,t)-cuts.

1. There exists a pair of DAGs of size $\mathcal{O}(m)$ that compactly store all minimum+1 (s,t)-cuts of G. Each minimum+1 (s,t)-cut appears as a (s,t)-cut in one of the 2 DAGs and is 3-transversal – it intersects any path in the DAG at most thrice.
2. There exists an $\mathcal{O}(n^2)$ size data structure that, given a pair of vertices $\{u,v\}$ which are not separated by an (s,t)-mincut, can determine in $\mathcal{O}(1)$ time if there exists a minimum+1 (s,t)-cut, say (A,B), such that $\{s,u\} \in A$ and $\{v,t\} \in B$; the corresponding cut can be reported in $\mathcal{O}(|B|)$ time.
3. There exists an $\mathcal{O}(n^2)$ size data structure that solves the dual edge sensitivity problem for (s,t)-mincuts. It takes $\mathcal{O}(1)$ time to report the value of a resulting (s,t)-mincut (A,B) and $\mathcal{O}(|B|)$ time to report the cut.
4. For the data structure problems addressed in (2) and (3) above, we also provide a matching conditional lower bound. We establish a close relationship among three seemingly unrelated problems – all-pairs directed reachability problem, the dual edge sensitivity problem for (s,t)-mincuts, and 2×2 maximum flow. Assuming the directed reachability hypothesis, this leads to $\tilde{\Omega}(n^2)$ lower bounds on the space for the latter two problems.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Network flows

Keywords and phrases mincut, maxflow, fault tolerant

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.15

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: http://www.cse.iitk.ac.in/users/sbaswana/Papers-published/icalp-2022-fv.pdf

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 15; pp. 15:1–15:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The concept of cuts of a graph is very fundamental in graph theory and has many algorithmic applications as well. There are mainly two types of cuts – global cuts and (s,t)-cuts. A set of edges whose removal disconnects a given undirected graph is called a *global* cut. Let $G = (V, E)$ be a directed multi-graph (E is a multiset) consisting of $n = |V|$ vertices and $m = |E|$ edges with a designated source vertex s and a designated sink vertex t. An (s,t)-cut of G is defined as follows.

▶ **Definition 1** ((s,t)-cut). *For a subset $C \subset V$ with $s \in C$ and $t \notin C$, the set of outgoing edges of C is called an (s,t)-cut.*

It follows from Definition 1 that for each (s,t)-cut, there exists at least one subset $C \subset V$ that defines it. For the simplicity of exposition and without causing any confusion, henceforth we shall use C to denote the corresponding (s,t)-cut as well. An edge is said to *contribute* to an (s,t)-cut C if it is an outgoing edge of C. The set of edges that have one endpoint in C and another endpoint in $\overline{C}$ is known as the *edge-set* of C, denoted by $E(C)$.

A (s,t)-cut (likewise a global cut) consisting of least number of contributing edges is called an (s,t)-mincut (likewise a global mincut).

There has been extensive research in designing efficient algorithms for global mincuts ([15, 18, 19]) as well as (s,t)-mincuts ([10, 16]). In addition, elegant graph structures have been invented that compactly store and characterize these cuts – Cactus graph for all global mincuts given by Dinitz, Karzanov, and Lomonosov [5], and a directed acyclic graph (DAG) given by Picard and Queyranne [23] for all (s,t)-mincuts. These structures can also serve as an efficient data structure for single edge sensitivity problem – to report the (s,t)-mincut (or global mincut) after the insertion/failure of an edge.

It is very natural to ask if there exists any compact structure for cuts of value greater than the value of the minimum cuts. For global minimum+1 cuts, Dinitz and Nutov answered this question in affirmative. In a seminal work [7], they showed that there exists an $\mathcal{O}(n)$ size 2-level cactus model that stores all minimum+1 cuts; they also gave a characterization of these cuts. An incremental maintenance of this structure solves the problem of maintaining minimum+2 edge connected components for any value of minimum cuts under insertion of edges; generalizing the results of Galil and Italiano [12] and Dinitz [6].

However, for minimum+1 (s,t)-cuts, to the best of our knowledge, no such compact structure exists till date. Note that the approach taken by Picard and Queyranne [23] for (s,t)-mincuts does not seem extendable for minimum+1 (s,t)-cuts. This is because their structure is based on the residual network resulting from a maximum (s,t)-flow and thus crucially exploits the equivalence between maximum flow and minimum cut (Ford and Fulkerson [10]); unfortunately, for a given minimum+1 (s,t)-cut, there is no equivalent (s,t)-flow.

While a compact structure for minimum+1 (s,t)-cuts is of significant importance from a graph theoretic perspective, equally important is a compact data structure that can efficiently answer the following fundamental query for any given edge $(u,v) \in E$.

$\boldsymbol{Q(u,v)}$**:** report a minimum+1 (s,t)-cut C, if exists, such that $u \in C$, $v \in \overline{C}$.

Interestingly, the DAG structure of Picard and Queyranne [23] for (s,t)-mincuts can answer efficiently the question stated above in case of (s,t)-mincuts. For this purpose, it crucially exploits the property that the set of (s,t)-mincuts are closed under intersection and union operations. Unfortunately, this property no longer holds for minimum+1 (s,t)-cuts, thus making it quite nontrivial to design a data structure for answering query Q.

The study of minimum+1 (s,t)-cuts has an immediate application as well. As is evident from the following fact, their study is indispensable for efficiently solving the dual edge sensitivity problem for (s,t)-mincuts.

▶ **Fact 2.** *The failure of a given pair of edges will reduce the value of (s,t)-mincut if and only if there is a minimum (s,t)-cut containing any failed edge or a minimum+1 (s,t)-cut containing both the failed edges.*

Sensitivity data structure for a given graph problem is motivated by the fact that graphs in the real world are prone to failures of vertices/edges. These failures are transient in nature. So while the set of failed vertices/edges keep changing with time, at any stage of time, the number of failed vertices/edges remains quite small. Therefore, the aim is to have a compact data structure that can efficiently report the solution of the given problem for a given set of failed vertices/edges. In a similar manner, we would like to efficiently report the solution of the given problem for a given set of newly added vertices/edges. This may help us determine which newly added vertex/edge changes the solution most significantly. In the past, many elegant fault-tolerant structures have been designed for various classical problems, like single-source reachability [17], shortest paths [4], breadth-first search [21], (s,t)-mincuts [23], all-pairs mincuts [1] etc, which can handle the failure of a single vertex/edge. It is certainly interesting and important to handle more than a single failure/addition. In this endeavour, it is quite natural to first design data structures that can handle dual failures (or dual insertions). This either helps, or exposes the difficulty, in solving the problem in its generality. It has turned out that the data structures that handle dual failures are often more complex and require deeper insight into the problem than the data structures that handle only single failure. This is evident at least for the following problems – single-source reachability [3], breadth-first search [20], shortest paths [9] etc. For the case of (s,t)-mincuts, note that the 40 years old data structure of Picard and Queyranne [23] is the only known sensitivity data structure and it can handle only a single edge failure/addition. It occupies $\mathcal{O}(m)$ space and can report the value of the resulting (s,t)-mincut and the corresponding cut in $\mathcal{O}(1)$ and $\mathcal{O}(m)$ time, respectively. No nontrivial data structure exists for the dual-edge sensitivity of (s,t)-mincuts till date.

1.1 New results and their overview

Let λ be the value of the (s,t)-mincut in G. Henceforth, we use $(\lambda+k)$ (s,t)-cut to denote a minimum+k (s,t)-cut, $k \in \{0,1\}$. Using just the sub-modularity of (s,t)-cuts (Lemma 10) and the relation between any pair of (s,t)-mincuts, we first present an alternate DAG structure, denoted by $\mathcal{D}_\lambda$, that compactly stores and characterizes all (s,t)-mincuts as follows. An (s,t)-cut in G is a (s,t)-mincut if and only if it appears as a 1-transversal cut in $\mathcal{D}_\lambda$ – the edge-set of an (s,t)-cut intersects any path in $\mathcal{D}_\lambda$ at most once. It can be easily observed that $\mathcal{D}_\lambda$, upon reversal of its edges, is identical to the DAG of Picard and Queyranne [23]. The major advantage of the approach taken for designing this alternate DAG is that it shows a way to design compact structures for $(\lambda+1)$ (s,t)-cuts using only the properties of (s,t)-cuts without exploiting the relation between cuts and flows. We now present an overview of our main results on the $(\lambda+1)$ (s,t)-cuts.

The set of minimum cuts are closed under intersection and union. This property has played a crucial role in designing compact structure as well as characterization of these mincuts – cactus graph for global mincuts by Dinitz, Karzanov, Lomonosov [5], the skeleton structure for Steiner mincuts by Dinitz and Vainshtein [8], and DAG for (s,t)-mincuts described in this paper. However, it turns out that $(\lambda+1)$ (s,t)-cuts are not closed under intersection as

well as union. Therefore, in order to design compact structure for $(\lambda+1)(s,t)$-cuts and their characterization, we analyse the relation between a pair of $(\lambda+1)(s,t)$-cuts. On the basis of the capacity of the cuts resulting from the intersection and union, each pair of $(\lambda+1)$ (s,t)-cuts is classified into exactly one of the three types – Type-1, Type-2, and Type-3. While designing compact structure for $(\lambda+1)$ (s,t)-cuts, the presence of pairs of Type-1 cuts poses a challenge in characterizing them. Likewise the presence of pairs of Type-2 cuts poses a challenge in determining the (s,t)-mincut upon the failure/addition of any pair of edges. A natural idea to conquer these challenges is to partition the set of $(\lambda+1)$ (s,t)-cuts into a *small* number of sets so that there is no pair of cuts from Type-1 (likewise Type-2) in a set. It can be observed that any arbitrary partitioning may produce a *large* number of sets which may defeat our objective of designing compact structures.

We present a technique called *covering* of (s,t)-cuts using which we can build a pair of graphs that cover all (s,t)-cuts of G and each of them has no pair of cuts from Type-1. It also helps in carefully handling pairs of cuts from Type-2. However, the covering technique works only for a graph with at most two (s,t)-mincuts, and hence can not be applied directly. In order to tackle this problem we introduce the concept of $(\lambda+1)$ (s,t)-class as follows.

▶ **Definition 3.** *A $(\lambda+1)$ (s,t)-class is a maximal set of vertices $A \subset V$ such that any pair of vertices from A are not separated by any (s,t)-mincut.*

It can be observed that the set of vertices of G that are assigned to a vertex of $\mathcal{D}_\lambda$ corresponds to a $(\lambda+1)$ (s,t)-class. We first form a partition of the set of all $(\lambda+1)$ (s,t)-cuts (excluding a set of *degenerate* cuts) with respect to the $(\lambda+1)$ (s,t)-classes. For each $(\lambda+1)$ (s,t)-class $\mathcal{W}$, we construct a new graph $G(\mathcal{W})$ that preserves all $(\lambda+1)$ (s,t)-cut of G that subdivides $\mathcal{W}$ and has at most two (s,t)-mincuts. We show that it is sufficient to work with $G(\mathcal{W})$ for each $\mathcal{W}$ to design compact structure for $(\lambda+1)$ (s,t)-cuts as well as dual edge sensitivity data structure for (s,t)-mincuts.

1. A 2-level DAG structure for $(\lambda+1)$ (s,t)-cuts. Along similar lines of $\mathcal{D}_\lambda$, we build a compact graph $\mathcal{D}_{\lambda+1}$ that stores all $(\lambda+1)$ (s,t)-cuts of $G(\mathcal{W})$. In order to establish the 1-transversality property of (s,t)-mincuts in $\mathcal{D}_\lambda$, the acyclicity of $\mathcal{D}_\lambda$ played a crucial role. Therefore, at first sight, we would expect $\mathcal{D}_{\lambda+1}$ to be acyclic and a $(\lambda+1)$ (s,t)-cut C in $G(\mathcal{W})$ to be a ℓ-transversal cut in $\mathcal{D}_{\lambda+1}$ – edge-set of C intersects any path in $\mathcal{D}_{\lambda+1}$ at most ℓ times, for some constant ℓ. We find that $\mathcal{D}_{\lambda+1}$ is not necessarily acyclic and, surprisingly enough, a $(\lambda+1)$ (s,t)-cut may appear in $\mathcal{D}_{\lambda+1}$ as an $\Omega(n)$-transversal cut. The root cause of non-acyclicity is the presence of pairs of cuts from Type-1. In order to tackle pairs of cuts from Type-1, we exploit the fact that $G(\mathcal{W})$ has at most two (s,t)-mincuts. We use the covering technique to construct a pair of graphs $G(\mathcal{W})^I$ and $G(\mathcal{W})^U$ that are Type-1 free. Now applying the same approach that was applied to obtain $\mathcal{D}_\lambda$, we construct a pair of DAGs – $\mathcal{D}_{\lambda+1}$ for graph $G(\mathcal{W})^I$ and $\mathcal{D}_{\lambda+1}$ for graph $G(\mathcal{W})^U$. This pair of DAGs is capable of characterizing each $(\lambda+1)$ (s,t)-cut in G that subdivides $\mathcal{W}$ as follows. A $(\lambda+1)$ (s,t)-cut in $G(\mathcal{W})$ appears as a *3-transversal* cut in exactly one of the two DAGs. In this way we obtain an $\mathcal{O}(m)$ size structure for compactly storing and characterizing all $(\lambda+1)$ (s,t)-cuts and it consists of two levels – (i) DAG $\mathcal{D}_\lambda$ and (ii) a pair of DAGs for each $(\lambda+1)$ (s,t)-class associated with a vertex of $\mathcal{D}_\lambda$.

Now we attempt to answer query $Q(u,v)$ using our 2-level DAG structure. Note that each 1-transversal cut in $\mathcal{D}_\lambda$ is also a (s,t)-mincut in G. Hence for the set of (s,t)-mincuts the query can be answered efficiently using a topological ordering of $\mathcal{D}_\lambda$. However, a 3-transversal cut in the pair of DAGs needs not be a $(\lambda+1)$ (s,t)-cut because of the existence of *certain* pairs of cuts from Type-2.

2. Data structure for $(\lambda+1)$ (s,t)-cuts. For each vertex u, there exists a unique (s,t)-mincut C, called *nearest (s,t)-mincut*, that keeps u on the side of s such that $C \subseteq C'$ for every other (s,t)-mincut C' that keeps u on the side of s. We can use this nearest (s,t)-mincut of u to determine if there exists a (s,t)-mincut C such that $u \in C$ and $v \in \overline{C}$ for any pair of vertices $\{u,v\}$. Unfortunately, there are multiple nearest $(\lambda+1)$ (s,t)-cuts that keep u on the side of s, and hence this approach fails.

Let $\mathcal{W}$ be the $(\lambda+1)$ (s,t)-class to which u belongs. Let C and C' be any pair of nearest $(\lambda+1)$ (s,t)-cuts of u. We show that C and C' do not *cross* in $\mathcal{W}$, that is, $\overline{C \cup C'} \cap \mathcal{W} = \emptyset$. Using this crucial insight, we are able to design an $\mathcal{O}(n^2)$ size data structure summarized in the following theorem.

▶ **Theorem 4.** *Let G be a directed multi-graph on n vertices and m edges with a designated source vertex s and a designated sink vertex t. There exists a data structure occupying $\mathcal{O}(n^2)$ space that can determine in $\mathcal{O}(k)$ time whether there exists a $(\lambda+1)$ (s,t)-cut C such that $u \in C$ and $v_1,\ldots,v_k \in \overline{C}$ for any given vertices $u, v_1,\ldots,v_k$ belonging to a $(\lambda+1)$ (s,t)-class. It can also report C in $\mathcal{O}(|\overline{C}|)$ time.*

3. An oracle for dual edge sensitivity for (s,t)-mincuts. We show that there is a data structure $\{\mathcal{F},\mathcal{I}\}$ occupying $\mathcal{O}(n^2)$ space which is capable of answering dual edge sensitivity query for (s,t)-mincuts in $\mathcal{O}(1)$ time. The data structure $\mathcal{F}$ for handling dual edge failure query is obtained as follows.

When both failed edges are not belonging to the same $(\lambda+1)$ (s,t)-class, data structures for (s,t)-mincuts are sufficient to answer the query. The main challenge arises when endpoints of both failed edges are belonging to the same $(\lambda+1)$ (s,t)-class $\mathcal{W}$. Notice that the data structure for $(\lambda+1)$ (s,t)-cut from Theorem 4 can determine whether there is a $(\lambda+1)$ (s,t)-cut in which a single failed edge is contributing, but cannot answer if both edges are contributing to a single $(\lambda+1)$ (s,t)-cut. Suppose there is a $(\lambda+1)$ (s,t)-cut C^* of G in which both failed edges, say (x,y) and (x',y'), are contributing. Then the necessary condition is that y' must not belong to the nearest $(\lambda+1)$ (s,t)-cut from x to y, and y must not belong to the nearest $(\lambda+1)$ (s,t)-cut from x' to y'. Therefore, if necessary condition holds then both edges are contributing to the union of the two nearest $(\lambda+1)$ (s,t)-cuts. However, the union needs not necessarily be a $(\lambda+1)$ (s,t)-cut because of the existence of certain pairs of $(\lambda+1)$ (s,t)-cuts from Type-2.

We employ the covering technique to tackle pairs of cuts from Type-2. Unfortunately, covering does not necessarily eliminate all Type-2 pairs of cuts like the way it does in case of Type-1 pairs. In order to tackle the problem arising due to the prevailing Type-2 pairs, we exploit the insight into the structure of $G(\mathcal{W})^I$ and $G(\mathcal{W})^U$. The end result is that, in order to determine whether any given pair of edges are contributing to a single $(\lambda+1)$ (s,t)-cut, we only have to perform a couple of nearest $(\lambda+1)$ (s,t)-cuts queries on $G(\mathcal{W})^I$ and $G(\mathcal{W})^U$.

4. Lower bound for various mincut data structures: We establish a close relationship between two seemingly unrelated problems – all-pairs directed reachability problem and dual edge sensitivity problem for (s,t)-mincuts. The classical problem of all-pairs directed reachability is defined as follows – Given a directed graph G on n vertices and m edges, preprocess it to form a data structure which can efficiently report if any given vertex v is reachable from another given vertex u. The problem becomes interesting when the underlying graph is sparse, that is, $m = o(n^2)$. It is natural to ask whether there is any data structure for the all-pairs directed reachability that takes $o(n^2)$ space and $o(m)$ query time ? Goldstein et al. [14], following the seminal work of Patrascu [22], stated a conjecture that concisely conveys the belief.

▶ **Conjecture 5** (Directed Reachability Hypothesis [14]). *Any data structure for the all-pairs reachability in a directed graph must either use $\tilde{\Omega}(n^2)$ space or $\tilde{\Omega}(m)$ query time. ($\tilde{\Omega}(.)$ denotes complexity upto polylogarithmic factors)*

We provide an $\mathcal{O}(m)$ time reduction from any instance of the all-pairs directed reachability problem for a given graph on n vertices and m edges to an instance of the dual edge sensitivity data structure for (s,t)-mincuts on a graph with $O(n)$ vertices and $\mathcal{O}(m)$ edges. Thus, assuming Conjecture 5 holds, our $\mathcal{O}(n^2)$ size data structure for dual edge sensitivity for (s,t)-mincuts is indeed the optimal size data structure for achieving $\mathcal{O}(1)$ query time. As a byproduct of this reduction, we establish conditional lower bounds on 2×2 flow tree. In particular, we show that if conjecture 5 holds, any data structure that can report the value of $(\{s,u\},\{v,t\})$-mincut for s,u,v and t being any four vertices of the graph must either use $\tilde{\Omega}(n^2)$ space or $\tilde{\Omega}(m)$ query time. An interesting implication of this result is a matching conditional lower bound for the data structure described in Theorem 4.

1.2 Related work

Benczur [2] gave an $\mathcal{O}(n^2)$ space geometric representation for all the global cuts of value within $\frac{6}{5}$ times the global minimum cut value. As an important application, this structure leads to the improvement in the time complexity of *splitting* algorithms [11].

Vazirani and Yannakakis [24] addressed the following fundamental question about (s,t)-cuts. – Is there a polynomial time algorithm to compute an (s,t)-cut of second minimum weight? They answered this question in the affirmative by showing that there is an algorithm which can compute a k^{th} minimum weight (s,t)-cut using only $\mathcal{O}(n^{2(k-1)})$ maximum flow computations. This algorithm gives an implicit structure for all (s,t)-cuts – a binary tree with ℓ leaves, that stores all (s,t)-cuts of G and the number of (s,t)-cuts is ℓ.

Another related work is on characterizing (s,t)-cuts using the polyhedron corresponding to the dual of linear programming (LP) formulation on maximum (s,t)-flow. Vazirani and Garg [13] showed that not all (s,t)-cuts can be characterized as the vertices of this polyhedron. Moreover, they modify the dual of the LP formulation by adding a polynomial number of constraints such that the corresponding polyhedron of the resulting LP formulation has all (s,t)-cuts as its vertices.

1.3 Organisation of the paper

Section 2 contains the basic preliminaries and the construction of a quotient graph for a set of $(\lambda + k)$ (s,t)-cuts. The technique of covering (s,t)-cuts is explained in Section 3. Section 4 contains the construction and properties of the alternate DAG structure for (s,t)-mincuts. The structure for $(\lambda + 1)$ (s,t)-cuts and their characterization is discussed in Section 5. The data structure for reporting $(\lambda + 1)$ (s,t)-cuts is constructed in Section 6. Section 7 contains the Oracle for dual-edge sensitivity of (s,t)-mincuts. Finally in Section 8 we give the lower bounds. Missing proofs of lemmas and theorems are provided in the full version.

2 Preliminaries

For any $X, Y \subseteq V$, the capacity of (X,Y) (denote by $c(X,Y)$) is the number of edges leaving X and entering Y; for brevity we use $c(X)$ to denote $c(X,\overline{X})$ where $\overline{X} = (V \setminus X)$. We say that an (s,t)-cut C *subdivides* $X \subseteq V$ if $C \cap X$ and $\overline{C} \cap X$ are non empty.

▶ **Definition 6** (Nearest $(\lambda+k)$ (s,t)-cut of a vertex). *The set $A \subset V$ with $s \in A$ and $t \in \overline{A}$ is said to be the nearest $(\lambda+k)$ (s,t)-cut of a vertex u if $u \in A$ and there is no $(\lambda+k)$ (s,t)-cut $B \subset V$ such that $u \in B$ and $B \subset A$. The set of all nearest $(\lambda+k)$ (s,t)-cuts of u is denoted by $N_k(u)$.*

▶ **Definition 7** (Nearest $(\lambda+k)$ (s,t)-cut for a pair of vertices). *Let A be a nearest $(\lambda+k)$ (s,t)-cut of a vertex u. For each vertex $v \in \overline{A}$, A is said to be a nearest $(\lambda+k)$ (s,t)-cut from vertex $\{u\}$ to a vertex $\{v\}$. The set of all nearest $(\lambda+k)$ (s,t)-cuts from u to v is denoted by $N_k(u,v)$.*

When $|N_1(u)| = 1$ (likewise $|N_1(u,v)| = 1$), without causing any ambiguity we use $N_1(u)$ (likewise $N_1(u,v)$) to denote the corresponding cut as well.

▶ **Definition 8** (ℓ-transversal cut). *A (s,t)-cut in a directed graph H is said to be ℓ-transversal, $\ell \geq 1$, if any path in H intersects with the edge-set of the (s,t)-cut at most ℓ times.*

▶ **Definition 9** (transpose of a graph). *Let H be a directed multi-graph with designated source vertex s and designated sink vertex t. The transpose of graph H (denoted by $(H)^{\mathcal{T}}$) is obtained by reversing the orientation of each edge of H with role of s and t swapped.*

▶ **Lemma 10** (Submodularity of cuts). *For any $A, B \subseteq V$, $c(A)+c(B) \geq c(A \cap B)+c(A \cup B)$.*

▶ **Lemma 11.** *Let A and B be any two (s,t)-mincuts in G that are crossing, that is, $A \setminus B$ as well as $B \setminus A$ are non-empty. There is no edge of G between $A \setminus B$ and $B \setminus A$.*

2.1 Quotient graph for a family of (s,t)-cuts

Let $\mathcal{C}$ be a set of $(\lambda+k)$ (s,t)-cuts in graph $G=(V,E)$ where $k \in \{0,1\}$. We define the following binary relation on the vertex set of G.

▶ **Definition 12** (Relation $R_{\lambda+k}$). *Any two vertices $\{x,y\} \in V$ are said to be related by $R_{\lambda+k}$ if and only if x and y are not separated by any $(\lambda+k)$ (s,t)-cut from $\mathcal{C}$.*

It is a simple exercise to show that $R_{\lambda+k}$ defines an equivalence relation on the vertex set. We call each equivalence class defined by $R_{\lambda+k}$ as a $(\lambda+k+1)$ (s,t)-class. It can be observed that any (s,t)-cut that subdivides a $(\lambda+k+1)$ (s,t)-class has capacity strictly larger than $\lambda+k$. The following lemma states how a $(\lambda+k)$ (s,t)-class is related to a $(\lambda+k)$ (s,t)-cut.

▶ **Lemma 13.** *A $(\lambda+k)$ (s,t)-cut can subdivide at most one $(\lambda+k)$ (s,t)-class.*

Since $R_{\lambda+k}$ is an equivalence relation, the $(\lambda+k+1)$ (s,t)-classes form a partition of the vertex set of G into disjoint subsets. Let $G_{\lambda+k}$ be the quotient graph of G obtained by contracting each of these subsets into single vertices. We call each vertex of the quotient graph as nodes. The node of $G_{\lambda+k}$ containing source s is denoted by S and the node containing sink t is denoted by T. We call an (S,T)-cut of $G_{\lambda+k}$ as (s,t)-cut without causing any ambiguity. The following theorem is immediate from the construction.

▶ **Theorem 14.** *For a directed multi-graph G and a set of $(\lambda+k)$ (s,t)-cuts $\mathcal{C}$ for any $k \in \{0,1\}$, there exists a quotient graph $G_{\lambda+k}$ of G such that $C \in \mathcal{C}$ is an (s,t)-cut in G if and only if C is a $(\lambda+k)$ (s,t)-cut in $G_{\lambda+k}$.*

An edge of $G_{\lambda+k}$ is classified as normal or *inverted* using the following definition.

▶ **Definition 15** (inverted edge for $(\lambda+k)$ (s,t)-cut). *An edge (x,y) of $G_{\lambda+k}$ is said to be an inverted edge if there exists no $(\lambda+k)$ (s,t)-cut $C \in \mathcal{C}$ such that $x \in C$ and $y \in \overline{C}$.*

▶ **Fact 16.** *For a directed multi-graph H and a set of $(\lambda+k)$ (s,t)-cuts $\mathcal{C}$, let $H_{\lambda+k}$ be the quotient graph of H. Given a pair of nodes μ, ν in graph $H_{\lambda+k}$, the contraction of all vertices which are mapped to μ or ν into a single vertex in H eliminates precisely all those $(\lambda+k)$ (s,t)-cuts from $\mathcal{C}$ that separate μ and ν.*

Consider two sets of $(\lambda+k)$ (s,t)-cuts $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \subset \mathcal{C}'$. Let $G_{\lambda+k}$ and $G'_{\lambda+k}$ be the quotient graphs for $\mathcal{C}$ and $\mathcal{C}'$ respectively. It follows from construction of quotient graphs that there exists equivalence classes of $G_{\lambda+k}$ which are further split into multiple equivalence classes of $G'_{\lambda+k}$. Therefore, the following lemma is immediate.

▶ **Lemma 17.** *Given a directed multi-graph G, let $G_{\lambda+k}$ and $G'_{\lambda+k}$ be a pair of quotient graphs formed on the sets of $(\lambda+k)$ (s,t)-cuts $\mathcal{C}$ and $\mathcal{C}'$ respectively. If $\mathcal{C} \subset \mathcal{C}'$ then $G_{\lambda+k}$ is a quotient graph of $G'_{\lambda+k}$.*

3 A covering of all (s,t)-cuts of a special graph

We first formalize the notion of *covering* the (s,t)-cuts of a graph using the following definition.

▶ **Definition 18** (Covering all (s,t)-cuts)**.** *Given a directed multi-graph H with a designated source vertex s and a designated sink vertex t, let $F = \{H^1, H^2, \ldots, H^\ell\}$ be a finite set of directed multi-graphs, each defined on the same vertex set as that of H. F is said to cover all (s,t)-cuts of H if and only if the following conditions hold.*

1. *For each (s,t)-cut C in H, there exists a (s,t)-cut C' in H^i for a unique $1 \leq i \leq \ell$ such that $C = C'$.*
2. *For each (s,t)-cut C of finite capacity in H^i for any $1 \leq i \leq \ell$, there exists a (s,t)-cut C' in H such that $C = C'$.*

Let H be a directed multi-graph with a designated source vertex s and a designated sink vertex t that has at most two (s,t)-mincuts – $\{s\}$ and (complement of) $\{t\}$. Our aim is to compute a small family $F = \{H^1, H^2, \ldots, H^\ell\}$ that covers all (s,t)-cuts of H and satisfies the following condition – for each pair C, C' of (s,t)-cuts in H^i for any $1 \leq i \leq \ell$, either their intersection or union is not a λ (s,t)-cut.

We shall show that there exists a family $F = \{H^U, H^I\}$ of only two graphs that *covers* all (s,t)-cuts of H and satisfies the said property. We build two graph H^U and H^I from H as follows. Let x be any arbitrary vertex other than s and t of H. H^I is formed by adding infinite capacity edge from vertex s to x. In a similar way, H^U is formed by adding infinite capacity edge from vertex x to t. The following lemma ensures that H^U and H^I together cover all $(\lambda+k)$ (s,t)-cuts of H. It exploits the following simple fact – Let x be a vertex of graph H and C be any (s,t)-cut; either $x \in C$ or $x \in \overline{C}$.

▶ **Lemma 19.** *Let $\lambda' > 0$ be a finite number. C is an (s,t)-cut in H of capacity λ' if and only if C is a cut of capacity λ' in H^U or H^I.*

It follows from the construction that the first (s,t)-mincut of H, that is $\{s\}$, is present in H^U while the second (s,t)-mincut of H, that is (complement of) $\{t\}$ is present in H^I. So using Lemma 19, we can state the following theorem.

▶ **Theorem 20.** *Let $H = (V, E)$ be a directed multi-graph on n vertices and m edges with a designated source vertex s and a designated sink vertex t. Suppose H has at most two (s,t)-mincuts, $\{s\}$ and $V \setminus \{t\}$. There exists a set of graphs $F = \{H^I, H^U\}$ satisfying the following properties:*

1. *F covers all $(\lambda+k)$ (s,t)-cuts of H.*
2. *$V \setminus \{t\}$ is the only λ (s,t)-cut in H^I, and $\{s\}$ is the only λ (s,t)-cut in H^U.*

▶ Note 21. Covering technique cannot be applied to a graph H that has more than two (s,t)-mincuts. This is because, for any vertex x in H, there will be more than one (s,t)-mincuts that keep x on side of s and/or more than one (s,t)-mincuts that keep x on side of t. Therefore, there will be multiple (s,t)-mincuts in H^I or multiple (s,t)-mincuts in H^U, hence violating Theorem 20(2).

4 An alternate DAG structure storing all (s,t)-mincuts

We now present an alternate compact structure $\mathcal{D}_\lambda$ for representing and characterizing all (s,t)-mincuts of G.

Construction of $\mathcal{D}_\lambda$: Let G_λ be the quotient graph defined by all λ (s,t)-cuts of G (Theorem 14). D_λ is the graph obtained by reversing the direction of each inverted edge of G_λ. Exploiting just the elementary properties of (s,t)-mincuts alone (Lemma 11 and submodularity of cuts), we show that D_λ is acyclic. As the reader may notice, D_λ is related to the DAG of Picard and Queyranne [23] – the mapping of vertices to the nodes of the DAG is identical while the direction of each edge is flipped. Each (s,t)-mincut of G is characterized by a 1-transversal cut in D_λ as stated in the following lemma.

▶ **Lemma 22** (1-transversality property). *An (s,t)-cut is an (s,t)-mincut of G if and only if it appears as a 1-transversal cut in $\mathcal{D}_\lambda$.*

The following lemma can be proved along similar lines as Lemma 22.

▶ **Lemma 23.** *An (s,t)-cut of G is an (s,t)-mincut if and only if it appears as an (s,t)-cut with no incoming edge in $\mathcal{D}_\lambda$.*

Interestingly, $\mathcal{D}_\lambda$ can serve as a compact data structure for efficiently answering the query $Q(u,v)$ for (s,t)-mincuts as follows. Let τ be a topological ordering of $\mathcal{D}_\lambda$. For each edge (μ,ν) of $\mathcal{D}_\lambda$, $\tau(\mu) < \tau(\nu)$. Hence, it is a simple corollary of Lemma 22 that every prefix of τ is a (s,t)-mincut. Therefore, given any edge $(u,v) \in E$, there is a (s,t)-mincut in G containing it if u and v appear in different nodes of $\mathcal{D}_\lambda$ and the node containing u appears before the node containing v.

5 Structure of $(\lambda+1)$ (s,t)-cuts

In order to construct a compact structure for $(\lambda+1)$ (s,t)-cuts, we take an approach similar to the construction of $\mathcal{D}_\lambda$ for (s,t)-mincuts and construct a graph $\mathcal{D}_{\lambda+1}$ as follows.

Construction of $\mathcal{D}_{\lambda+1}$: Let $G_{\lambda+1}$ be the quotient graph defined by all $(\lambda+1)$ (s,t)-cuts (Theorem 14). $D_{\lambda+1}$ is the graph obtained by reversing the direction of each inverted edge of $G_{\lambda+1}$. Henceforth $\mathcal{D}_{\lambda+1}$ for a graph H is denoted by $\mathcal{D}_{\lambda+1}(H)$.

$\mathcal{D}_\lambda$ characterizes (s,t)-mincuts of G as 1-transversal cuts. Unfortunately, it turns out that $\mathcal{D}_{\lambda+1}(G)$ is not sufficient for characterizing $(\lambda+1)$ (s,t)-cuts in terms of ℓ-tranversal cuts for some constant ℓ as stated in the following theorem.

▶ **Theorem 24** ($\Omega(n)$-transversality). *There exists a directed multi-graph H on n vertices with a designated source vertex s and a designated sink vertex t having only two (s,t)-mincuts and a $(\lambda+1)$ (s,t)-cut C such that C appears as a $\Omega(n)$-transversal cut in $\mathcal{D}_{\lambda+1}(H)$.*

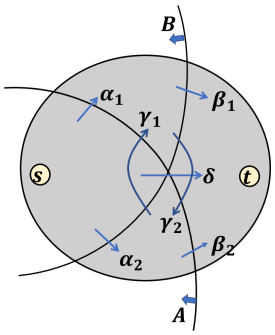


Figure 1 Contributing edges of a pair of (s,t)-cuts A and B between different regions.

We now provide a classification of all-pairs of $(\lambda+1)$ (s,t)-cuts. This classification will act as a crucial tool in the analysis of $\mathcal{D}_{\lambda+1}(G)$ as well as establishing properties of the compact structure for $(\lambda+1)$ (s,t)-cuts in Section 5.1.

Suppose A and B are any two $(\lambda+1)$ (s,t)-cuts. Using sub-modularity of cuts (Lemma 10), we arrive at the following inequality, $c(A\cap B)+c(A\cup B)\leq c(A)+c(B)=2(\lambda+1)$. Note that $c(A\cap B)\geq\lambda$ and $c(A\cup B)\geq\lambda$. This implies that the value of $c(A\cap B)+c(A\cup B)$ will always belong to $\{2\lambda,\ 2\lambda+1,\ 2\lambda+2\}$. Therefore, each pair (A,B) of $(\lambda+1)$ (s,t)-cuts can be classified uniquely into one of the three types based on the value of $c(A\cap B)+c(A\cup B)$. We now state the following lemma that will provide an alternate characterization of these three types based on the number of edges between $A\setminus B$ and $B\setminus A$.

▶ **Lemma 25.** *Suppose A and B are any two $(\lambda+1)$ (s,t)-cuts. Let γ_1 and γ_2 denote the number of edges from $A\setminus B$ to $B\setminus A$ and from $B\setminus A$ to $A\setminus B$ respectively. If $c(A\cap B)=\lambda+p$ and $c(A\cup B)=\lambda+q$ for some $p,q\geq 0$, then $\gamma_1+\gamma_2=2-p-q$.*

Proof. The cuts A and B partition G into at most 4 regions. Refer to Figure 1 for the illustration of these regions and the edges contributing to the respective cuts.

The following equations follow directly from Figure 1.

$$c(A\cap B)=\alpha_1+\alpha_2+\delta=\lambda+p \tag{1}$$
$$c(A\cup B)=\beta_1+\beta_2+\delta=\lambda+q \tag{2}$$
$$c(B)=\beta_1+\alpha_2+\gamma_2+\delta=\lambda+1 \tag{3}$$
$$c(A)=\beta_2+\alpha_1+\gamma_1+\delta=\lambda+1 \tag{4}$$

Using equations (1) and (3) we get $\alpha_1=\beta_1+\gamma_2+p-1$. Similarly using equations (1) and (4) we get $\alpha_2=\beta_2+\gamma_1+p-1$. By combining these two equalities, we get $\alpha_1+\alpha_2=\beta_1+\beta_2+\gamma_1+\gamma_2+2p-2$. Hence $\gamma_1+\gamma_2=2-p-q$, since $\alpha_1+\alpha_2=\beta_1+\beta_2+p-q$ from equations (2) and (1). ◀

Refer to Table 1 for two ways to characterize the three types of pairs of $(\lambda+1)$ (s,t)-cuts.

Table 1 Classification of any pair (A,B) of $(\lambda+1)$ (s,t)-cuts.

	$c(A\cap B)+c(A\cup B)$	$c(A\setminus B,B\setminus A)+c(B\setminus A,A\setminus B)$
Type-1	2λ	2
Type-2	$2\lambda+2$	0
Type-3	$2\lambda+1$	1

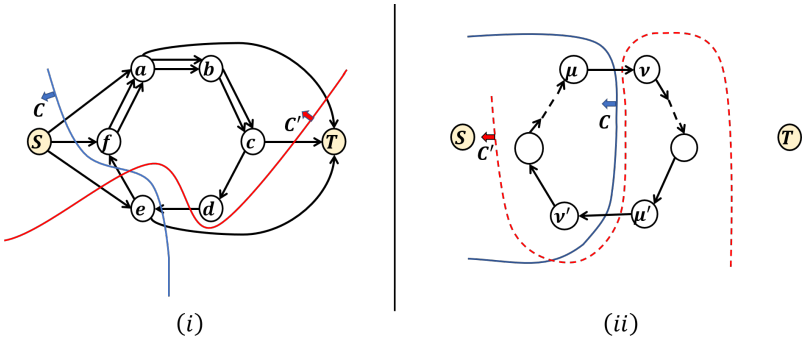


Figure 2 (i) An example of $\mathcal{D}_{\lambda+1}(G)$ with cycle $\langle a, b, c, d, e, f, a\rangle$. Note that here $\mathcal{D}_{\lambda+1}(G)$ turns out to be same as G. (ii) Cycle O in graph $\mathcal{D}_{\lambda+1}(H)$ from the proof of Lemma 27.

It follows from Theorem 24 that transversality of $(\lambda+1)$ (s,t)-cuts cannot be bounded by a constant even if a graph has at most two (s,t)-mincuts – $\{s\}$ and (complement of) $\{t\}$. In the following section, we shall first give a compact structure for $(\lambda+1)$ (s,t)-cuts for graph G assuming that G has at most two (s,t)-mincuts. Finally, we shall extend it to general graphs with the help of the structure of $\mathcal{D}_\lambda$ in Section 5.1.1.

5.1 Compact representation and characterization of $(\lambda+1)$ (s,t)-cuts

The characterization of (s,t)-mincuts crucially exploits the fact that $\mathcal{D}_\lambda$ is a DAG. Unfortunately, as shown in Figure 2(i), $\mathcal{D}_{\lambda+1}(G)$ can have cycles. The following lemma states the source of any 2-length cycle in $\mathcal{D}_{\lambda+1}$.

▶ **Lemma 26.** *If graph G does not have any pair of cuts from Type-1, then $\mathcal{D}_{\lambda+1}(G)$ cannot have a cycle of length two.*

Proof. Suppose we have a 2-length cycle $\langle \mu, \nu, \mu\rangle$ in $\mathcal{D}_{\lambda+1}(G)$. It follows from construction of $\mathcal{D}_{\lambda+1}$ that there exists a $(\lambda+1)$ (s,t)-cut C for the edge (μ,ν) such that $\mu \in C$ and $\nu \in \overline{C}$. In a similar manner there exists a $(\lambda+1)$ (s,t)-cut C' for the edge (ν,μ) such that $\nu \in C'$ and $\mu \in \overline{C'}$. It can be observed that (C, C') forms a pair of cuts from Type-1, a contradiction. ◀

In fact, the absence of pairs of $(\lambda+1)$ (s,t)-cuts from Type-1 is a sufficient condition for acyclicity as stated in the following lemma.

▶ **Lemma 27** (acyclicity property)**.** *If graph G does not have any pair of cuts from Type-1, then $\mathcal{D}_{\lambda+1}(G)$ is a directed acyclic graph.*

Proof. We begin with stating the following assertion.
$\mathcal{A}(i)$: For any graph H that has no pair of $(\lambda+1)$ (s,t)-cuts from Type 1, there is no cycle of length i in $\mathcal{D}_{\lambda+1}(H)$.
We shall now prove, by induction on i, that $\mathcal{A}(i)$ holds for all $i \geq 2$, and this would establish the lemma. The base case, $\mathcal{A}(2)$ follows directly from Lemma 26. Suppose $\mathcal{A}(j)$ holds for all $j < i$. We shall now prove that $\mathcal{A}(i)$ holds.

Suppose there is a cycle O of length i in $\mathcal{D}_{\lambda+1}(H)$ (see Figure 2(ii)). Consider any arbitrary edge (μ,ν) in this cycle. It follows from construction of $\mathcal{D}_{\lambda+1}(H)$ that there exists a $(\lambda+1)$ (s,t)-cut C of H such that $\mu \in C$ and $\nu \in \overline{C}$. Let $\mathcal{U}$ be the set of vertices of H that are mapped to either μ or ν. Contracting the set $\mathcal{U}$ into a single vertex we obtain a new graph H' from H.

It follows from Fact 16 and Lemma 17 that $\mathcal{D}_{\lambda+1}(H')$ has to be a quotient graph of $\mathcal{D}_{\lambda+1}(H)$. As a result, cycle O in $\mathcal{D}_{\lambda+1}(H)$ will be mapped to either a cycle of length strictly less than i or a single node in $\mathcal{D}_{\lambda+1}(H')$. The Induction Hypothesis rules out the possibility of the former case. We shall now rule out the possibility of the latter case. This will establish the validity of $\mathcal{A}(i)$.

The cut C must contain at least one more edge, say (μ', ν'), of cycle O because the cycle O must intersect the edge-set of C at least twice. It follows from the construction of $\mathcal{D}_{\lambda+1}(H)$ that there exists a $(\lambda+1)$ (s,t)-cut, say C', in $\mathcal{D}_{\lambda+1}(H)$ such that $\mu' \in C'$ and $\nu' \in \overline{C'}$. Since μ' and ν' belong to the same node in $\mathcal{D}_{\lambda+1}(H')$, the cut C' is not present in H'. This observation, in the light of Fact 16, implies that C' must separate μ and ν (dashed curve in Figure 2). This would imply that for the pair of cuts (C, C'), $c(C \setminus C', C' \setminus C) + c(C' \setminus C, C \setminus C') = 2$. Hence (C, C') forms a Type-1 pair of $(\lambda+1)$ (s,t)-cut in H, a contradiction. ◀

In order to have acyclic structures that collectively preserve all $(\lambda+1)$ (s,t)-cuts, Lemma 27 raises the following question – can we partition the set of $(\lambda+1)$ (s,t)-cuts such that each partition does not contain any pair of $(\lambda+1)$ (s,t)-cut from Type-1? The covering technique (Theorem 20) gives an affirmative answer to this question. In particular, it outputs just a pair of graphs $\{G^I, G^U\}$ such that all $(\lambda+1)$ (s,t)-cuts of G are covered by G^I and G^U together. Moreover, both G^U and G^I will contain exactly one (s,t)-mincut each, and this ensures that there are no pairs of $(\lambda+1)$ (s,t)-cuts from Type-1 in G^I or G^U. Therefore, it follows from Lemma 27 that $\mathcal{D}_{\lambda+1}(G^I)$ and $\mathcal{D}_{\lambda+1}(G^U)$ are acyclic. In the case when G has exactly one (s,t)-mincut, $\mathcal{D}_{\lambda+1}(G)$ itself is acyclic.

Not only $\mathcal{D}_{\lambda+1}(G^I)$ and $\mathcal{D}_{\lambda+1}(G^U)$ are acyclic but also help in characterizing $(\lambda+1)$ (s,t)-cuts as follows.

We show that each $(\lambda+1)$ (s,t)-cut of G is 3-transversal in $\mathcal{D}_{\lambda+1}(G^I)$ or in $\mathcal{D}_{\lambda+1}(G^U)$. Without loss of generality let us consider the graph $\mathcal{D}_{\lambda+1}(G^U)$. In order to accomplish 3-transversality of $(\lambda+1)$ (s,t)-cuts in $\mathcal{D}_{\lambda+1}(G^U)$, we first show that for any path in $\mathcal{D}_{\lambda+1}(G^U)$ whose first node is not S, there is no $(\lambda+1)$ (s,t)-cut which can keep both the first and the last node of the path on side of S, and remaining nodes on side of T. As a warm-up, the following lemma validates this assertion for all paths of length two.

▶ **Lemma 28.** *Let $P = \langle v_0 \neq S, v_1, v_2 \rangle$ be a path in $\mathcal{D}_{\lambda+1}(G^U)$. There can not exist any $(\lambda+1)$ (s,t)-cut C such that $v_0, v_2 \in C$ and $v_1 \in \overline{C}$.*

Proof. We give a proof by contradiction. Assume that there is such a $(\lambda+1)$ (s,t)-cut C. It follows from the construction of $\mathcal{D}_{\lambda+1}(G^U)$ that there is a $(\lambda+1)$ (s,t)-cut C' for the edge (v_1, v_2) such that $v_1 \in C'$ and $v_2 \in \overline{C'}$. It can be observed that the edge (v_1, v_2) is an edge from $C' \setminus C$ to $C \setminus C'$. Therefore, $\{C, C'\}$ cannot be a pair from Type-2. $\{C, C'\}$ cannot be pair from Type-1 as well since there is no pair of $(\lambda+1)$ (s,t)-cuts from Type-1 in $\mathcal{D}_{\lambda+1}(G^U)$. So, $\{C, C'\}$ must be a pair from Type-3. Notice that $C \cup C'$ has capacity $\lambda+1$ since the only λ (s,t)-cut in G^U is $\{s\}$. Hence, $C \cap C'$ must be containing the node S only. Since $v_0 \in C$, therefore, v_0 cannot belong to C' and hence the edge-set of C' must intersect path P again at edge (v_0, v_1). Then, in that case, there are two edges (v_0, v_1) and (v_1, v_2) between $C \setminus C'$ and $C' \setminus C$. This implies that (C, C') are from Type-1, a contradiction. ◀

Generalizing Lemma 28 for any arbitrary length path, we give the following lemma which will provide the foundation for establishing 3-transversality of $(\lambda+1)$ (s,t)-cuts.

▶ **Lemma 29.** *Let $P = \langle v_0 \neq S, v_1, \ldots, v_i \rangle$ be a simple path in $\mathcal{D}_{\lambda+1}(G^U)$. There does not exist a $(\lambda+1)$ (s,t)-cut C such that $v_0, v_i \in C$ and $v_1, \ldots, v_{i-1} \in \overline{C}$.*

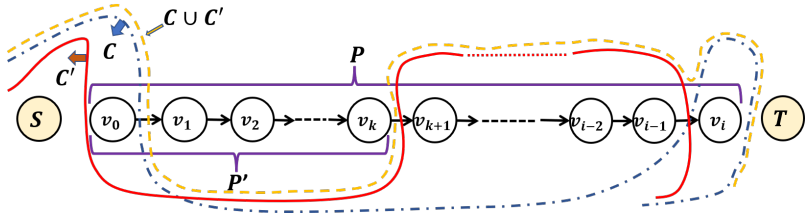


Figure 3 C: dash-dot curve, C': solid curve, $C \cup C'$: dashed curve, (v_{i-1}, v_i) is in $E(C) \cap E(C')$.

Proof. We begin with stating the following assertion.

$\mathcal{A}(i)$: There does not exist a $(\lambda+1)$ (s,t)-cut C such that $v_0, v_i \in C$ and $v_1, \ldots, v_{i-1} \in \overline{C}$ for a path $P = \langle v_0 \neq S, v_1, \ldots, v_{i-1}, v_i \rangle$ of length i.

We shall use proof by induction on the length of the path $i \geq 2$ to show that $\mathcal{A}(i)$ holds. The base case, $\mathcal{A}(2)$ follows directly from Lemma 28.

Let us now assume that $\mathcal{A}(j)$ holds for all $2 \leq j < i$. We shall prove that $\mathcal{A}(i)$ holds using a proof by contradiction. Assume to the contrary that there is a $(\lambda+1)$ (s,t)-cut C such that $v_0, v_i \in C$ and $v_1, \ldots, v_{i-1} \in \overline{C}$ (refer to Figure 3). It follows from the construction of $\mathcal{D}_{\lambda+1}(G^U)$ that there is a $(\lambda+1)$ (s,t)-cut C' for the edge (v_{i-1}, v_i) such that $v_{i-1} \in C'$ and $v_i \in \overline{C'}$. Along similar lines of the proof of Lemma 28 we can argue that (C, C') are pairs from Type-3 with $c(C \cup C') = \lambda + 1$, and edge-set of C' must be intersecting path P at least one more time before the edge (v_{i-1}, v_i). So, let (v_k, v_{k+1}), $0 \leq k < i-1$, be the first edge on P that intersects edge-set of C'. Now, there are two possible cases, either $k = 0$ and $v_{k+1} = v_1$ or $k > 0$. In the former case, it is easy to observe that two edges (v_0, v_1) and (v_{i-1}, v_i) lie between $C' \setminus C$ and $C \setminus C'$. Therefore, (C, C') must be from Type-1, a contradiction. In the latter case, observe that $v_0, \ldots, v_k \notin C'$ and $v_{k+1} \in C'$ because (v_k, v_{k+1}) is the nearest edge from v_0 in P that intersects edge-set of C'. Therefore, $v_0, v_{k+1} \in C \cup C'$ and $v_1, \ldots, v_k \notin C \cup C'$. So, we get a path $P' = \langle v_0, \ldots . v_{k+1} \rangle$ and a $(\lambda+1)$ (s,t)-cut $C \cup C'$ such that $v_0, v_{k+1} \in C \cup C'$ and $v_1, \ldots, v_k \in \overline{C \cup C'}$. Moreover, path P' has length strictly smaller than i because $k < i-1$. Therefore, $\mathcal{A}(k+1)$ fails to hold and $k+1 < i$, a contradiction. ◀

Now in the following lemma, using Lemma 29 we argue that there cannot exist any $(\lambda+1)$ (s,t)-cut C in $\mathcal{D}_{\lambda+1}(G^U)$ such that edge-set of C intersects a path P more than thrice.

▶ **Lemma 30** (3-transversality property). *Each $(\lambda+1)$ (s,t)-cut of G is 3-transversal in $\mathcal{D}_{\lambda+1}(G^I)$ or in $\mathcal{D}_{\lambda+1}(G^U)$.*

Proof. Without loss of generality let us consider $\mathcal{D}_{\lambda+1}(G^U)$. The proof is along similar lines for $\mathcal{D}_{\lambda+1}(G^I)$. We give a proof by contradiction. Assume to the contrary that there exists a $(\lambda+1)$ (s,t)-cut C in G and a path P in $\mathcal{D}_{\lambda+1}(G^U)$ such that edge-set of C intersects path P more than thrice. Then, path P can be divided into at least 5 contiguous disjoint subpaths $\{P_1, P_2, P_3, P_4, P_5\}$. For the cut C, observe that either each node of P_i is on side of S for all odd i and each node of P_i is on side of T for all even i or vice versa. In the former case, let us consider the subpath $P' = \langle P_3, P_4, P_5 \rangle$ of P. For this path P', we have each node of P_3, P_5 in C and each node of P_4 in $\overline{C}$. Moreover, S cannot belong to P_3 because by construction of $\mathcal{D}_{\lambda+1}(G^U)$ there is no incoming edge to S. Therefore, we have a path P' with first node (not S) and last node in $(\lambda+1)$ (s,t)-cut C and other nodes are in $\overline{C}$. This contradicts Lemma 29. In a similar way we can argue the latter case using subpath $P' = \langle P_2, P_3, P_4 \rangle$. ◀

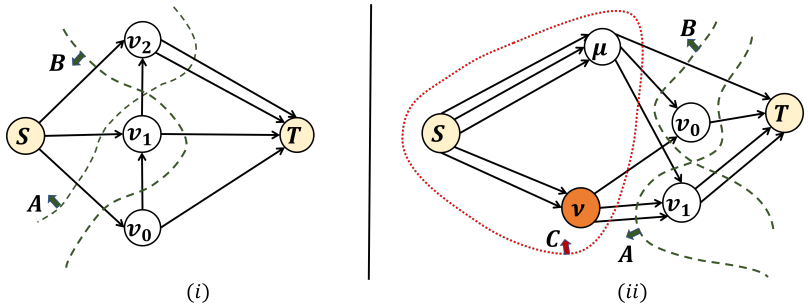


Figure 4 (i) The edge-set of $(\lambda+1)$ (s,t)-cut A intersects path $\langle S, v_1, v_2, T\rangle$ thrice; A and B are $(\lambda+1)$ (s,t)-cuts from Type-3. (ii) A 1-transversal cut $C = A \cap B$ with capacity greater than $\lambda+1$.

We also show that there are $(\lambda+1)$ (s,t)-cuts which may not appear in $\mathcal{D}_{\lambda+1}$ as 1-transversal cut (refer to Figure 4(i)). This is because, for each 3-transversal $(\lambda+1)$ (s,t)-cut A, there exists a $(\lambda+1)$ (s,t)-cut B such that A, B are pairs from Type-3. Therefore, our bound of 3-transversality is tight.

5.1.1 Extension to general graphs

The compact structure and characterization of $(\lambda+1)$ (s,t)-cuts are valid if the graph has at most two (s,t)-mincuts. However, in general, graphs can have exponential number of (s,t)-mincuts. To tackle this difficulty we explore how a $(\lambda+1)$ (s,t)-cut is related to the $(\lambda+1)$ (s,t)-classes of a graph G. There are $(\lambda+1)$ (s,t)-cuts in G which do not subdivide any $(\lambda+1)$ (s,t)-class. We call them *degenerate* $(\lambda+1)$ (s,t)-cuts. These cuts appear in $\mathcal{D}_\lambda$ and have the following characterization.

▶ **Lemma 31.** *An (s,t)-cut of G is a degenerate $(\lambda+1)$ (s,t)-cut if and only if it appears as an (s,t)-cut with exactly one incoming edge in $\mathcal{D}_\lambda$.*

Henceforth, our main focus is on the non-degenerate $(\lambda+1)$ (s,t)-cuts – cuts that subdivide $(\lambda+1)$ (s,t)-classes. It follows from Lemma 13 that each such $(\lambda+1)$ (s,t)-cut subdivides precisely one $(\lambda+1)$ (s,t)-class. Therefore, the set of all $(\lambda+1)$ (s,t)-cuts can be partitioned into disjoint subsets where each subset subdivides exactly one $(\lambda+1)$ (s,t)-class. This partitioning allows us to work separately with each $(\lambda+1)$ (s,t)-class. Let $\mathcal{W}$ be a $(\lambda+1)$ (s,t)-class. In order to build a compact structure that stores all $(\lambda+1)$ (s,t)-cuts that subdivide $\mathcal{W}$, we now define a graph $G(\mathcal{W})$ associated with $\mathcal{W}$ as follows.

Construction of $G(\mathcal{W})$: Let τ be a topological ordering of $\mathcal{D}_\lambda$ where source node S (likewise T) has the smallest (likewise highest) topological number. $G(\mathcal{W})$ is obtained by forming a quotient graph of G using τ as follows. Let μ be the node of $\mathcal{D}_\lambda$ corresponding to $\mathcal{W}$. All nodes that precede μ in the topological ordering τ are contracted into a single source node S' and every node that succeeds μ are contracted to a single sink node T'.

It is easy to observe that $s \in S'$ (likewise $t \in T'$) since $s \in S$ (likewise $t \in T$). Henceforth without causing any ambiguity we denote an (S', T')-cut in $G(\mathcal{W})$ as an (s,t)-cut in $G(\mathcal{W})$. The following lemma establishes the mapping between the $(\lambda+1)$ (s,t)-cuts of G and $G(\mathcal{W})$.

▶ **Lemma 32.** *A $(\lambda+1)$ (s,t)-cut C in G subdivides a $(\lambda+1)$ (s,t)-class $\mathcal{W}$ into $(\mathcal{W}_1, \mathcal{W}\setminus\mathcal{W}_1)$ if and only if there exists a $(\lambda+1)$ (s,t)-cut $C' = \mathcal{W}_1 \cup \{S'\}$ in $G(\mathcal{W})$.*

It is easy to observe that graph $G(\mathcal{W})$ can have at most two (s,t)-mincuts – S' and complement of T'. Therefore, we construct pair of DAGs, $\mathcal{D}_{\lambda+1}((G(\mathcal{W}))^I)$ and $\mathcal{D}_{\lambda+1}((G(\mathcal{W}))^U)$, for each $(\lambda+1)$ (s,t)-class $\mathcal{W}$. Now we summarize these structural and characterization results in the following theorem.

▶ **Theorem 33.** *For any directed multi-graph G, on n vertices and m edges with a designated source vertex s and a designated sink vertex t, there exists a 2-level DAG structure of $\mathcal{O}(m)$ size–(i) $\mathcal{D}_\lambda$ and (ii) a pair of DAGs associated with each node of $\mathcal{D}_\lambda$, such that all $(\lambda+1)$ (s,t)-cuts of G are compactly stored and characterized as follows.*

1. *a non-degenerate $(\lambda+1)$ (s,t)-cut of G subdividing a $(\lambda+1)$ (s,t)-class $\mathcal{W}$ is a 3-transversal cut in one of the two DAGs associated with $\mathcal{W}$, and*
2. *an (s,t)-cut of G is a degenerate $(\lambda+1)$ (s,t)-cut if and only if it has exactly one incoming edge in $\mathcal{D}_\lambda$.*

We now explore the possibility of answering query $Q(u,v)$ using the pair of DAGs, $\mathcal{D}_{\lambda+1}(G^I)$ or $\mathcal{D}_{\lambda+1}(G^U)$. Recall that for the case of (s,t)-mincuts, just storing a topological ordering of $\mathcal{D}_\lambda$ is sufficient to answer query $Q(u,v)$ because each 1-transversal cut in $\mathcal{D}_\lambda$ is also an (s,t)-mincut. However, a 1-transversal cut in $\mathcal{D}_{\lambda+1}(G^I)$ or $\mathcal{D}_{\lambda+1}(G^U)$ needs not be a $(\lambda+1)$ (s,t)-cut. For example, cut $A \cap B$ as shown in Figure 4(ii) is 1-transversal but has capacity $\lambda+2$. Note that A and B form a $(\lambda+1)$ (s,t)-cuts from Type-2. In the following section, we provide a compact data structure that can answer query $Q(u,v)$ even when (u,v) is not necessarily an edge.

6 Compact data structures for reporting $(\lambda+1)$ (s,t)-cuts

In this section we address the very fundamental problem of reporting any $(\lambda+1)$ (s,t)-cut C for a given pair of vertices $\{u,v\}$ such that $u \in C$ and $v \in \overline{C}$, if exists. In order to answer the query it is sufficient to verify if v is separated by at least one of the cuts from $N_1(u)$. Note that in case of (s,t)-mincut, it suffices to store the $N_0(u)$ which turns out to be unique for each vertex u. This structure occupies $\mathcal{O}(n^2)$ space and achieves $\mathcal{O}(|C|)$ time for reporting the (s,t)-mincut C such that $u \in C$ and $v \in \overline{C}$. Unfortunately, $N_1(u)$ can have more than one elements (as shown in Figure 4 that both A and B belong to $N_1(\nu)$). This is because unlike for (s,t)-mincuts, the set of all $(\lambda+1)$ (s,t)-cuts that keep a vertex u on the side of s is not closed under intersection and union operations.

In order to design compact data structure for reporting $(\lambda+1)$ (s,t)-cut, as discussed in Section 5.1.1, we work on each $(\lambda+1)$ (s,t)-class. Let $\mathcal{W}$ be a $(\lambda+1)$ (s,t)-class in graph G and $u,v \in \mathcal{W}$. Although $N_1(u)$ can have multiple elements, it can be shown using sub-modularity of cuts (Lemma 10) that $N_1(u,v)$ is unique. However, this fact alone guarantees $\mathcal{O}(|\mathcal{W}|^3)$ space data structure for determining the existence of $N_1(u,v)$ for any pair of vertices $u,v \in \mathcal{W}$. In order to achieve more compact data structure we explore how $N_1(u,v)$ is related to $N_1(u,w)$ for any $w \in \mathcal{W}$. The following lemma provides an important insight into this relationship.

▶ **Lemma 34.** *If $N_1(u,v) \neq N_1(u,w)$ for any $\{u,v,w\}$ in $\mathcal{W}$, then $\mathcal{W} \subseteq N_1(u,v) \cup N_1(u,w)$.*

Proof. The proof is by contradiction. Let $C = N_1(u,v)$ and $C' = N_1(u,w)$. Assume that there is a vertex in $\mathcal{W}$ which is also in $\overline{C \cup C'}$. In that case both $C \cap C'$ and $C \cup C'$ subdivide $\mathcal{W}$; therefore, capacity of each of them is strictly larger than λ. So it immediately follows from sub-modularity of cuts (Lemma 10) that $c(C \cap C')$ is $(\lambda+1)$. Therefore we have a $(\lambda+1)$ (s,t)-cut $C \cap C'$ which is a proper subset of at least one of C and C'. Therefore, C or C' fails to satisfy Definition 7 – a contradiction. ◀

Now let $N_1(u) = \{C_1, C_2, \ldots, C_l\}$. Consider any vertex $v \in \mathcal{W}$. If v belongs to $\bigcap_{i=1}^{l} C_i$, then $N_1(u, v)$ does not exist; otherwise, Lemma 34 implies that v is separated from u by exactly one of the cuts from $N_1(u)$. As a result the sets $\overline{C_i} \cap \mathcal{W}$ for each $i \in [l]$ are disjoint. In order to determine the position of any other vertex, $w \in \mathcal{W}$, with respect to $N_1(u, v)$, we formulate the following query.

$$\textsc{belong}(u, v, w) = \begin{cases} 1 & \text{if } N_1(u, v) \text{ exists and } w \in N_1(u, v) \\ 0 & \text{otherwise.} \end{cases}$$

For each vertex $x \in \mathcal{W} \setminus \{u\}$, if $x \in \overline{C_i}$ for any $i \in [l]$ then mark x with label i. Now if label of v and w are same, then w is not present in $N_1(u, v)$, otherwise w belongs to $N_1(u, v)$. In this way we can answer query $\textsc{belong}(u, v, w)$ in $\mathcal{O}(1)$ time. Therefore, the following lemma is immediate.

▶ **Lemma 35.** *Let $\mathcal{W}$ be a $(\lambda + 1)$ (s, t)-class and $u \in \mathcal{W}$. There exists an $\mathcal{O}(|\mathcal{W}|)$ size data structure $\mathcal{N}_{\lambda+1}(u)$ that can determine in $\mathcal{O}(k)$ time whether there exists a $(\lambda+1)$ (s, t)-cut C such that $u \in C$ and $v_1, \ldots, v_k \in \overline{C}$ for any $v_1, \ldots, v_k \in \mathcal{W}$. If C exists, the data structure can output $\overline{C} \cap \mathcal{W}$ in $\mathcal{O}(|\overline{C} \cap \mathcal{W}|)$ time.*

Moreover, given vertices $u, v_1, \ldots, v_k \in \mathcal{W}$, $\mathcal{N}_{\lambda+1}(u)$ can be used to report a $(\lambda + 1)$ (s, t)-cut C, if exists, in $\mathcal{O}(|\overline{C}|)$ time using an auxiliary $\mathcal{O}(n)$ space topological ordering of $\mathcal{D}_\lambda$ of G such that $u \in C$ and $v_1, \ldots, v_k \in \overline{C}$. Now based on Lemma 35 we construct the following data structure for $(\lambda + 1)$ (s, t)-class $\mathcal{W}$.

Description of $\mathcal{N}_{\lambda+1}$: It consists of $\mathcal{N}_{\lambda+1}(u)$ for each u in $\mathcal{W}$.

$\mathcal{N}_{\lambda+1}$ for a $(\lambda + 1)$ (s, t)-class $\mathcal{W}$ occupies $\mathcal{O}(|\mathcal{W}|^2)$ space. Each $(\lambda + 1)$ (s, t)-class is disjoint from each other. Hence by constructing $\mathcal{N}_{\lambda+1}$ for each $(\lambda + 1)$ (s, t)-class of G we get an $\mathcal{O}(n^2)$ size data structure and it completes the proof of Theorem 4. We complement this result with a conditional lower bound of $\tilde{\Omega}(n^2)$ space based on Conjecture 5.

7 Dual edge sensitivity oracle for (s, t)-mincuts

In this Section we shall present an oracle that can efficiently report (s, t)-mincut upon the failure of a pair of edges in graph G. We say that an edge (u, v) belongs to a $(\lambda+1)$ (s, t)-class $\mathcal{W}$ if both $u, v \in \mathcal{W}$.

7.1 Handling dual edge failures

Consider the failure of two edges $e = (x, y), e' = (x', y')$ in G. Suppose at least one of $\{e, e'\}$ does not belong to any $(\lambda + 1)$ (s, t)-class. In this case the value of (s, t)-mincut decreases by at least 1 if $N_0(x, y)$ or $N_0(x', y')$ exists. The value of (s, t)-mincut decreases by exactly 2 if and only if both e, e' are contributing to a single (s, t)-mincut. To determine the existence of such an (s, t)-mincut, since (s, t)-mincuts are closed under union operation, it is sufficient to verify whether $y \notin N_0(x', y')$ and $y' \notin N_0(x, y)$. So we construct a data structure, denoted by $\mathcal{N}_\lambda$, which consists of $N_0(u)$ for each vertex u of G. This $\mathcal{O}(n^2)$ space data structure achieves $\mathcal{O}(1)$ time to report the resulting value of (s, t)-mincut on failure of $\{e, e'\}$ in this case.

If e and e' belong to distinct $(\lambda + 1)$ (s, t)-classes, then it follows from Lemma 13 that the (s, t)-mincut value remains unchanged. Let us consider case when both e and e' belong to the same $(\lambda + 1)$ (s, t)-class, say $\mathcal{W}$. It follows as a simple corollary of Lemma 32 that we just need to verify if there is a $(\lambda + 1)$ (s, t)-cut in $G(\mathcal{W})$ in which e and e' are contributing. Note that the only $(\lambda + 1)$ (s, t)-class of $G(\mathcal{W})$ is $\mathcal{W}$.

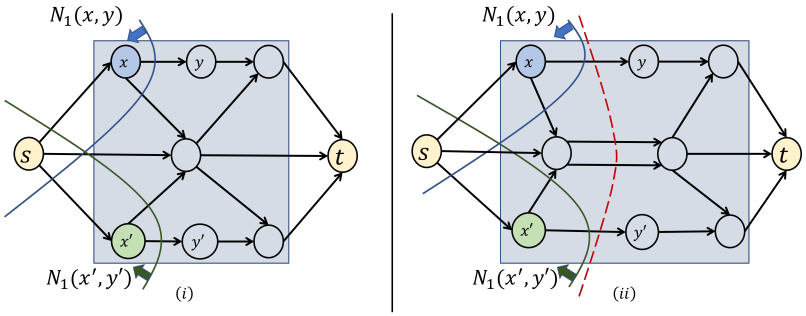


Figure 5 Graph (i) has no $\lambda+1$ (s,t)-cut containing edges (x,y) and (x',y'), but Graph (ii) does have one such cut shown by dashed curve.

Consider the data structure $\mathcal{N}_{\lambda+1}$ for $\mathcal{W}$ in $G(\mathcal{W})$. For the existence of a $(\lambda+1)$ (s,t)-cut in which both e and e' are contributing, observe that a necessary condition is that $y \notin N_1(x',y')$ and $y' \notin N_1(x,y)$. These conditions can be verified using $\mathcal{N}_{\lambda+1}$ in $\mathcal{O}(1)$ time. Upon checking these conditions, a natural approach would be to explore whether the union of these two cuts is a $(\lambda+1)$ (s,t)-cut. Unfortunately, we can not infer anything conclusively from the union of these cuts as illustrated by the two graphs in Figure 5. In both these graphs, $N_1(x,y) \cup N_1(x',y')$ is not a $(\lambda+1)$ (s,t)-cut . However, for graph (i), no $(\lambda+1)$ (s,t)-cut exists that contains the two edges; for graph (ii), there is still a $(\lambda+1)$ (s,t)-cut containing the two edges. Note that in these graphs, $N_1(x,y)$ and $N_1(x',y')$ are pairs of $(\lambda+1)$ (s,t)-cuts from Type-2. Looking at this hurdle carefully, we get the following insight. Since y, y' both lie outside $N_1(x',y') \cup N_1(x,y)$, hence $c(N_1(x,y) \cup N_1(x',y'))$ has to be $> \lambda$. Therefore, if $c(N_1(x,y) \cap N_1(x',y'))$ is also $> \lambda$, then using sub-modularity of cuts (Lemma 10), their union is bound to be a $(\lambda+1)$ (s,t)-cut and this will serve our purpose. Unfortunately, $G(\mathcal{W})$ does not ensure that $c(N_1(x,y) \cap N_1(x',y'))$ is greater than λ as shown in Figure 5.

In order to materialize the above insight, we use covering technique (Theorem 20) to build the pair of graphs $\{G(\mathcal{W})^I, G(\mathcal{W})^U\}$ that partition all $(\lambda+1)$ (s,t)-cuts of $G(\mathcal{W})$. It follows from Theorem 20(2) that the capacity of the intersection (likewise union) of each pair of $(\lambda+1)$ (s,t)-cut in $G(\mathcal{W})^I$ (likewise $G(\mathcal{W})^U$) is greater than λ. This is because the only (s,t)-mincut of $G(\mathcal{W})^I$ is $\overline{T'}$ and the only (s,t)-mincut of $G(\mathcal{W})^U$ is S'. Theorem 20(1) states that $\{G(\mathcal{W})^I, G(\mathcal{W})^U\}$ covers all the $(\lambda+1)$ (s,t)-cuts of $G(\mathcal{W})$. Therefore, if $y' \notin N_1(x,y)$ and $y \notin N_1(x',y')$ in $G(\mathcal{W})^I$ or $x \notin N_1(y',x')$ and $x' \notin N_1(y,x))$ in $(G(\mathcal{W})^U)^{\mathcal{T}}$ then there is a $(\lambda+1)$ (s,t)-cut in $G(\mathcal{W})$ in which e, e' are contributing. Now we shall establish the converse of this assertion.

Suppose there exists a $(\lambda+1)$ (s,t)-cut C in $G(\mathcal{W})$ to which both edges (x,y) and (x',y') are contributing. Without loss of generality assume that C is present in $G(\mathcal{W})^I$. It can be observed that the cuts $N_1(x,y)$ and $N_1(x',y')$ in $G(\mathcal{W})^I$ are subsets of C. Hence $y \notin N_1(x',y')$ and $y' \notin N_1(x,y)$ in $G(\mathcal{W})^I$. If C is present in $G(\mathcal{W})^U$, exactly the same analysis can be carried out on $(G(\mathcal{W})^U)^{\mathcal{T}}$. So we can state the following lemma.

▶ **Lemma 36.** *A pair of edges $e=(x,y)$, $e'=(x',y')$ from $(\lambda+1)$ class $\mathcal{W}$ are outgoing edges of a $(\lambda+1)$ (s,t)-cut in $G(\mathcal{W})$ if and only if (i) $y \notin N_1(x',y')$ and $y' \notin N_1(x,y)$ in $G(\mathcal{W})^I$ or (ii) $x \notin N_1(y',x')$ and $x' \notin N_1(y,x)$ in $(G(\mathcal{W})^U)^{\mathcal{T}}$.*

Using the data structure $\mathcal{N}_{\lambda+1}$, it requires a constant number of BELONG queries to verify the conditions mentioned in Lemma 36. Therefore, the data structure $\mathcal{F}$ for dual edge failure is as follows.

$\mathcal{F}$ consists of the following data structures:
- $\mathcal{N}_\lambda$ for graph G.
- $\mathcal{N}_{\lambda+1}$ for $G(\mathcal{W})^I$ and $(G(\mathcal{W})^U)^\mathcal{T}$ for each $(\lambda+1)$ (s,t)-class $\mathcal{W}$.

It can be observed that the resulting (s,t)-mincut value can be reported in $\mathcal{O}(1)$ time upon failure of any pair of edges from G using $\mathcal{F}$. Handling of dual edge insertion is covered in the full version of this paper. We summarize the results of this section in the following theorem.

▶ **Theorem 37.** *A directed multi-graph $G=(V,E)$, on $|V|=n$ vertices and $|E|=m$ edges with a designated source vertex s and a designated sink vertex t, can be preprocessed for constructing $\mathcal{O}(n^2)$ space Oracle $\{\mathcal{F},\mathcal{I}\}$ that takes $\mathcal{O}(1)$ time to report the value of resultant (s,t)-mincut upon*
1. *failure of any given pair of edges $(x,y),(x',y')\in E$ using $\mathcal{F}$, or*
2. *insertion of any given pair of edges $(x,y),(x',y')\in V\times V$ using $\mathcal{I}$.*

▶ Remark 38. For a $(\lambda+1)$ (s,t)-class $\mathcal{W}$ and any pair of disjoint subsets $A,B\subset\mathcal{W}$, $\mathcal{F}$ can determine in $\mathcal{O}(|A||B|)$ time if there exists a $(\lambda+1)$ (s,t)-cut C such that $A\subseteq C$ and $B\subseteq\overline{C}$. If C exists, then it is possible to report C using $\mathcal{F}$ in $\mathcal{O}((|A|+|B|)|\mathcal{W}|+|\overline{C}|)$ time.

8 Conditional lower bound for dual edge sensitivity for (s,t)-mincuts

The problem of reachability in directed graph is as follows – Given a simple directed graph G with n vertices and m edges, preprocess it to form a data structure which can efficiently report if a given vertex v is reachable from another vertex u. The reachability in G is same as reachability in G_{SCC}, a directed acyclic graph which can be obtained by contracting each of the Strongly Connected Components to a single vertex. Henceforth, we shall assume that G is a directed acyclic graph. We transform the directed acyclic graph G into a graph $\mathcal{D}$ as follows.

Construction of $\mathcal{D}$: Create two additional vertices, namely s and t. Suppose Δ_v denotes the difference in the number of incoming and outgoing edges of any vertex v of G. For each vertex v in G, if $\Delta_v>0$ we add Δ_v edges from v to t. Likewise, if $\Delta_v<0$ we add Δ_v edges from s to v. Lastly, add two additional edge(s) from s to v and v to t for all v in G. Observe that the number of edges in this graph is only $\mathcal{O}(m)$. Thus, we state the following lemma.

▶ **Lemma 39.** *For a directed graph G with n vertices and m edges, there exists a directed acyclic multi-graph $\mathcal{D}$ with $\mathcal{O}(m)$ edges and $n+2$ vertices such that a vertex v is reachable from a vertex u in G if and only if vertex v is reachable from vertex u in $\mathcal{D}$.*

The graph $\mathcal{D}$, that we have constructed, has a very interesting property. $\mathcal{D}$ is identical to the DAG $\mathcal{D}_\lambda$ that stores all (s,t)-mincuts in graph $\mathcal{D}$. We crucially exploit this property to derive an equivalence between reachability queries in graph $\mathcal{D}$ and dual edge failure (or insertion) query for (s,t)-mincut in $\mathcal{D}$. Since, the reachability structure of $\mathcal{D}$ and G is identical, we state the following lemma.

▶ **Lemma 40.** *Let G be a directed graph. A vertex v is reachable from a vertex u in G if and only if the value of (s,t)-mincut reduces by exactly 1 on removal of the edges $\{(s,u),(v,t)\}$ from graph $\mathcal{D}$ which is obtained from G using Lemma 39.*

Proof sketch. We know that $\mathcal{D}$ is same as $\mathcal{D}_\lambda$ of $\mathcal{D}$. So, removal of any edge from $\mathcal{D}$ reduces value of (s,t)-mincut by 1. Any (s,t)-cut in which both edges, $\{(s,u),(v,t)\}$, are contributing cannot be 1-transversal because of the u to v path. Every (s,t)-mincut is 1-transversal (Lemma 22), therefore, there cannot exist (s,t)-mincut in which both edges are contributing.

If there is no path from u to v, then both edges contribute to (s,t)-cut $C = \overline{R(\{u\}) \cup T}$; where $R(\{u\})$ defines the set of vertices reachable from u. We can show that C is also an (s,t)-mincut. Thus, upon removal of edges $\{(s,u),(v,t)\}$ the value of (s,t)-mincut reduces by 2. ◀

Using Conjecture 5 and Lemma 40 we state the following conditional lower bound.

▶ **Theorem 41.** *Assuming Directed Reachability Hypothesis holds, any data structure that can report the value of (s,t)-mincut for a designated source s and a designated sink t upon failure or addition of any pair of edges in a directed multi-graph with n vertices and m edges must either use $\tilde{\Omega}(n^2)$ space, or $\tilde{\Omega}(m)$ time.*

References

1. Surender Baswana and Abhyuday Pandey. Sensitivity oracles for all-pairs mincuts. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 581–609. SIAM, 2022. doi:10.1137/1.9781611977073.27.
2. András A. Benczúr. A representation of cuts within 6/5 times the edge connectivity with applications. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 92–102. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492466.
3. Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 130:1–130:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.130.
4. Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008. doi:10.1137/S0097539705429847.
5. Efim A Dinitz, Alexander V Karzanov, and Michael V Lomonosov. On the structure of the system of minimum edge cuts in a graph. *Issledovaniya po Diskretnoi Optimizatsii*, pages 290–306, 1976.
6. Yefim Dinitz. Maintaining the 4-edge-connected components of a graph on-line. In *Second Israel Symposium on Theory of Computing Systems, ISTCS 1993, Natanya, Israel, June 7-9, 1993, Proceedings*, pages 88–97. IEEE Computer Society, 1993. doi:10.1109/ISTCS.1993.253480.
7. Yefim Dinitz and Zeev Nutov. A 2-level cactus model for the system of minimum and minimum+1 edge-cuts in a graph and its incremental maintenance. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 509–518. ACM, 1995. doi:10.1145/225058.225268.
8. Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM J. Comput.*, 30(3):753–808, 2000. doi:10.1137/S0097539797330045.
9. Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 506–515. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496826.
10. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.
11. Harold N. Gabow. Efficient splitting off algorithms for graphs. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 696–705. ACM, 1994. doi:10.1145/195058.195436.

12 Zvi Galil and Giuseppe F. Italiano. Maintaining the 3-edge-connected components of a graph on-line. *SIAM J. Comput.*, 22(1):11–28, 1993. doi:10.1137/0222002.

13 Naveen Garg and Vijay V. Vazirani. A polyhedron with all$s$ - t cuts as vertices, and adjacency of cuts. *Math. Program.*, 70:17–25, 1995. doi:10.1007/BF01585926.

14 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2017. doi:10.1007/978-3-319-62127-2_36.

15 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. doi:10.1145/3274663.

16 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.52.

17 Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979. doi:10.1145/357062.357071.

18 Jason Li. Deterministic mincut in almost-linear time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 384–395. ACM, 2021. doi:10.1145/3406325.3451114.

19 Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 85–92. IEEE, 2020. doi:10.1109/FOCS46700.2020.00017.

20 Merav Parter. Dual failure resilient BFS structure. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490. ACM, 2015. doi:10.1145/2767386.2767408.

21 Merav Parter and David Peleg. Sparse fault-tolerant BFS structures. *ACM Trans. Algorithms*, 13(1):11:1–11:24, 2016. doi:10.1145/2976741.

22 Mihai Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011. doi:10.1137/09075336X.

23 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *In Rayward-Smith V.J. (eds) Combinatorial Optimization II. Mathematical Programming Studies*, 13(1):8–16, 1980. doi:10.1007/BFb0120902.

24 Vijay V. Vazirani and Mihalis Yannakakis. Suboptimal cuts: Their enumeration, weight and number (extended abstract). In Werner Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *Lecture Notes in Computer Science*, pages 366–377. Springer, 1992. doi:10.1007/3-540-55719-9_88.

Counting and Enumerating Optimum Cut Sets for Hypergraph k-Partitioning Problems for Fixed k

Calvin Beideman
University of Illinois Urbana-Champaign, IL, USA

Karthekeyan Chandrasekaran
University of Illinois Urbana-Champaign, IL, USA

Weihang Wang
University of Illinois Urbana-Champaign, IL, USA

Abstract

We consider the problem of enumerating optimal solutions for two hypergraph k-partitioning problems – namely, Hypergraph-k-Cut and Minmax-Hypergraph-k-Partition. The input in hypergraph k-partitioning problems is a hypergraph $G = (V, E)$ with positive hyperedge costs along with a fixed positive integer k. The goal is to find a partition of V into k non-empty parts $(V_1, V_2, \ldots, V_k)$ – known as a k-partition – so as to minimize an objective of interest.

1. If the objective of interest is the maximum cut value of the parts, then the problem is known as Minmax-Hypergraph-k-Partition. A subset of hyperedges is a minmax-k-cut-set if it is the subset of hyperedges crossing an optimum k-partition for Minmax-Hypergraph-k-Partition.
2. If the objective of interest is the total cost of hyperedges crossing the k-partition, then the problem is known as Hypergraph-k-Cut. A subset of hyperedges is a min-k-cut-set if it is the subset of hyperedges crossing an optimum k-partition for Hypergraph-k-Cut.

We give the first polynomial bound on the number of minmax-k-cut-sets and a polynomial-time algorithm to enumerate all of them in hypergraphs for every fixed k. Our technique is strong enough to also enable an $n^{O(k)}p$-time deterministic algorithm to enumerate all min-k-cut-sets in hypergraphs, thus improving on the previously known $n^{O(k^2)}p$-time deterministic algorithm, where n is the number of vertices and p is the size of the hypergraph. The correctness analysis of our enumeration approach relies on a structural result that is a strong and unifying generalization of known structural results for Hypergraph-k-Cut and Minmax-Hypergraph-k-Partition. We believe that our structural result is likely to be of independent interest in the theory of hypergraphs (and graphs).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases hypergraphs, k-partitioning, counting, enumeration

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.16

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2204.09178` [7]

Funding *Calvin Beideman*: Supported in part by NSF grants CCF-1814613 and CCF-1907937.
Karthekeyan Chandrasekaran: Supported in part by NSF grants CCF-1814613 and CCF-1907937.
Weihang Wang: Supported in part by NSF grants CCF-1814613 and CCF-1907937.

1 Introduction

In hypergraph k-partitioning problems, the input consists of a hypergraph $G = (V, E)$ with positive hyperedge-costs $c : E \to \mathbb{R}_+$ and a fixed positive integer k (e.g., $k = 2, 3, 4, \ldots$). The goal is to find a partition of the vertex set into k *non-empty* parts $V_1, V_2, \ldots, V_k$ so as to minimize an objective of interest. There are several natural objectives of interest in hypergraph k-partitioning problems. In this work, we focus on two particular objectives: Minmax-Hypergraph-k-Partition and Hypergraph-k-Cut:

1. In Minmax-Hypergraph-k-Partition, the objective is to minimize the maximum cut value of the parts of the k-partition – i.e., minimize $\max_{i=1}^{k} c(\delta(V_i))$; here $\delta(V_i)$ is the set of hyperedges intersecting both V_i and $V \setminus V_i$ and $c(\delta(V_i)) = \sum_{e \in \delta(V_i)} c(e)$ is the total cost of hyperedges in $\delta(V_i)$.
2. In Hypergraph-k-Cut[1], the objective is to minimize the cost of hyperedges crossing the k-partition – i.e., minimize $c(\delta(V_1, \ldots, V_k))$; here $\delta(V_1, \ldots, V_k)$ is the set of hyperedges that intersect at least two sets in $\{V_1, \ldots, V_k\}$ and $c(\delta(V_1, \ldots, V_k)) = \sum_{e \in \delta(V_1, \ldots, V_k)} c(e)$ is the total cost of hyperedges in $\delta(V_1, \ldots, V_k)$.

If the input G is a graph, then we will refer to these problems as Minmax-Graph-k-Partition and Graph-k-Cut respectively. We note that the case of $k = 2$ corresponds to global minimum cut in both objectives. In this work, we focus on the problem of enumerating all optimum solutions to Minmax-Hypergraph-k-Partition and Hypergraph-k-Cut.

Motivations and Related Problems. We consider the problem of counting and enumerating optimum solutions for partitioning problems over hypergraphs for three reasons. Firstly, hyperedges provide more powerful modeling capabilities than edges and consequently, several problems in hypergraphs become non-trivial in comparison to graphs. Although hypergraphs and partitioning problems over hypergraphs (including Minmax-Hypergraph-k-Partition) were discussed as early as 1973 by Lawler [33], most of these problems still remain open. The powerful modeling capability of hyperedges has been useful in a variety of modern applications, which in turn, has led to a resurgence in the study of hypergraphs with recent works focusing on min-cuts, cut-sparsifiers, spectral-sparsifiers, etc. [6, 8, 12, 15, 17, 18, 20, 21, 28, 32, 38]. Our work adds to this rich and emerging theory of hypergraphs.

Secondly, hypergraph k-partitioning problems are special cases of submodular k-partitioning problems. In submodular k-partitioning problems, the input is a finite ground set V, a submodular function[2] $f : 2^V \to \mathbb{R}$ provided by an evaluation oracle[3] and a positive integer k (e.g., $k = 2, 3, 4, \ldots$). The goal is to partition the ground set V into k non-empty parts $V_1, V_2, \ldots, V_k$ so as to minimize an objective of interest. Two natural objectives are of interest: (1) In Minmax-Submod-k-Partition, the objective is to minimize $\max_{i=1}^{k} f(V_i)$ and (2) In Minsum-Submod-k-Partition, the objective is to minimize $\sum_{i=1}^{k} f(V_i)$. If the given submodular function is symmetric[4], then we denote the resulting problems as Minmax-SymSubmod-k-Partition and Minsum-SymSubmod-k-Partition respectively. Since the hypergraph cut function is symmetric submodular, it follows that Minmax-Hypergraph-k-Partition is a special case of Minmax-SymSubmod-k-Partition. Moreover, Hypergraph-k-Cut is a special case of Minsum-Submod-k-Partition (this reduction is slightly non-trivial with the submodular function in the reduction being asymmetric – e.g., see [36] for the reduction). Queyranne claimed, in 1999, a polynomial-time algorithm for Minsum-SymSubmod-k-Partition for every fixed k [37], however the claim was retracted subsequently (see [24]). The complexity status of submodular k-partitioning problems (for fixed $k \geq 4$) are open, so recent works have focused on hypergraph k-partitioning problems as a stepping stone towards submodular k-partitioning [8, 12, 13, 24, 36, 41, 42]. Our work contributes to this stepping stone by advancing the state of the art in hypergraph k-partitioning problems. We emphasize that the

[1] We emphasize that the objective of Hypergraph-k-Cut is not equivalent to minimizing $\sum_{i=1}^{k} c(\delta(V_i))$.
[2] A real-valued set function $f : 2^V \to \mathbb{R}$ is submodular if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)\ \forall\ A, B \subseteq V$.
[3] An evaluation oracle for a set function f over a ground set V returns the value of $f(S)$ given $S \subseteq V$.
[4] A real-valued set function $f : 2^V \to \mathbb{R}$ is symmetric if $f(A) = f(V \setminus A)\ \forall\ A \subseteq V$.

complexity status of two other variants of hypergraph k-partitioning problems which are also special cases of Minsum-Submod-k-Partition are still open (see [36, 41, 42] for these variants).

Thirdly, counting and enumeration of optimum solutions for *graph* k-partitioning problems are fundamental to graph theory and extremal combinatorics. They have found farther reaching applications than initially envisioned. We discuss some of the results and applications for $k = 2$ and $k > 2$ now. For $k = 2$ in connected graphs, it is well-known that the number of min-cuts and the number of α-approximate min-cuts are at most $\binom{n}{2}$ and $O(n^{2\alpha})$ respectively, and they can all be enumerated in polynomial time for constant α. These combinatorial results have been the crucial ingredients of several algorithmic and representation results in graphs. On the algorithmic front, these results enable fast randomized construction of graph skeletons which, in turn, plays a crucial role in fast algorithms to solve graph min-cut [29]. On the representation front, counting results form the backbone of cut sparsifiers which in turn have found applications in sketching and streaming [2–4, 32]. A polygon representation of the family of 6/5-approximate min-cuts in graphs was given by Benczur and Goemans in 1997 (see [9–11]) – this representation was used in the recent groundbreaking $(3/2 - \epsilon)$-approximation for metric TSP [31]. On the approximation front, in addition to the $(3/2 - \epsilon)$-approximation for metric TSP [31], counting results also led to the recent 1.5-approximation for path TSP [40]. For $k > 2$, we note that fast algorithms for Graph-k-Cut have been of interest since they help in generating cutting planes while solving TSP [5, 19]. A recent series of works aimed towards improving the bounds on the number of optimum solutions for Graph-k-Cut culminated in a drastic improvement in the run-time to solve Graph-k-Cut [25–27]. Given the status of counting and enumeration results for k-partitioning in graphs and their algorithmic and representation implications that were discovered subsequently, we believe that a similar understanding in hypergraphs could serve as an important ingredient in the algorithmic and representation theory of hypergraphs.

The Enumeration Problem. There is a fundamental structural distinction between hypergraphs and graphs that becomes apparent while enumerating optimum solutions to k-partitioning problems. In connected graphs, the number of optimum k-partitions for Graph-k-Cut and for Minmax-Graph-k-Partition are $n^{O(k)}$ and $n^{O(k^2)}$ respectively and they can all be enumerated in polynomial time, where n is the number of vertices in the input graph [14, 16, 25, 27, 30, 39]. In contrast, a connected hypergraph could have exponentially many optimum k-partitions for both Minmax-Hypergraph-k-Partition and Hypergraph-k-Cut even for $k = 2$ – e.g., consider the hypergraph with a single hyperedge containing all vertices; we will denote this as the spanning-hyperedge-example. Hence, enumerating all optimum k-partitions for hypergraph k-partitioning problems in polynomial time is impossible. Instead, our goal in the enumeration problems is to enumerate *k-cut-sets* corresponding to optimum k-partitions. We will call a subset $F \subseteq E$ of hyperedges to be a k-cut-set if there exists a k-partition $(V_1, \ldots, V_k)$ such that $F = \delta(V_1, \ldots, V_k)$; we will call a 2-cut-set as a cut-set. In the enumeration problems that we will consider, the input consists of a hypergraph $G = (V, E)$ with positive hyperedge-costs $c : E \to \mathbb{R}_+$ and a fixed positive integer k (e.g., $k = 2, 3, 4, \ldots$).

1. For an optimum k-partition $(V_1, \ldots, V_k)$ for Minmax-Hypergraph-k-Partition in (G, c), we will denote $\delta(V_1, \ldots, V_k)$ as a minmax-k-cut-set. In Enum-MinMax-Hypergraph-k-Partition, the goal is to enumerate all minmax-k-cut-sets.
2. For an optimum k-partition $(V_1, \ldots, V_k)$ for Hypergraph-k-Cut in (G, c), we will denote $\delta(V_1, \ldots, V_k)$ as a min-k-cut-set. In Enum-Hypergraph-k-Cut, the goal is to enumerate all min-k-cut-sets.

We observe that in the spanning-hyperedge-example, although the number of optimum k-partitions for MINMAX-HYPERGRAPH-k-PARTITION (as well as HYPERGRAPH-k-CUT) is exponential, the number of MINMAX-k-CUT-SETs (as well as MIN-k-CUT-SETs) is only one.

1.1 Results

In contrast to graphs, whose representation size is the number of edges, the representation size of a hypergraph $G = (V, E)$ is $p := \sum_{e \in E} |e|$. Throughout, our algorithmic discussion will focus on the case of fixed k (e.g., $k = 2, 3, 4, \ldots$).

There are no prior results regarding ENUM-MINMAX-HYPERGRAPH-k-PARTITION in the literature. We recall the status of MINMAX-HYPERGRAPH-k-PARTITION. As mentioned earlier, MINMAX-HYPERGRAPH-k-PARTITION was discussed as early as 1973 by Lawler [33] with its complexity status being open until recently. We note that the objective here could be viewed as aiming to find a *fair* k-partition, i.e., a k-partition where no part pays too much in cut value. Motivated by this connection to fairness, Chandrasekaran and Chekuri (2021) [13] studied the more general problem of MINMAX-SYMSUBMOD-k-PARTITION. They gave the first (deterministic) polynomial-time algorithm to solve MINMAX-SYMSUBMOD-k-PARTITION and as a consequence, obtained the first polynomial-time algorithm to solve MINMAX-HYPERGRAPH-k-PARTITION. Their algorithm does not show any bound on the number of MINMAX-k-CUT-SETs since it solves the more general problem of MINMAX-SYMSUBMOD-k-PARTITION for which the number of optimum k-partitions can indeed be exponential (recall the spanning-hyperedge-example). Focusing on hypergraphs raises the question of whether all k-cut-sets corresponding to optimum solutions can be enumerated efficiently for every fixed k. We answer this question affirmatively by giving the first polynomial-time algorithm for ENUM-MINMAX-HYPERGRAPH-k-PARTITION.

▶ **Theorem 1.** *There exists a deterministic algorithm to solve ENUM-MINMAX-HYPERGRAPH-k-PARTITION that runs in time $O(kn^{4k^2-2k+1}p)$, where n is the number of vertices and p is the size of the input hypergraph. Moreover, the number of MINMAX-k-CUT-SETs in a n-vertex hypergraph is $O(n^{4k^2-2k})$.*

We emphasize that our result shows the first polynomial bound on the number of MINMAX-k-CUT-SETs in hypergraphs for every fixed k (in addition to a polynomial-time algorithm to enumerate all of them for every fixed k). Our upper bound of $n^{O(k^2)}$ on the number of MINMAX-k-CUT-SETs is tight – there exist n-vertex connected *graphs* for which the number of MINMAX-k-CUT-SETs is $n^{\Theta(k^2)}$.

Next, we briefly recall the status of HYPERGRAPH-k-CUT and ENUM-HYPERGRAPH-k-CUT. HYPERGRAPH-k-CUT was shown to be solvable in randomized polynomial time only recently [15, 20]; the randomized algorithms also showed that the number of MIN-k-CUT-SETs is $O(n^{2k-2})$ and they can all be enumerated in randomized polynomial time. A subsequent deterministic algorithm was designed to solve HYPERGRAPH-k-CUT in time $n^{O(k)}p$ by Chandrasekaran and Chekuri [12]. Chandrasekaran and Chekuri's techniques were extended to design the first deterministic polynomial-time algorithm to solve ENUM-HYPERGRAPH-k-CUT in [8]. The algorithm for ENUM-HYPERGRAPH-k-CUT given in [8] runs in time $n^{O(k^2)}p$. We note that this run-time has a quadratic dependence on k in the exponent of n although the number of MIN-k-CUT-SETs has only linear dependence on k in the exponent of n (since it is $O(n^{2k-2})$). So, an open question that remained from [8] is whether one can obtain an $n^{O(k)}p$-time deterministic algorithm for ENUM-HYPERGRAPH-k-CUT. We resolve this question affirmatively.

▶ **Theorem 2.** *There exists a deterministic algorithm to solve* Enum-Hypergraph-k-Cut *that runs in time* $O(n^{16k-25}p)$*, where* n *is the number of vertices and* p *is the size of the input hypergraph.*

Our algorithms for both Enum-MinMax-Hypergraph-k-Partition and Enum-Hypergraph-k-Cut are based on a structural theorem that allows for efficient recovery of optimum k-cut-sets via minimum (s,t)-terminal cuts (see Theorem 4). Our structural theorem builds on structural theorems that have appeared in previous works on Minmax-Hypergraph-k-Partition and Hypergraph-k-Cut [8, 12, 13]. Our structural theorem may appear to be natural/incremental in comparison to ones that appeared in previous works, but formalizing the theorem and proving it is a significant part of our contribution. Moreover, our single structural theorem is strong enough to enable efficient algorithms for both Enum-Hypergraph-k-Cut as well as Enum-MinMax-Hypergraph-k-Partition in contrast to previously known structural theorems. In this sense, our structural theorem can be viewed as a strong and unifying generalization of structural theorems that have appeared in previous works. We believe that our structural theorem will be of independent interest in the theory of cuts and partitioning in hypergraphs (as well as graphs).

1.2 Technical overview and main structural result

We focus on the unit-cost variant of Enum-Hypergraph-k-Cut and Enum-MinMax-Hypergraph-k-Partition in the rest of this work for the sake of notational simplicity – i.e., the cost of every hyperedge is 1. Throughout, we will allow multigraphs and hence, this is without loss of generality. Our algorithms extend in a straightforward manner to arbitrary hyperedge costs. They rely only on minimum (s,t)-terminal cut computations and hence, they are strongly polynomial-time algorithms.

Notation and background. Let $G = (V, E)$ be a hypergraph. Throughout this work, n will denote the number of vertices in G, m will denote the number of hyperedges in G, and $p := \sum_{e \in E} |e|$ will denote the representation size of G. We will denote a partition of the vertex set into h non-empty parts by an ordered tuple $(V_1, \ldots, V_h)$ and call such an ordered tuple as an h-partition. For a partition $\mathcal{P} = (V_1, V_2, \ldots, V_h)$, we will say that a hyperedge e crosses the partition $\mathcal{P}$ if it intersects at least two parts of the partition. We will refer to a 2-partition as a cut. For a non-empty proper subset U of vertices, we will use $\overline{U}$ to denote $V \setminus U$, $\delta(U)$ to denote the set of hyperedges crossing the cut $(U, \overline{U})$, and $d(U) := |\delta(U)|$ to denote the cut value of U. We observe that $\delta(U) = \delta(\overline{U})$, so we will use $d(U)$ to denote the value of the cut $(U, \overline{U})$. More generally, given a partition $\mathcal{P} = (V_1, V_2, \ldots, V_h)$, we denote the set of hyperedges crossing the partition by $\delta(V_1, V_2, \ldots, V_h)$ (also by $\delta(\mathcal{P})$ for brevity) and the number of hyperedges crossing the partition by $|\delta(V_1, V_2, \ldots, V_h)|$. We will denote the optimum value of Minmax-Hypergraph-k-Partition and Hypergraph-k-Cut respectively by

$$OPT_{\text{minmax-}k\text{-partition}} := \min \left\{ \max_{i \in [k]} |\delta(V_i)| : (V_1, \ldots, V_k) \text{ is a } k\text{-partition of } V \right\} \text{ and}$$

$$OPT_{k\text{-cut}} := \min \left\{ |\delta(V_1, \ldots, V_k)| : (V_1, \ldots, V_k) \text{ is a } k\text{-partition of } V \right\}.$$

A key algorithmic tool will be the use of fixed-terminal cuts. Let S, T be disjoint non-empty subsets of vertices. A 2-partition $(U, \overline{U})$ is an (S,T)-terminal cut if $S \subseteq U \subseteq V \setminus T$. Here, the set U is known as the source set and the set $\overline{U}$ is known as the sink set. A minimum-valued (S,T)-terminal cut is known as a *minimum* (S,T)*-terminal cut*. Since there

could be multiple minimum (S,T)-terminal cuts, we will be interested in *source minimal* minimum (S,T)-terminal cuts. For every pair of disjoint non-empty subsets S and T of vertices, there exists a unique source minimal minimum (S,T)-terminal cut and it can be found in deterministic polynomial time via standard maxflow algorithms. In particular, the source minimal minimum (S,T)-terminal cut can be found in time $O(np)$ [17].

Our technique to enumerate all MINMAX-k-CUT-SETs and all MIN-k-CUT-SETs will build on the approaches of Chandrasekaran and Chekuri for HYPERGRAPH-k-CUT and MINMAX-SYMSUBMOD-k-PARTITION [8, 12, 13]. We need the following structural theorem that was shown in [8].

▶ **Theorem 3** ([8]). *Let $G=(V,E)$ be a hypergraph and let $OPT_{k\text{-}cut}$ be the optimum value of HYPERGRAPH-k-CUT in G for some integer $k \geq 2$. Suppose $(U,\overline{U})$ is a 2-partition of V with $d(U) < OPT_{k\text{-}cut}$. Then, for every pair of vertices $s \in U$ and $t \in \overline{U}$, there exist subsets $S \subseteq U \setminus \{s\}$ and $T \subseteq \overline{U} \setminus \{t\}$ with $|S| \leq 2k-3$ and $|T| \leq 2k-3$ such that $(U,\overline{U})$ is the unique minimum $(S \cup \{s\}, T \cup \{t\})$-terminal cut in G.*

Enum-Hypergraph-k-Cut. We first focus on ENUM-HYPERGRAPH-k-CUT. We note that Theorem 3 will allow us to recover those parts V_i of an optimum k-partition $(V_1,\ldots,V_k)$ for which $d(V_i) < OPT_{k\text{-cut}}$. However, recall that our goal is *not* to recover all optimum k-partitions for HYPERGRAPH-k-CUT, but rather to recover all MIN-k-CUT-SETs (i.e., not to recover the parts of every optimum k-partition, but rather only to recover the k-cut-set of every optimum k-partition). The previous work [8] that designed an $n^{O(k^2)}p$-time deterministic enumeration algorithm achieved this by proving the following structural result: suppose $(V_1,\ldots,V_k)$ is an optimum k-partition for HYPERGRAPH-k-CUT for which $d(V_1) = OPT_{k\text{-cut}}$. Then, they showed that for every subset $T \subseteq \overline{V_1}$ satisfying $T \cap V_j \neq \emptyset$ for all $j \in \{2,\ldots,k\}$, there exists a subset $S \subseteq V_1$ with $|S| \leq 2k$ such that the source minimal minimum (S,T)-terminal cut $(A,\overline{A})$ satisfies $\delta(A) = \delta(V_1)$. This structural theorem in conjunction with Theorem 3 allows one to enumerate a candidate family $\mathcal{F}$ of $n^{O(k^2)}$ subsets of hyperedges such that every MIN-k-CUT-SET is present in the family. The drawback of their structural theorem is that it is driven towards recovering the cut-set $\delta(V_i)$ of every part V_i of every optimum k-partition $(V_1,\ldots,V_k)$. Hence, their algorithmic approach ends up with a run-time of $n^{O(k^2)}p$. In order to improve the run-time, we prove a stronger result: we show that for an *arbitrary* cut $(U,\overline{U})$ with cut value $OPT_{k\text{-cut}}$ (as opposed to only those sets V_i of an optimum k-partition $(V_1,\ldots,V_k)$), its cut-set $\delta(U)$ can be recovered as the cut-set of *any* minimum (S,T)-terminal cut for some S and T of small size. The following is the main structural theorem of this work.

▶ **Theorem 4.** *Let $G=(V,E)$ be a hypergraph and let $OPT_{k\text{-}cut}$ be the optimum value of HYPERGRAPH-k-CUT in G for some integer $k \geq 2$. Suppose $(U,\overline{U})$ is a 2-partition of V with $d(U) = OPT_{k\text{-}cut}$. Then, there exist sets $S \subseteq U$, $T \subseteq \overline{U}$ with $|S| \leq 2k-1$ and $|T| \leq 2k-1$ such that every minimum (S,T)-terminal cut $(A,\overline{A})$ satisfies $\delta(A) = \delta(U)$.*

We encourage the reader to compare and contrast Theorems 3 and 4. The former helps to recover cuts whose cut value is strictly smaller than $OPT_{k\text{-cut}}$ while the latter helps to recover *cut-sets* whose size is equal to $OPT_{k\text{-cut}}$. So, the latter theorem is weaker since it only recovers cut-sets, but we emphasize that this is the best possible that one can hope to do (as seen from the spanning-hyperedge-example). However, proving the latter theorem requires us to work with cut-sets (as opposed to cuts) which is a technical barrier to overcome. Indeed, our proof of Theorem 4 deviates significantly from the proof of Theorem 3 since we have to work with cut-sets. Our proof also deviates from the structural result in [8] that was mentioned in the paragraph above Theorem 4 since our result is stronger than

their result – our result helps to recover the cut-set $\delta(U)$ of an arbitrary cut $(U, \overline{U})$ whose cut value is $d(U) = OPT_{k\text{-cut}}$ while their result helps only to recover the cut-set $\delta(V_i)$ of a part V_i of an optimum k-partition $(V_1, \ldots, V_k)$ for Hypergraph-k-Cut whose cut value is $d(V_i) = OPT_{k\text{-cut}}$; moreover, their proof technique crucially relies on a containment property with respect to the part V_i, whereas under the hypothesis of our structural theorem, the containment property fails with respect to the set U and consequently, our proof technique differs from theirs.

Theorems 3 and 4 lead to a deterministic $n^{O(k)}$-time algorithm to enumerate all min-k-cut-sets via a divide-and-conquer approach. We describe this algorithm now: For each pair (S, T) of disjoint subsets of vertices S and T with $|S|, |T| \leq 2k-1$, compute the source minimal minimum (S, T)-terminal cut $(A, \overline{A})$; (i) if $G - \delta(A)$ has at least k connected components, then add $\delta(A)$ to the candidate family $\mathcal{F}$; (ii) otherwise, add the set A to a collection $\mathcal{C}$. We note that the sizes of the family $\mathcal{F}$ and the collection $\mathcal{C}$ are $O(n^{4k-2})$. Next, for each subset A in the collection $\mathcal{C}$, recursively enumerate all min-$k/2$-cut-sets in the subhypergraphs induced by A and $\overline{A}$ respectively[5] – denoted $G[A]$ and $G[\overline{A}]$ respectively – and add $\delta(A) \cup F_1 \cup F_2$ to the family $\mathcal{F}$ for each F_1 and F_2 being min-$k/2$-cut-set in $G[A]$ and $G[\overline{A}]$ respectively. Finally, return the subfamily of k-cut-sets from the family $\mathcal{F}$ that are of smallest size.

We sketch the correctness analysis of the above approach: let $F = \delta(V_1, \ldots, V_k)$ be a min-k-cut-set with $(V_1, \ldots, V_k)$ being an optimum k-partition for Hypergraph-k-Cut. We will show that the family $\mathcal{F}$ contains F. Let $U := \cup_{i=1}^{k/2} V_i$. We note that $\delta(U) \subseteq F$. We have two possibilities: (1) Say $d(U) = |F|$. Then, $d(U) = OPT_{k\text{-cut}}$. Consequently, by Theorem 4, the min-k-cut-set F will be added to the family $\mathcal{F}$ by step (i). (2) Say $d(U) < |F|$. Then, by Theorem 3, the set $U = \cup_{i=1}^{k/2} V_i$ will be added to the collection $\mathcal{C}$ by step (ii); moreover, $F_1 := F \cap E(G[U])$ and $F_2 := F \cap E(G[\overline{U}])$ are min-$k/2$-cut-sets in $G[U]$ and $G[\overline{U}]$ respectively and they would have been enumerated by recursion, and hence, the set $\delta(U) \cup F_1 \cup F_2 = F$ will be added to the family $\mathcal{F}$. The size of the family $\mathcal{F}$ can be shown to be $n^{O(k \log k)}$ and the run-time is $n^{O(k \log k)}p$. Using the known fact that the number of min-k-cut-sets in a n-vertex hypergraph is $O(n^{2k-2})$, we can improve the run-time analysis of this approach to $n^{O(k)}p$.

Enum-MinMax-Hypergraph-k-Partition. Next, we focus on Enum-MinMax-Hypergraph-k-Partition. There is a fundamental technical issue in enumerating minmax-k-cut-sets as opposed to min-k-cut-sets. We highlight this technical issue now. Suppose we find an optimum k-partition $(V_1, \ldots, V_k)$ for Minmax-Hypergraph-k-Partition (say via Chandrasekaran and Chekuri's algorithm [13]) and store only the minmax-k-cut-set $F = \delta(V_1, \ldots, V_k)$ but forget to store the partition $(V_1, \ldots, V_k)$; now, by knowing a minmax-k-cut-set F, can we recover *some* optimum k-partition for Minmax-Hypergraph-k-Partition (not necessarily $(V_1, \ldots, V_k)$)? Or by knowing a minmax-k-cut-set F, is it even possible to find the value $OPT_{\text{minmax-}k\text{-partition}}$ without solving Minmax-Hypergraph-k-Partition from scratch again – i.e., is there an advantage to knowing a minmax-k-cut-set in order to solve Minmax-Hypergraph-k-Partition? We are not aware of such an advantage. This is in stark contrast to Hypergraph-k-Cut where knowing a min-k-cut-set enables a linear-time solution to Hypergraph-k-Cut[6].

[5] Subhypergraph $G[A]$ has vertex set A and contains all hyperedges of G which are entirely contained within A.

[6] Suppose we know a min-k-cut-set F. Then consider the connected components $Q_1, \ldots, Q_t$ in $G - F$ and create a partition $(P_1, \ldots, P_k)$ by taking $P_i = Q_i$ for every $i \in [k-1]$ and $P_k = \cup_{j=k}^{t} Q_j$; such a k-partition $(P_1, \ldots, P_k)$ will be an optimum k-partition for Hypergraph-k-Cut.

Why is this issue significant while solving Enum-MinMax-Hypergraph-k-Partition? We recall that in our approach for Enum-Hypergraph-k-Cut, the algorithm computed a polynomial-sized family $\mathcal{F}$ containing all min-k-cut-sets and returned the ones with smallest size – the smallest size ones will exactly be min-k-cut-sets. It is unclear if a similar approach could work for enumerating minmax-k-cut-sets: suppose we do have an algorithm to enumerate a polynomial-sized family $\mathcal{F}$ containing all minmax-k-cut-sets; now, in order to return all minmax-k-cut-sets (which is a subfamily of $\mathcal{F}$), note that we need to identify them among the ones in the family $\mathcal{F}$ – i.e., we need to verify if a given subset $F \in \mathcal{F}$ of hyperedges is a minmax-k-cut-set; this verification problem is closely related to the question mentioned in the previous paragraph. We do not know how to address this verification problem directly. So, our algorithmic approach for Enum-MinMax-Hypergraph-k-Partition has to overcome this technical issue.

Our ingredient to overcome this technical issue is to enumerate *representatives* for minmax-k-cut-sets. For a k-partition $(V_1, \ldots, V_k)$ and disjoint subsets $U_1, \ldots, U_k \subseteq V$, we will call the k-tuple $(U_1, \ldots, U_k)$ to be a *k-cut-set representative* of $(V_1, \ldots, V_k)$ if $U_i \subseteq V_i$ and $\delta(U_i) = \delta(V_i)$ for all $i \in [k]$. We note that a fixed k-partition $(V_1, \ldots, V_k)$ could have several k-cut-set representatives and a fixed k-tuple $(U_1, \ldots, U_k)$ could be the k-cut-set representative of several k-partitions. Yet, it is possible to efficiently verify if a given k-tuple $(U_1, \ldots, U_k)$ is a k-cut-set representative. Moreover, knowing a k-cut-set representative $(U_1, \ldots, U_k)$ of a k-partition $(V_1, V_2, \ldots, V_k)$ allows one to recover the k-cut-set $F := \delta(V_1, \ldots, V_k)$ since $F = \cup_{i=1}^{k} \delta(U_i)$. Thus, in order to enumerate all minmax-k-cut-sets, it suffices to enumerate k-cut-set representatives of *all* optimum k-partitions for Minmax-Hypergraph-k-Partition. At this point, the astute reader may wonder if there exists a polynomial-sized family of k-cut-set representatives of all optimum k-partitions for Minmax-Hypergraph-k-Partition given that the number of optimum k-partitions for Minmax-Hypergraph-k-Partition could be exponential. For example, is there a polynomial-sized family of k-cut-set representatives of all optimum k-partitions for Minmax-Hypergraph-k-Partition in the spanning-hyperedge-example? Indeed, in the spanning-hyperedge-example, even though the number of optimum k-partitions for Minmax-Hypergraph-k-Partition is exponential, there exists a $(k!\binom{n}{k})$-sized family of k-cut-set representatives of all optimum k-partitions: consider the family $\{(\{v_1\}, \ldots, \{v_k\}) : v_1, \ldots, v_k \in V, v_i \neq v_j \ \forall \text{ distinct } i, j \in [k]\}$.

It turns out that Theorems 3 and 4 are strong enough to enable efficient enumeration of k-cut-set representatives of all optimum k-partitions for Minmax-Hypergraph-k-Partition. We describe the algorithm to achieve this: For each pair (S, T) of disjoint subsets of vertices with $|S|, |T| \leq 2k-1$, compute the source minimal minimum (S, T)-terminal cut $(U, \overline{U})$ and add U to a candidate collection $\mathcal{C}$. We note that the size of the collection $\mathcal{C}$ is $O(n^{4k-2})$. Next, for each k-tuple $(U_1, \ldots, U_k) \in \mathcal{C}^k$, verify if $(U_1, \ldots, U_k)$ is a k-cut-set representative and if so, then add the k-tuple to the candidate family $\mathcal{D}$. Finally, return $\arg\min\{\max_{i=1}^{k} d(U_i) : (U_1, \ldots, U_k) \in \mathcal{D}\}$, i.e., prune and return the subfamily of k-cut-set representatives $(U_1, \ldots, U_k)$ from the family $\mathcal{D}$ that have minimum $\max_{i=1}^{k} d(U_i)$.

We note that the size of the family $\mathcal{D}$ is $n^{O(k^2)}$ and consequently, the run-time is $n^{O(k^2)}p$. We sketch the correctness analysis of the above approach: let $(V_1, \ldots, V_k)$ be an optimum k-partition for Minmax-Hypergraph-k-Partition. We will show that the family $\mathcal{D}$ contains a k-cut-set representative of $(V_1, \ldots, V_k)$. By noting that $OPT_{\text{minmax-}k\text{-partition}} \leq OPT_{k\text{-cut}}$ and by Theorems 3 and 4, for every $i \in [k]$, we have a set U_i in the collection $\mathcal{C}$ with $U_i \subseteq V_i$ and $\delta(U_i) = \delta(V_i)$. Hence, the k-tuple $(U_1, \ldots, U_k) \in \mathcal{C}^k$ is a k-cut-set representative and it will be added to the family $\mathcal{D}$. The final pruning step will not remove $(U_1, \ldots, U_k)$ from the family $\mathcal{D}$ and hence, it will be in the subfamily returned by the algorithm.

Significance of our technique. As mentioned earlier, our techniques build on the structural theorems that appeared in previous works [8, 12, 13]. The main technical novelty of our contribution lies in Theorem 4 which can be viewed as the culmination of structural theorems developed in those previous works. We also emphasize that using minimum (s,t)-terminal cuts to solve global partitioning problems is not a new technique per se (e.g., minimum (s,t)-terminal cut is the first and most natural approach to solve global minimum cut). This technique of using minimum (s,t)-terminal cuts to solve global partitioning problems has a rich variety of applications in combinatorial optimization: e.g., (1) it was used to design the first efficient algorithm for Graph-k-Cut for fixed k [23], (2) it was used to design efficient algorithms for certain constrained submodular minimization problems [22, 35], and (3) more recently, it was used to design fast algorithms for global minimum cut in graphs as well as to obtain fast Gomory-Hu trees in unweighted graphs [1, 34]. The applicability of this technique relies on identifying and proving appropriate structural results. Our Theorem 4 is such a structural result. The merit of the structural result lies in its ability to solve two different enumeration problems in hypergraph k-partitioning which was not possible via structural theorems that were developed before. Moreover, it leads to the first polynomial bound on the number of minmax-k-cut-sets in hypergraphs for every fixed k.

Organization. In Section 1.3, we recall properties of the hypergraph cut function. In Section 2, we prove a special case of Theorem 4. In Section 3, we use this special case to prove Theorem 4. We refer the reader to the full version [7] for a discussion of related work, our algorithms to prove Theorems 1 and 2, and a lower bound example. We conclude with some open questions in Section 4.

1.3 Preliminaries

Let $G = (V, E)$ be a hypergraph. Throughout, we will follow the notation mentioned in the second paragraph of Section 1.2. For disjoint $A, B \subseteq V$, we define $E(A, B) := \{e \in E : e \subseteq A \cup B, e \cap A \neq \emptyset, e \cap B \neq \emptyset\}$, and $E[A] := \{e \in E : e \subseteq A\}$. We will repeatedly rely on the fact that the hypergraph cut function $d : 2^V \to \mathbb{R}_+$ is symmetric and submodular. We recall that a set function $f : 2^V \to \mathbb{R}$ is *symmetric* if $f(U) = f(\overline{U})$ for all subsets $U \subseteq V$ and is *submodular* if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all subsets $A, B \subseteq V$.

We will need the following partition uncrossing theorem that was proved in previous works on Hypergraph-k-Cut and Enum-Hypergraph-k-Cut (see Figure 1 for an illustration of the sets that appear in the statement of Theorem 5):

▶ **Theorem 5** ([8, 12]). *Let $G = (V, E)$ be a hypergraph, $k \geq 2$ be an integer and $\emptyset \neq R \subsetneq U \subsetneq V$. Let $S = \{u_1, \ldots, u_p\} \subseteq U \setminus R$ for $p \geq 2k - 2$. Let $(\overline{A_i}, A_i)$ be a minimum $((S \cup R) \setminus \{u_i\}, \overline{U})$-terminal cut. Suppose that $u_i \in A_i \setminus (\cup_{j \in [p] \setminus \{i\}} A_j)$ for every $i \in [p]$. Then, the following two hold:*

1. *There exists a k-partition $(P_1, \ldots, P_k)$ of V with $\overline{U} \subsetneq P_k$ such that*

$$|\delta(P_1, \ldots, P_k)| \leq \frac{1}{2} \min\{d(A_i) + d(A_j) : i, j \in [p], i \neq j\}.$$

2. *Moreover, if there exists a hyperedge $e \in E$ such that e intersects $W := \cup_{1 \leq i < j \leq p} (A_i \cap A_j)$, e intersects $Z := \cap_{i \in [p]} \overline{A_i}$, and e is contained in $W \cup Z$, then the inequality in the previous conclusion is strict.*

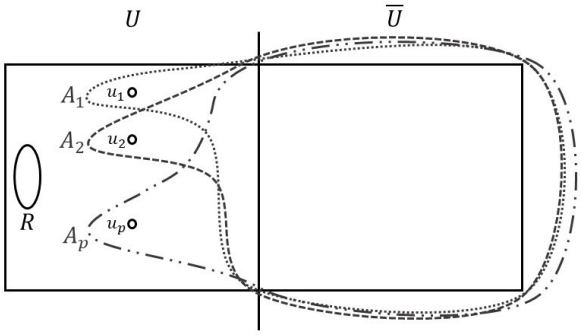


Figure 1 Illustration of the sets that appear in the statement of Theorem 5.

2 A special case of Theorem 4

The following is the main theorem of this section. Theorem 6 implies Theorem 4 in the special case where the 2-partition $(U, \overline{U})$ of interest to Theorem 4 is such that $|\overline{U}| \leq 2k-1$.

▶ **Theorem 6.** *Let $G = (V, E)$ be a hypergraph and let $OPT_{k\text{-}cut}$ be the optimum value of* Hypergraph-k-Cut *in G for some integer $k \geq 2$. Suppose $(U, \overline{U})$ is a 2-partition of V with $d(U) = OPT_{k\text{-}cut}$. Then, there exists a set $S \subseteq U$ with $|S| \leq 2k-1$ such that every minimum $(S, \overline{U})$-terminal cut $(A, \overline{A})$ satisfies $\delta(A) = \delta(U)$.*

Proof. Consider the collection

$$\mathcal{C} := \{Q \subseteq V : \ \overline{U} \subsetneq Q, \ d(Q) \leq d(U), \text{ and } \delta(Q) \neq \delta(U)\}.$$

Let S be an inclusion-wise minimal subset of U such that $S \cap Q \neq \emptyset$ for all $Q \in \mathcal{C}$, i.e., the set S is completely contained in U and is a minimal transversal of the collection $\mathcal{C}$. Proposition 7 and Lemma 8 complete the proof of Theorem 6 for this choice of S. ◀

▶ **Proposition 7.** *Every minimum $(S, \overline{U})$-terminal cut $(A, \overline{A})$ has $\delta(A) = \delta(U)$.*

Proof. Let $(A, \overline{A})$ be a minimum $(S, \overline{U})$-terminal cut. If $A = U$, then we are done, so we may assume that $A \neq U$. This implies that $S \subseteq A$ and $\overline{U} \subsetneq \overline{A}$. Since $(U, \overline{U})$ is a $(S, \overline{U})$-terminal cut, we have that $d(\overline{A}) = d(A) \leq d(U)$. Since S intersects every set in the collection $\mathcal{C}$, we have that $\overline{A} \notin \mathcal{C}$. Hence, $\delta(\overline{A}) = \delta(U)$, and by symmetry of cut-sets, $\delta(A) = \delta(U)$. ◀

▶ **Lemma 8.** *The size of the subset S is at most $2k-1$.*

Proof. For the sake of contradiction, suppose $|S| \geq 2k$. Our proof strategy is to show the existence of a k-partition with fewer crossing hyperedges than $OPT_{k\text{-cut}}$, thus contradicting the definition of $OPT_{k\text{-cut}}$. Let $S := \{u_1, u_2, \ldots, u_p\}$ for some $p \geq 2k$. For notational convenience, we will use $S - u_i$ to denote $S \setminus \{u_i\}$ and $S - u_i - u_j$ to denote $S \setminus \{u_i, u_j\}$. For a subset $X \subseteq U$, we denote the source minimal minimum $(X, \overline{U})$-terminal cut by $(H_X, \overline{H_X})$.

Our strategy to arrive at a k-partition with fewer crossing hyperedges than $OPT_{k\text{-cut}}$ is to apply the second conclusion of Theorem 5. The next few claims will set us up to obtain sets that satisfy the hypothesis of Theorem 5.

▷ Claim 9. For every $i \in [p]$, we have $\overline{H_{S-u_i}} \in \mathcal{C}$.

Proof. Let $i \in [p]$. Since S is a minimal transversal of the collection $\mathcal{C}$, there exists a set $B_i \in \mathcal{C}$ such that $B_i \cap S = \{u_i\}$. Hence, $(\overline{B_i}, B_i)$ is a $(S - u_i, \overline{U})$-terminal cut. Therefore,

$$d(\overline{H_{S-u_i}}) \leq d(B_i) \leq d(U).$$

Since $(H_{S-u_i}, \overline{H_{S-u_i}})$ is a $(S - u_i, \overline{U})$-terminal cut, we have that $\overline{U} \subseteq \overline{H_{S-u_i}}$. If $d(\overline{H_{S-u_i}}) < d(U)$, then $\delta(\overline{H_{S-u_i}}) \neq \delta(U)$ and $\overline{U} \subsetneq \overline{H_{S-u_i}}$, and consequently, $\overline{H_{S-u_i}} \in \mathcal{C}$. So, we will assume henceforth that $d(\overline{H_{S-u_i}}) = d(U)$.

Since $(H_{S-u_i} \cap \overline{B_i}, \overline{H_{S-u_i} \cap \overline{B_i}})$ is a $(S - u_i, \overline{U})$-terminal cut, we have that

$$d(H_{S-u_i} \cap \overline{B_i}) \geq d(H_{S-u_i}).$$

Since $(H_{S-u_i} \cup \overline{B_i}, \overline{H_{S-u_i} \cup \overline{B_i}})$ is a $(S - u_i, \overline{U})$-terminal cut, we have that

$$d(H_{S-u_i} \cup \overline{B_i}) \geq d(H_{S-u_i}).$$

Therefore, by submodularity of the hypergraph cut function, we have that

$$2d(U) \geq d(H_{S-u_i}) + d(B_i) \geq d(H_{S-u_i} \cap \overline{B_i}) + d(H_{S-u_i} \cup \overline{B_i}) \geq 2d(H_{S-u_i}) = 2d(U). \quad (1)$$

Therefore, all inequalities above should be equations. In particular, we have that $d(H_{S-u_i} \cap \overline{B_i}) = d(U) = d(B_i) = d(H_{S-u_i})$ and hence, $(H_{S-u_i} \cap \overline{B_i}, \overline{H_{S-u_i} \cap \overline{B_i}})$ is a minimum $(S - u_i, \overline{U})$-terminal cut. Since $(H_{S-u_i}, \overline{H_{S-u_i}})$ is a source minimal minimum $(S - u_i, \overline{U})$-terminal cut, we must have $H_{S-u_i} \cap \overline{B_i} = H_{S-u_i}$, and thus $H_{S-u_i} \subseteq \overline{B_i}$. Therefore, $B_i \subseteq \overline{H_{S-u_i}}$. Since $B_i \in \mathcal{C}$, we have $\delta(B_i) \neq \delta(U)$. However, $d(B_i) = d(U)$. Therefore $\delta(U) \setminus \delta(B_i) \neq \emptyset$. Let $e \in \delta(U) \setminus \delta(B_i)$. Since $e \in \delta(\overline{U})$, but $e \notin \delta(B_i)$, and $\overline{U} \subseteq B_i$, we have that $e \subseteq B_i$, and thus $e \subseteq \overline{H_{S-u_i}}$. Thus, we conclude that $\delta(U) \setminus \delta(\overline{H_{S-u_i}}) \neq \emptyset$, and so $\delta(\overline{H_S - u_i}) \neq \delta(U)$. This also implies that $\overline{U} \subsetneq \overline{H_{S-u_i}}$. Thus, $\overline{H_{S-u_i}} \in \mathcal{C}$. ◁

Claim 9 implies the following Corollary.

▶ **Corollary 10.** *For every $i \in [p]$, we have $u_i \in \overline{H_{S-u_i}}$.*

Proof. By definition, $S - u_i \subseteq H_{S-u_i}$, so $S \cap \overline{H_{S-u_i}} \subseteq \{u_i\}$. By Claim 9 we have that $\overline{H_{S-u_i}} \in \mathcal{C}$. Since S is a transversal of the collection $\mathcal{C}$, we have that $S \cap \overline{H_{S-u_i}} \neq \emptyset$. So, the vertex u_i must be in $\overline{H_{S-u_i}}$. ◀

Having obtained Corollary 10, the next few claims (Claims 11, 13, 14, and 15) are similar to the claims appearing in the proof of a structural theorem that appeared in [8]. Since the hypothesis of the structural theorem that we are proving here is different from theirs, we present the complete proofs of these claims here. The way in which we use the claims will also be different from [8].

The following claim will help in showing that $u_i, u_j \notin H_{S-u_i-u_j}$, which in turn, will be used to show that the hypothesis of Theorem 5 is satisfied by suitably chosen sets.

▷ Claim 11. For every $i, j \in [p]$, we have $H_{S-u_i-u_j} \subseteq H_{S-u_i}$.

Proof. We may assume that $i \neq j$. We note that $(H_{S-u_i-u_j} \cap H_{S-u_i}, \overline{H_{S-u_i-u_j} \cap H_{S-u_i}})$ is a $(S - u_i - u_j, \overline{U})$-terminal cut. Therefore,

$$d(H_{S-u_i-u_j} \cap H_{S-u_i}) \geq d(H_{S-u_i-u_j}). \quad (2)$$

Also, $(H_{S-u_i-u_j} \cup H_{S-u_i}, \overline{H_{S-u_i-u_j} \cup H_{S-u_i}})$ is a $(S - u_i, \overline{U})$-terminal cut. Therefore,

$$d(H_{S-u_i-u_j} \cup H_{S-u_i}) \geq d(H_{S-u_i}). \quad (3)$$

By submodularity of the hypergraph cut function and inequalities (2) and (3), we have that

$$\begin{aligned} d(H_{S-u_i-u_j}) + d(H_{S-u_i}) &\geq d(H_{S-u_i-u_j} \cap H_{S-u_i}) + d(H_{S-u_i-u_j} \cup H_{S-u_i}) \\ &\geq d(H_{S-u_i-u_j}) + d(H_{S-u_i}). \end{aligned}$$

Therefore, inequality (2) is an equation, and consequently, $(H_{S-u_i-u_j} \cap H_{S-u_i}, \overline{H_{S-u_i-u_j} \cap H_{S-u_i}})$ is a minimum $(S-u_i-u_j, \overline{U})$-terminal cut. If $H_{S-u_i-u_j} \setminus H_{S-u_i} \neq \emptyset$, then $(H_{S-u_i-u_j} \cap H_{S-u_i}, \overline{H_{S-u_i-u_j} \cap H_{S-u_i}})$ contradicts source minimality of the minimum $(S-u_i-u_j, \overline{U})$-terminal cut $(H_{S-u_i-u_j}, \overline{H_{S-u_i-u_j}})$. Hence, $H_{S-u_i-u_j} \setminus H_{S-u_i} = \emptyset$ and consequently, $H_{S-u_i-u_j} \subseteq H_{S-u_i}$. ◁

Claim 11 implies the following Corollary.

▶ **Corollary 12.** *For every $i, j \in [p]$, we have $u_i, u_j \notin H_{S-u_i-u_j}$.*

Proof. By Corollary 10, we have that $u_i \notin H_{S-u_i}$. Therefore, $u_i, u_j \notin H_{S-u_i} \cap H_{S-u_j}$. By Claim 11, $H_{S-u_i-u_j} \subseteq H_{S-u_i}$ and $H_{S-u_i-u_j} \subseteq H_{S-u_j}$. Therefore, $H_{S-u_i-u_j} \subseteq H_{S-u_i} \cap H_{S-u_j}$, and thus, $u_i, u_j \notin H_{S-u_i-u_j}$. ◀

The next claim will help in controlling the cost of the k-partition that we will obtain by applying Theorem 5.

▷ Claim 13. For every $i, j \in [p]$, we have $d(H_{S-u_i}) = d(U) = d(H_{S-u_i-u_j})$.

Proof. Let $a, b \in [p]$. We will show that $d(H_{S-u_a}) = d(U) = d(H_{S-u_a-u_b})$. Since $(U, \overline{U})$ is a $(S-u_a, \overline{U})$-terminal cut, we have that $d(H_{S-u_a}) \leq d(U)$. Since $(H_{S-u_a}, \overline{H_{S-u_a}})$ is a $(S-u_a-u_b, \overline{U})$-terminal cut, we have that $d(H_{S-u_a-u_b}) \leq d(H_{S-u_a}) \leq d(U)$. Thus, in order to prove the claim, it suffices to show that $d(H_{S-u_a-u_b}) \geq d(U)$.

Suppose for contradiction that $d(H_{S-u_a-u_b}) < d(U)$. Let $\ell \in [p] \setminus \{a, b\}$ be an arbitrary element (which exists since we have assumed that $p \geq 2k$ and $k \geq 2$). Let $R := \{u_\ell\}$, $S' := S - u_a - u_\ell$, and $A_i := \overline{H_{S-u_a-u_i}}$ for every $i \in [p] \setminus \{a, \ell\}$. We note that $|S'| = p-2 \geq 2k-2$. By definition, $(\overline{A_i}, A_i)$ is a minimum $(S-u_a-u_i, \overline{U})$-terminal cut for every $i \in [p] \setminus \{a, \ell\}$. Moreover, by Corollary 12, we have that $u_i \in A_i \setminus (\cup_{j \in [p] \setminus \{a,i,\ell\}} A_j)$ for every $i \in [p] \setminus \{a, \ell\}$. Hence, the sets U, R, and S', and the cuts $(\overline{A_i}, A_i)$ for $i \in [p] \setminus \{a, \ell\}$ satisfy the conditions of Theorem 5. Therefore, by the first conclusion of Theorem 5, there exists a k-partition $\mathcal{P}'$ with

$$|\delta(\mathcal{P}')| \leq \frac{1}{2} \min\{d(H_{S-u_a-u_i}) + d(H_{S-u_a-u_j}) \colon i, j \in [p] \setminus \{a, \ell\}\}.$$

By assumption, $d(H_{S-u_a-u_b}) < d(U)$ and $b \in [p] \setminus \{a, \ell\}$, so $\min\{d(H_{S-u_a-u_i}) \colon i \in [p] \setminus \{a, \ell\}\} < d(U)$. Since $(U, \overline{U})$ is a $(S-u_a-u_i, \overline{U})$-terminal cut, we have that $d(H_{S-u_a-u_i}) \leq d(U)$ for every $i \in [p] \setminus \{a, \ell\}$. Therefore,

$$\frac{1}{2} \min\{d(H_{S-u_a-u_i}) + d(H_{S-u_a-u_j}) \colon i, j \in [p] \setminus \{a, \ell\}\} < d(U) = OPT_{k\text{-cut}}.$$

Thus, we have that $|\delta(\mathcal{P}')| < OPT_{k\text{-cut}}$, which is a contradiction. ◁

The next two claims will help in arguing the existence of a hyperedge satisfying the conditions of the second conclusion of Theorem 5. In particular, we will need Claim 15. The following claim will help in proving Claim 15.

▷ Claim 14. For every $i, j \in [p]$, we have

$$d(H_{S-u_i} \cap H_{S-u_j}) = d(U) = d(H_{S-u_i} \cup H_{S-u_j}).$$

Proof. Since $(H_{S-u_i} \cap H_{S-u_j}, \overline{H_{S-u_i} \cap H_{S-u_j}})$ is a $(S - u_i - u_j, \overline{U})$-terminal cut, we have that $d(H_{S-u_i} \cap H_{S-u_j}) \geq d(H_{S-u_i-u_j})$. By Claim 13, we have that $d(H_{S-u_i-u_j}) = d(U) = d(H_{S-u_i})$. Therefore,

$$d(H_{S-u_i} \cap H_{S-u_j}) \geq d(H_{S-u_i}). \tag{4}$$

Since $(H_{S-u_i} \cup H_{S-u_j}, \overline{H_{S-u_i} \cup H_{S-u_j}})$ is a $(S - u_j, \overline{U})$-terminal cut, we have that

$$d(H_{S-u_i} \cup H_{S-u_j}) \geq d(H_{S-u_j}). \tag{5}$$

By submodularity of the hypergraph cut function and inequalities (4) and (5), we have that

$$d(H_{S-u_i}) + d(H_{S-u_j}) \geq d(H_{S-u_i} \cap H_{S-u_j}) + d(H_{S-u_i} \cup H_{S-u_j}) \geq d(H_{S-u_i}) + d(H_{S-u_j}).$$

Therefore, inequalities (4) and (5) are equations. Thus, by Claim 13, we have that

$$d(H_{S-u_i} \cap H_{S-u_j}) = d(H_{S-u_i}) = d(U),$$

and

$$d(H_{S-u_i} \cup H_{S-u_j}) = d(H_{S-u_j}) = d(U).$$ ◁

▷ Claim 15. For every $i, j, \ell \in [p]$ with $i \neq j$, we have $H_{S-u_\ell} \subseteq H_{S-u_i} \cup H_{S-u_j}$.

Proof. If $\ell = i$ or $\ell = j$ the claim is immediate. Thus, we assume that $\ell \notin \{i, j\}$. Let $Q := H_{S-u_\ell} \setminus (H_{S-u_i} \cup H_{S-u_j})$. We need to show that $Q = \emptyset$. We will show that $(H_{S-u_\ell} \setminus Q, \overline{H_{S-u_\ell} \setminus Q})$ is a minimum $(S - u_\ell, \overline{U})$-terminal cut. Consequently, Q must be empty (otherwise, $H_{S-u_\ell} \setminus Q \subsetneq H_{S-u_\ell}$ and hence, $(H_{S-u_\ell} \setminus Q, \overline{H_{S-u_\ell} \setminus Q})$ contradicts source minimality of the minimum $(S - u_\ell, \overline{U})$-terminal cut $(H_{S-u_\ell}, \overline{H_{S-u_\ell}})$).

We now show that $(H_{S-u_\ell} \setminus Q, \overline{H_{S-u_\ell} \setminus Q})$ is a minimum $(S - u_\ell, \overline{U})$-terminal cut. Since $H_{S-u_\ell} \setminus Q = H_{S-u_\ell} \cap (H_{S-u_i} \cup H_{S-u_j})$, we have that $S - u_i - u_j - u_\ell \subseteq H_{S-u_\ell} \setminus Q$. We also know that u_i and u_j are contained in both H_{S-u_ℓ} and $H_{S-u_i} \cup H_{S-u_j}$. Therefore, $S - u_\ell \subseteq H_{S-u_\ell} \setminus Q$. Thus, $(H_{S-u_\ell} \setminus Q, \overline{H_{S-u_\ell} \setminus Q})$ is a $(S - u_\ell, \overline{U})$-terminal cut. Therefore,

$$d(H_{S-u_\ell} \cap (H_{S-u_i} \cup H_{S-u_j})) = d(H_{S-u_\ell} \setminus Q) \geq d(H_{S-u_\ell}). \tag{6}$$

We also have that $(H_{S-u_\ell} \cup (H_{S-u_i} \cup H_{S-u_j}), \overline{H_{S-u_\ell} \cup (H_{S-u_i} \cup H_{S-u_j})})$ is a $(S - u_i, \overline{U})$-terminal cut. Therefore, $d(H_{S-u_\ell} \cup (H_{S-u_i} \cup H_{S-u_j})) \geq d(H_{S-u_i})$. By Claims 13 and 14, we have that $d(H_{S-u_i}) = d(V_1) = d(H_{S-u_i} \cup H_{S-u_j})$. Therefore,

$$d(H_{S-u_\ell} \cup (H_{S-u_i} \cup H_{S-u_j})) \geq d(H_{S-u_i} \cup H_{S-u_j}). \tag{7}$$

By submodularity of the hypergraph cut function and inequalities (6) and (7), we have that

$$\begin{aligned} d(H_{S-u_\ell}) + d(H_{S-u_i} \cup H_{S-u_j}) &\geq d(H_{S-u_\ell} \cap (H_{S-u_i} \cup H_{S-u_j})) + d(H_{S-u_\ell} \cup (H_{S-u_i} \cup H_{S-u_j})) \\ &\geq d(H_{S-u_\ell}) + d(H_{S-u_i} \cup H_{S-u_j}). \end{aligned}$$

Therefore, inequalities (6) and (7) are equations, so $(H_{S-u_\ell} \setminus Q, \overline{H_{S-u_\ell} \setminus Q})$ is a minimum $(S - u_\ell, \overline{U})$-terminal cut. ◁

Let $R := \{u_p\}$, $S' := S - u_p$, and $(\overline{A_i}, A_i) := (H_{S-u_i}, \overline{H_{S-u_i}})$ for every $i \in [p-1]$. By definition, $(\overline{A_i}, A_i)$ is a minimum $(S - u_i, \overline{U})$-terminal cut for every $i \in [p-1]$. Moreover, by Corollary 10, we have that $u_i \in A_i \setminus (\cup_{j \in [p-1] \setminus \{i\}} A_j)$. Hence, the sets U, R, and S', and the cuts $(\overline{A_i}, A_i)$ for $i \in [p-1]$ satisfy the conditions of Theorem 5. We will use the second conclusion of Theorem 5. We now show that there exists a hyperedge satisfying the conditions mentioned in the second conclusion of Theorem 5. We will use Claim 16 below to prove this. Let $W := \cup_{1 \le i < j \le p-1}(A_i \cap A_j)$ and $Z := \cap_{i \in [p-1]} \overline{A_i}$ as in the statement of Theorem 5.

▷ Claim 16. There exists a hyperedge $e \in E$ such that $e \cap W \neq \emptyset$, $e \cap Z \neq \emptyset$, and $e \subseteq W \cup Z$.

Proof. We note that $S \subseteq (S - u_i) \cup (S - u_j) \subseteq H_{S-u_i} \cup H_{S-u_j}$ for every distinct $i, j \in [p-1]$. Therefore, $S \cap (A_i \cap A_j) = \emptyset$ for every distinct $i, j \in [p-1]$, and thus $S \cap W = \emptyset$. Since S is a transversal of the collection $\mathcal{C}$, it follows that the set W is not in the collection $\mathcal{C}$.

By definition, $\overline{U} \subseteq A_i$ for every $i \in [p-1]$, and thus $\overline{U} \subseteq W$. Since $W \notin \mathcal{C}$, either $d(W) > d(U)$ or $\delta(W) = \delta(U)$. By Claim 9, we have that $\overline{H_{S-u_p}} \in \mathcal{C}$, and thus, $d(\overline{H_{S-u_p}}) \le d(U)$ and $\delta(\overline{H_{S-u_p}}) \neq \delta(U)$. Consequently, $d(W) \ge d(\overline{H_{S-u_p}})$, and $\delta(W) \neq \delta(\overline{H_{S-u_p}})$, and thus, $\delta(W) \setminus \delta(\overline{H_{S-u_p}}) \neq \emptyset$. Let $e \in \delta(W) \setminus \delta(\overline{H_{S-u_p}})$. We will show that this choice of e achieves the desired properties.

For each $i \in [p]$, let $Y_i := \overline{H_{S-u_i}} \setminus W$. By Claim 15, for every $i, j, \ell \in [p]$ with $i \neq j$ we have that $H_{S-u_\ell} \subseteq H_{S-u_i} \cup H_{S-u_j}$. Therefore $\overline{H_{S-u_i}} \cap \overline{H_{S-u_j}} \subseteq \overline{H_{S-u_\ell}}$ for every such $i, j, \ell \in [p]$, and hence $W \subseteq \overline{H_{S-u_\ell}}$ for every $\ell \in [p]$. Thus, $W \subseteq \overline{H_{S-u_p}}$. Since $e \in \delta(W) \setminus \delta(\overline{H_{S-u_p}})$, we have that $e \subseteq W \cup Y_p$, $e \cap W \neq \emptyset$ and $e \cap Y_p \neq \emptyset$. Therefore, in order to show that e has the three desired properties as in the claim, it suffices to show that $Y_p \subseteq Z$. We prove this next.

By definition, $Y_p \cap W = \emptyset$. By Claim 15, for every $i \in [p-1]$, we have that $\overline{H_{S-u_p}} \cap \overline{H_{S-u_i}} \subseteq \overline{H_{S-u_1}}$ and $\overline{H_{S-u_p}} \cap \overline{H_{S-u_i}} \subseteq \overline{H_{S-u_2}}$, so $\overline{H_{S-u_p}} \cap \overline{H_{S-u_i}} \subseteq \overline{H_{S-u_1}} \cap \overline{H_{S-u_2}} \subseteq W$. Thus, for every $i \in [p-1]$, $Y_p \cap Y_i \subseteq \overline{H_{S-u_p}} \cap \overline{H_{S-u_i}} \subseteq W$, so since $Y_p \cap W = \emptyset$, we have that $Y_p \cap Y_i = \emptyset$ for every $i \in [p-1]$. Therefore,

$$Y_p \subseteq \overline{W \cup \left(\bigcup_{i=1}^{p-1} Y_i\right)} = \overline{\bigcup_{i=1}^{p-1} \overline{H_{S-u_i}}} = \bigcap_{i=1}^{p-1} H_{S-u_i} = Z. \qquad \triangleleft$$

By Claim 16, there is a hyperedge e satisfying the conditions of the second conclusion of Theorem 5. Therefore, by Theorem 5, there exists a k-partition $\mathcal{P}'$ with

$$\begin{aligned} |\delta(\mathcal{P}')| &< \frac{1}{2} \min\{d(A_i) + d(A_j) : i, j \in [p-1], i \neq j\} \\ &= d(U) && \text{(By Claim 13)} \\ &= OPT_{k\text{-cut}}. && \text{(By assumption of the theorem)} \end{aligned}$$

Thus, we have obtained a k-partition $\mathcal{P}'$ with $|\delta(\mathcal{P}')| < OPT_{k\text{-cut}}$, which is a contradiction. ◀

3 Proof of Theorem 4

We prove Theorem 4 in this section. Applying Theorem 6 to $(\overline{U}, U)$ yields the following corollary.

▶ **Corollary 17.** *Let $G = (V, E)$ be a hypergraph and let $OPT_{k\text{-cut}}$ be the optimum value of Hypergraph-k-Cut in G for some integer $k \ge 2$. Suppose $(U, \overline{U})$ is a 2-partition of V with $d(U) = OPT_{k\text{-cut}}$. Then, there exists a set $T \subseteq \overline{U}$ with $|T| \le 2k - 1$ such that every minimum (U, T)-terminal cut $(A, \overline{A})$ satisfies $\delta(A) = \delta(U)$.*

We now restate Theorem 4 and prove it using Theorem 6 and Corollary 17.

▶ **Theorem 4.** *Let $G = (V, E)$ be a hypergraph and let $OPT_{k\text{-}cut}$ be the optimum value of* Hypergraph-k-Cut *in G for some integer $k \geq 2$. Suppose $(U, \overline{U})$ is a 2-partition of V with $d(U) = OPT_{k\text{-}cut}$. Then, there exist sets $S \subseteq U$, $T \subseteq \overline{U}$ with $|S| \leq 2k-1$ and $|T| \leq 2k-1$ such that every minimum (S,T)-terminal cut $(A, \overline{A})$ satisfies $\delta(A) = \delta(U)$.*

Proof. By Theorem 6, there exists a subset $S \subseteq U$ with $|S| \leq 2k-1$ such that every minimum $(S, \overline{U})$-terminal cut $(A, \overline{A})$ has $\delta(A) = \delta(U)$. By Corollary 17, there exists a subset $T \subseteq \overline{U}$ with $|T| \leq 2k-1$ such that every minimum (U,T)-terminal cut $(A, \overline{A})$ has $\delta(A) = \delta(U)$. We will show that every minimum (S,T)-terminal cut $(A, \overline{A})$ has $\delta(A) = \delta(U)$. We will need the following claim.

▷ Claim 18. Let $(Y, \overline{Y})$ be the source minimal minimum (S,T)-terminal cut. Then $\delta(Y) = \delta(U)$.

Proof. Since $(U, \overline{U})$ is a (S,T)-terminal cut, and $(Y, \overline{Y})$ is a minimum (S,T)-terminal cut, we have that

$$d(U) \geq d(Y).$$

Since $(U \cap Y, \overline{U \cap Y})$ is a $(S, \overline{U})$-terminal cut, we have that

$$d(U \cap Y) \geq d(U).$$

Since $(U \cup Y, \overline{U \cup Y})$ is a (U,T)-terminal cut, we have that

$$d(U \cup Y) \geq d(U).$$

Thus, by the submodularity of the hypergraph cut function we have that

$$2d(U) \geq d(U) + d(Y) \geq d(U \cap Y) + d(U \cup Y) \geq 2d(U).$$

Therefore, we have that $d(U \cap Y) = d(U)$, so $(U \cap Y, \overline{U \cap Y})$ is a minimum (S,T)-terminal cut. Since $(Y, \overline{Y})$ is the source minimal (S,T)-terminal cut, we have that $U \cap Y = Y$, and hence $Y \subseteq U$. Therefore, $(Y, \overline{Y})$ is a minimum $(S, \overline{U})$-terminal cut. By the choice of S, we have that $\delta(Y) = \delta(U)$. ◁

Applying Claim 18 to both sides of the partition $(U, \overline{U})$, we have that the source minimal minimum (S,T)-terminal cut $(Y, \overline{Y})$ has $\delta(Y) = \delta(U)$, and the source minimal minimum (T,S)-terminal cut $(Z, \overline{Z})$ has $\delta(Z) = \delta(U)$. Therefore, for every $e \in \delta(U)$, we have that $e \cap Y \neq \emptyset$ and $e \cap Z \neq \emptyset$.

Let $(A, \overline{A})$ be a minimum (S,T)-terminal cut. Since $(Y, \overline{Y})$ is the source minimal minimum (S,T)-terminal cut, we have that $Y \subseteq A$. Since $(Z, \overline{Z})$ is the source minimal minimum (T,S)-terminal cut, we have that $Z \subseteq \overline{A}$. Since every $e \in \delta(U)$ intersects both Y and Z, it follows that every $e \in \delta(U)$ intersects both A and $\overline{A}$, and hence, $\delta(U) \subseteq \delta(A)$. Since $(A, \overline{A})$ is a minimum (S,T)-terminal cut, $d(A) \leq d(U)$, and thus we have that $\delta(A) = \delta(U)$. ◀

4 Conclusion

We showed the first polynomial bound on the number of MINMAX-k-CUT-SETs in hypergraphs for every fixed k and gave a polynomial-time algorithm to enumerate all MINMAX-k-CUT-SETs as well as all MIN-k-CUT-SETs in hypergraphs for every fixed k. Our main contribution is

a structural theorem that is the backbone of the correctness analysis of our enumeration algorithms. In order to enumerate MINMAX-k-CUT-SETS in hypergraphs, we introduced the notion of k-cut-set representatives and enumerated k-cut-set representatives of all optimum k-partitions for MINMAX-HYPERGRAPH-k-PARTITION. Our technique builds on known structural results for HYPERGRAPH-k-CUT and MINMAX-HYPERGRAPH-k-PARTITION [8, 12, 13].

The technique underlying our enumeration algorithms is not necessarily novel – we simply rely on minimum (s,t)-terminal cuts. Using fixed-terminal cuts to address global partitioning problems is not a novel technique by itself – it is common knowledge that minimum (s,t)-terminal cuts can be used to solve global minimum cut. However, there are several problems where naive use of this technique fails to lead to efficient algorithms: e.g., multiway cut does not help in solving GRAPH-k-CUT since multiway cut is NP-hard. Adapting this technique for specific partitioning problems requires careful identification of structural properties. In fact, beautiful structural properties have been shown for a rich variety of partitioning problems in combinatorial optimization in order to exploit this technique: for example, it was used (1) to design the first efficient algorithm for GRAPH-k-CUT [23], (2) to solve certain constrained submodular minimization problems [22, 35], and (3) more recently, to design fast algorithms for global minimum cut in graphs and for Gomory-Hu tree in unweighted graphs [1, 34]. Our use of this technique also relies on identifying and proving a suitable structural property, namely Theorem 4. The advantage of our structural property is that it simultaneously enables enumeration of MIN-k-CUT-SETS as well as MINMAX-k-CUT-SETS in hypergraphs which was not possible via structural theorems that were developed before. Furthermore, it helps in showing the first polynomial bound on the number of MINMAX-k-CUT-SETS in hypergraphs for every fixed k.

We also emphasize a limitation of our technique. Although it helps in solving ENUM-HYPERGRAPH-k-CUT and ENUM-MINMAX-HYPERGRAPH-k-PARTITION, it does not help in solving a seemingly related hypergraph k-partitioning problem – namely, given a hypergraph $G = (V, E)$ and a fixed integer k, find a k-partition $(V_1, \ldots, V_k)$ of the vertex set that minimizes $\sum_{i=1}^{k} |\delta(V_i)|$. Natural variants of our structural theorem fail to hold for this objective. Resolving the complexity of this variant of the hypergraph k-partitioning problem for $k \geq 5$ remains open.

We mention an open question concerning HYPERGRAPH-k-CUT and the enumeration of MIN-k-CUT-SETS in hypergraphs for fixed k. We recall the status in graphs: the number of minimum k-partitions in a connected graph was known to be $O(n^{2k-2})$ via Karger-Stein's algorithm [30] and $\Omega(n^k)$ via the cycle example, where n is the number of vertices; recent works have improved on the upper bound to match the lower bound for fixed k – this improvement in upper bound also led to the best possible $O(n^k)$-time algorithm for GRAPH-k-CUT for fixed k [25–27]. For hypergraphs, the number of MIN-k-CUT-SETS is known to be $O(n^{2k-2})$ and $\Omega(n^k)$. Can we improve the upper/lower bound? Is it possible to design an algorithm for HYPERGRAPH-k-CUT that runs in time $O(n^k p)$?

References

1 A. Abboud, R. Krauthgamer, and O. Trabelsi. Subcubic Algorithms for Gomory–Hu Tree in Unweighted Graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 1725–1737, 2021.

2 K. Ahn and S. Guha. Graph sparsification in the semi-streaming model. In *Proceeings of the 36th International Colloquium on Automata, Languages and Programming: Part II*, ICALP, pages 328–338, 2009.

3 K. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 459–467, 2012.

4 K. Ahn, S. Guha, and A. McGregor. Graph sketches: Sparsification, spanners, and subgraphs. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS, pages 5–14, 2012.

5 D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

6 N. Bansal, O. Svensson, and L. Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 910–928, 2019.

7 C. Beideman, K. Chandrasekaran, and W. Wang. Counting and enumerating optimum cut sets for hypergraph k-partitioning problems for fixed k, 2022. `arXiv:2204.09178`.

8 C. Beideman, K. Chandrasekaran, and W. Wang. Deterministic enumeration of all minimum k-cut-sets in hypergraphs for fixed k. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, 2022.

9 A. Benczur. A representation of cuts within 6/5 times the edge connectivity with applications. In *Proceedings of the 36th Annual IEEE Foundations of Computer Science*, FOCS, pages 92–102, 1995.

10 A. Benczur. *Cut structures and randomized algorithms in edge-connectivity problems*. PhD thesis, MIT, 1997.

11 A. Benczur and M. Goemans. Deformable polygon representation and near-mincuts. *Building Bridges: Between Mathematics and Computer Science, M. Groetschel and G.O.H. Katona, Eds., Bolyai Society Mathematical Studies*, 19:103–135, 2008.

12 K. Chandrasekaran and C. Chekuri. Hypergraph k-cut for fixed k in deterministic polynomial time. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science*, FOCS, pages 810–821, 2020.

13 K. Chandrasekaran and C. Chekuri. Min-max partitioning of hypergraphs and symmetric submodular functions. In *Proceedings of the 32nd annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1026–1038, 2021.

14 K. Chandrasekaran and W. Wang. Fixed Parameter Approximation Scheme for Min-max k-cut. In *Integer Programming and Combinatorial Optimization*, IPCO, pages 354–367, 2021.

15 K. Chandrasekaran, C. Xu, and X. Yu. Hypergraph k-cut in randomized polynomial time. *Mathematical Programming (Preliminary version in SODA 2018)*, 186:85–113, 2019.

16 C. Chekuri, K. Quanrud, and C. Xu. LP relaxation and tree packing for minimum k-cut. *SIAM Journal on Discrete Mathematics*, 34(2):1334–1353, 2020.

17 C. Chekuri and C. Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018.

18 Y. Chen, S. Khanna, and A. Nagda. Near-linear size hypergraph cut sparsifiers. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 61–72, 2020.

19 W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, 1997.

20 K. Fox, D. Panigrahi, and F. Zhang. Minimum cut and minimum k-cut in hypergraphs via branching contractions. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 881–896, 2019.

21 M. Ghaffari, D. Karger, and D. Panigrahi. Random contractions and sampling for hypergraph and hedge connectivity. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1101–1114, 2017.

22 M. X. Goemans and V. S. Ramakrishnan. Minimizing submodular functions over families of sets. *Combinatorica*, 15:499–513, 1995.

23 O. Goldschmidt and D. Hochbaum. A Polynomial Algorithm for the k-cut Problem for Fixed k. *Mathematics of Operations Research (Preliminary version in FOCS 1988)*, 19(1):24–37, February 1994.

24 F. Guinez and M. Queyranne. The size-constrained submodular k-partition problem. Unpublished manuscript. Available at `https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxmbGF2aW9ndWluZXpob21lcGFnZXxneDo0NDVlMThkMDg4ZWRlOGI1`. See also `https://smartech.gatech.edu/bitstream/handle/1853/43309/Queyranne.pdf`, 2012.

25 A. Gupta, D. Harris, E. Lee, and J. Li. Optimal Bounds for the k-cut Problem. *arXiv preprint*, 2020. `arXiv:2005.08301`.

26 A. Gupta, E. Lee, and J. Li. The number of minimum k-cuts: improving the Karger-Stein bound. In *Proceedings of the 51st ACM Symposium on Theory of Computing*, STOC, pages 229–240, 2019.

27 A. Gupta, E. Lee, and J. Li. The Karger-Stein algorithm is optimal for k-cut. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing*, STOC, pages 473–484, 2020.

28 M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 598–611, 2021.

29 D. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000.

30 D. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, July 1996.

31 A. Karlin, N. Klein, and S-O. Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 32–45, 2021.

32 D. Kogan and R. Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 367–376, 2015.

33 E. Lawler. Cutsets and Partitions of Hypergraphs. *Networks*, 3:275–285, 1973.

34 J. Li and D. Panigrahi. Deterministic Min-cut in Poly-logarithmic Max-flows. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science*, FOCS, pages 85–92, 2020.

35 M. Nägele, B. Sudakov, and R. Zenklusen. Submodular minimization under congruency constraints. *Combinatorica*, 39:1351–1386, 2019.

36 K. Okumoto, T. Fukunaga, and H. Nagamochi. Divide-and-conquer algorithms for partitioning hypergraphs and submodular systems. *Algorithmica*, 62(3):787–806, 2012.

37 M. Queyranne. On optimum size-constrained set partitions, 1999. Talk at Aussiois workshop on Combinatorial Optimization.

38 T. Soma and Y. Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2570–2581, 2019.

39 M. Thorup. Minimum k-way Cuts via Deterministic Greedy Tree Packing. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC, pages 159–166, 2008.

40 R. Zenklusen. A 1.5-Approximation for Path TSP. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1539–1549, 2019.

41 L. Zhao. *Approximation algorithms for partition and design problems in networks*. PhD thesis, Graduate School of Informatics, Kyoto University, Japan, 2002.

42 L. Zhao, H. Nagamochi, and T. Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming*, 102(1):167–183, 2005.

Finding Monotone Patterns in Sublinear Time, Adaptively

Omri Ben-Eliezer
Massachusetts Institute of Technology, Cambridge, MA, USA

Shoham Letzter
University College London, UK

Erik Waingarten
Stanford University, CA, USA

Abstract

We investigate adaptive sublinear algorithms for finding monotone patterns in sequential data. Given fixed $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$, consider the problem of finding a length-k increasing subsequence in a sequence $f\colon [n] \to \mathbb{R}$, provided that f is ε-far from free of such subsequences. It was shown by Ben-Eliezer et al. [FOCS 2019] that the non-adaptive query complexity of the above task is $\Theta((\log n)^{\lfloor \log_2 k \rfloor})$. In this work, we break the non-adaptive lower bound, presenting an adaptive algorithm for this problem which makes $O(\log n)$ queries. This is optimal, matching the classical $\Omega(\log n)$ adaptive lower bound by Fischer [Inf. Comp. 2004] for monotonicity testing (which corresponds to the case $k = 2$). Equivalently, our result implies that testing whether a sequence decomposes into k monotone subsequences can be done with $O(\log n)$ queries.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases property testing, monotone patterns, monotone decomposition, adaptivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.17

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/1911.01169`

Funding *Shoham Letzter*: Research supported by Dr. Max Rössler, the Walter Haefner Foundation and by the ETH Zurich Foundation.
Erik Waingarten: This work is supported by the National Science Foundation under Award No. 2002201 and Moses Charikar's Simons Investigator award.

1 Introduction

Pattern avoidance and detection in sequential data is a central problem in theoretical computer science and combinatorics [57], dating back to the seminal work of Knuth [38] (from a computer science perspective), and Simion and Schmidt [55] (from a combinatorial perspective). Studying the computational problem within the framework of sublinear algorithms, Newman, Rabinovich, Rajendraprasad, and Sohler [44, 45] considered the problem of property testing for forbidden order patterns in a sequence, where one of the central special cases they considered was that of *monotone patterns*. The property testing problem of detecting monotone patterns generalizes classical monotonicity testing in sequences, and is tightly connected to the longest increasing subsequence (LIS) problem [46].

For an integer $k \in \mathbb{N}$ and a sequence $f\colon [n] \to \mathbb{R}$, a *length-$k$ monotone subsequence of f* is a tuple of k indices, $(i_1, \ldots, i_k) \in [n]^k$, such that $i_1 < \cdots < i_k$ and $f(i_1) < \cdots < f(i_k)$. More generally, for a permutation $\pi\colon [k] \to [k]$, a *π-pattern of f* is given by a tuple of k indices $i_1 < \cdots < i_k$ such that $f(i_{j_1}) < f(i_{j_2})$ whenever $j_1, j_2 \in [k]$ satisfy $\pi(j_1) < \pi(j_2)$. A

sequence f is π-free if there are no subsequences of f with order pattern π. For a fixed $k \in \mathbb{N}$ and a pattern π of length k, the goal is to test whether a function $f\colon [n] \to \mathbb{R}$ is π-free or ε-far from π-free (that is, any π-free function g differs from f on at least εn inputs). The algorithmic task proposed in [45] and studied in this paper is as follows.

> *For $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$, design a randomized algorithm that, given query access to a function $f\colon [n] \to \mathbb{R}$, distinguishes with probability at least $9/10$ between the case that f is free of length-k monotone subsequences and the case that it is ε-far from free of length-k monotone subsequences.*

The above algorithmic formulation is equivalent to the following property testing problem (with one-sided error). For a given $2 \leq k \in \mathbb{N}$ and $f\colon [n] \to \mathbb{R}$, test whether there exists a decomposition of f into fewer than k non-increasing subsequences, or f is ε-far from having such a decomposition. The equivalence of the two formulations is a consequence of Dilworth's theorem [22]. One direction is trivial: if there exists a length-k increasing subsequence $(i_1, \ldots, i_k) \in [n]^k$, then any partition of f into fewer than k subsequences must contain two indices i_j and $i_{j'}$ within the same subsequence, hence, the subsequences are not non-increasing. The other direction follows from considering the poset $([n], \prec_f)$, where $i \prec_f j$ iff $i \leq j$ and $f(i) \geq f(j)$; every anti-chain of $\prec_f$ is an increasing subsequence of f, and every chain of $\prec_f$ is a non-increasing subsequence. If there are no length-k increasing subsequences, the maximum anti-chain of $\prec_f$ has size at most $k-1$, and by Dilworth's theorem, there is a partition of $([n], \prec_f)$ into at most $k-1$ chains, i.e., non-increasing subsequences.[1]

This paper gives an algorithm with optimal dependence in n for the above problems. We state the main theorem next, and discuss connections to monotonicity testing and to the longest increasing subsequence (LIS) problem shortly after.

▶ **Theorem 1.** *Fix $k \in \mathbb{N}$ and $\varepsilon > 0$. There exists an algorithm that, given query access to a function $f\colon [n] \to \mathbb{R}$ which is ε-far from free of length-k monotone subsequences, outputs a length-k monotone subsequence of f with probability $9/10$, with query complexity and running time of*

$$\left(k^k \cdot (\log(1/\varepsilon))^k \cdot \frac{1}{\varepsilon} \cdot \log(1/\delta)\right)^{O(k)} \cdot \log n.$$

Thus, for fixed k and ε, the query complexity and running time are of order $O(\log n)$. The above result can be stated analogously in the language of monotone decompositions.

▶ **Corollary 2.** *Fix $k \in \mathbb{N}$ and $\varepsilon > 0$. There is an algorithm with query complexity and running time $O(\log n)$ for ε-testing whether a sequence $f\colon [n] \to \mathbb{R}$ is decomposable into k monotone subsequences.*

The algorithm underlying Theorem 1 is *adaptive*[2] and solves the testing problem with one-sided error, since a length-k monotone subsequence is evidence for not being free of such subsequences. The algorithm improves on a recent result of Ben-Eliezer, Canonne, Letzter

[1] A similar equivalence, between being decomposable into k increasing (or decreasing) subsequences and not containing non-increasing (or non-decreasing, respectively) patterns of length $k+1$ holds as well. We note that all results stated here in terms of "strong" monotonicity, e.g., being increasing, will also hold for their "weak" monotonicity analogue, e.g., being non-decreasing.

[2] An algorithm is *non-adaptive* if its queries do not depend on the answers to previous queries, or, equivalently, if all queries to the function can be made in parallel. Otherwise, if the queries of an algorithm may depend on the outputs of previous queries, then the algorithm is *adaptive*.

and Waingarten [7] who gave a non-adaptive algorithm for finding length-k monotone patterns with query complexity $O_{k,\varepsilon}((\log n)^{\lfloor \log_2 k \rfloor})$, which in itself improved upon a $O_{k,\varepsilon}((\log n)^{O(k^2)})$ upper bound by [45]. The focus of [7] was on *non-adaptive* algorithms, and they gave a lower bound of $\Omega\big((\log n)^{\lfloor \log_2 k \rfloor}\big)$ queries for non-adaptive algorithms achieving one-sided error. Hence, Theorem 1 implies a natural separation between the power of adaptive and non-adaptive algorithms for finding monotone subsequences.

Theorem 1 is optimal, even among two-sided error algorithms. In the case $k = 2$, corresponding to monotonicity testing, there is a $\Omega(\log n/\varepsilon)$ lower bound (as long as, say, $\varepsilon > n^{-0.99}$) for both non-adaptive and adaptive algorithms [25, 27, 15], even with two-sided error. A simple reduction suggested in [45] shows that the same lower bound (up to a multiplicative factor depending on k) holds for any fixed $k \geq 2$. Thus, an appealing consequence of Theorem 1 is that the natural generalization of monotonicity testing, which considers forbidden monotone patterns of fixed length longer than 2, does not affect the dependence on n in the query complexity by more than a constant factor. Interestingly, [27] shows that for any adaptive algorithm for monotonicity testing on $f\colon [n] \to \mathbb{R}$ there is a non-adaptive algorithm which is at least as good in terms of query complexity (even if we only restrict ourselves to one-sided error algorithms). That is, adaptivity does not help at all for $k = 2$. In contrast, the separation between our $O(\log n)$ adaptive upper bound and the $\Omega\big((\log n)^{\lfloor \log_2 k \rfloor}\big)$ non-adaptive lower bound of [7] means this is no longer true for $k \geq 4$.

While our work settles the dependence in n in the query complexity of adaptive monotone pattern testing, and [7] settles the non-adaptive dependence in n, the following interesting question remains wide open.

▶ **Question 3.** *What is the optimal dependence of the query complexity in k and ε for the monotone subsequence testing problem discussed in this paper?*

Thus far, all known (adaptive and non-adaptive) results on this problem have a $k^{O(k^2)}$ type dependence in k in the query complexity; see Theorem 3.1 in [45] and Lemma 3.2 in [7]. The best known dependence in ε is of the form $(1/\varepsilon)^{\log_2 k + O(1)}$ for fixed k [7].

On the role of adaptivity in order pattern detection

Harnessing adaptivity to improve algorithmic performance is a notoriously difficult problem in many branches of property testing, typically requiring a good structural understanding of the task at hand. In the context of testing for forbidden order patterns, non-adaptive algorithms are quite weak: the non-adaptive query complexity is $\Omega(n^{1/2})$ for all non-monotone order patterns [45], and as high as $n^{1-1/(k-\Theta(1))}$ for almost all patterns of length k [6]. A recent (and independent) work of [47] gave new adaptive algorithms for general patterns with query complexity $n^{o(1)}$ for fixed constant $\varepsilon > 0$ and $k \in \mathbb{N}$, showing that for non-monotone patterns, too, adaptive algorithms may significantly improve upon non-adaptive ones. We note that the query complexity obtained in [47] is not polylogarithmic in n, and so their result is incomparable to ours. Their proof techniques are also very different from ours: at the core of their proof is a sophisticated sparsification framework, which makes use of a beautiful result of Marcus and Tardos [40] on pattern-avoidance in matrices.

Connections to the Longest Increasing Subsequence (LIS) problem

As an immediate consequence, Theorem 1 gives an optimal testing algorithm for the longest increasing subsequence (LIS) problem in a certain regime. The classical LIS problem asks to determine, given a sequence $f\colon [n] \to \mathbb{R}$, the maximum k for which f contains a

length-k increasing subsequence. It is very closely related to other fundamental algorithmic problems in sequences, such as computing the edit distance, Ulam distance, or distance from monotonicity (for example, the latter equals n minus the LIS length), and has been thoroughly investigated from the perspective of classical algorithms [29, 50], sublinear-time algorithms [49, 1, 54, 52, 46, 42, 2], streaming algorithms [34, 56, 30, 53, 24, 43], dynamic algorithms [18, 31, 39, 41] and massively parallel computation [36, 12]. In the property testing regime, the corresponding decision task is to distinguish between the case where f has LIS length at most k (where k is given as part of the input) and the case that f is ε-far from having such a LIS length. Theorem 1 in combination with the aforementioned $\Omega(\log n)$ lower bounds (which readily carry over to this setting) yield a tight bound on the query complexity of testing whether the LIS length is a constant.

▶ **Corollary 4.** *Fix $2 \leq k \in \mathbb{N}$ and $\varepsilon > 0$. The query complexity of ε-testing whether $f\colon [n] \to \mathbb{R}$ has LIS length at most k is $\Theta(\log n)$.*

1.1 Related Work

Considering general permutations π of length k and *exact* computation, [35] showed how to find a π-pattern in a sequence f in time $2^{O(k^2 \log k)}n$, later improved by [28] to $2^{O(k^2)}n$. In the regime $k = \Omega(\log n)$, an algorithm of [8] running in time $n^{k/4+o(k)}$ provides the state-of-the-art. The analogous *counting* problem has also been actively studied, see [26] and the references within.

For *approximate* computation of general patterns π, the works of [45, 6] investigate the query complexity of property testing for forbidden order patterns. When π is of length 2, the problem considered is equivalent to testing monotonicity, one of the most widely-studied problems in property testing, with works spanning the past two decades. Over the years, variants of monotonicity testing over various partially ordered sets have been considered, including the line $[n]$ [25, 27, 3, 48, 5], the Boolean hypercube $\{0,1\}^d$ [23, 10, 13, 14, 20, 19, 37, 4, 16, 21, 17], and the hypergrid $[n]^d$ [11, 15, 9]. We refer the reader to [32, Chapter 4] for more on monotonicity testing, and a general overview of the field of property testing (introduced in [51, 33]).

1.2 Main Ideas and Techniques

We now describe some intuition behind the proof of Theorem 1. We note that the algorithm considers several cases and combines ideas from [45] and [7] with new structural and algorithmic components. In this overview, technical details established in [45] and [7] are noted but excluded; the purpose is to highlight the challenges and novel ideas arising specifically from this work. (See the Appendix in the full-version of the work for a short technical overview of these previous results.)

Fix $k \in \mathbb{N}$ and $\varepsilon > 0$, and suppose that $f\colon [n] \to \mathbb{R}$ is ε-far from $(12\dots k)$-free, that is, ε-far from free of length-k increasing subsequences. Notice that f must contain a collection $\mathcal{C}$ of at least $\varepsilon n/k$ pairwise-disjoint increasing subsequences of length k; indeed, otherwise, greedily eliminating these subsequences gives a $(12\dots k)$-free function differing in strictly fewer than εn inputs.

For simplicity in this overview, assume that k is even and that all $\varepsilon n/k$ length-k increasing subsequences of f in $\mathcal{C}$, $(x_1, x_2, \dots, x_k) \in [n]^k$, satisfy that $|x_{k/2+1} - x_{k/2}| \geq |x_{i+1} - x_i|$ for all $i \in [k-1]$ (the non-adaptive lower bound of $\Omega_\varepsilon((\log n)^{\lfloor \log_2 k \rfloor})$ holds even in this restricted case) – intuitively, the largest "gap" in successive indices is between the $k/2$-th and $(k/2+1)$-th position. A goal, common to [45, 7] and this work, is to recursively

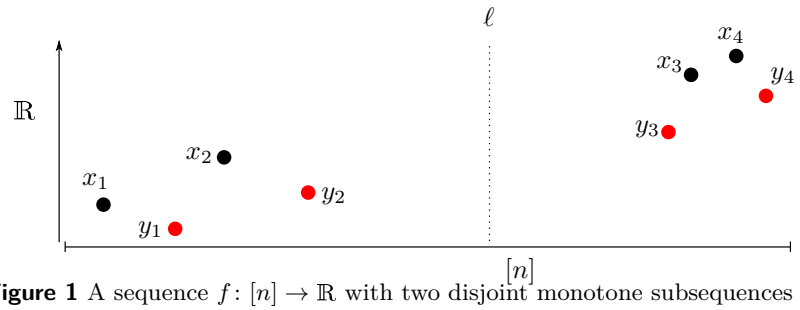


Figure 1 A sequence $f\colon [n] \to \mathbb{R}$ with two disjoint monotone subsequences of length 4, and an index $\ell \in [n]$. The sequences are $x = (x_1, x_2, x_3, x_4)$ and $y = (y_1, y_2, y_3, y_4)$. Note that both x and y have the largest gap between consecutive elements at index 2, i.e., $|x_3 - x_2|$ and $|y_3 - y_2|$ are the largest gaps between consecutive indices in x and y. Furthermore, ℓ cuts both x and y with slack.

find a $(12\dots k/2)$-pattern of indices $(i_1, \dots, i_{k/2}) \in [n]^{k/2}$, as well as $(12\dots k/2)$-pattern of indices $(i_{k/2+1}, \dots, i_k) \in [n]^{k/2}$ that can be combined into one $(12\dots k)$-pattern. Toward this recursive approach, we say that an index $\ell \in [n]$ *cuts* $(x_1, \dots, x_k)$ *with slack* if

$$x_{k/2} + \frac{x_{k/2+1} - x_{k/2}}{3} \;\le\; \ell \;\le\; x_{k/2+1} - \frac{x_{k/2+1} - x_{k/2}}{3},$$

or, informally, if ℓ lies "roughly in the middle" between $x_{k/2}$ and $x_{k/2+1}$ – which, by the above assumption, form the largest gap among consecutive indices of the increasing subsequence (see Figure 1). The index $\ell \in [n]$ allows us to recurse on an interval before ℓ, as well as an interval after ℓ. Additionally, the *width* of $(x_1, \dots, x_k)$ is set to be $\lfloor \log(x_{k/2+1} - x_{k/2}) \rfloor$. We consider the subset of $\mathcal{C}$ consisting of length-k monotone subsequences of width w which are cut by ℓ with slack,

$$\mathcal{C}_{\ell,w} = \{(x_1, \dots, x_k) \in \mathcal{C} : \text{width}(x_1, \dots, x_k) = w,\ \ell \text{ cuts } (x_1, \dots, x_k) \text{ with slack}\},$$

and note that if $(x_1, \dots, x_k) \in \mathcal{C}_{\ell,w}$, then $x_1, \dots, x_{k/2} \in [\ell - k \cdot 2^w, \ell]$ and $x_{k/2+1}, \dots, x_k \in [\ell, \ell + k \cdot 2^w]$, since $|x_{k/2+1} - x_{k/2}|$ was maximal. Motivated by this observation, the *density* of width-w copies in $\mathcal{C}$ around ℓ is measured by

$$\tau_{\mathcal{C}}(\ell, w) = \frac{1}{2^w} \cdot |\mathcal{C}_{\ell,w}|,$$

and the total density (over all widths) of $\mathcal{C}$ around ℓ is measured by

$$\tau_{\mathcal{C}}(\ell) = \sum_{w=1}^{\log n} \tau_{\mathcal{C}}(\ell, w).$$

The algorithms (ours and those in [45, 7]) proceed in a recursive manner. Each step considers an index $\ell \in [n]$ where the total density $\tau_{\mathcal{C}}(\ell)$ is high, namely at least $\Omega_k(\varepsilon)$, as well as a width w where $\tau_{\mathcal{C}}(\ell, w)$ is high. At a very high level, the algorithm can recurse on the sub-intervals $[\ell - k \cdot 2^w, \ell]$ and $[\ell, \ell + k \cdot 2^w]$, where the lower bound on $\tau_{\mathcal{C}}(\ell, w)$ implies sufficiently many increasing subsequences exist in each interval. If we choose the index ℓ and width w correctly, we have reduced the problem of finding a $(12\dots k)$-pattern to finding two $(12\dots k/2)$-patterns in subsequences of size $k \cdot 2^w$ to the left and right of ℓ which are themselves $\Omega_{\varepsilon,k}(1)$-far from free of $(12\dots, k/2)$-patterns.

While ℓ may be chosen randomly, choosing the correct width w becomes analytically trickier, and is the step where the algorithms differ. The number of possible widths w is $\Theta(\log n)$ (since these are powers of 2 between 1 and n), and a *non-adaptive* algorithm cannot know what a correct choice of w is. The non-adaptive algorithms consider all $\Theta(\log n)$ options

and recursively apply the algorithm for each width, thereby losing a $\Theta(\log n)$ factor in the query complexity at each recursive step. The main challenge of [45, 7] is obtaining the "best" lower bound on $\tau_{\mathcal{C}}(\ell, w)$ for some $w \in [\log n]$ and determining the number of recursive steps necessary. The fact that a non-adaptive algorithm must explore $\Omega(\log n)$ widths is inevitable, and what the non-adapive lower bound in [7] formalizes.

With adaptivity, the hope is that an algorithm considering an index $\ell \in [n]$ with $\tau_{\mathcal{C}}(\ell) = \Omega_k(\varepsilon)$ can choose *one* width w satisfying $\tau_{\mathcal{C}}(\ell, w) = \Omega_k(\varepsilon)$, and recurse only on that width. The algorithm may devote $\Theta_{k,\varepsilon}(\log n)$ queries to consider all $\Theta(\log n)$ possible widths, and the benefit is that recursing on a single width incurs a $\Theta_{k,\varepsilon}(\log n)$ *additive* loss in the query complexity, as opposed to the $\Theta_{k,\varepsilon}(\log n)$ multiplicative loss incurred by [45, 7]. We describe how we accomplish this next.

First, there is a simple $O_{k,\varepsilon}(\log n)$-query procedure which can choose a width $\widehat{w}$ where $\widehat{w} \geq w$. For example, for every possible width w_0, the algorithm queries $O_{k,\varepsilon}(1)$ randomly sampled indices from $[\ell - k \cdot 2^{w_0}, \ell]$ and $[\ell, \ell + k \cdot 2^{w_0}]$. Then, let $\widehat{w}$ be the largest w_0 where some increasing pair is found. The fact that the unknown $w \in [\log n]$ satisfies $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ implies that with high constant probability, there exists two $(x_1, \ldots, x_k), (y_1, \ldots, y_k) \in \mathcal{C}_{\ell,w}$ where indices x_1 and y_k are sampled and by an observation from [45], with high enough probability, $f(x_1) \leq f(y_k)$ (see the appendix in the full-version for a more thorough discussion on this point). This, in turn, implies $\widehat{w} \geq w$.

If the simple procedure happened to choose $\widehat{w}$ which is not much larger than w, then we may recurse on $\widehat{w}$, similarly to [45, 7]; we call this the *fitting* case. The problem is that $\widehat{w}$ may be too large, a case we refer to as *overshooting*. Consider the execution selecting a width $\widehat{w}$ which is too large, in particular, the "correct" width w satisfies $w \ll \widehat{w}$. Intuitively, the problem is the following: the promise that $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ ensures that the subsequence $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ is sufficiently dense with $(12\ldots k)$-patterns; however, when $\widehat{w}$ is much larger, the subsequence $[\ell - k \cdot 2^{\widehat{w}}, \ell + k \cdot 2^{\widehat{w}}]$ is much larger than the subsequence $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$; hence, the length-$k$ increasing subsequences in $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ constitute a tiny (at most $O_k(2^{w-\widehat{w}})$) fraction of the interval $[\ell - k \cdot 2^{\widehat{w}}, \ell + k \cdot 2^{\widehat{w}}]$ the algorithm would recurse on.

Due to the density $\tau_{\mathcal{C}}(\ell, \widehat{w})$ being potentially very small, at this point, it is not clear how to proceed with our wrong (too large) choice of $\widehat{w}$ as the width to recurse on. To overcome this, we prove a robust structural theorem, drawing a much more favorable picture as to which widths are good for recursion. The robust structural theorem asserts the following. For sufficiently many possible $\ell \in [n]$ and widths w where $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$, *every* interval J containing $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ has $\Omega_k(\varepsilon|J|)$ pairwise-disjoint length-k increasing subsequences. At a high level, the prior structural results ensured that $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ is dense with $(12\ldots k)$-patterns cut by ℓ; our robust version ensures that any interval J containing $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$ remains dense with $(12\ldots k)$-patterns. In particular, the choice of interval is robust to picking a width $\widehat{w}$ which is larger than w. These length-k increasing subsequences are not cut with slack by ℓ, a condition which was crucial for [45, 7]; however, the algorithm's choice of $\widehat{w}$ means it found an increasing pair at distance $\Theta_k(2^{\widehat{w}})$. We exploit this with an adaptive algorithm in a somewhat surprising manner, which we expand on now.

New algorithm when overshooting

Let $\ell \in [n]$ be an index with $\tau_{\mathcal{C}}(\ell) \geq \Omega_k(\varepsilon)$, and let w be the unknown width where $\tau_{\mathcal{C}}(\ell, w) \geq \Omega_k(\varepsilon)$ with the above-mentioned robustness property. Suppose that the widest increasing pair $(\boldsymbol{x}, \boldsymbol{y})$ found by the algorithm (which sets $\widehat{w} \approx \log_2 |\boldsymbol{y} - \boldsymbol{x}|$), satisfies $\widehat{w} \gg w$.

Even though the algorithm has "committed" to a width $\widehat{w}$ which is too large, we will algorithmically exploit the fact that $(\boldsymbol{x}, \boldsymbol{y})$ is an increasing pair lying very far apart, and containing the interval $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$. Specifically, since $(\boldsymbol{x}, \boldsymbol{y})$ are very far away, the algorithm may fit $k-2$ intervals $J_1, \ldots, J_{k-2}$ between $\boldsymbol{x}$ and $\boldsymbol{y}$ which lie adjacent to each other, satisfying the following conditions:

- J_1 contains the interval $[\ell - k \cdot 2^w, \ell + k \cdot 2^w]$.
- J_{i+1} lies immediately after J_i, for any $i \in [k-3]$.
- $|J_{i+1}| \geq |J_i| \cdot \alpha_{k,\varepsilon}$ for all $i \in [k-3]$, and a large fixed constant $\alpha_{k,\varepsilon} > 1$.

A consequence of the robust structural theorem, and the fact that $J_1, \ldots, J_{k-2}$ have exponentially increasing lengths is that each J_i contains a collection $\mathcal{T}_i$ of $\Omega_k(\varepsilon|J_i|)$ disjoint length-k increasing subsequences. For each $i \in [k-2]$, define two sets $\mathcal{A}_i$ and $\mathcal{B}_i$ as follows. Let $\mathcal{A}_i$ be the collection of prefixes $(a_1, \ldots, a_{i+1})$ of $\mathcal{T}_i$ with $f(a_{i+1}) < f(\boldsymbol{y})$, and let $\mathcal{B}_i$ be the collection of suffixes $(a_{i+1}, \ldots, a_k)$ of $\mathcal{T}_i$ with $f(a_{i+1}) \geq f(\boldsymbol{y})$. As $|\mathcal{T}_i| = |\mathcal{A}_i| + |\mathcal{B}_i|$, one of $\mathcal{A}_i$ and $\mathcal{B}_i$ is large (i.e. has size at least $\Omega_k(\varepsilon|J_i|)$). This seemingly innocent combinatorial idea can be exploited non-trivially to find an increasing subsequence of length k. Specifically, the algorithm to handle overshooting aims to (recursively) find shorter increasing subsequences in $J_1, \ldots, J_{k-2}$, with the hope of combining them together into an increasing subsequence of length k. Concretely, for any $i \in [k-2]$, we make two recursive calls of our algorithm on J_i: one for an $(i+1)$-increasing subsequence in J_i, with values smaller than $f(\boldsymbol{y})$,[3] and a second one for a $(k-i)$-increasing subsequence in J_i whose values are at least $f(\boldsymbol{y})$. By induction, the first recursive call succeeds with good probability if $|\mathcal{A}_i|$ is large, while the second call succeeds with good probability if $|\mathcal{B}_i|$ is large. Since for any i either $|\mathcal{A}_i|$ or $|\mathcal{B}_i|$ must be large, at least one of the following must hold.

- $\mathcal{B}_1$ is large. In this case we are likely to find a length-$(k-1)$ monotone pattern in J_1 with values at least $f(\boldsymbol{y}) > f(\boldsymbol{x})$, which combines with $\boldsymbol{x}$ to form a length-k monotone pattern.
- $\mathcal{A}_{k-2}$ is large. Here we are likely to find a length-$(k-1)$ monotone pattern in J_{k-2} whose values lie below $f(\boldsymbol{y})$, which combines with $\boldsymbol{y}$ to form a length-k monotone pattern.
- There exists $i \in [k-3]$ where both $\mathcal{A}_i$ and $\mathcal{B}_{i+1}$ are large. Here we will find, with good probability, a length-$(i+1)$ monotone pattern in J_i with values below $f(\boldsymbol{y})$, and a length-$(k-i-1)$ monotone pattern in J_{i+1} with values at or above $f(\boldsymbol{y})$; together these two patterns combine to form a $(12\ldots k)$-pattern.

In all cases, a k-increasing subsequence is found with good probability. See Figure 2 for an example. The benefit is that the algorithm spends $\Theta_{k,\varepsilon}(\log n)$ queries to identify one fixed width $\widehat{w} \in [\log n]$. Then, there are $2(k-2)$ recursive calls each aiming to find an increasing subsequence of length strictly less than k. The $\Theta_{\varepsilon,k}(\log n)$ loss in the query complexity is additive per recursive step; this leads to the $\Theta_{\varepsilon,k}(\log n)$ query complexity bound which was impossible in the non-adaptive algorithms of [45, 7], as these had to explore all possible widths $\widehat{w} \in [\log n]$ in each recursive step.

Organization

The rest of the paper is organized as follows. Relevant notation can be found in Section 1.3. Section 2 establishes the stronger structural result required for our adaptive algorithm. Section 3 contains the new algorithmic components and the formal statements regarding the

[3] Technically speaking, our algorithm can be configured to only look for increasing subsequences whose values lie in some range; we use this to make sure that shorter increasing subsequences obtained from the recursive calls of the algorithm can eventually be concatenated into a valid length-k one.

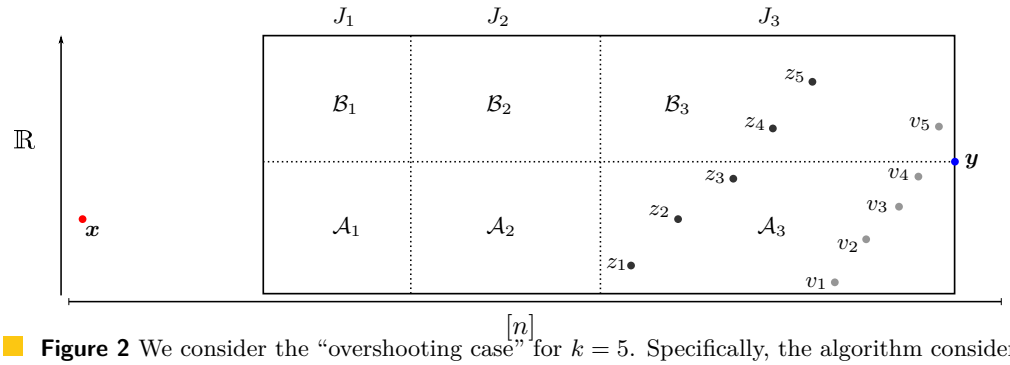


Figure 2 We consider the "overshooting case" for $k = 5$. Specifically, the algorithm considers an index $\ell \in [n]$ with $\tau_{\mathcal{C}}(\ell) = \Omega_k(\varepsilon)$ and, for some unknown $w \in [\log n]$, $\tau_{\mathcal{C}}(\ell, w) = \Omega_k(\varepsilon)$. Furthermore, in trying to identify a correct width $\widehat{w}$, the algorithm samples an increasing pair $(\boldsymbol{x}, \boldsymbol{y})$ with $\log_2 |\boldsymbol{x} - \boldsymbol{y}| \approx \widehat{w} \gg w$. The algorithm will consider at least $k-2$ geometrically increasing intervals between $\boldsymbol{x}$ and $\boldsymbol{y}$; these are displayed as J_1, J_2, and J_3; by virtue of the robust structural theorem, each J_i contains $\Omega_k(\varepsilon|J_i|)$ disjoint length-k monotone subsequences. $\mathcal{A}_i$ contains those length-k monotone subsequences where the $(i+1)$-th index is above $f(\boldsymbol{y})$ and $\mathcal{B}_i$ contains those whose $(i+1)$-th index is below $f(\boldsymbol{y})$. As an example, $(z_1, z_2, z_3, z_4, z_5) \in \mathcal{B}_4$ and $(v_1, v_2, v_3, v_4, v_5) \in \mathcal{A}_4$. The crucial properties are: (i) for all $i \in [k-2]$ any $(12\dots i)$-pattern in $\mathcal{A}_i$ and any $(12\dots(k-i))$-pattern in $\mathcal{B}_{i+1}$ may be combined into a $(12\dots k)$-pattern, (ii) any $(12\dots(k-1))$-pattern in $\mathcal{B}_1$ may be combined with $\boldsymbol{x}$ since $f(\boldsymbol{y}) > f(\boldsymbol{x})$, and (iii) any $(12\dots(k-1))$-pattern in $\mathcal{A}_4$ may be combined with $\boldsymbol{y}$. The reasoning may proceed as follows: if $|\mathcal{B}_1|$ is large, we find a $(12\dots(k-1))$-pattern and combine it with $\boldsymbol{x}$; so, assume $|\mathcal{B}_1|$ is small, which implies $|\mathcal{A}_1|$ must be large. If $|\mathcal{B}_2|$ is large, then a (12)-pattern from $\mathcal{A}_1$ and a $(12\dots(k-2))$-pattern from $\mathcal{B}_2$ may be combined; so assume $|\mathcal{B}_2|$ is small which implies $|\mathcal{A}_2|$ is large, ... Eventually, we deduce that we may assume $|\mathcal{A}_{k-2}|$ is large, and a $(12\dots(k-1))$-pattern in $\mathcal{A}_{k-2}$ may be combined with $\boldsymbol{y}$.

correctness of our algorithm and its query complexity. The appendices in the full-version provide a brief description of the previous (non-adaptive) testing results on $(12\dots k)$-freeness from [45, 7], as well as the remaining proofs, relegated from the main body due to space constraints.

1.3 Notation

All logarithms considered are base 2. We consider functions $f\colon I \to \mathbb{R}$, where $I \subseteq [n]$, as the inputs and main objects of study. An *interval* in $[n]$ is a set $I \subseteq [n]$ of the form $I = \{a, a+1, \dots, b\}$. At many places throughout the paper, we think of augmenting the image with a special character $*$ to consider $f\colon I \to \mathbb{R} \cup \{*\}$. The character $*$ can be thought of as a *masking* operation: In many cases, we will only be interested in entries x of f so that $f(x)$ lies in some prescribed (known in advance) range of values $R \subseteq \mathbb{R}$, so that entries outside this range will be marked by $*$. Whenever the algorithm queries $f(x)$ and observes $*$, it will interpret this as an incomparable value (with respect to ordering) in $\mathbb{R}$. As a result, $*$-values will never be part of monotone subsequences. We note that augmenting the image with $*$ was unnecessary in [45, 7] because they only considered non-adaptive algorithms. We say that for a fixed $f\colon I \to \mathbb{R} \cup \{*\}$, the set T is a collection of disjoint monotone subsequences of length k if it consists of tuples $(i_1, \dots, i_k) \in I^k$, where $i_1 < \dots < i_k$ and $f(i_1) < \dots < f(i_k)$ (in particular, $f(i_1), \dots, f(i_k) \neq *$), and furthermore, for any two tuples $(i_1, \dots, i_k)$ and $(i'_1, \dots, i'_k)$, their intersection (as sets) is empty. We also denote $E(T)$ as the union of indices in k-tuples of T, i.e., $E(T) = \cup_{(i_1,\dots,i_k)\in T}\{i_1, \dots, i_k\}$. Finally, we let $\mathrm{poly}(\cdot)$ denote a large enough polynomial whose degree is (bounded by) a universal constant.

2 Stronger Structural Dichotomy

In this section, we prove a robust structural dichotomy for functions $f\colon [n] \to \mathbb{R}$ that are ε-far from $(12\dots k)$-free, which strengthens the dichotomy proved in [7]. In their paper, it is shown that any f which is ε-far from $(12\dots k)$-free satisfies at least one of two conditions: either f contains many *growing suffixes*, or it can be decomposed into *splittable intervals*. In Section 2.1, we define and describe these notions and state the original (non-robust) structural result from [7]. Then, in Section 2.2, we establish a substantially stronger structural dichotomy, better suited for our purposes. The proof of the stronger dichotomy combines the original one as a black-box with additional combinatorial ideas.

2.1 The Non-Robust Structural Decomposition

For completeness, we first introduce the non-robust structural result from [7]. As the formal definitions are somewhat complicated, we start with an informal description of the growing suffixes and splittable intervals conditions. For the purpose of this discussion, let $\mathcal{C}$ be any collection of $\Theta_{k,\varepsilon}(n)$ disjoint $(12\dots k)$-copies in f. We use the notation from Section 1.2.

- **Growing suffixes:** there exist $\Omega_{k,\varepsilon}(n)$ values of $\ell \in [n]$ where $\tau_{\mathcal{C}}(\ell) \geq \Theta_k(\varepsilon)$ and $\tau_{\mathcal{C}}(\ell, w) \ll \tau_{\mathcal{C}}(\ell)$ for every $w \in [\log n]$. In words, many $\ell \in [n]$ are such that the sum of local densities, $\tau_{\mathcal{C}}(\ell)$, of $(12\dots k)$-patterns in intervals of growing widths is not too small, and furthermore, the densities are not concentrated on any small set of widths w. Any such ℓ is said to be the starting point of a growing suffix.
- **Splittable intervals (non-robust):** there exist $c \in [k-1]$ and a collection of pairwise-disjoint intervals $I_1, \dots, I_s \subset [n]$ with $\sum_{i=1}^{s} |I_i| = \Theta_{k,\varepsilon}(n)$, so that each I_i contains a dense collection of disjoint $(12\dots k)$-patterns of a particular structure. Specifically, each such interval I_i can be partitioned into three disjoint intervals L_i, M_i, R_i (in this order), each of size $\Omega_k(|I_i|)$, where I_i fully contains $\Omega_{k,\varepsilon}(|I_i|)$ disjoint copies of $(12\dots k)$-patterns, in which the first c entries lie in L_i, the last $k-c$ entries lie in R_i (none of these entries lies in M_i), and every such c entry lies below every $c+1$ entry.

Informally, the non-robust structural dichotomy from [7] asserts that any f that is ε-far from $(12\dots k)$-free either satisfies the growing suffixes condition, or the non-robust splittable intervals condition (or both). These two notions are formally defined next; the precise definition for growing suffixes is slightly more complicated than described above (but understanding it is not essential for this work, as the growing suffixes procedure from [7] will eventually only be used as a black box). For what follows, for an index $\ell \in [n]$ define $\eta_\ell = \lceil \log_2(n-\ell) \rceil$, and for any $t \in [\eta_\ell]$ set $S_t(\ell) = [a + 2^{t-1}, a + 2^t) \cap [n]$. Note that the intervals $S_1, \dots, S_{\eta_\ell}$ are a partition of $(\ell, n]$ into intervals of geometrically increasing length (except for maybe the last one). Finally, the tuple $S(\ell) = (S_t(\ell))_{t \in [\eta_\ell]}$ is called the *growing suffix* starting at ℓ.

▶ **Definition 5** (Growing suffixes (see [7], Definition 2.4)). *Let $\alpha, \beta \in [0,1]$. We say that an index $\ell \in [n]$ starts an (α,β)-*growing suffix *if, when considering the collection of intervals $S(\ell) = \{S_t(\ell) : t \in [\eta_\ell]\}$, for each $t \in [\eta_\ell]$ there is a subset $D_t(\ell) \subseteq S_t(\ell)$ of indices such that the following properties hold.*

1. *We have $|D_t(\ell)|/|S_t(\ell)| \leq \alpha$ for all $t \in [\eta_\ell]$, and $\sum_{t=1}^{\eta_\ell} |D_t(\ell)|/|S_t(\ell)| \geq \beta$.*
2. *For every $t, t' \in [\eta_a]$ where $t < t'$, if $a \in D_t(\ell)$ and $a' \in D_{t'}(\ell)$, then $f(a) < f(a')$.*

The second definition, also from [7], describes the (non-robust) splittable intervals setting.

▶ **Definition 6** (Splittable intervals (see [7], Definition 2.5)). *Let $\alpha, \beta \in (0,1]$ and $c \in [k-1]$. Let $I \subseteq [n]$ be an interval, let $T \subseteq I^k$ be a set of disjoint, length-k monotone subsequences of f lying in I, and define*

$$T^{(L)} = \{(i_1, \ldots, i_c) \in I^c : (i_1, \ldots, i_c) \text{ is a prefix of a } k\text{-tuple in } T\}, \text{ and}$$
$$T^{(R)} = \{(j_1, \ldots, j_{k-c}) \in I^{k-c} : (j_1, \ldots, j_{k-c}) \text{ is a suffix of a } k\text{-tuple in } T\}.$$

*We say that the pair (I,T) is (c,α,β)-*splittable *if $|T|/|I| \geq \beta$; $f(i_c) < f(j_1)$ for every $(i_1, \ldots, i_c) \in T^{(L)}$ and $(j_1, \ldots, j_{k-c}) \in T^{(R)}$; and there is a partition of I into three consecutive intervals $L, M, R \subseteq I$ (that appear in this order, from left to right) of size at least $\alpha|I|$, satisfying $T^{(L)} \subseteq L^c$ and $T^{(R)} \subseteq R^{k-c}$.*

A collection of disjoint interval-tuple pairs $(I_1,T_1), \ldots, (I_s,T_s)$ is called a (c,α,β)-splittable collection of T *if each (I_j,T_j) is (c,α,β)-splittable and the sets $(T_j : j \in [s])$ partition T.*

The following theorem presents the growing suffixes versus (non-robust) splittable intervals dichotomy, which is among the main structural results of [7]. We remark that in their paper, the theorem is stated with respect to two parameters, k, k_0; for our purpose it suffices to set $k_0 = k$. Also, here we allow f to take the value $*$, which is not the case in [7]. Nevertheless, as their proof takes into account only the elements of a given family T^0 of disjoint length-k increasing subsequences, which in particular are non-$*$ elements, the same proof works here.

▶ **Theorem 7** ([7], Theorem 2.2). *Let $k, n \in \mathbb{N}$, $\varepsilon \in (0,1)$, and $C > 0$, and let $I \subseteq [n]$ be an interval. Let $f\colon I \to \mathbb{R} \cup \{*\}$ be a function and let $T^0 \subseteq I^k$ be a set of at least $\varepsilon|I|$ disjoint monotone subsequences of f of length k. Then there exist $\alpha \in (0,1)$ and $p > 0$ satisfying $\alpha \geq \Omega(\varepsilon/k^5)$ and $p \leq \mathrm{poly}(k \log(1/\varepsilon))$ such that at least one of the following conditions holds.*

1. ***Growing suffixes:*** *There exists a set $H \subseteq [n]$, of indices that start an $(\alpha, Ck\alpha)$-growing suffix, satisfying $\alpha|H| \geq (\varepsilon/p)n$.*
2. ***Splittable intervals (non-robust):*** *There exist a positive integer $c < k$, a set T of disjoint length-k monotone subsequences satisfying $E(T) \subseteq E(T^0)$, and a $(c, 1/(6k), \alpha)$-splittable collection of T consisting of disjoint interval-tuple pairs $(I_1,T_1), \ldots, (I_s,T_s)$, such that $\alpha \sum_{h=1}^{s} |I_h| \geq |T^0|/p$.*

2.2 Robustifying the Structural Result

We are now ready to establish the robust structural foundations – specifically, a *growing suffixes* versus *robust splittable intervals* dichotomy – lying at the heart of our adaptive algorithm. The next lemma will eventually imply that the splittable intervals condition can be robustified by merely throwing away a subset of "bad" splittable intervals.

▶ **Lemma 8.** *Let $\alpha \in (0,1)$ and let $I \subset \mathbb{N}$ be an interval. Suppose that $I_1, \ldots, I_s \subset I$ are disjoint intervals such that $\sum_{h=1}^{s} |I_h| \geq \alpha|I|$. Then there exists a set $G \subset [s]$ such that $\sum_{h \in G} |I_h| \geq (\alpha/4)|I|$, and for every interval $J \subset I$ that contains an interval I_h with $h \in G$, $\sum_{h \in [s]\colon I_h \subset J} |I_h| \geq (\alpha/4)|J|$.*

The full proof appears in the Appendix of the full verion. The idea is to consider a minimal subset $\mathcal{J}$ of the collection of all "problematic" intervals J which *do not satisfy* the conditions of the lemma. For each $J \in \mathcal{J}$, less than an $\alpha/4$-fraction of J is covered by intervals from $\mathcal{I} = \{I_1, \ldots, I_s\}$. Conversely, as we show, the minimality of $\mathcal{J}$ entails that any element in I is covered by at most three intervals from $\mathcal{J}$. The combination of

these conditions implies that, if we remove from $\mathcal{I}$ all intervals I_j contained in some interval $J \in \mathcal{J}$, then at the end of the process $\sum_{I_j \in \mathcal{I}} |I_j| = \Omega(\alpha|I|)$, and no "problematic" choices of J survive. Thus, the set of surviving intervals from $\mathcal{I}$ satisfy the conditions of the lemma.

The robust version of the structural dichotomy is stated below; for the proof, combining the basic structural dichotomy with the last lemma, see the appendices of the full-version.

▶ **Theorem 9** (Robust structural theorem). *Let $k, n \in \mathbb{N}$, $\varepsilon \in (0,1)$, and $C > 0$, and let $I \subseteq [n]$ be an interval. Let $f : I \to \mathbb{R} \cup \{*\}$ be an array and let $T^0 \subseteq I^k$ be a set of at least $\varepsilon|I|$ disjoint length-k monotone subsequences of f. Then there exist $\alpha \in (0,1)$ and $p > 0$ with $\alpha \geq \Omega(\varepsilon/k^5)$ and $p \leq \mathrm{poly}(k \log(1/\varepsilon))$ such that at least one of the following holds.*

1. ***Growing suffixes:*** *There exists a set $H \subseteq [n]$, of indices that start an $(\alpha, Ck\alpha)$-growing suffix, satisfying $\alpha|H| \geq (\varepsilon/p)n$.*
2. ***Robust splittable intervals:*** *There exist an integer c with $1 \leq c < k$, a set T, with $E(T) \subseteq E(T^0)$, of disjoint length-k monotone subsequences, and a $(c, 1/(6k), \alpha)$-splittable collection of T, consisting of disjoint interval-tuple pairs $(I_1, T_1), \ldots, (I_s, T_s)$, such that*

$$\alpha \sum_{h=1}^{s} |I_h| \geq \frac{\varepsilon}{p} \cdot |I|.$$

Moreover, if $J \subset I$ is an interval where $J \supset I_h$ for some $h \in [s]$, then J contains at least $(\varepsilon/p)|J|$ disjoint $(12\ldots k)$-patterns from T^0.

3 The Algorithm

In this section we prove the existence of a randomized algorithm, $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$, that receives as input a function $f : I \to \mathbb{R} \cup \{*\}$ (where $I \subset \mathbb{N}$ is an interval), and parameters $\varepsilon, \delta \in (0,1)$, and satisfies the following: if f contains $\varepsilon|I|$ disjoint $(12\ldots k)$-patterns, then the algorithm outputs such a pattern with probability at least $1 - \delta$. The running time is $O_{k,\varepsilon}(\log n)$. The algorithm is described in Figure 5 below. It uses three subroutines: `Sample-Suffix`, `Find-Within-Interval`, and `Find-Good-Split`, the first of which is given in [7], and the latter two are described below, in Figures 3 and 4. The majority of the section is devoted to the proof that `Find-Monotone` indeed outputs a $(12\ldots k)$-pattern with high probability as claimed. Specifically, we shall prove the following theorem.

▶ **Theorem 10.** *Let $k \in \mathbb{N}$. The randomized algorithm $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$, described in Figure 5, satisfies the following. Given a function $f : I \to \mathbb{R} \cup \{*\}$ and parameters $\varepsilon, \delta \in (0,1)$, if f contains at least $\varepsilon|I|$ disjoint $(12\ldots k)$-patterns, then $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$ outputs a $(12\ldots k)$-pattern of f with probability at least $1 - \delta$.*

Our proof proceeds by induction on k. It relies on Lemmas 12, 13, 14, the former is taken from [7] whereas the proofs of the latter two assume that Theorem 10 holds for smaller k. We first state and prove these lemmas, and then we prove Theorem 10.

To complete the picture, in the following lemma we provide an upper bound on the query complexity and running time of `Find-Monotone`. For the proof, see the appendices of the full version.

▶ **Lemma 11.** *Let $f : I \to \mathbb{R} \cup \{*\}$, where I is an interval of length at most n. The query complexity and running time of $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$ are at most*

$$\left(k^k \cdot (\log(1/\varepsilon))^k \cdot \frac{1}{\varepsilon} \cdot \log(1/\delta)\right)^{O(k)} \cdot \log n.$$

3.1 The `Sample-Suffix` Sub-Routine

We restate Lemma 3.1 from [7] which gives the $\texttt{Sample-Suffix}_k$ subroutine, with a few adaptations to fit our needs.

▶ **Lemma 12** ([7]). *Fix $k \in \mathbb{N}$ and let $C > 0$ be a large enough constant. There exists a non-adaptive and randomized algorithm, $\texttt{Sample-Suffix}_k(f, \varepsilon, \delta)$ which takes three inputs: query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, where $I \subset \mathbb{N}$ is an interval, a parameter $\varepsilon \in (0,1)$, and an error probability bound $\delta \in (0,1)$. Suppose there exists $\alpha \in (0,1)$, and a set $H \subseteq I$ of $(\alpha, Ck\alpha)$-growing suffixes of f satisfying $\alpha|H| \geq \varepsilon|I|$. Then, $\texttt{Sample-Suffix}_k(f, \varepsilon, \delta)$ finds a length-k monotone subsequence of f with probability at least $1 - \delta$. The query complexity of $\texttt{Sample-Suffix}_k(f, \varepsilon, \delta)$ is at most $\log(1/\delta) \cdot \mathrm{polylog}(1/\varepsilon) \cdot \frac{1}{\varepsilon} \cdot \log n$.*

For additional technical remarks about Lemma 12 and `Sample-Suffix`, see the appendices of the full versions.

3.2 Handling Overshooting: The `Find-Within-Interval` Sub-Routine

In this section, we describe the `Find-Within-Interval` subroutine, addressing the overshooting case as explained in Section 1.2. As the algorithm may appear unintuitive, let us

Subroutine $\texttt{Find-Within-Interval}_k(f, \varepsilon, \delta, x, y, \mathcal{J})$.

Input: Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$, two inputs $x, y \in I$ where $x < y$ and $f(x) < f(y)$, and $\mathcal{J} = (J_1, \ldots, J_{k-2})$ which is a collection of disjoint intervals appearing in order inside $[x, y]$.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. For every $\kappa \in [k-2]$, let $f_\kappa, f'_\kappa\colon J_\kappa \to \mathbb{R} \cup \{*\}$ be given by:

$$f_\kappa(i) = \begin{cases} f(i) & f(i) < f(y) \\ * & \text{o.w.} \end{cases} \qquad \text{and} \qquad f'_\kappa(i) = \begin{cases} f(i) & f(i) \geq f(y) \\ * & \text{o.w.} \end{cases} \tag{1}$$

2. Call $\texttt{Find-Monotone}_{\kappa+1}(f_\kappa, \varepsilon/2, \delta/(2k))$ for every $\kappa \in [k-2]$.
3. Call $\texttt{Find-Monotone}_{k-\kappa}(f'_\kappa, \varepsilon/2, \delta/(2k))$ for every $\kappa \in [k-2]$.
4. Consider the set of all indices that are output in Lines 2 and 3, together with x and y. If $\mathcal{S}$ contains a length-k increasing subsequence among these indices, output it. Otherwise, output fail.

Figure 3 Description of the `Find-Within-Interval` subroutine.

remind the reader of the setup in which this subroutine is relevant (see also Section 1.2). By Theorem 9, either the growing suffixes condition or the splittable intervals condition hold. The former case is handled by Lemma 12, so we assume that the latter holds. Now assume that we sampled an element $\boldsymbol{x}$ which is the first element of a length-c increasing subsequence from a set L_i as described in Definition 6. We then sample, uniformly at random, elements $\boldsymbol{y}$ from $[\boldsymbol{x}, \boldsymbol{x} + 2^t]$. The splittable intervals condition implies that we will find, with high probability, an element $\boldsymbol{y}$ which is the last element of a length-$(k-c)$ increasing subsequence from R_i. In particular, $f(\boldsymbol{y}) > f(\boldsymbol{x})$. However, even if we did indeed sample such $\boldsymbol{y}$, we may have sampled many other values of $\boldsymbol{y}'$ with $f(\boldsymbol{y}') > f(\boldsymbol{x})$, and we do not know of a way of determining which of these values is the "correct" one. Instead, we take $\boldsymbol{y}_0$ to be the largest sampled $\boldsymbol{y}'$ such that $f(\boldsymbol{y}') > f(\boldsymbol{x})$. The case where $\boldsymbol{y}_0$ is close to $\boldsymbol{y}$ is taken care of by Lemma 13, so we assume that $\boldsymbol{y}_0$ is much larger than $\boldsymbol{y}$.

We now have elements $\boldsymbol{x}$ and $\boldsymbol{y}_0$, and all that we know is that they contain a large portion of an interval I_i from the splittable intervals condition. It is not hard to see (this is shown in the proof of Theorem 10) that $[\boldsymbol{x}, \boldsymbol{y}_0]$ can be partitioned into $k-2$ intervals $J_1, \ldots, J_{k-2}$, each of which contains many disjoint length-k increasing subsequences. To continue, out only hope is use the induction hypothesis to find shorter increasing subsequences in the intervals. For example, if there are many disjoint length-$(k-1)$ increasing subsequences in J_1 that lie above $\boldsymbol{x}$, then one such subsequence is likely to be detected by a recursive call to the main algorithm, and together with $\boldsymbol{x}$ it will form a length-k increasing subsequence. If there are few such length-$(k-1)$ subsequences, this means that there are many disjoint length-2 increasing subsequences in J_1 that lie below $\boldsymbol{x}$ (because for every length-k increasing subsequence, either its $(k-1)$-suffix lies above $\boldsymbol{x}$, or its 2-prefix lies above $\boldsymbol{x}$). We can then use a recursive call to detect such a sequence, and hope to complete it to a length-k subsequence using a length-$(k-2)$ subsequnece from J_2 that lies above $\boldsymbol{x}$. Continuing with this logic, it follows that with high probability we can find an increasing subsequence of length k using $\boldsymbol{x}$ and J_1, J_i and J_{i+1} for some i, or J_{k-2} and $\boldsymbol{y}_0$.

▶ **Lemma 13.** *Consider the randomized algorithm,* $\texttt{Find-Within-Interval}_k(f, \varepsilon, \delta, x, y, \mathcal{J})$, *described in Figure 3, which takes six inputs:*

- *Query access to a function* $f \colon I \to \mathbb{R} \cup \{*\}$,
- *Two parameters* $\varepsilon, \delta \in (0,1)$,
- *Two points* $x, y \in I$ *where* $x < y$ *and* $f(x) < f(y)$, *and*
- *A collection* $\mathcal{J} = (J_1, \ldots, J_{k-2})$ *of* $k-2$ *disjoint intervals that appear in order (i.e.,* J_κ *comes before* $J_{\kappa+1}$*) within the interval* $[x, y]$,

and outputs either a length-k increasing subsequence of f, *or* fail.

Suppose that for every $\kappa \in [k-2]$, *the function* $f|_{J_\kappa} \colon J_\kappa \to \mathbb{R} \cup \{*\}$, *contains* $\varepsilon |J_\kappa|$ *disjoint* $(12\ldots k)$*-patterns. Then, assuming that Theorem 10 holds for every* k' *with* $1 \le k' < k$, *the procedure* $\texttt{Find-Within-Interval}_k(f, \varepsilon, \delta, x, y, \mathcal{J})$ *outputs a length-k monotone subsequence of* f *with probability at least* $1 - \delta$.

The full proof appears in the appendices of the full version.

3.3 Handling the Fitting Case: The `Find-Good-Split` Sub-Routine

In this section, we describe the `Find-Good-Split` subroutine, which corresponds to the fitting case from Section 1.2. The proof of the lemma below appears in the appendices of the full version.

▶ **Lemma 14.** *Consider the randomized algorithm* $\texttt{Find-Good-Split}_k(f, \varepsilon, \delta, c, \xi)$, *described in Figure 4, which takes as input five parameters: (i) query access to a function* $f \colon I \to \mathbb{R} \cup \{*\}$; *(ii) two parameters* $\varepsilon, \delta \in (0,1)$; *(iii) an integer* $c \in [k-1]$; *and (iv) a parameter* $\xi \in (0,1]$; *and outputs either a length-k increasing subsequence or* fail.

Suppose that there exists an interval-tuple pair (I', T) *which is* $(c, 1/(6k), \varepsilon)$*-splittable and* $|I'|/|I| \ge \xi$. *Then, the algorithms* $\texttt{Find-Good-Split}_k(f, \varepsilon, \delta, c, \xi)$ *finds a* $(12\ldots k)$*-pattern of* f *with probability* $1 - \delta$.

3.4 The Main Algorithm

Consider the description of the main algorithm in Figure 5. The proof uses Lemma 12, Lemma 13, and Lemma 14.

Subroutine $\texttt{Find-Good-Split}_k(f, \varepsilon, \delta, c, \xi)$.

Input: Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$, and $c \in [k-1]$. We let $c_1 > 1$ be a large enough (absolute) constant.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. Repeat the following procedure $t = c_1 k/(\varepsilon\xi^2) \cdot \log(1/\delta)$ times:
   a. Sample $\boldsymbol{w}, \boldsymbol{z} \sim I$, and consider the functions $f_{\boldsymbol{z},\boldsymbol{w}}\colon I \cap (-\infty, \boldsymbol{z}) \to \mathbb{R} \cup \{*\}$ and $f'_{\boldsymbol{z},\boldsymbol{w}}\colon I \cap [\boldsymbol{z}, \infty) \to \mathbb{R} \cup \{*\}$ given by

$$f_{\boldsymbol{z},\boldsymbol{w}}(i) = \begin{cases} f(i) & f(i) < f(\boldsymbol{w}) \\ * & \text{o.w.} \end{cases} \quad \text{and} \quad f'_{\boldsymbol{z},\boldsymbol{w}}(i) = \begin{cases} f(i) & f(i) \geq f(\boldsymbol{w}) \\ * & \text{o.w.} \end{cases} . \quad (2)$$

   b. Run $\texttt{Find-Monotone}_c(f_{\boldsymbol{z},\boldsymbol{w}}, \varepsilon\xi/3, \delta/3)$ and $\texttt{Find-Monotone}_{k-c}(f'_{\boldsymbol{z},\boldsymbol{w}}, \varepsilon\xi/3, \delta/3)$.
2. If both runs of Line 1b are successful for some iteration and some $\boldsymbol{w}$, $\boldsymbol{z}$ and c, then we output the combination of their outputs which forms a length-k increasing subsequence of f; otherwise, output fail.

Figure 4 Description of the `Find-Good-Split` subroutine.

Proof of Theorem 10. The proof is by induction on k. For the base case of $k = 1$, recall that f has at least $\varepsilon|I|$ non-$*$ values. Thus, with probability at least $1 - \delta$, a non-$*$ value is observed after sampling $\boldsymbol{x} \sim I$ at least $(1/\varepsilon) \cdot \log(1/\delta)$ times. It follows that with probability at least $1 - \delta$, Line 2a of our main algorithm, given in Figure 5, samples $\boldsymbol{x} \neq *$ in one of its iterations. We next proceed to the inductive Step: namely, we prove Theorem 10 for $k \geq 2$, under the assumption that it holds for every k' with $1 \leq k' < k$.

Let $p = P(k \log(1/\varepsilon))$ (recall that $P(\cdot)$ is a polynomial of sufficiently large (constant) degree). Apply Theorem 9 to f.

Suppose, first, that (1) of Theorem 9 holds. So, there exists a set $H \subset [n]$ of indices that start an $(\alpha, Ck\alpha)$-growing suffix, with $\alpha|H| \geq (\varepsilon/p)n$, for some $\alpha \in (0,1)$. By Lemma 12, the call for $\texttt{Sample-Suffix}_k(f, \varepsilon/p, \delta)$ in Line 1 outputs a length-k monotone subsequence of f with probability at least $1 - \delta$.

Now suppose that (2) of Theorem 9 holds, and let $(I_1, T_1), \ldots, (I_s, T_s)$ be a $(c, 1/(6k), \alpha)$-splittable collection for some $\alpha \geq \Omega(\varepsilon/k^5)$ and $c \in [k-1]$, satisfying the robust splittable intervals condition and, moreover, that any $J \subset I$ with $J \supset I_h$ for some $h \in [s]$ contains $(\varepsilon/p)|J|$ disjoint $(12\ldots k)$-patterns. Let `Event` be the event that, for a particular iteration of Lines 2a and 2b, $\boldsymbol{x}$ is the 1-entry of some k-tuple from T_h, for some $h \in [s]$, and $\boldsymbol{y}_t$ is the $(c+1)$-entry of some (possibly other) k-tuple in T_h, where t is such that $|I_h| \leq 2^t < 2|I_h|$.

▷ Claim 15. $\mathbf{Pr}[\texttt{Event}] \geq \varepsilon\alpha/(2p)$.

Proof. For each $h \in [s]$, let A_h and B_h be the collections of 1- and $(c+1)$-entries of patterns in T_h. Then

$$\sum_{h=1}^{s} |A_h| = \sum_{h=1}^{s} |T_h| \geq \alpha \sum_{h=1}^{s} |I_h| \geq \frac{\varepsilon}{p} \cdot |I|.$$

The first inequality follows from the assumption that (I_h, T_h) is $(c, 1/(6k), \alpha)$-splittable, and the second inequality follows from the assumption that the robust splittable condition of Theorem 9 holds.

Subroutine $\texttt{Find-Monotone}_k(f, \varepsilon, \delta)$.

Input: Query access to a function $f\colon I \to \mathbb{R} \cup \{*\}$, parameters $\varepsilon, \delta \in (0,1)$. We let $c_1, c_2, c_3 > 0$ be large enough constants, and let $p = P(k \log(1/\varepsilon))$, where $P\colon \mathbb{R} \to \mathbb{R}$ is a polynomial of large enough (constant) degree.

Output: a sequence $i_1 < \ldots < i_k$ with $f(i_1) < \ldots < f(i_k)$, or fail.

1. Run $\texttt{Sample-Suffix}_k(f, \varepsilon/p, \delta)$.
2. Repeat the following for $c_1 \log(1/\delta) \cdot p \cdot k^5/\varepsilon^2$ many iterations:

   a. Sample $\boldsymbol{x} \sim I$ uniformly at random. If $f(\boldsymbol{x}) = *$, proceed to the next iteration. Otherwise, if $k = 1$ output $\boldsymbol{x}$ and proceed to Step 3, and if $k \geq 2$ proceed to the next step.

   b. For each $t \in [\log n]$, sample $\boldsymbol{y}_t \sim [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ uniformly at random. If there exists at least one t where $f(\boldsymbol{y}_t) > f(\boldsymbol{x})$, set

   $$\boldsymbol{y} = \max\left\{\boldsymbol{y}_t : t \in [\log n] \text{ and } f(\boldsymbol{y}_t) > f(\boldsymbol{x})\right\}, \tag{3}$$

   let $t^* \in [\log n]$ be the index for which $\boldsymbol{y}_{t^*} = \boldsymbol{y}$, and continue to the next line. Otherwise, i.e. if $f(\boldsymbol{y}_t) \not> f(\boldsymbol{x})$ for every t, continue to the next iteration.

   c. If $k = 2$, output $(\boldsymbol{x}, \boldsymbol{y})$ and proceed to Step 3. If $k > 2$, continue to the next line.

   d. Here $k \geq 3$. Set $\ell = 4p/\varepsilon$ and perform the following.

      i. Consider the collection $\mathcal{J}$ of $k-2$ intervals $J_1, \ldots, J_{k-2}$ appearing in order within $[\boldsymbol{x}, \boldsymbol{y}]$, given by setting, for every $i \in [k-2]$,

      $$J_i = \left[\boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-1-i)}, \boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)}\right), \tag{4}$$

      and run $\texttt{Find-Within-Interval}_k(f, \varepsilon/2p, \delta/2, \boldsymbol{x}, \boldsymbol{y}, \mathcal{J})$.

      ii. For each $t' \in [t^* - 3k \log \ell, t^*]$ do the following.

      Consider the interval $J_{t'} = [\boldsymbol{x} - 2^{t'}, \boldsymbol{x} + 2^{t'}]$, and the restricted function $g_{t'}\colon J_{t'} \to \mathbb{R} \cup \{*\}$ given by $g_{t'} = f|_{J_{t'}}$. For every $c_0 \in [k-1]$, run $\texttt{Find-Good-Split}_k(g_{t'}, \varepsilon/(c_2 k^5), \delta/2, c_0, 1/4)$.

3. If a length-k monotone subsequence of f is found, output it. Otherwise, output fail.

Figure 5 Description of the Find-Monotone subroutine.

As a result, the probability over the draw of $\boldsymbol{x} \sim I$ in Line 2a that $\boldsymbol{x} \in A_h$ is at least ε/p. Fix such an $\boldsymbol{x}$, and consider $t \in [\log n]$ for which $|I_h| \leq 2^t < 2|I_h|$. Notice that $B_h \subset [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ since $2^{t-1} \leq |I_h| < 2^t$, and that the distance between any index of A_h and B_h is at least $|I_h|/(6k) \geq 2^t/(12k)$ since (I_h, T_h) is $(c, 1/(6k), \alpha)$-splittable. Therefore, the probability over the draw of $\boldsymbol{y}_t \sim [\boldsymbol{x} + 2^t/(12k), \boldsymbol{x} + 2^t]$ that $\boldsymbol{y}_t \in B_h$ is at least $|B_h|/2^t \geq |T_h|/(2|I_h|) \geq \alpha/2$. ◁

By the previous claim, since we have $c_1 \cdot \log(1/\delta) \cdot p \cdot k^5/\varepsilon^2$ iterations of Lines 2a and 2b, with probability at least $1 - \delta/2$, Event holds in some iteration (using the lower bound $\alpha \geq \Omega(\varepsilon/k^5)$ and the choice of c_1 as a large constant).

Consider the first execution of Line 2a and Line 2b where `Event` holds (assuming such an execution exists). Let $h \in [s]$ and $t \in [\log n]$ be the corresponding parameters, i.e., h and t are set so $\boldsymbol{x}$ is the first index of a k-tuple in T_h, $\boldsymbol{y}_t$ is the $(c+1)$-th index in another k-tuple in T_h, and $|I_h| \leq 2^t < 2|I_h|$. We consider this iteration of Line 2, and assume that `Event` holds with these parameters for the rest of the proof. Notice that $\boldsymbol{y}$, as defined in (3), satisfies $\boldsymbol{y} \geq \boldsymbol{y}_t$ (as $f(\boldsymbol{y}) > f(\boldsymbol{x})$) and hence $t^* \geq t$.

Note that if $k = 2$, the pair $(\boldsymbol{x}, \boldsymbol{y})$, which is a (12)-pattern in f, is output in Line 2c, so the proof is complete in this case. From now on, we assume that $k \geq 3$. We break up the analysis into two cases: $t^* \geq t + 3k \log \ell$ and $t^* < t + 3k \log \ell$.

Suppose $t^* \geq t + 3k \log \ell$. We now observe a few facts about the collection $\mathcal{J}$ specified in (4). First, notice that $J_1, \ldots, J_{k-2}$ appear in order from left-to-right, and they lie in $[\boldsymbol{x}, \boldsymbol{y}]$ (as $\boldsymbol{y} = \boldsymbol{y}_{t^*} \in [\boldsymbol{x} + 2^{t^*}/(12k), 2^{t^*}]$). Second, in the next claim we show that for every $i \in [k-2]$, the interval J_i contains $(\varepsilon/2p)|J_i|$ disjoint $(12\ldots k)$-patterns.

▷ Claim 16. J_i contains $(\varepsilon/2p)|J_i|$ disjoint $(12\ldots k)$-patterns.

Proof. Let J_i' be the interval given by $J_i' = I_h \cup \left[\boldsymbol{x}, \boldsymbol{x} + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)}\right]$. Observe that

$$|J_i' \setminus J_i| \leq 2^t + \frac{2^{t^*}}{12k} \cdot \ell^{-(k-1-i)} \leq \frac{2^{t^*}}{6k} \cdot \ell^{-(k-1-i)} = \frac{2}{\ell} \cdot \frac{2^{t^*}}{12k} \cdot \ell^{-(k-2-i)} \geq \frac{2}{\ell} \cdot |J_i'| = \frac{\varepsilon}{2p} \cdot |J_i'|,$$

where for the second inequality we used the bound $t^* - t \geq 3k \log \ell \geq \log(12) + \log k + (k-2)\log \ell$, and that $\ell = 4p/\varepsilon$. By Theorem 9, J_i' contains at least $(\varepsilon/p)|J_i'|$ disjoint $(12\ldots k)$-patterns in f. Hence, the number of disjoint $(12\ldots k)$-patterns in J_i is at least:

$$\frac{\varepsilon}{p} \cdot |J_i'| - |J_i' \setminus J_i| \geq \frac{\varepsilon}{2p} \cdot |J_i'| \geq \frac{\varepsilon}{2p} \cdot |J_i|,$$

as required. ◁

By Lemma 13, Line 2(d)i outputs a $(12\ldots k)$-pattern in f with probability at least $1 - \delta/2$. By a union bound, we obtain the desired result.

Suppose, on the other hand, that $t^* \leq t + 3k \log \ell$. In this case, as $2^{t-1} \leq |I_h| \leq 2^{t^*}$ (by choice of t), for one of the values of t' considered in Line 2(d)ii we have $2^{t'-1} \leq |I_h| < 2^{t'}$; fix this t'. The interval $J_{t'}$, defined in Line 2(d)ii, hence satisfies $|I_h|/|J_{t'}| \geq 1/4$. As a result, and since $I_h \subset J_{t'}$ (because $t \leq t^*$), the function $g\colon J \to \mathbb{R} \cup \{*\}$ contains an interval-tuple pair (I_h, T_h) which is $(c, 1/(6k), \alpha)$-splittable. By Lemma 14, once Line 2(d)ii considers $c_0 = c$, the sub-routine $\texttt{Find-Good-Split}_k(g, \varepsilon/(c_2 k^5), \delta/2, c, 1/4)$ will output a $(12\ldots k)$-pattern of $g_{t'}$ (which is also a $(12\ldots k)$-pattern of f) with probability at least $1 - \delta/2$. Hence, we obtain the result by a union bound. ◀

References

1 Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31(3):371–383, 2007.

2 Alexandr Andoni, Negev Shekel Nosatzki, Sandip Sinha, and Clifford Stein. Estimating the longest increasing subsequence in nearly optimal time. *arXiv preprint*, 2021. `arXiv:2112.05106`.

3 Aleksandrs Belovs. Adaptive Lower Bound for Testing Monotonicity on the Line. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 31:1–31:10, 2018.

4 Aleksandrs Belovs and Eric Blais. Quantum algorithm for monotonicity testing on the hypercube. *Theory of Computing*, 11(16):403–412, 2015.

5 Omri Ben-Eliezer. Testing local properties of arrays. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 11:1–11:20, 2019.

6 Omri Ben-Eliezer and Clément L. Canonne. Improved bounds for testing forbidden order patterns. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2093–2112, 2018.

7 Omri Ben-Eliezer, Clément L. Canonne, Shoham Letzter, and Erik Waingarten. Finding monotone patterns in sublinear time. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1469–1494. IEEE Computer Society, 2019.

8 Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and Counting Permutations via CSPs. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

9 Hadley Black, Deeparnab Chakrabarty, and C. Seshadhri. A $o(d) \cdot \text{polylog} n$ monotonicity tester for boolean functions over the hypergrid $[n]^d$. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2133–2151, 2018.

10 Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

11 Eric Blais, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Lower bounds for testing properties of functions over hypergrid domains. In *Proceedings of the 29th Conference on Computational Complexity (CCC)*, pages 309–320, 2014.

12 Mahdi Boroujeni and Saeed Seddighin. Improved MPC algorithms for edit distance and ulam distance. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 31–40, 2019.

13 Jop Briët, Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Monotonicity testing and shortest-path routing on the cube. *Combinatorica*, 32(1):35–53, 2012.

14 Deeparnab Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC)*, pages 419–428, 2013.

15 Deeparnab Chakrabarty and C. Seshadhri. An optimal lower bound for monotonicity testing over hypergrids. *Theory of Computing*, 10(17):453–464, 2014.

16 Deeparnab Chakrabarty and C. Seshadhri. An $o(n)$ monotonicity tester for boolean functions over the hypercube. *SIAM Journal on Computing*, 45(2):461–472, 2016.

17 Deeparnab Chakrabarty and C Seshadhri. Adaptive boolean monotonicity testing in total influence time. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 20:1–20:7, 2019.

18 Alex Chen, Timothy Chu, and Nathan Pinsker. The dynamic longest increasing subsequence problem. *arXiv preprint*, 2013. `arXiv:1309.7724`.

19 Xi Chen, Anindya De, Rocco A. Servedio, and Li-Yang Tan. Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proceedings of the 47th ACM Symposium on the Theory of Computing (STOC)*, pages 519–528, 2015.

20 Xi Chen, Rocco A. Servedio, and Li-Yang Tan. New algorithms and lower bounds for monotonicity testing. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–295, 2014.

21 Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond Talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th ACM Symposium on the Theory of Computing (STOC)*, pages 523–536, 2017.

22 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

23 Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 97–108, 1999.

24 Funda Ergün and Hossein Jowhari. On the monotonicity of a data stream. *Combinatorica*, 35(6):641–653, 2015.

25 Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Vishwanthan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000.

26 Chaim Even-Zohar and Calvin Leng. Counting small permutation patterns. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2288–2302, 2021.

27 Eldar Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004.

28 Jacob Fox. Stanley–Wilf limits are typically exponential. *arXiv*, 2013. `arXiv:1310-8378`.

29 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

30 Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. *SICOMP*, 39(8):3463–3479, 2010. Short version in FOCS'07.

31 Paweł Gawrychowski and Wojciech Janczewski. Fully dynamic approximation of LIS in polylogarithmic time. In *Proceedings of the 53rd ACM Symposium on the Theory of Computing (STOC)*, pages 654–667, 2021.

32 Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

33 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

34 Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007.

35 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–101, 2014.

36 Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient massively parallel methods for dynamic programming. In *Proceedings of the 49th ACM Symposium on the Theory of Computing (STOC)*, pages 798–811, 2017.

37 Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric type theorems. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 52–58, 2015.

38 Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

39 Tomasz Kociumaka and Saeed Seddighin. Improved dynamic algorithms for longest increasing subsequence. In *53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 640–653, 2021.

40 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley–Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107:153–160, 2004.

41 Michael Mitzenmacher and Saeed Seddighin. Dynamic algorithms for LIS and distance to monotonicity. In *Proceedings of the 52nd ACM Symposium on the Theory of Computing (STOC)*, pages 671–684, 2020.

42 Michael Mitzenmacher and Saeed Seddighin. Improved sublinear time algorithm for longest increasing subsequence. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1934–1947, 2021.

43 Timothy Naumovitz and Michael E. Saks. A polylogarithmic space deterministic streaming algorithm for approximating distance to monotonicity. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1252–1262, 2015.

44 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1582–1597, 2017.

45 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. *Random Structures and Algorithms*, 55:402–426, 2019. Extended abstract in SODA 2017 [44].

46 Ilan Newman and Nithin Varma. New sublinear algorithms and lower bounds for LIS estimation. In *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 100:1–100:20, 2021. `arXiv:2010.05805`.

47 Ilan Newman and Nitin Varma. Strongly sublinear algorithms for testing pattern freeness. *arXiv preprint*, 2021. To appear in ICALP 2022. `arXiv:2106.04856`.

48 Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and Nithin M. Varma. Parameterized property testing of functions. *ACM Transactions on Computation Theory*, 9(4):17:1–17:19, 2018.

49 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.

50 Prakash Ramanan. Tight $\omega(n \log n)$ lower bound for finding a longest increasing subsequence. *International Journal of Computer Mathematics*, 65(3–4):161–164, 1997.

51 Ronitt Rubinfeld and Madhu Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

52 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1121–1145. IEEE Computer Society, 2019.

53 Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1698–1709, 2013.

54 Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM J. Comput.*, 46(2):774–823, 2017. Short version in FOCS 2010.

55 Rodica Simion and Frank W. Schmidt. Restricted permutations. *European Journal of Combinatorics*, 6(4):383–406, 1985.

56 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007.

57 Vincent Vatter. Permutation classes. In *Handbook of Enumerative Combinatorics*, pages 777–858. Chapman and Hall/CRC, 2015.

Deciding Twin-Width at Most 4 Is NP-Complete

Pierre Bergé
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Hugues Déprés
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We show that determining if an n-vertex graph has twin-width at most 4 is NP-complete, and requires time $2^{\Omega(n/\log n)}$ unless the Exponential-Time Hypothesis fails. Along the way, we give an elementary proof that n-vertex graphs subdivided at least $2\log n$ times have twin-width at most 4. We also show how to encode trigraphs H (2-edge colored graphs involved in the definition of twin-width) into graphs G, in the sense that every d-sequence (sequence of vertex contractions witnessing that the twin-width is at most d) of G inevitably creates H as an induced subtrigraph, whereas there exists a partial d-sequence that actually goes from G to H. We believe that these facts and their proofs can be of independent interest.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Twin-width, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.18

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2112.08953`

Funding This paper was supported by the ANR projects TWIN-WIDTH (ANR-21-CE48-0014-01) and Digraphs (ANR-19-CE48-0013-01).

Acknowledgements We wish to thank Eunjung Kim, Stéphan Thomassé, and Rémi Watrigant.

1 Introduction

A *trigraph* is a graph with some edges colored black, and some colored red. A (vertex) *contraction* consists of merging two (non-necessarily adjacent) vertices, say, u, v into a vertex w, and keeping every edge wz black if and only if uz and vz were previously black edges. The other edges incident to w become red (if not already), and the rest of the trigraph stays the same. A *contraction sequence* of an n-vertex graph G is a sequence of trigraphs $G = G_n, \ldots, G_1 = K_1$ such that G_i is obtained from G_{i+1} by performing one contraction. A *d-sequence* is a contraction sequence where all the trigraphs have red degree at most d. The *twin-width* of G, denoted by $tww(G)$, is then the minimum integer d such that G admits a d-sequence. See Figure 1 for an example of a graph admitting a 2-sequence. The *red graph* of a trigraph is obtained by simply deleting its black edges. A *partial d-sequence* is similar to a d-sequence but ends on any trigraph G_i, instead of on the 1-vertex (tri)graph G_1. Twin-width can be naturally extended to matrices over a finite alphabet (in an unordered [6], or an ordered setting [4]), and hence to any binary structure.

Surprisingly many classes turn out to be of bounded twin-width. Such is the case of graphs with bounded clique-width, H-minor free graphs for any fixed H, posets with antichains of bounded size, strict subclasses of permutation graphs, map graphs, bounded-degree string

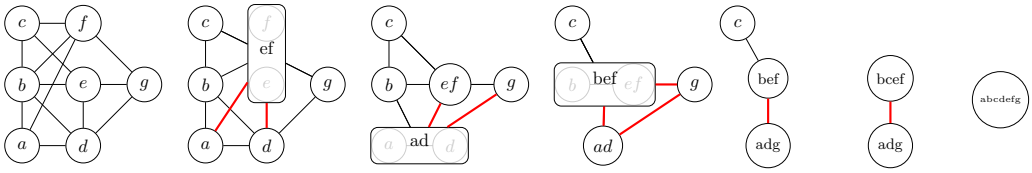


Figure 1 A 2-sequence witnessing that the initial graph has twin-width at most 2.

graphs [6], as well as $\Omega(\log n)$-subdivisions of n-vertex graphs, and some classes of cubic expanders [3]. One of the main algorithmic interests with twin-width is that first-order (FO) model checking, that is, deciding if a first-order sentence φ holds in a graph G, can be decided in fixed-parameter time (FPT) $f(|\varphi|, d) \cdot |V(G)|$ for some computable function f, when given a d-sequence of G [6]. As for most classes known to have bounded twin-width, one can compute $O(1)$-sequences in polynomial time for members of the class, the latter result unifies and extends several known results [14, 17, 18, 16, 21] for hereditary (but not necessarily monotone) classes.

For monotone (i.e., subgraph-closed) classes, the FPT algorithm of Grohe, Kreutzer, and Siebertz [20] for FO model checking on nowhere dense classes, is complemented by W[1]-hardness on classes that are somewhere dense (i.e., *not* nowhere dense) [11], and even AW[$*$]-hardness on classes that are *effectively* somewhere dense [9]. The latter results mean that, for monotone classes, FO model checking is unlikely to be FPT beyond nowhere dense classes.

The missing piece for an FO model-checking algorithm in FPT time on any class of bounded twin-width is a polynomial-time algorithm and a computable function f, that given a constant integer bound c and a graph G, either finds an $f(c)$-sequence for G, or correctly reports that the twin-width of G is greater than c. The running time of the algorithm could be $n^{g(c)}$, for some function g. However to get an FPT algorithm in the combined parameter *size of the sentence + bound on the twin-width*, one would further require that the approximation algorithm takes FPT time in c (now seen as a parameter), i.e., $g(c)n^{O(1)}$. We know such an algorithm for instance on ordered graphs (more generally, ordered binary structures) [4], graphs of bounded clique-width, proper minor-closed classes [6], but not in general graphs.

On the other hand, prior to this paper, no algorithmic lower bound was known for computing the twin-width. Our main result rules out an (exact) XP algorithm to decide $tww(G) \leqslant k$, that is, an algorithm running in time $n^{f(k)}$ for some computable function f. Indeed we show that deciding if the twin-width of a graph is at most 4 is intractable. We refer the reader to Section 2 for some context on the Exponential-Time Hypothesis (ETH), which implies that n-variable 3-SAT cannot be solved in time $2^{o(n)}$.

▶ **Theorem 1.** *Deciding if a graph has twin-width at most 4 is NP-complete. Furthermore, no algorithm running in time $2^{o(n/\log n)}$ can decide if an n-vertex graph has twin-width at most 4, unless the ETH fails.*

As far as approximation algorithms are concerned, our result only rules out a ratio better than 5/4 for determining the twin-width. This still leaves plenty of room for an $f(\mathrm{OPT})$-approximation, which would be good enough for most of the (theoretical) algorithmic applications. Note that such algorithms exist for treewidth in polytime [12] and FPT time [27], for pathwidth [12], and for clique-width via rank-width [30].

Is Theorem 1 surprising? On the one hand, it had to be expected that deciding, given a graph G and an integer k, whether $tww(G) \leqslant k$ would be NP-complete. This is the case for example of treewidth [1], pathwidth [29, 26, 28], clique-width [13], rank-width [23],

mim-width [32], and bandwidth [19]. On the other hand, the parameterized complexity of these width parameters is more diverse and harder to predict. Famously, Bodlaender's algorithm is a linear FPT algorithm to exactly compute treewidth [2] (and a non-uniform FPT algorithm came from the Graph Minor series [31]). In contrast, while there is an XP algorithm to compute bandwidth [33], an FPT algorithm is highly unlikely [10]. It is a long-standing open whether an FPT or a mere XP algorithm exist for computing clique-width exactly, or even simply if one can recognize graphs of clique-width at most 4 in polynomial time (deciding clique-width at most 3 is indeed tractable [7]).

Theorem 1 almost completely resolves the parameterized complexity of exactly computing twin-width on general graphs. Two questions remain: can graphs of twin-width at most 2, respectively at most 3, be recognized in polynomial time. Graphs of twin-width 0 are cographs, which can be recognized in linear time [22], while it was recently shown that graphs of twin-width at most 1 can be recognized in polynomial time [5]. In the course of establishing Theorem 1 we show and generalize the following, where an $(\geqslant s)$-*subdivision* of a graph is obtained by subdividing each of its edges at least s times.

▶ **Theorem 2.** *Any $(\geqslant 2\log n)$-subdivision of an n-vertex graph has twin-width at most 4.*

That those graphs have bounded twin-width was known [3], but not with the explicit bound of 4. Another family of graphs with twin-width at most 4 is the set of grids, walls, their subgraphs, and subdivisions. Even if there is no proof of that fact, sufficiently large grids, walls, or their subdivisions likely have twin-width *at least* 4; it is actually surprising that the 6×8 grid still has twin-width 3 [34]. We also believe that long subdivisions of "sufficiently complicated" graphs have twin-width *at least* 4. That would make graphs of twin-width at most 3 considerably simpler than those of twin-width at most 4, especially among sparse graphs.

Contrary to the hardness proof for treewidth [1], which involves some structural characterizations by chordal completions, and the intermediate problems Minimum Cut Linear Arrangement, Max Cut, and Max 2-SAT [19], our reduction is "direct" from 3-SAT. This makes the proven hardness of twin-width more robust, and easier to extend to restricted classes of graphs, especially sparse ones. Theorem 1 holds for bounded-degree input graphs. For instance, performing our reduction from Planar 3-SAT produces subgraphs of constant powers of the planar grid (while admittedly weakening the ETH lower bound from $2^{\Omega(n/\log n)}$ to $2^{\Omega(\sqrt{n}/\log n)}$). Hence, while the complexity status of computing treewidth on planar graphs is a famous long-standing open question, one can probably extend the NP-hardness of *twin-width at most 4* to planar graphs, by tuning and/or replacing the few non-planar gadgets of our reduction.

Let us point out that, in contrast to subset problems, there is no $2^{O(n)}$-time algorithm known to compute twin-width. The exhaustive search takes time $n^{2n+O(1)}$ by considering all sequences of $n-1$ pairs of vertices. We leave as an open question whether the ETH lower bound of computing twin-width can be brought from $2^{\Omega(n/\log n)}$ to $2^{\Omega(n)}$, or even $2^{\Omega(n\log n)}$. The latter lower bound is known to hold for Subgraph Isomorphism [8] (precisely, given a graph H and an n-vertex graph G, deciding if H is isomorphic to a subgraph of G requires time $2^{\Omega(n\log n)}$, unless the ETH fails), or computing the Hadwiger number [15] (i.e., the size of the largest clique minor).

1.1 Outline of the proof of Theorem 1

We propose a quasilinear reduction from 3-SAT. Given an n-variable instance I of 3-SAT, we shall construct an $O(n\log n)$-vertex graph $G = G(I)$ which has twin-width at most 4 if and only if I is satisfiable.

Half of our task is to ensure that no 4-sequence will exist if I is unsatisfiable. This is challenging since many contraction strategies are to be considered and addressed. We make this task more tractable by attaching *fence gadgets* to some chosen vertex subsets. The effect of the fence *enclosing* S is that no contraction can involve vertices in S with vertices outside of S, while S is not contracted into a single vertex. The *maximal or outermost* fences (we may nest two or more fence gadgets) partition the rest of the vertices. This significantly tames the potential 4-sequences of G.

Our basic building block, the *vertical set*, consists of a pair of vertices (*vertical pair*) enclosed by a fence. It can be thought of as a bit set to 0 as long as the pair is not contracted, and to 1 when the pair gets contracted. It is easy to assemble vertical sets as prescribed by an auxiliary digraph D (of maximum degree 3), in such a way that, to contract (by a partial 4-sequence) the pair of a vertical set V, one first has to contract all the vertical sets that can reach V in D. This allows to propagate and duplicate a bit in a so-called *wire* (corresponding to an out-tree in D), and to perform the logical AND of two bits.

The bit propagation originates from a variable gadget (we naturally have one per variable appearing in I) that offers two alternatives. One can contract the "top half" of the gadget of variable x_i, which then lets one contract the vertical sets in the wire of literal x_i, or one can contract instead the "bottom half" of the gadget, as well as the vertical sets in the wire of literal $\neg x_i$. Concretely, these two contraction schemes represent the two possible assignments for variable x_i. A special "lock" on the variable gadget (called *half-guards*) prevents its complete contraction, and in particular, performing contractions in both the wires of a literal and its negation.

The leaves of the literal wires serve as inputs for 3-clause gadgets. One can contract the output (also a vertical set) of a clause gadget if and only if one of its input is previously contracted. We then progressively make the AND of the clauses via a "path" of binary AND gadgets fed by the clause outputs. We eventually get a vertical set, called *global output*, which can be contracted by a partial 4-sequence only if I is satisfiable. Indeed at this point, the variable gadgets are still locked so at most one of their literals can be propagated. This ticks one of our objective off. We should now ensure that a 4-sequence is possible from there, when I is satisfiable.

For that purpose, we add a wire from the global output back to the half-guards (or locks) of the variable gadgets. One can contract the vertical sets of that wire, and in particular the half-guards. Once the variable gadgets are "unlocked," they can be fully contracted. As a consequence, one can next contract the wires of literals set to false, and *all* the remaining vertical sets involved in clause gadgets.

At this point, the current trigraph H roughly has one vertex per outermost fence with red edges linking two adjacent gadgets (and no black edge). We will guarantee that the (red) degree of H is at most 4, its number of vertices of degree at least 3 is at most βn, for some constant β. Besides we will separate gadgets by degree-2 wires of length $2\log(\beta n)$ beforehand. This is crucial so that the red graph of H is a $(2\log n')$-subdivision of an n'-vertex graph. We indeed show that such trigraphs have twin-width at most 4. A complicated proof in [3] shows that $\Theta(\log n')$-subdivisions of n'-vertex graphs have bounded twin-width. Here we give an elementary proof of a similar fact with an explicit upper bound of 4.

This finishes to describe our overall plan for the reduction and its correctness. It happens that fence gadgets are easier to build as trigraphs, while the rest of the gadgetry can be directly encoded by graphs. We thus show how to encode trigraphs by graphs, as follows. For any trigraph J whose red graph has degree at most d, and component size at most h,

there is a graph G on at most $f(d,h)\cdot|V(J)|$ vertices such that J has twin-width at most $2d$ if and only if G has twin-width at most $2d$. This uses some local replacements and confluence properties of certain partial contraction sequences.

The proofs of the theorems and lemmas marked with a $\star$ can be found in the full version.

2 Preliminaries

For i and j two integers, we denote by $[i,j]$ the set of integers that are at least i and at most j. For every integer i, $[i]$ is a shorthand for $[1,i]$. We use the standard graph-theoretic notations: $V(G)$ denotes the vertex set of a graph G, $E(G)$ denotes its edge set, $G[S]$ denotes the subgraph of G induced by S, etc. An $(\geqslant s)$-subdivision (resp. s-subdivision) of a graph G is obtained by subdividing every edge of G at least s times (resp. exactly s times).

2.1 Definitions and notations related to twin-width

A *trigraph* G has vertex set $V(G)$, black edge set $E(G)$, red edge set $R(G)$. Its *red graph* $(V(G),R(G))$ may be denoted $\mathcal{R}(G)$. The *red degree* of a trigraph is the degree of its red graph. We say that $u\in V(G)$ is a *black neighbor* (respectively *red neighbor*) of $v\in V(G)$ when $(u,v)\in E(G)$ (respectively $(u,v)\in R(G)$). A trigraph G' is an *induced subtrigraph* of trigraph G if $V(G')\subseteq V(G)$, $E(G')=E(G)\cap\binom{V(G')}{2}$, and $R(G')=R(G)\cap\binom{V(G')}{2}$. Then we say that G is a supertrigraph of G', and we may also denote G' by $G[V(G')]$. A *(partial) d-sequence* of a (tri)graph G is a (partial) contraction sequence starting at G and admitting trigraphs of red degree at most d.

The *twin-width* of a graph, introduced in [6], can be defined in the following way. A *partition sequence* of an n-vertex graph G, is a sequence $\mathcal{P}_n,\dots,\mathcal{P}_1$ of partitions of its vertex set $V(G)$, such that $\mathcal{P}_n$ is the set of singletons $\{\{v\} : v\in V(G)\}$, $\mathcal{P}_1$ is the singleton set $\{V(G)\}$, and for every $2\leqslant i\leqslant n$, $\mathcal{P}_{i-1}$ is obtained from $\mathcal{P}_i$ by merging two of its parts into one. Two parts P,P' of a same partition $\mathcal{P}$ of $V(G)$ are said *homogeneous* if either every pair of vertices $u\in P, v\in P'$ are non-adjacent, or every pair of vertices $u\in P, v\in P'$ are adjacent. Finally the twin-width of G, denoted by $tww(G)$, is the least integer d such that there is partition sequence $\mathcal{P}_n,\dots,\mathcal{P}_1$ of G with each part P of each $\mathcal{P}_i$ $(1\leqslant i\leqslant n)$ being homogeneous to every part of $\mathcal{P}_i\setminus\{P\}$ but at most d.

The reason we gave two (equivalent) definitions of twin-width is that both viewpoints are incomparably useful and convenient. To navigate between these two worlds, we use the following notations and vocabulary.

Assume there is a partial contraction sequence from (tri)graph G to trigraph H. If u is a vertex of H, then $u(G)$ denotes the set of vertices eventually contracted into u in H. We denote by $\mathcal{P}(H)$ the partition $\{u(G) : u\in V(H)\}$ of $V(G)$. If G is clear from the context, we may refer to a *part* of H as any set in $\{u(G) : u\in V(H)\}$. We say that a contraction of two vertices $u,u'\in V(H)$ *involves* a vertex $v\in V(G)$ if $v\in u(G)$ or $v\in u'(G)$. A contraction *involves* a pair of vertices v,v' if $v\in u(G)$ and $v'\in u'(G)$ (or $v\in u'(G)$ and $v'\in u(G)$). A contraction *involves* a set S, if it involves a vertex of S, or a pair of sets S,T if it involves a pair in $S\times T$.

2.2 Useful observations

The twin-width can only decrease when taking induced subgraphs or turning red into black.

▶ **Observation 3.** *Let G' be an induced subtrigraph of trigraph G. Then $tww(G')\leqslant tww(G)$.*

▶ **Observation 4.** *Let G be a trigraph and G' another trigraph obtained from G by turning some non-edges and some edges into red edges. Then $tww(G') \geqslant tww(G)$.*

Trees admit a simple 2-sequence, that gives a d-sequence on red trees of degree at most d.

▶ **Lemma 5** ([6]). *Every (black) tree has twin-width at most 2. Every red tree has twin-width at most its maximum degree.*

2.3 The Exponential-Time Hypothesis

The Exponential-Time Hypothesis (ETH) was proposed by Impagliazzo and Paturi [24] and asserts that there is no subexponential-time algorithm solving 3-SAT. More precisely, there is an $\varepsilon > 0$ such that n-variable 3-SAT cannot be solved in time $2^{\varepsilon n}$. A classic reduction [35] linear in the number of clauses, and the Sparsification Lemma [25] imply that:

▶ **Theorem 6** ([35, 25]). *The n-variable 3-SAT problem where each variable appears at most twice positively, and at most twice negatively, is NP-complete, and cannot be solved in time $2^{o(n)}$, unless the ETH fails.*

3 Long subdivisions have twin-width at most four

In [3], it is proved that the $\Omega(\log n)$-subdivision of any n-vertex graph has bounded twin-width. The proof is rather involved, relies on a characterization by *mixed minors* established in [6], and does not give an explicit constant bound. Here we give an elementary proof that any ($\geqslant 2\lceil \log n \rceil - 1$)-subdivision of an n-vertex graph has twin-width at most 4.

▶ **Theorem 7.** *Let G be a trigraph obtained by subdividing each edge of an n-vertex graph H at least $2\lceil \log n \rceil - 1$ times, and by turning red any subset of its edges as long as the red degree of G remains at most 4, and no vertex with red degree 4 has a black neighbor. Then $tww(G) \leqslant 4$.*

Proof. By no more than doubling the number of vertices of H, we can assume that n is a power of 2. Indeed, padding H with isolated vertices up to the next power of 2 does not change the quantity $\lceil \log |V(H)| \rceil$.

Let G' be a supertrigraph of G obtained by arbitrarily arranging the vertices of H (in G) at the leaves of a "virtual" full binary tree of height $\log n$. So that the red degree does not exceed 4, we so far omit the edges of the tree incident to a leaf (i.e., a vertex of H), while we

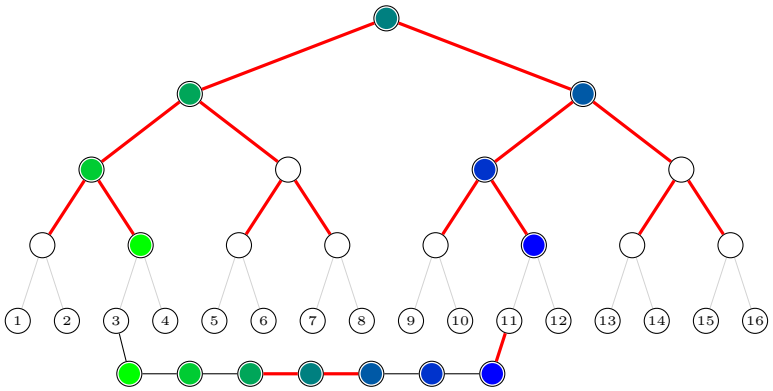


Figure 2 Contracting the pairs of vertices with the same color, from the greenest to the bluest, is a partial 4-sequence, which acts as a deletion of the subdivided edge $(3, 11)$.

put in red all the other edges of the tree. (The missing edges of the tree will naturally appear in red.) The internal nodes of the tree are all fresh vertices, not present in G. See Figures 2 and 3 for an illustration. We show that $tww(G') \leqslant 4$, hence by Observation 3, $tww(G) \leqslant 4$ since G is an induced subtrigraph of G'.

We label $1, 2, \ldots, n$ the vertices of H. If there is an edge $ij \in E(H)$, it is subdivided into a path, say, $i, s(ij,1), s(ij,2), \ldots, s(ij,z), j$ in G with $z \geqslant 2\log n - 1$. First, we repeatedly contract adjacent vertices in the middle of this path until it consists of exactly $2\log n$ edges. If $z > 2\log n - 1$, we had to contract at least one pair of adjacent vertices. Thus the vertex in the middle of the path necessarily has now two red edges incident to it. Note that the other edges of the path can be black or red indifferently. To avoid cumbersome notations, we rename the inner vertices of the path $s(ij,1), s(ij,2), \ldots, s(ij,z)$ with now $z = 2\log n - 1$.

▷ Claim 8. There is a partial 4-sequence from G' to $G' - \{s(ij,1), \ldots, s(ij,z)\}$.

Proof. Intuitively we "zip" the subdivision of ij with the walk made by the union of the path from leaf i to the root, and the path from the root to leaf j. Let $i, v_1, v_2, \ldots, v_z, j$ be the concatenation of the simple path from i to the root of the tree, and the simple path from the root to j. Its length is thus $2\log n = z + 1$. For h going from 1 to $z = 2\log n - 1$, we contract v_h and $s(ij,h)$ (see Figure 2). After each contraction, the newly formed vertex has red degree at most 4. The red degree of vertices that are neither the new vertex nor a leaf of the tree is either unchanged or at most 2. The red degree of a leaf ℓ of the tree may increase by 1. This may only happen the first time a neighbor of ℓ is involved in a contraction, and that contraction merges a black neighbor of ℓ with the parent of ℓ in the tree (like is the case for leaf 3 from Figure 2 to Figure 3). By assumption, this implies that ℓ had red degree at most 3, thus its red degree does not exceed 4. Thus, what we defined is indeed a partial 4-sequence. One can finally notice that after these z contractions, we indeed reach trigraph $G' - \{s(ij,1), \ldots, s(ij,z)\}$. ◁

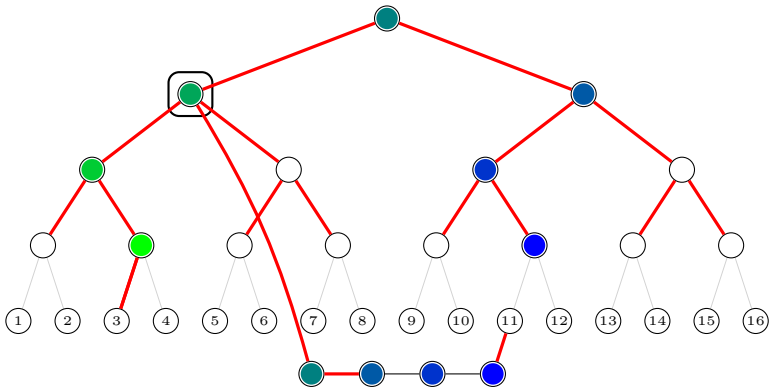


Figure 3 The picture after the first three contractions. The newly formed vertex has red degree 4.

We apply Claim 8 for each edge of H (or rather, subdivided edge in G). We are then left with a red full binary tree which admits a 3-sequence by Lemma 5. Hence there is a 4-sequence for G', and in particular, for G. ◀

We will only use the following consequence.

▶ **Lemma 9.** *Let G be a trigraph obtained by subdividing at least $2\lceil \log n \rceil - 1$ times each edge of an n-vertex graph H of degree at most 4, and by turning red all its edges. Then $tww(G) \leqslant 4$.*

4 Hardness of determining if the twin-width is at most four

Here we show the main result of the paper.

▶ **Theorem 1.** *Deciding if a graph has twin-width at most 4 is NP-complete. Furthermore, no algorithm running in time $2^{o(n/\log n)}$ can decide if an n-vertex graph has twin-width at most 4, unless the ETH fails.*

The membership to NP is ensured by the d-sequence: a polynomial-sized certificate that a graph has twin-width at most d, checkable in polynomial time. We thus focus on the hardness part of the statement, and design a quasilinear reduction from 3-SAT.

4.1 Foreword to the reduction

Let us start by a construction allowing to encode trigraphs into (plain) graphs. Given a trigraph H with red degree at most d, we can produce a graph G such that H admits a $2d$-sequence iff G admits a $2d$-sequence.

▶ **Lemma 10** ($\star$)**.** *Given any trigraph H whose red graph has degree at most d and connected components of size at most h, one can compute in time $O_{d,h}(|V(H)|)$ a graph G on $O_{d,h}(|V(H)|)$ vertices such that H has a $2d$-sequence if and only if G has a $2d$-sequence.*

In what follows, we only need the following scaled-down version.

▶ **Lemma 11.** *Given any trigraph H whose red graph is a disjoint union of 12-vertex paths and isolated vertices, one can compute in polynomial time a graph G on $O(|V(H)|)$ vertices such that H has twin-width at most 4 if and only if G has twin-width at most 4.*

Our task is now slightly simpler. Given a 3-SAT instance I, we may design a *trigraph* satisfying the requirements of Lemma 11 with twin-width at most 4 if and only if I is satisfiable.

We will motivate all the gadgets along the way, by exhibiting key properties that they impose on a potential 4-sequence. These properties readily lead to a satisfying assignment for I. We also describe partial 4-sequences to reduce most of the gadgets. However some preconditions (specifying the context in which a particular gadget stands) tend to be technical, and make more sense after the construction of G. In those cases, to avoid unnecessarily lengthy lemmas, we only give an informal strategy, and postpone the adequate contraction sequence.

4.2 Fence gadget

The vertex set of a fence gadget is $A \cup B$ with $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ and $B = \{b_1, b_2, b_3, b_4, b_5, b_6\}$. Its black edge set consists of 13 edges: the cycles $a_1a_2a_3a_4a_5a_6a_1$ and $b_1b_2b_3b_4b_5b_6b_1$, plus the edge b_1a_6. Its red edge set consists of 11 edges: a_ib_i for each $i \in [6]$, and a_ib_{i+1} for each $i \in [5]$. A fence gadget is *attached* to a vertex subset S by making A fully adjacent to S. See Figure 4 for an illustration.

We will later nest fence gadgets. Thus we have to tolerate that F has other neighbors than S in G. Actually we even allow $V(F)$ to have neighbors outside of S and the fence gadgets surrounding F. We however always observe the following rule.

▶ **Definition 12** (Attachment rule)**.** *A fence gadget F with vertex bipartition (A, B), and attached to S, satisfies the* attachment rule *in a trigraph H if F is a connected component of $\mathcal{R}(H)$, and there is a set $X \subseteq V(H) \setminus (A \cup B \cup S)$ such that:*

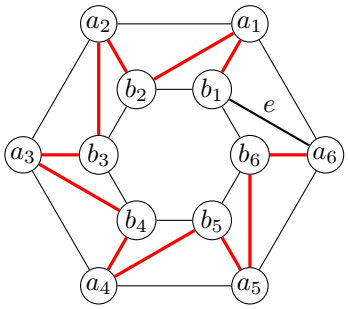


Figure 4 The fence gadget F, with $A = \{a_i \mid 1 \leqslant i \leqslant 6\}$ and $B = \{B_i \mid 1 \leqslant i \leqslant 6\}$.

- $\forall x \in A$, $N(x) \setminus V(F) = X \cup S$,
- $\forall x \in B$, $N(x) \setminus V(F) = X$, *and*
- $\forall x \in X$, $S \subset N(x)$.

Initially in G, we make sure that all the fence gadgets satisfy the attachment rule. This will remain so until we decide to contract them.

For each fence gadget F satisfying the attachment rule, there is only one adequate set X, it is $X = N(F) \setminus (V(F) \cup S)$. We denote by Y the set $V(G) \setminus (V(F) \cup S \cup X)$. We make the following observations on the three possible neighborhoods that vertices outside of F have within $V(F)$.

▶ **Observation 13.** *The fence gadget definition and the attachment rule implies:*

- $\forall x \in S$, *it holds* $N(x) \cap V(F) = A$,
- $\forall x \in X$, *it holds* $N(x) \cap V(F) = V(F) = A \cup B$, *and*
- $\forall x \in Y$, *it holds* $N(x) \cap V(F) = \emptyset$.

Henceforth we will represent every fence gadget as a brown rectangle surrounding the set S it is attached to. The vertices of X are linked to the brown rectangle, as they are fully adjacent to $S \cup V(F)$. See Figure 5 for an illustration of the attachment rule, and a compact representation of fence gadgets.

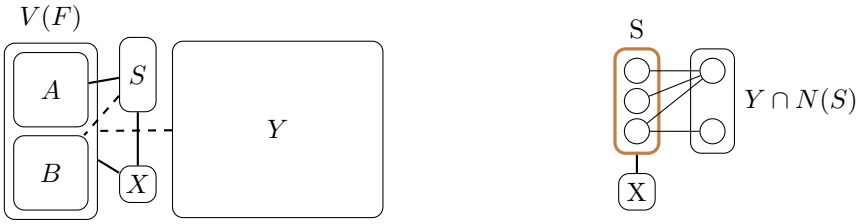


Figure 5 Left: The forced adjacencies (solid lines, all edges between the two sets) and non-adjacencies (dashed lines, no edge between the two sets), as specified by the attachment rule. Right: Symbolic representation of the fence gadget attached to S by a brown rectangle.

Constraints of the fence gadget on a 4-sequence. The following lemmas are preparatory steps for the milestone that no part in a 4-sequence of G can overlap S (that is, intersects S without containing it).

▶ **Lemma 14** ($\star$). *The first contraction involving two vertices of F results in a vertex of red degree at least 5, except if it is a contraction of some $a_i \in A$ with some $b_j \in B$.*

▶ **Lemma 15** (⋆). *If the first contraction involving two vertices of F is of some $a_i \in A$ with some $b_j \in B$, the red degree within F of the created vertex is at least 3.*

The last preparatory step is this easy lemma.

▶ **Lemma 16.** *Before a contraction involves two vertices of $V(F)$, the following holds in a partial 4-sequence:*

- *no part intersects both X and S,*
- *no part intersects both Y and S, and*
- *no part intersects both X and Y.*

Proof. By Observation 13, such a part would have red degree $|B| = 6$, $|A| = 6$, and $|A \cup B| = 12$, respectively. ◀

As a consequence we obtain the following.

▶ **Lemma 17.** *In a partial 4-sequence of G, the first contraction involving a vertex in $V(F)$ and a vertex in $V(F) \cup S$ has to be done after S is contracted into a single vertex.*

Proof. We consider the first time a vertex $u \in V(F)$ is involved in a contraction with a vertex of $V(F) \cup S$. Either (case 1) the part of u, P_u, is contracted with a part P_v containing $v \in V(F)$, or (case 2) P_u is contracted with a part P intersecting S but not $V(F)$.

In case 1, by Lemma 14, u and v hit both A and B. Thus, by Lemma 15, the red degree within F of the resulting vertex z is at least 3. Moreover z is linked by a red edge to every part within S, since S is fully adjacent to A, and fully non-adjacent to B. Thus S should at this point consist of a single part.

We now argue that case 2 is impossible in a partial 4-sequence. By Lemma 16, part P cannot intersect $X \cup Y$ (nor $V(F)$, by construction). Thus $P \subseteq S$. If $u \in A$, then the contraction of P_u and P has incident red edges toward at least 5 vertices: three vertices of A non-adjacent to u and two private neighbors of u (in the total graph) within B. If instead $u \in B$, the red degree of the contracted part is at least 6, as witnessed by two neighbors of u in B, and four non-neighbors of u in A. ◀

We can now establish the main lemma on how a fence gadget constrains a 4-sequence. Lemmas 16 and 17 have the following announced consequence: While S is not contracted into a single vertex, no part within S can be contracted with a part outside of S, and similarly vertices of X cannot be contracted with vertices of Y.

▶ **Lemma 18** (⋆). *In a partial 4-sequence, while S is not contracted to a single vertex,*

- *(i) no part intersects both S and $V(G) \setminus S$, nor*
- *(ii) both X and Y.*

Contracting the fence gadget. The previous lemmas establish some constraints that the fence gadget imposes on a supposed (partial) 4-sequence. We now see how a partial 4-sequence actually contracts a fence gadget.

Every time we are about to contract a fence gadget F attached to S, we will ensure that the following properties hold:

- no prior contraction has involved a vertex of $V(F)$,
- no red edge has one endpoint in $V(F)$ and one endpoint outside $V(F)$, and
- S is contracted into a single vertex with red degree at most 3.

In particular, the fence gadget F still satisfies the attachment rule.

▶ **Lemma 19** (⋆). *Let H be a trigraph containing a fence gadget F attached to a single vertex s of red degree at most 3. We assume that F satisfies the attachment rule in H.*

Then there is a partial 4-sequence from H to H', where H' is the trigraph obtained from H by contracting $V(F)$ into a single vertex.

4.3 Propagation, wire, and long chain

A *vertical set* V consists of two vertices x, y combined with a fence gadget F attached to $\{x, y\}$. Thus $V = \{x, y\} \cup V(F)$. We call *vertical pair* the vertices x and y. We will usually add a superscript to identify the different copies of vertical sets: Every vertex whose label is of the form u^j belongs to the vertical set V^j.

The *propagation gadget* from V^j to V^k puts all the edges between V^j and x^k (and no other edge). We call these edges an *arc from V^j to V^k*. We also say that the vertical set V^k is *guarded* by V^j. The pair V^j, V^k is said *adjacent*. Here, singleton $\{x^k\}$ plays the role of X (Definition 12) for the attachment rule of vertical set V_j.

The *propagation digraph of G*, denoted by $\mathcal{D}(G)$, has one vertex per vertical set and an arc between two vertical sets linked by an arc (in the previous sense). A (maximal) *wire* W is an induced subgraph of G corresponding in $\mathcal{D}(G)$ to a (maximal) out-tree on at least two vertices. See Figure 6 for an illustration of a wire made by simply concatenating propagation gadgets. Eventually $\mathcal{D}(G)$ will have out-degree at most 2, in-degree at most 2, and total degree at most 3.

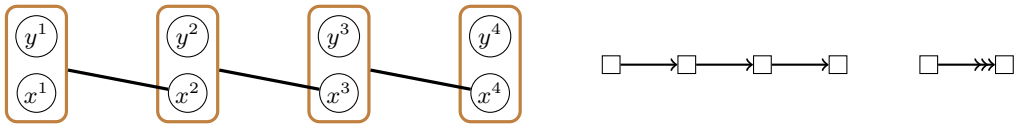


Figure 6 Left: A non-branching wire made of 4 vertical sets and 3 propagation gadgets. Every vertical set is guarded by the vertical set just to its left. Center: A more compact representation, which corresponds to the propagation digraph. Right: Symbolic representation of the long chain, that is, of the represented wire if $L = 4$.

The *children* of a vertical set V are the vertical sets that V guards. The *root of wire W* is the unique vertical set of in-degree 0 in $\mathcal{D}(W)$. The *leaves of wire W* are the vertical sets of out-degree 0 in $\mathcal{D}(W)$. A wire is said *primed* when the vertical pair of its root has been contracted.

A wire W is *non-branching* if every vertex of $\mathcal{D}(W)$ has out-degree at most 1; hence, $\mathcal{D}(W)$ is a directed path. A *long chain* is a wire W such that $\mathcal{D}(W)$ is a directed path on L vertices, where integer L will be specified later (and can be thought as logarithmic in the total number of fences which do not belong to vertical sets). Otherwise, if $\mathcal{D}(W)$ has at least one vertex with out-degree at least 2, wire W is said *branching*. A vertical set with two children is also said *branching*.

Constraints of the propagation gadget on a 4-sequence. We provide the proof that a contraction in a vertical set V is only possible when the vertical pair of all the vertical sets V' with a directed path to V in $\mathcal{D}(G)$ has been contracted.

▶ **Lemma 20.** *Let V^j and V^k be two vertical sets with an arc from V^j to V^k. In a partial 4-sequence from G, any contraction involving two vertices of V^k has to be preceded by the contraction of x^j and y^j.*

Proof. We recall the notations of Section 4.2. Let F be the fence gadget attached to $S = \{x^j, y^j\}$, X the neighborhood of $V(F)$ outside of $S \cup V(F)$, and Y the vertices that are not in $S \cup V(F) \cup X$. (We always assume that the attachment rule is satisfied.) We have $x^{j'} \in X$ and $y^k \in Y$, therefore by the second item of Lemma 18, their contraction has to be preceded by the contraction of x^j and y^j. Now applying the first item of Lemma 18 to the fence gadget F' attached to $S' = \{x^k, y^k\}$, and Lemma 17, any contraction involving a pair of V^k distinct from S' has to be preceded by the contraction of x^k and y^k. ◀

Henceforth, when we say that a contraction is *preceded* by another contraction, it includes the case that the two contractions are in fact the same. By a straightforward induction, we obtain the following from Lemma 20 (and Lemma 17).

▶ **Lemma 21.** *In a partial 4-sequence from G, any contraction involving a pair of vertices in a vertical set V has to be preceded by the contraction of the vertical pair of every vertical set V' such that there is a directed path from V' to V in $\mathcal{D}(G)$.*

Contracting wires. As the roots and leaves of wires will be connected to other gadgets, we postpone the description of how to contract wires until after building the overall construction G. Intuitively though, contracting a wire (in the vacuum) consists of contracting the vertical pair of its root, then its fence gadget by applying Lemma 19, and finally recursively contracting the subtrees rooted at its children. Since $\mathcal{D}(G)$ has total degree at most 3, every vertex has red degree at most 4 (for the at most 3 adjacent vertical sets, plus the pendant vertex of the fence gadget).

4.4 Binary AND gate

The *binary* AND *gate* (AND gadget, for short) simply consists of three vertical sets V^1, V^2, V^3 with an arc from V^1 to V^3, and an arc from V^2 to V^3. As usual, the vertical pairs of V^1, V^2, and V^3, are $\{x^1, y^1\}$, $\{x^2, y^2\}$, and $\{x^3, y^3\}$, respectively. We call the vertical sets V^1, V^2 the *inputs* of the AND gadget, and the vertical set V^3 the *output* of the AND gadget.

Constraint of the AND gadget on a 4-sequence. By Lemma 20, we readily derive:

▶ **Lemma 22.** *Assume G contains an AND gadget with inputs V^1, V^2, and output V^3. In a partial 4-sequence from G, any contraction involving two vertices of V^3 has to be preceded by the contraction of x^1 and y^1, and the contraction of x^2 and y^2.*

Contraction of an AND gadget. Once V^1 and V^2 are contracted into single vertices, one can contract the vertical pair x^3, y^3. This results in a vertex of red degree 2. Thus one can contract the fence gadget of V^3 by applying Lemma 19.

4.5 Binary OR gate

The *binary* OR *gate* (OR *gadget*) is connected to three vertical sets: two *inputs* V^1, V^2, and one *output* V^3. We start by building two vertical sets V, V' whose vertical pairs are $\{a, b\}$ and $\{c, d\}$, respectively. The edges ac and bd are added, as well as a vertex e adjacent to a and to c. Finally a fence is attached to $\{e\} \cup V \cup V'$.

The OR gadget is connected to its inputs and output, in the following way. Vertex a is made adjacent to x^1 and to y^1 (but not to their fence gadget). Similarly vertex b is linked to x^2 and y^2. Finally x^3 is adjacent to all the vertices of the OR gadget, that is, $\{e\} \cup V \cup V'$ plus the vertices of the outermost fence. See Figure 7 for a representation of the OR gadget.

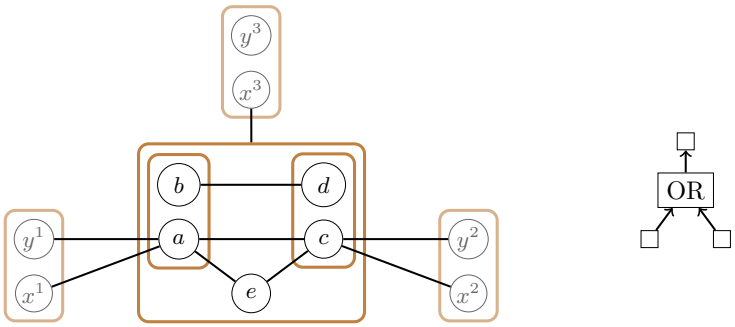


Figure 7 An OR gadget attached to inputs V^1, V^2 and output V^3, and its symbolic representation. These vertical sets are technically not part of the OR gate, so we represent them slightly dimmer.

Constraint of the OR gadget on a 4-sequence. By design, one can only start contracting the OR gadget after the vertical pair of at least one of its two inputs has been contracted. This implies that no contraction can involve V^3 before at least one the vertical pairs $\{x^1, y^1\}$ and $\{x^2, y^2\}$ is contracted.

▶ **Lemma 23.** *Assume G contains an OR gadget attached to inputs V^1 and V^2. In a partial 4-sequence from G, the contractions of a, b and of c, d have to be preceded by the contraction of x^1, y^1 or the contraction of x^2, y^2.*

Proof. Assume none of the pairs $\{a, b\}, \{c, d\}, \{x^1, y^1\}, \{x^2, y^2\}$ have been contracted. Because of the fences, by Lemma 18, all the vertices $x^1, y^1, a, b, c, d, x^2, y^2$ and e are in distinct parts. Therefore contracting a and b would create a vertex of red degree at least 5, considering the (singleton) parts of x^1, y^1, c, d, e. Symmetrically contracting c and d yields at least five red neighbors, considering the (singleton) parts of x^2, y^2, a, b, e. ◀

From Lemma 23 we get the following.

▶ **Lemma 24.** *Assume G contains an OR gadget attached to inputs V^1 and V^2. In a partial 4-sequence from G, no contraction involving a vertex of V^3 can happen before either the pair x^1, y^1 or the pair x^2, y^2 is contracted.*

Proof. Suppose neither x^1, y^1 nor x^2, y^2 is contracted. By Lemma 23, the pairs $\{a, b\}$ and $\{c, d\}$ cannot be contracted. By the first item of Lemma 18, no contraction can involve a vertex of $\{a, b, c, d, e\}$. As x^3 is adjacent to all these vertices but not y^3, one cannot contract the vertical pair $\{x^3, y^3\}$. Hence by Lemma 17 no contraction can involve a vertex of V^3. ◀

Contraction of the OR gadget. We now show how to contract the OR gate when the vertical pair of one of its inputs has been contracted.

▶ **Lemma 25.** *Assume that x^1 and y^1 have been contracted into z^1, and that z^1, x^2, and y^2 all have red degree at most 3. Then there is a partial 4-sequence that contracts the whole OR gadget to a single vertex with only three red neighbors: z^1, x^2, and y^2. (The same holds symmetrically if x^2 and y^2 have been contracted into a single vertex.)*

Proof. First contract a and b into vertex α of red degree 4. At this point the fence of $\{a, b\}$ cannot be contracted yet, as this would make the red degree of α go above 4. Hence we next contract c and d into γ, decreasing the red degree of α to 3. Note that γ has only 4 red neighbors: α, e, x^2, y^2.

By Lemma 19, we can now contract the fence gadget of $\{a, b\}$ to a single vertex. Next we contract this latter vertex with α, and the resulting vertex with t; we call α' the obtained vertex. Now γ has only red degree 3, so we can contract the fence gadget of $\{c, d\}$ to a single vertex that we further contract with γ; we call γ' the obtained vertex. We contract α' and γ' in a vertex ε of red degree 3; its three red neighbors are z^1, x^2, and y^2. Again by Lemma 19, the outermost fence of the OR gadget can be contracted into a single vertex, that we finally contract with ε. This results in a vertex with three red neighbors: z^1, x^2, and y^2.

Throughout this process the red degree of z^1, x^2, and y^2 never goes above 4. Indeed the red degree of these vertices is initially at most 3, while they have exactly one black neighbor in the entire OR gadget (so at most one part to be in conflict with). ◀

4.6 Variable gadget

The variable gadget is represented in Figure 8 (see long version for a textual description).

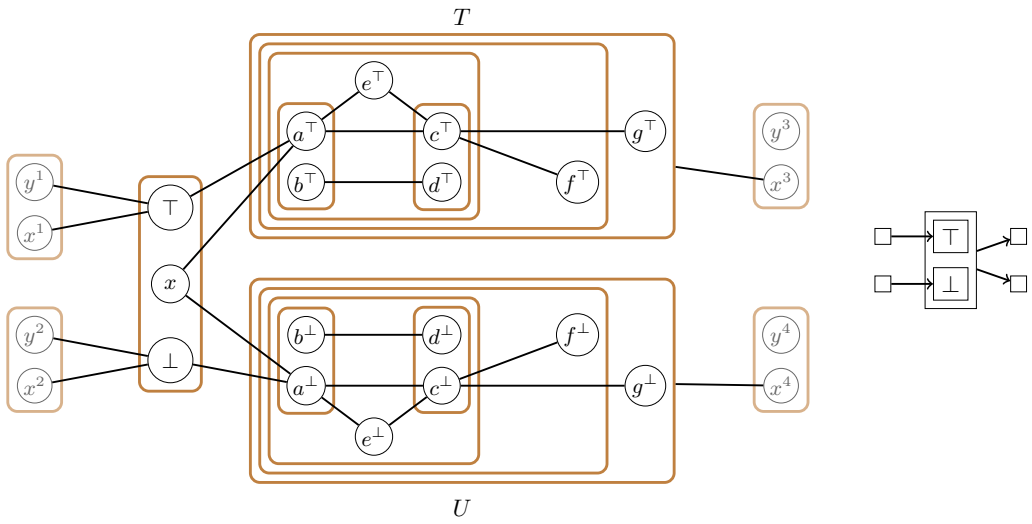


Figure 8 A variable gadget half-guarded by V^1, V^2, and with outputs V^3, V^4, and its compact representation (right).

Constraints of the variable gadget on a 4-sequence. Because of the fence gadget attached to $\{\top, x, \bot\}$, one has at some point to contract $\top$ and $\bot$ (be it with or without x). A first observation is that this has to wait that the vertical pair of V^1 or V^2 is contracted.

▶ **Lemma 26.** *Assume G has a variable gadget half-guarded by vertical sets V^1 and V^2. In a partial 4-sequence from G, the contraction of $\top$ and $\bot$ has to be preceded by the contraction of the pair x^1, y^1 or of the pair x^2, y^2.*

Proof. The pairs $\{x^1, y^1\}$, $\{x^2, y^2\}$, and $\{\top, \bot\}$ are contained in three disjoint sets S^1, S^2, S, respectively, to which a fence is attached. Thus before any of these pairs are contracted, by Lemma 18, a vertex outside $S^1 \cup S^2 \cup S$, like $a^\bot$, is in a different part than the six vertices $x^1, y^1, x^2, y^2, \top, \bot$. Therefore contracting $\top$ and $\bot$ would create a vertex with five red neighbors, considering the parts of $x^1, y^1, x^2, y^2, a^\bot$. ◀

We next show that no contraction is possible in U (resp. in T), while x and $\bot$ (resp. x and $\top$) are not contracted.

▶ **Lemma 27.** *In a partial 4-sequence, the contractions of $a^\perp$ and $b^\perp$ (resp. $a^\top$ and $b^\top$) and of $c^\perp$ and $d^\perp$ (resp. $c^\top$ and $d^\top$) have to be preceded by the contraction of x and $\perp$ (resp. x and $\top$). Therefore no contraction is possible in U (resp. T) before x and $\perp$ (resp. x and $\top$) are contracted.*

Proof. Since the two statements are symmetric, we only show it with $\perp$. Assume none of the pairs $\{x, \perp\}$, $\{a^\perp, b^\perp\}$, $\{c^\perp, d^\perp\}$ are contracted. Because of the fence gadgets, by the first item of Lemma 18, the vertices $x, \perp, a^\perp, b^\perp, c^\perp, d^\perp, e^\perp, f^\perp$ are pairwise in distinct parts. Therefore contracting $a^\perp$ and $b^\perp$ or $c^\perp$ and $d^\perp$ would create a vertex of red degree at least 5. The structure of the fence gadgets in U thus prevents any contraction. ◀

We deduce that priming the wire of $\neg x$ (resp. $+x$) can only be done after x and $\perp$ (resp. x and $\top$) are contracted.

▶ **Lemma 28.** *Assume that G has a variable gadget with outputs the vertical sets V^3 (root of the wire of $+x$) and V_4 (root of the wire of $\neg x$). In a partial 4-sequence from G, the contraction of x^4 and y^4 (resp. x^3 and y^3) has to be preceded by the contraction of x and $\perp$ (resp. x and $\top$).*

Proof. By the second item of Lemma 18 applied to the fence attached to U, the pair x^4, y^4 cannot be contracted until U is not contracted to a single vertex. Thus by Lemma 27, the pair x^4, y^4 can only be contracted after the pair $x, \perp$ is contracted. The other statement is obtained symmetrically. ◀

Contraction of the variable gadget. We show two options to contract a "half" of the variable gadget, either T and its fence, or U and its fence, into a single vertex.

▶ **Lemma 29.** *There is a partial 4-sequence that contracts x and $\top$ together, and $T \cup F^\top$ into a single vertex. Symmetrically there is a partial 4-sequence that contracts x and $\perp$ together, and $U \cup F^\perp$ into a single vertex.*

Proof. We first contract x and $\top$ into a vertex $+x$. Observe that $+x$ has exactly three red neighbors: x^1, y^1, and $a^\perp$. Thus $\{a^\top, b^\top, c^\top, d^\top, e^\top\}$ and their three fences can be contracted exactly like an OR gadget. So by Lemma 25, there is a partial 4-sequence that contracts all these vertices to a single vertex u, with three red neighbors ($+x$, $f^\top$, and $g^\top$). We can now contract u and $f^\top$ into u', followed by contracting their fence gadget $F'^\top$ into a single vertex, by Lemma 19. That pendant vertex can be contracted to u', and the result to $g^\top$, forming vertex v. Finally, again by Lemma 19, the fence $F^\top$ attached to T can be contracted into a single vertex, which can be contracted with v.

The other sequence is the symmetric. ◀

The second "half" of the variable gadget can be contracted once the vertical pairs of the *half-guards* V^1, V^2 have been contracted.

▶ **Lemma 30.** *Assume $T \cup F^\top$ (resp. $U \cup F^\perp$) has been contracted into a single vertex u, and that the pairs $\{\top, x\}$ (resp. $\{\perp, x\}$), $\{x^1, y^1\}$, and $\{x^2, y^2\}$ have been contracted into $+x$ (resp. $\neg x$), z^1, and z^2, respectively. We further assume that the red degree of z^2 (resp. z^1) is at most 3. Then there is partial 4-sequence that contracts $\top, x, \perp$ and their fence into a single vertex, and $U \cup F^\perp$ (resp. $T \cup F^\top$) into a single vertex.*

Proof. We contract $\perp$ with $+x$ into v of red degree 4. This increases the red degree of z^2 by one, which remains at most 4. We then contract $U \cup F^\perp$ into a single vertex w, like in Lemma 29. We contract u and w into y, a vertex of red degree at most 3. Now v has

degree 3. So we can contract its fence gadget by Lemma 19. We further contract v with its pendant neighbor, and finally with y. What results is a unique vertex with four red neighbors.

The other partial sequence is symmetric. ◀

4.7 Clause gadget

A *3-clause gadget* (or simply *clause gadget*) has for *inputs* three vertical sets V^1, V^2, V^3, and for *output* one vertical set V^4. It consists of combining two OR gadgets, and using long chains to make the OR gadgets *distant enough*. We add an OR gadget with input V^1 and V^2, and output V. We then add a long chain from V to V', and an OR gadget with input V' and V^3, and output V^4. The clause gadget is depicted in Figure 9. We call *first OR gadget* of the clause gadget, the one with output V, and *second OR gadget*, the one with output V^4.

Figure 9 Left: A 3-clause gadget. Right: A shorthand for the gadget.

Constraint of the clause gadget on a 4-sequence. As a consequence of Lemmas 21 and 24, we get that once a contraction involves the output, at least one of the vertical pairs of the inputs has been contracted.

▶ **Lemma 31** (⋆). *Assume G contains a clause gadget with inputs V^1, V^2, V^3 and output V^4. In a partial 4-sequence from G, any contraction involving a vertex of V^4 is preceded by the contraction of the vertical pair of one of V^1, V^2, or V^3.*

Contraction of the clause gadget. The OR gates of the clause gadgets will be contracted as specified in Lemma 25, while we wait the overall construction to describe the contraction of the wires.

4.8 Overall construction and correctness

Let $I = (C_1, \ldots, C_m)$ be an instance of 3-SAT, that is, a collection of m 3-clauses over the variables $x_1, \ldots, x_n$. We further assume that each variable appears at most twice positively, and at most twice negatively in I. The 3-SAT problem remains NP-complete with that restriction, and without $2^{o(n)}$-time algorithm unless the ETH fails; see Theorem 6. We build a trigraph H that has twin-width at most 4 if and only if I is satisfiable. As trigraph H will satisfy the condition of Lemma 11, it can be replaced by a graph G on $O(|V(H)|)$ vertices. We set L the length of the long chain to $2\lceil \log(5n+3m) \rceil = O(\log n)$, so that G has $O(n \log n)$ vertices. We now piece the gadgets described in the previous sections together.

Variable to clause gadgets. For every variable x_i, we add a variable gadget half-guarded by V_i^1, V_i^2, and with outputs V_i^3, V_i^4. We add a long chain starting at vertical set V_i^3 (resp. V_i^4), and ending at a branching vertical set from which starts two long chains stopping at vertical sets $V_i^{\top,1}$ and $V_i^{\top,2}$ (resp. $V_i^{\bot,1}$ and $V_i^{\bot,2}$). Vertical set $V_i^{\top,1}$ (resp. $V_i^{\bot,1}$) serves as the

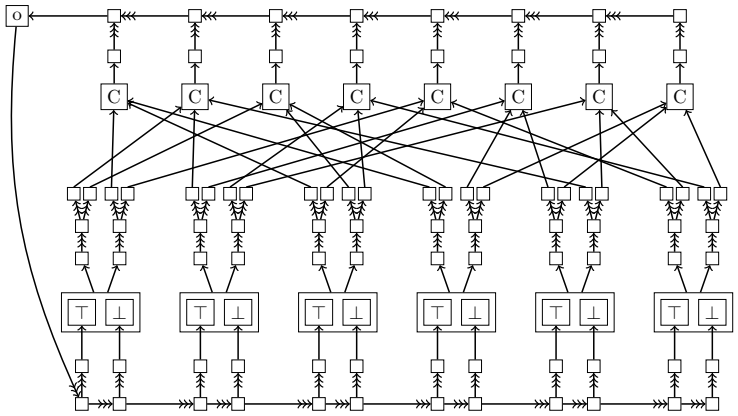


Figure 10 An example of the overall construction G on a 3-SAT instance with 6 variables and 8 clauses. The first two clauses are $\neg x_1 \vee x_3 \vee x_4$ and $x_1 \vee x_2 \vee \neg x_5$.

input of the first clause gadget in which x_i appears positively (resp. negatively), while $V_i^{\top,2}$ (resp. $V_i^{\bot,2}$) becomes the input of the second clause gadget in which x_i appears positively (resp. negatively). If a literal has only one occurrence, then we omit the corresponding vertical set, and the long chain leading to it. We nevertheless assume that each literal has at least one occurrence, otherwise the corresponding variable could be safely assigned.

Clause gadgets to global output. For every $j \in [m]$, we add a long chain from the output of the clause gadget of C_j, to a vertical set, denoted by V_j^c. For every $j \in [2, m]$, we then add a long chain starting at V_j^c and ending at V_{j-1}^c. We add an arc from V_1^c to a new vertical set V^o, which we call the *global output*.

Global output back to half-guarding the variable gadgets. For every $i \in [n]$, we add two vertical sets $V_i^{1,r}, V_i^{2,r}$, and puts a long chain starting at $V_i^{1,r}$ and ending at $V_i^{2,r}$. We add a long chain from V^o to $V_1^{1,r}$. We also add a long chain from $V_i^{2,r}$ to $V_{i+1}^{2,r}$, for every $i \in [n-1]$. Finally we add a long chain from $V_i^{a,r}$ to V_i^a for every $a \in \{1, 2\}$ and every $i \in [n]$. Recall that V_i^1 and V_i^2 are half-guarding the variable gadget of x_i.

This finishes the construction of the graph $G = G(I)$. See Figure 10 for an illustration.

If G has twin-width at most 4, then I is satisfiable. Let us consider the trigraph H obtained after the vertical pair of the global output V^o is contracted. This has to happen in a 4-sequence by the first item of Lemma 18 applied to the fence of V^o. By Lemma 21, no contraction involving vertices of the vertical sets V_i^1, V_i^2 can have happened (for any $i \in [n]$). This is because there is a directed path in the propagation digraph $\mathcal{D}(G)$ from V^o to V_i^1 and V_i^2.

Thus by Lemmas 26 and 28, for every variable x_i ($i \in [n]$), at most one of the wire of $+x_i$ and the wire of $\neg x_i$ has been primed. We can define the corresponding truth assignment $\mathcal{A}$: x_i is set to true if the wire of $\neg x_i$ is *not* primed, and to false if instead the wire of $+x_i$ is *not* primed. Besides, by Lemma 21 applied to the contraction in V^o, every vertical pair of a clause-gadget output has been contracted. Then Lemma 31 implies that the vertical pair of at least one input of each clause gadget has been contracted. But such a vertical pair can be contracted only if it corresponds to a literal set to true by $\mathcal{A}$. For otherwise, the root of the wire of that literal cannot be contracted. This implies that $\mathcal{A}$ is a satisfying assignment.

If I is satisfiable, then G has twin-width at most 4. In what follows, when we write that we *can* contract a vertex, a set, or make a sequence of contractions, we mean that there is a partial 4-sequence that does the job. Let $\mathcal{A}$ be a satisfying assignment of I. We start by contracting "half" of the variable gadget of each x_i. We add a subscript matching the variable index to the vertex and set labels of Figure 8. For every $i \in [n]$, we contract vertices x_i and $\top_i$ together, and $T_i \cup F_i^\top$ to a single vertex v_i, if $\mathcal{A}$ sets variable x_i to true, and x_i and $\bot_i$ together, and $U_i \cup F_i^\bot$ to a single vertex v_i, if instead $\mathcal{A}$ sets x_i to false. By Lemma 29, this can be done by a partial 4-sequence.

Next we contract the wire of $+x_i$ if $\mathcal{A}(x_i)$ is true, or the wire of $\neg x_i$ if $\mathcal{A}(x_i)$ is false. This is done as described in the end of Section 4.3. We contract the vertical pair of the root of the wire into z_i (the red degree of v_i goes from 1 to 2). We then contract its fence by Lemma 19. We can now contract the resulting vertex with z_i. Inductively, we contract the children of the current vertical set to single vertices, in a similar fashion. As the propagation digraph has degree at most 3, this never creates a vertex of red degree more than 4.

At the leaves of the wire (the vertical sets $V_i^{\top,1}, V_i^{\top,2}$ or $V_i^{\bot,1}, V_i^{\bot,2}$), we make an exception, and only contract the vertical pair. We then contract, by Lemma 25, all the (non-already reduced) OR gadgets (involved in clause gadgets) at least one input of which has its vertical pair contracted. After that, applying Lemma 19, we finish the contraction of the OR inputs whose vertical pairs was contracted. Next we contract each output of a contracted OR gadget into a single vertex.

In those clauses where the third literal is not satisfied by $\mathcal{A}$, the output of the clause gadget is not contracted at that point. However, as $\mathcal{A}$ is a satisfying assignment, the output of the first OR gate of the gadget is contracted. We then contract each vertical set of the long chain leading to the input of the second OR gate. We only contract the vertical pair of that input, and contract the incident OR gadget, by Lemma 25. We finally proceed by contracting the input vertical set into a single vertex, and the output vertical set into a single vertex.

At this point, each output of the clause gadgets is contracted into a single vertex. The (*not* strongly) connected component C of the propagation digraph $\mathcal{D}(G)$ containing the global output V_o is acyclic and has total degree at most 3. All the vertical sets of C with in-degree 0 (ant out-degree 1) in $\mathcal{D}(G)$ are the clause outputs, which have been contracted to single vertices. Thus each vertical set of C can be contracted to single vertices, by repeated use of Lemma 19, followed by contracting the pendant vertex (resulting from the fence contraction) with its unique (red) neighbor. Note that this process terminates by contracting the half-guards V_i^1, V_i^2 (for every $i \in [n]$). The conditions of Lemma 30 are now satisfied, so we can finish contracting each variable gadget into two vertices that we further contract together. This results in a vertex of red degree 4.

We can then contract the wire of the literal that was set to false by $\mathcal{A}$. Again, we only contract the vertical pair of the inputs of OR gates that are not contracted yet. Then we contract those OR gadgets, and finish by fully contracting the vertical set of those inputs. We next contract each output of the newly contracted OR gates. We eventually contract into a single vertex each vertical set of the long chain in clause gadgets where this was not already done.

At this point, the current trigraph H has only red edges. Thus it can be interpreted as a graph, and we write *degree* instead or *red degree*. H has $4n + 3m$ vertices of degree 3, n vertices of degree 4, and the rest of its vertices have degree 2. Because we added long chains to separate what now corresponds to vertices of degree at least 3, H is an $(\geqslant L)$-subdivision of a graph on $5n + 3m$ vertices. Since $L = 2\lceil \log(5n + 3m) \rceil$, by Lemma 9, H finally admits a 4-sequence.

References

1 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

3 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.

4 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021. arXiv:2102.03117.

5 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.10.

6 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 601–612. IEEE, 2020. doi:10.1109/FOCS46700.2020.00062.

7 Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce A. Reed, and Udi Rotics. Polynomial-time recognition of clique-width $\leqslant 3$ graphs. *Discret. Appl. Math.*, 160(6):834–865, 2012. doi:10.1016/j.dam.2011.03.020.

8 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017. doi:10.1145/3051094.

9 Anuj Dawar and Stephan Kreutzer. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009. URL: http://eccc.hpi-web.de/report/2009/131.

10 Markus Sortland Dregi and Daniel Lokshtanov. Parameterized complexity of bandwidth on trees. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 405–416. Springer, 2014. doi:10.1007/978-3-662-43948-7_34.

11 Zdenek Dvorák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

12 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.

13 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. doi:10.1137/070687256.

14 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001. doi:10.1137/S0097539799360768.

15 Fedor V. Fomin, Daniel Lokshtanov, Ivan Mihajlin, Saket Saurabh, and Meirav Zehavi. Computation of hadwiger number and related contraction problems: Tight lower bounds. *ACM Trans. Comput. Theory*, 13(2):10:1–10:25, 2021. doi:10.1145/3448639.

16 Jakub Gajarský, Petr Hlinený, Daniel Lokshtanov, Jan Obdrzálek, Sebastian Ordyniak, M. S. Ramanujan, and Saket Saurabh. FO model checking on posets of bounded width. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 963–974, 2015. doi:10.1109/FOCS.2015.63.

17 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, and Sebastian Ordyniak. Faster existential FO model checking on posets. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:8)2015.

18 Robert Ganian, Petr Hlinený, Daniel Král, Jan Obdrzálek, Jarett Schwartz, and Jakub Teska. FO model checking of interval graphs. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:11)2015.

19 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

20 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

21 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

22 Michel Habib and Christophe Paul. A simple linear time algorithm for cograph recognition. *Discret. Appl. Math.*, 145(2):183–197, 2005. doi:10.1016/j.dam.2004.01.011.

23 Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

24 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

25 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

26 Toshinobu Kashiwabara. NP-completeness of the problem of finding a minimal-clique number interval graph containing a given graph as a subgraph. In *Proc. 1979 Int. Symp. Circuit Syst*, pages 657–660, 1979.

27 Tuukka Korhonen. Single-exponential time 2-approximation algorithm for treewidth. *CoRR*, abs/2104.07463, 2021. arXiv:2104.07463.

28 Thomas Lengauer. Black-white pebbles and graph separation. *Acta Informatica*, 16:465–475, 1981. doi:10.1007/BF00264496.

29 Tatsuo Ohtsuki, Hajimu Mori, Ernest S. Kuh, Toshinobu Kashiwabara, and Toshio Fujisawa. One-dimensional logic gate assignment and interval graphs. In *The IEEE Computer Society's Third International Computer Software and Applications Conference, COMPSAC 1979, 6-8 November, 1979, Chicago, Illinois, USA*, pages 101–106. IEEE, 1979. doi:10.1109/CMPSAC.1979.762474.

30 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

31 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

32 Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theor. Comput. Sci.*, 615:120–125, 2016. doi:10.1016/j.tcs.2015.11.039.

33 James B. Saxe. Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM J. Algebraic Discret. Methods*, 1(4):363–369, 1980. doi:10.1137/0601042.

34 André Schidler and Stefan Szeider. A SAT approach to twin-width. *CoRR (accepted at ALENEX '22)*, abs/2110.06146, 2021. arXiv:2110.06146.

35 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discret. Appl. Math.*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

Memoryless Worker-Task Assignment with Polylogarithmic Switching Cost

Aaron Berger
MIT, Cambridge, MA, USA

William Kuszmaul
MIT, Cambridge, MA, USA

Adam Polak
EPFL, Lausanne, Switzerland

Jonathan Tidor
MIT, Cambridge, MA, USA

Nicole Wein
DIMACS, Piscataway, NJ, USA

Abstract

We study the basic problem of assigning memoryless workers to tasks with dynamically changing demands. Given a set of w workers and a multiset $T \subseteq [t]$ of $|T| = w$ tasks, a memoryless worker-task assignment function is any function ϕ that assigns the workers $[w]$ to the tasks T based only on the current value of T. The assignment function ϕ is said to have switching cost at most k if, for every task multiset T, changing the contents of T by one task changes $\phi(T)$ by at most k worker assignments. The goal of memoryless worker task assignment is to construct an assignment function with the smallest possible switching cost.

In past work, the problem of determining the optimal switching cost has been posed as an open question. There are no known sub-linear upper bounds, and after considerable effort, the best known lower bound remains 4 (ICALP 2020).

We show that it is possible to achieve polylogarithmic switching cost. We give a construction via the probabilistic method that achieves switching cost $O(\log w \log(wt))$ and an explicit construction that achieves switching cost $\text{polylog}(wt)$. We also prove a super-constant lower bound on switching cost: we show that for any value of w, there exists a value of t for which the optimal switching cost is w. Thus it is not possible to achieve a switching cost that is sublinear *strictly* as a function of w.

Finally, we present an application of the worker-task assignment problem to a metric embeddings problem. In particular, we use our results to give the first low-distortion embedding from sparse binary vectors into low-dimensional Hamming space.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Random projections and metric embeddings; Mathematics of computing → Combinatorics

Keywords and phrases Distributed Task Allocation, Metric Embeddings, Probabilistic Method

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.19

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2008.10709

Funding *Aaron Berger*: NSF Graduate Research Fellowship Program DGE-1745302.
William Kuszmaul: NSF GRFP fellowship and a Fannie and John Hertz Fellowship. Research was partially sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 19; pp. 19:1–19:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
Adam Polak: Swiss National Science Foundation project *Lattice Algorithms and Integer Programming* (185030). Part of this work was done at Jagiellonian University, supported by National Science Center of Poland grant 2017/27/N/ST6/01334.
Jonathan Tidor: NSF Graduate Research Fellowship Program DGE-1745302.
Nicole Wein: This work was done at MIT, supported by NSF Grant CCF-1514339.

1 Introduction

The general problem of **distributed task allocation**, where a group of agents must collectively allocate themselves to tasks, has been studied in a wide variety of settings where the agents have varying degrees of communication, memory, knowledge of the system, and faultiness (see Georgiou and Shvartsman's book [15] for a survey). In this paper we are interested in the dynamic version of a distributed task allocation problem, that is, where the demands for each task are changing over time. Dynamic task allocation has been the focus of a great deal of both empirical and theoretical work in areas such as swarm robotics [20, 32, 26, 27, 21, 23, 34] and collective insect behavior [4, 31, 30, 33, 34].

Although there are many possible variations of the dynamic task allocation problem (in particular, in terms of what the capabilities of the workers are), most share a basic common structure. For positive integers w and t, there are w workers $1, 2, \ldots, w$, and there is a multiset $T \subseteq [t]$ of $|T| = w$ tasks.[1] The multiset T changes gradually over time: in each time step, one new task is added and one old task is removed. The goal is to maintain a dynamic assignment of workers $\{1, 2, \ldots, w\}$ to tasks T such that the **switching cost** is as small as possible, i.e., the number of worker-task assignments that change each time that T changes is bounded by some small quantity.

The goal of studying dynamic distributed task allocation is to answer the following question: to what degrees are various capabilities (i.e., memory, communication, knowledge of the system, computational power, etc.) needed for the workers to guarantee a small switching cost?

Motivated in part by applications to swarm robotics and collective ant behavior, recent work [33, 34] has focused on the question of what happens when workers are *completely memoryless*. At any given moment each worker $i \in \{1, 2, \ldots, w\}$ must determine which task $\tau \in T$ they are assigned to solely as a function of the current task multiset T; the workers do not remember anything about the past system states or worker assignments. Note that this memoryless-ness requirement is sufficiently strong that it also implies communicationless-ness – indeed, if workers cannot remember where they were in previous steps, and all that each worker knows is the current multiset of tasks, then there is no worker-specific information to be shared. Being memoryless and communicationless is especially important to settings where a worker might suffer a fault and thus "reboot"; in this case, the worker can determine which task they are assigned to without relying on what was stored in its memory and without knowledge of which tasks other workers are assigned to at the moment [34].

[1] If a task j appears $m_T(j)$ times in the multiset T, then one should think of the task as having a current demand of $m_T(j)$ workers. By writing $T \subseteq [t]$ we mean that the elements of multiset T are from $\{1, 2, \ldots, t\}$, but the multiplicity of each element can be arbitrarily large.

It remains an open question [34, 33] whether memoryless workers can achieve even a *sub-linear* switching cost. In this paper we show that, not only is sub-linear switching cost possible, but it is even possible to achieve a polylogarithmic switching cost of $\text{polylog}(wt)$. We also prove a lower bound for any algorithm that wishes to parameterize by only w and not t: for any w, if $t \gg w$ is sufficiently large, then the trivial switching cost of w becomes optimal.

Formal problem statement. Throughout this paper, we shall refer to the memoryless dynamic distributed task allocation problem simply as the **worker-task assignment problem**. Formally, the worker-task assignment problem is defined as follows. There are w workers $1, 2, \ldots, w$ and t tasks $1, 2, \ldots, t$. A **worker-task assignment function** ϕ is a function that takes as input a multiset T of w tasks, and produces an assignment of workers to tasks such that the number of workers assigned to a given task $\tau \in T$ is equal to the multiplicity of τ in T.

Two task multisets T_1, T_2 of size w are said to be *adjacent* if they agree on exactly $w - 1$ elements; that is, $|T_1 \setminus T_2| = |T_2 \setminus T_1| = 1$.[2] The switching cost between two adjacent task multisets T_1, T_2 of size w is defined as the number of workers whose assignment changes between $\phi(T_1)$ and $\phi(T_2)$. The **switching cost of** ϕ is defined to be the maximum switching cost over *all* pairs of adjacent task multisets. The goal of the worker-task assignment problem is to design a worker-task assignment function with the minimum possible switching cost.

Prior work in the memoryless setting. The optimal switching cost is trivially between 1 and w. Improving either of these bounds substantially has proven difficult, however.

Su, Su, Dornhaus, and Lynch [33] initiated the study of the worker-task assignment problem and observed that assigning the workers to tasks in numerical order achieves a switching cost of $\min(t - 1, w)$. They also proved a lower bound of 2 on switching cost, and showed a matching upper bound in the case where $w \leq 6$ and $t \leq 4$.

Subsequent work by Su and Wein [34], in ICALP 2020, pushed further on the lower-bound side of the problem. They proved that a switching cost of 2 is not always possible in general. They show that, if $t \geq 5$ and $w \geq 3$, then any worker-task assignment function must have switching cost at least 3; and if t is sufficiently large in terms of w (i.e., it is a tower of height $w - 1$), then the switching cost must be at least 4.

The bounds by [33] and [34] have until now remained state-of-the-art. It remains unknown whether the optimal switching cost is small (it could be as small as 4) or large (it could be as large as $\min(t - 1, w)$). And even achieving a lower bound of 4 on switching cost [34] has required a quite involved argument.

1.1 This paper

The contributions of this paper are twofold. First, we present significant progress on the worker-task assignment problem, resulting in both a polylogarithmic upper bound and a super-constant lower bound for the optimal switching cost – our results are interesting in part because of their use of randomized techniques to construct a deterministic assignment function. Second, we explore a natural connection between the worker-task assignment problem and the metric embedding problem of densification into Hamming space, and we transform our progress on the former into new results for the latter.

[2] Let $m_A(i)$ denote the number of times element i appears in multiset A. Then, for any two multisets A and B, we define multisets $A \setminus B$, $A \cup B$, and $A \cap B$ to be such that $m_{A \setminus B}(i) = \max(0, m_A(i) - m_B(i))$, $m_{A \cup B}(i) = \max(m_A(i), m_B(i))$, and $m_{A \cap B}(i) = \min(m_A(i), m_B(i))$, for every element i.

Results on worker-task assignment. Our first result establishes that it is possible to construct a worker-task assignment function with $O(\log w \log(wt))$ switching cost. This resolves the open question as to whether memoryless worker can allocate themselves to tasks with strong worst-case guarantees.

▶ **Theorem 1.** *There exists a worker-task assignment function that achieves switching cost* $O(\log w \log(wt))$.

Theorem 1 is proven via the probabilistic method and is thus non-constructive. By replacing random hash functions with strong dispersers, however, we show that one can construct an explicit worker-task assignment function with polylogarithmic switching cost.

▶ **Theorem 2.** *There is an explicit worker-task assignment function that achieves switching cost* $O(\text{polylog}(wt))$.

Both Theorems 1 and 2 continue to hold in the more general setting where the size of T changes over time. That is, T is permitted to be any multiset of $[t]$ of size w or smaller. Two task multisets T_1, T_2 of different sizes are considered adjacent if they satisfy $\big||T_1| - |T_2|\big| = 1$ and $|(T_1 \cup T_2) \setminus (T_1 \cap T_2)| = 1$. If $|T| < w$, then our worker-task assignment function assign workers $1, \ldots, |T|$ to tasks, and leaves workers $|T| + 1, \ldots, w$ unassigned.

Finally, from the lower bounds side, we prove that no algorithm can achieve sub-linear switching cost as a function of only w. Theorem 3 says that, if t is sufficiently larger than w, then for any worker-task assignment function, there must exist a pair of adjacent task multi-sets that forces *all* of the w workers to be reassigned. In the statement of the theorem, and throughout the paper, $\text{tow}(n)$ is defined to be a tower of twos of height n (i.e., the inverse of the $\log^*$ function).

▶ **Theorem 3.** *For every* w *and* $t \geq \text{tow}(\Omega(w))$*, every worker-task assignment function has switching cost* w.

This represents the first super-constant lower bound for the switching cost of a worker-task assignment function. Another way to think about the theorem is that for every t, there is some w for which any worker-task assignment function has switching cost at least $\Omega(\log^*(t))$. Therefore our bounds leave a gap between $\log^*$ and polylog in terms of dependence on t.

An application to metric embeddings: Densification into Hamming space. The problem of embedding one metric space $\mathcal{M}_1$ into another metric space $\mathcal{M}_2$ with small distortion has been widely studied in many contexts and has found many algorithmic applications [29, 7, 9, 8, 13, 2, 6, 10, 1, 5, 19, 24, 3, 22].

Bourgain [5] initiated the study of metric embeddings (into normed spaces) by showing that $O(\log |M|)$-distortion embeddings into ℓ_2 are possible for any space M. Much of the subsequent work has focused either on embeddings between exponentially large metric spaces [29, 7, 9, 8, 13, 2, 10, 3], or on embeddings with sub-logarithmic distortion [19, 24, 3, 10].

One natural question is that of **densification**: can one embed sparse vectors from a high-dimensional ℓ_1-space into a low-dimensional ℓ_1-space? That is, if V_n^k is the set of n-dimensional vectors with k non-zero entries, what is the smallest m for which V_n^k can be embedded into m-dimensional ℓ_1-space with low distortion? Charikar and Sahai [10] were the first to consider this problem, and showed how to achieve an output dimension of $m = O((k/\epsilon)^2 \log n)$ with distortion $1 + \epsilon$. They also showed how to apply densification to the related problem of embedding arbitrary tree metrics into low-dimensional ℓ_1-space [10]. Subsequently, Berinde et al. [3] used expander graphs in order to achieve $m = O(k \log(n/k)/\epsilon^2)$ with distortion

$1+\epsilon$. They then used their densification embedding as a tool to perform sparse signal recovery [3, 17, 16, 18]. Both of the known densification algorithms [10, 3] rely on linear sketches, in which each vector $\vec{x} \in V_n^k$ is mapped to a vector of the form $\sum_i x_i \vec{b_i}$ for some set of vectors $\vec{b_1}, \ldots, \vec{b_n}$.

The prior work on densification [10, 3] has focused on embedding into ℓ_1-space. In Section 5, we consider the same problem over Hamming space, where the distance between two vectors $\vec{x}, \vec{y}$ is given by $\mathrm{Ham}(\vec{x}, \vec{y}) = |\{i \mid x_i \neq y_i\}|$. Densification over Hamming space requires new techniques due to the fact that summations of vectors (and thus linear sketches) do not behave well in Hamming space.

Let $\mathcal{H}_n^k$ denote the set of n-dimensional binary vectors with k ones. Let $\mathcal{H}_k(n)$ denote the set of k-dimensional vectors with entries from $[n]$. We show that $\mathcal{H}_n^k$ can be embedded into $\mathcal{H}_k(n)$ with distortion $O(\log n \log k)$.

▶ **Theorem 4.** *There exists a map $\phi : \mathcal{H}_n^k \to \mathcal{H}_k(n)$ such that, for every $\vec{x}, \vec{y} \in \mathcal{H}_n^k$,*

$$\mathrm{Ham}(\vec{x}, \vec{y})/2 \leq \mathrm{Ham}(\phi(\vec{x}), \phi(\vec{y})) \leq O(\log n \log k)\, \mathrm{Ham}(\vec{x}, \vec{y}).$$

The densification embedding is a simple application of the worker-task assignment problem. In order to embed a vector $\vec{x} \in \mathcal{H}_n^k$ into $\mathcal{H}_k(n)$, we simply assign the workers $\{1, 2, \ldots, k\}$ to the task set $T = \{i \mid \vec{x}_i = 1\}$, and then we construct the vector $\vec{y}$ whose i-th coordinate denotes the task in T to which worker i is assigned. This map transforms the switching cost in the worker-task assignment problem into the distortion of the metric embedding.

The densification embedding is optimal in two senses. First, the target space of the embedding must have $\Omega(k)$ coordinates simply in order to allow for distances of $\Omega(k)$. Second, when $k \ll n$, any embedding of $\mathcal{H}_n^k$ to k-dimensional Hamming space must use $\Omega(\log n)$ bits per coordinate, simply in order so that the embedding is an injection. It is not clear whether the distortion achieved by our embedding is optimal, however, and it remains open whether smaller distortion can be achieved by allowing for a larger target-space dimension.

We remark that the basic relationship between worker-task assignment and densification embeddings problem has already implicitly been observed in previous work on lower bounds, as a way to formalize what makes the worker-task assignment problem difficult [34]. In contrast, here we are using the relationship as an avenue to obtain improved upper bounds for the densification problem.

2 Technical overview

This section gives an overview of the main technical ideas in the paper. For simplicity, the section will treat the task multiset $T \subseteq [t]$ as always being a set (rather than a multiset). As discussed in Section 3, one can formally reduce from the multiset case to the set case, at the cost of t being replaced with $t' = wt$.

2.1 A warmup: The random-permutation algorithm

We begin by describing a simple assignment function that we call the **random-permutation algorithm**. The random-permutation algorithm does not necessarily achieve small switching cost, but it does have the property that for any two adjacent task sets $T_1, T_2 \subseteq [t]$, the switching cost between T_1 and T_2 is $O(\log w)$ with high probability in w.

The algorithm. The random-permutation algorithm assigns to each worker $i \in [w]$ a random permutation

$$\sigma_i = \langle \sigma_i(1), \sigma_i(2), \ldots, \sigma_i(t) \rangle$$

of the numbers $[t]$. We think of worker i as **preferring** task $\sigma_i(j)$ over task $\sigma_i(j+1)$ for all $j \in [t-1]$.

Suppose we wish to assign workers to tasks T. The random-permutation algorithm assigns the workers $1, 2, \ldots, w$ to tasks $\tau_1, \tau_2, \ldots, \tau_w \in T$ one by one in order of worker ID, assigning worker i to the task that it most prefers out of the tasks in T that have not yet been assigned a worker.

For each $i \in [w]$, we define the ***i*-remainder tasks** to be the tasks $T \setminus \{\tau_1, \ldots, \tau_i\}$. That is, the i-remainder tasks are the tasks that remain after the first i workers are assigned. This means that worker $i+1$ is assigned to the i-remainder task that it most prefers.

Analyzing expected switching cost. Let $T_1, T_2 \subseteq [t]$ be adjacent task sets of size w. We begin by showing that the expected switching cost from T_1 to T_2 is $O(\log w)$.

Let r and s be such that $T_1 = (T_2 \cup \{r\}) \setminus \{s\}$. Let ψ_1 and ψ_2 denote the assignments produced by the random permutation algorithm for T_1 and T_2, respectively. Let A_i and B_i denote the set of i-remainder tasks during the constructions of ψ_1 and ψ_2, respectively.

The key to analyzing the random-permutation algorithm is to compare the i-remainder sets A_i and B_i for each $i \in [w]$. We claim that $|A_i \setminus B_i| \leq 1$ for all $i \in [w]$. We prove this by induction on i: suppose that $A_{i-1} = (B_{i-1} \cup \{a\}) \setminus \{b\}$, and suppose for contradiction that $|A_i \setminus B_i| \geq 2$. If either ψ_1 assigns worker i to task a, or ψ_2 assigns worker i to task b, then we would be guaranteed that $|A_i \setminus B_i| \leq 1$, a contradiction. Thus ψ_1 and ψ_2 must each assign worker i to a task in $A_{i-1} \cap B_{i-1}$. But this means that, in both assignments, worker i is assigned to the task in $A_{i-1} \cap B_{i-1}$ that worker i most prefers. Thus ψ_1 and ψ_2 assign worker i to the same task, again contradicting that $|A_i \setminus B_i| \geq 2$.

We now analyze the probability of ψ_1 and ψ_2 differing in their assignment of worker i. Since A_i contains at most one element a not in B_i, the probability that worker i prefers a over all elements in B_i is at most $1/|B_i| = 1/(w-i+1)$. Similarly, since B_i contains at most one element b not in A_i, the probability that worker i prefers b over all elements in A_i is at most $1/|A_i| = 1/(w-i+1)$. By the union bound, it follows that the probability of ψ_1 and ψ_2 assigning worker i to different tasks is at most $2/(w-i+1)$.

By linearity of expectation, the expected switching cost between T_1 and T_2 is at most

$$\sum_{i=1}^{w} \frac{2}{w-i+1} = O(\log w).$$

Why a union bound fails for worst-case switching cost. By using Chernoff-style bounds, one can modify the above analysis of the random-permutation algorithm to show that, with high probability in w (i.e., probability $1 - 1/\operatorname{poly} w$), the switching cost between T_1 and T_2 is $O(\log w)$.

On the other hand, achieving a switching cost of $O(\log w)$ for *all* pairs (T_1, T_2) of adjacent task sets presents a challenge because there are $\binom{w+1}{2}\binom{t}{w+1}$ such pairs that must be considered. When $w = t/2$, the number of distinct pairs (T_1, T_2) of adjacent task sets exceeds $2^t \geq 2^w$.

Thus, the probability bounds achieved by the random-permutation algorithm are nowhere near high enough to enable a union bound over all adjacent worker-set pairs. We call this the **union-bound magnitude issue**.

2.2 An algorithm with small switching cost

We now describe a randomized assignment algorithm $\mathcal{A}$ that, with high probability in t, achieves switching cost $O(\log w \log t)$ on *all* adjacent task-sets $T_1, T_2 \subseteq [t]$ of size w. That is, with high probability, $\mathcal{A}$ produces an assignment function satisfying the requirements of Theorem 1. The algorithm $\mathcal{A}$ is called the **multi-round balls-to-bins algorithm**.

The multi-round balls-to-bins algorithm essentially flips the approach taken by the random-permutation algorithm. One can think of the random-permutation algorithm as consisting of w phases in which each phase *deterministically* assigns exactly one worker to a task, and then the phases *probablistically* incur small switching cost. In contrast, the multi-round balls-to-bins algorithm consists of $O(\log w)$ phases, where each phase *probabilistically* assigns some number of workers to tasks, and each phase *deterministically* incurs small switching cost. Whereas the failure mode of the random-permutation algorithm is that a high-switching cost may occur, the failure mode of the multi-round balls-to-bins algorithm is that some workers may be left unassigned. As we shall see later, this distinction plays an important role in solving the union-bound magnitude issue.

Structure of the multi-round balls-to-bins algorithm. We begin with a succinct description of the algorithm $\mathcal{A}$. For each i from 1 to $\log_{1.1} w$, repeat the following hashing procedure $c \log t$ many times. Initialize a hash table consisting of $w/(1.1)^i$ bins and randomly hash each unassigned worker and each unassigned task into this table. For each bin that contains at least one worker and one task, assign the minimum worker in that bin to the minimum task in that bin.

In more detail, the algorithm $\mathcal{A}$ is the composition of $O(\log w)$ sub-algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots$. Each of $\mathcal{A}_1, \mathcal{A}_2, \ldots$ are **partial-assignment** algorithms, meaning that $\mathcal{A}_i$ assigns some subset of the workers to some subset of the tasks in T, possibly leaving workers and tasks unassigned. Note that the input to algorithm $\mathcal{A}_i$ is the set of workers/tasks that remain unassigned by $\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}$. Thus one can think of the input to $\mathcal{A}_i$ as being a pair (W, T) where $W \subseteq [w]$ is a set of workers, $T \subseteq [t]$ is a set of tasks, and $|W| = |T|$.

The algorithm $\mathcal{A}_1$'s responsibility is to assign enough workers to tasks so that at most $w/1.1$ workers remain unassigned. Algorithm $\mathcal{A}_2$ is then executed on the remaining (i.e., not-yet-assigned) workers and tasks, and is responsible for assigning enough workers to tasks so that at most $w/(1.1)^2$ workers remain unassigned. Continuing like this, algorithm $\mathcal{A}_i$ is executed on the workers/tasks that remain unassigned by all of $\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}$, and is responsible for assigning enough workers to tasks that at most $r_i = w/(1.1)^i$ workers in W remain unassigned.

Each of the $\mathcal{A}_i$'s are randomized algorithms, meaning that they have some probability of failure. The failure mode for $\mathcal{A}_i$ is *not* high-switching cost, however. In fact, as we shall see later, each $\mathcal{A}_i$ deterministically contributes at most $O(\log w)$ to the switching cost. Instead, the way in which $\mathcal{A}_i$ can fail is that it may leave more than r_i workers unassigned. This means that the failure mode for the full algorithm $\mathcal{A}$ is that it may fail to assign all of the workers in W to tasks in T.

Applying the probabilistic method to $\mathcal{A}_1, \mathcal{A}_2, \ldots$. Before describing the partial-assignment algorithms $\mathcal{A}_i$ in detail, we first describe how our analysis of algorithm $\mathcal{A}$ overcomes the union-bound magnitude issue.

Recall that each algorithm $\mathcal{A}_i$ is responsible for reducing the number of remaining workers to $r_i = w/(1.1)^i$. We will later see that each $\mathcal{A}_i$ has a failure probability p_i that is a function of r_i and t, namely,

$$p_i = \frac{1}{t^{\Omega(r_i)}}.$$

As i grows, the failure probability p_i of $\mathcal{A}_i$ becomes larger, making it impossible to union-bound over exponentially many pairs of task sets T_1, T_2.

An important insight is that, if all of $\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}$ succeed (i.e., they each assign the number of workers that they are responsible for assigning) then the number of workers and tasks that $\mathcal{A}_{i-1}$ is executed on is only $O(r_i)$. That is, if we think of the inputs to $\mathcal{A}_i$ as being pairs (W, T) where $W \subseteq [w]$ is a set of workers and $T \subseteq [t]$ is a set of tasks, the set of inputs (W, T) that algorithm $\mathcal{A}_{i-1}$ must succeed on is only the inputs for which $|W| = |T| \leq O(r_i)$. The number of such inputs is at most $t^{O(r_i)}$. In other words, even though the failure probability p_i of algorithm $\mathcal{A}_i$ increases with i, the number of inputs over which we must apply a union bound decreases. By a union bound, we can deduce that $\mathcal{A}_i$ has a high probability in t of succeeding on all relevant inputs (W, T). Combining this analysis over all of the partial-assigning algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots$, we get that the full assignment algorithm $\mathcal{A}$ also succeeds with high probability in t. In particular, we have proven that there exists a deterministic assignment function with the desired switching cost, and that such a function can be obtained with high probability by the randomized algorithm $\mathcal{A}$.

Designing $\mathcal{A}_i$. Each algorithm $\mathcal{A}_i$ is a composition of $\Theta(\log t)$ algorithms $\mathcal{A}_{i,1}, \mathcal{A}_{i,2}, \mathcal{A}_{i,3}, \ldots$, each of which individually is a partial assignment algorithm.

Each algorithm $\mathcal{A}_{i,j}$ takes a simple balls-in-bins approach to assigning some subset of the remaining workers to some subset of the remaining tasks.

In particular, $\mathcal{A}_{i,j}$ places the workers into bins $1, 2, \ldots, r_i$ by hashing each worker to a bin (using a random function from $[w]$ to $[r_i]$). Similarly, the tasks are placed into bins $1, 2, \ldots, r_i$ by hashing each task to a bin. If a bin b contains both at least one worker and at least one task, then the smallest-numbered worker in bin b is assigned to the smallest-number task in bin b.

Note that each of the algorithms $\mathcal{A}_{i,1}, \mathcal{A}_{i,2}, \mathcal{A}_{i,3}, \ldots$ are identical copies of one-another, except using different random bits. Also note all of the $\mathcal{A}_i$'s are defined in the same way as each other, except the number of bins hashed to decreases as i increases. As we shall see shortly, the reason for having $\mathcal{A}_i$ consist of $\Theta(\log t)$ sub-algorithms is to enable probability amplification later in the analysis.

Bounding the switching cost. The partial assignment algorithms $\mathcal{A}_{i,j}$ are designed to satisfy two essential properties, which we prove formally in the full proof. These two properties can then be combined to bound the switching cost of the full algorithm $\mathcal{A}$.

Compatibility: Let $I_1 = (W_1, T_1)$ and $I_2 = (W_2, T_2)$ be inputs to $\mathcal{A}_{i,j}$. Suppose I_1 and I_2 are **unit distance**, meaning that

$$|W_1 \setminus W_2| + |W_2 \setminus W_1| + |T_1 \setminus T_2| + |T_2 \setminus T_1| \leq 2.$$

Let $I_1' = (W_1', T_1')$ and $I_2' = (W_2', T_2')$ be the workers and tasks that remain unassigned when $\mathcal{A}_{i,j}$ is executed on each of I_1 and I_2, respectively. Then I_1' and I_2' are guaranteed to also be unit-distance.

Low Switching Cost: The switching cost of $\mathcal{A}_{i,j}$ is $O(1)$. That is, if $I_1 = (W_1, T_2)$ and $I_2 = (W_2, T_2)$ are inputs to $\mathcal{A}_{i,j}$, and I_1 and I_2 are unit-distance, then the worker-task assignments made by $\mathcal{A}_{i,j}$ on each of I_1 and I_2 differ by at most $O(1)$ assignments.

Consider two adjacent task sets T_1 and T_2. When we execute $\mathcal{A}$ on T_1 and T_2, respectively, we use $I_1^{i,j}$ and $I_2^{i,j}$, respectively, to denote the worker/task input that are given to partial-assignment algorithm $\mathcal{A}_{i,j}$.

The Compatibility property of the $\mathcal{A}_{i,j}$'s guarantees by induction that, for each $\mathcal{A}_{i,j}$ the worker/task inputs $I_1^{i,j}$ and $I_2^{i,j}$ are unit-distance (or zero-distance). The Low-Switching-Cost property then guarantees that each $\mathcal{A}_{i,j}$ contributes at most $O(1)$ to the switching cost of $\mathcal{A}$. Since there are $O(\log t \log w)$ $\mathcal{A}_{i,j}$'s, this bounds the total switching cost of $\mathcal{A}$ by $O(\log t \log w)$.

Deriving the success probabilities. Next we analyze the probability of $\mathcal{A}_i$ failing on a given worker/task input (W, T). Recall that the only way in which $\mathcal{A}_i$ might fail is if more than r_i workers remain unassigned after $\mathcal{A}_i$ finishes. Additionally, since we need only consider cases where $\mathcal{A}_{i-1}$ succeeds, we can assume that $r_i \le |W|, |T| \le 1.1 r_i$.

Let q denote the number of workers that $\mathcal{A}_{i,1}$ assigns to tasks. Given that $r_i \le |W|, |T| \le 1.1 r_i$, a simple analysis of $\mathcal{A}_{i,1}$ shows that $\mathbb{E}[q] \ge r_i/5$. On the other hand, using McDiarmid's inequality, one can perform a balls-in-bins style analysis in order to show that $\Pr[\mathbb{E}[q] - q > r_i/10] \le 2^{-\Omega(r_i)}$. This means that $\mathcal{A}_{i,1}$ has probability at most $2^{-\Omega(r_i)}$ of leaving more than r_i workers unassigned.

In order for $\mathcal{A}_i$ to fail (i.e., $\mathcal{A}$ leaves more than r_i workers unassigned), *all* of sub-algorithms $\mathcal{A}_{i,1}, \mathcal{A}_{i,2}, \ldots$ would have to individually fail. Since there are $\Theta(\log t)$ sub-algorithms, the probability of them all failing is

$$p_i = 2^{-\Omega(r_i \log t)} = t^{-\Omega(r_i)}.$$

This allows us to apply the probabilistic method to the $\mathcal{A}_i$'s in order to bound the probability of any $\mathcal{A}_i$ failing on any input, as desired.

An explicit construction with polylogarithmic switching cost. The multi-round balls-to-bins algorithm gives a non-explicit approach to constructing an assignment function with low switching cost. The approach is non-explicit because it relies on the probabilistic method.

In the full version of the paper[3] we show how to obtain an explicit algorithm with switching cost polylog wt. The basic idea is to replace random hash functions, used to place workers and tasks into bins, with functions obtained from pseudorandom objects called strong dispersers. Instead of trying a number of random hash functions within the $\mathcal{A}_{i,j}$'s, we instead iterate over all of the hash functions from a small family given by a strong disperser [28].

2.3 A lower bound on switching cost

Define $s_{w,t}$ to be the optimal switching cost for assignment functions that assign workers $1, 2, \ldots, w$ to multisets of w tasks from the universe $[t]$. The upper bounds in this paper establish that $s_{w,t} \le O(\log w \log(wt))$. It is natural to wonder whether smaller bounds can be achieved, and in particular, whether a small switching cost that depends only on w can be achieved.

It trivially holds that $s_{w,t} \le w$. We show that when t is sufficiently large relative to w, there is a matching lower bound of $s_{w,t} \ge w$. In fact, our lower bound only uses the evaluation of the assignment function on sets (as opposed to multisets).

Consider an assignment function ϕ that, given a multiset T of tasks with elements from $[t]$ of w tasks, produces an assignment of workers $[w]$ to tasks T. Our goal will be to find tasks $\tau_1 < \tau_2 < \cdots < \tau_{w+1}$ such that if $\phi(\{\tau_1, \ldots, \tau_w\})$ assigns worker i to task $\tau_{\pi(i)}$ for some permutation π of $[w]$, then $\phi(\{\tau_2, \ldots, \tau_{w+1}\})$ assigns worker i to task $\tau_{\pi(i)+1}$. The existence of such a configuration immediately implies that ϕ has switching cost w.

[3] https://arxiv.org/abs/2008.10709

We use an application of the hypergraph Ramsey theorem to show that, when t is large enough, a configuration of the type described in the above paragraph must exist. Let $K_t^{(w)}$ denote the complete w-uniform hypergraph on t vertices. This is just the set of w-element subsets of $[t]$, which correspond to sets of tasks. For each hyperedge $T = \{\tau_1, \ldots, \tau_w\}$, where $1 \leq \tau_1 < \cdots < \tau_w \leq t$, we color the hyperedge T by a color π where $\tau_{\pi(i)}$ is the task assigned to worker i.

This gives a coloring of the hyperedges of $K_t^{(w)}$ by $w!$ colors, each color being a permutation of $[w]$. By the hypergraph Ramsey theorem, if t is large enough in terms of w, there must exist $w+1$ vertices $\tau_1, \ldots, \tau_{w+1}$ so all the hyperedges formed by the vertices have the same color π. By examining the hyperedges $\{\tau_1, \ldots, \tau_w\}$ and $\{\tau_2, \ldots, \tau_{w+1}\}$, it follows that $\phi(\{\tau_1, \ldots, \tau_w\})$ assigns each worker i to task $\tau_{\pi(i)}$ and that $\phi(\{\tau_2, \ldots, \tau_{w+1}\})$ assigns each worker i to task $\tau_{\pi(i)+1}$, as desired.

3 Achieving switching cost $O(\log w \log(wt))$

In this section, we prove the following theorem.

▶ **Theorem 1.** *There exists a worker-task assignment function that achieves switching cost* $O(\log w \log(wt))$.

We demonstrate the existence of such a function via the probabilistic method, showing that there is a randomized construction that produces a low-switching cost worker-task assignment function with nonzero probability. In the full version of the paper, we also show how to derandomize the construction at the cost of a few extra log factors.

From multisets to sets. We begin by showing that, without loss of generality, we can restrict our attention to task multisets T that are sets (rather than multisets). We reduce from the multiset version of the problem with w workers and t tasks to the set version of the problem with w workers and wt tasks.

▶ **Lemma 5.** *Define $n = wt$. Let ϕ be a worker-task assignment function that assigns workers $[w]$ to task sets $T \subseteq [n]$ (note that ϕ is defined only on task* sets *T, and not on multisets). Let s be the switching cost of ϕ (considering only pairs of adjacent subsets of $[n]$, rather than adjacent sub-multisets). Then there exists a worker-task assignment function ϕ' assigning workers $[w]$ to task multisets $T \subseteq [t]$, such that ϕ' also has switching cost s.*

Proof. When discussing the assignment function ϕ, we think of its input task-set T as consisting of elements from $[t] \times [w]$ rather than elements of $[tw]$.

With this in mind, we construct ϕ' as follows. Given a task multiset $T \subseteq [t]$, define the set $\mathbf{S}(T) \subseteq [t] \times [w]$ to be $\bigcup_{i=1}^{t} \{(i,1), \ldots, (i, m_T(i))\}$, where $m_T(i)$ is the multiplicity of i in T. The worker-task assignment ϕ produces some bijection $\psi_{\mathbf{S}(T)} : [w] \to \mathbf{S}(T)$. Similarly, ϕ' should produce some bijection $\psi'_T : [w] \to T$. This bijection is defined naturally by projection: if $\psi_{\mathbf{S}(T)}$ assigns worker j to task (i, x), let ψ'_T assign worker j to task i.

We now compute the switching cost of ϕ'. Let T and T' be two adjacent task multisets, so $T' = T \cup \{a\} \setminus \{b\}$ for some $a, b \in [t]$. Then $\mathbf{S}(T') = \mathbf{S}(T) \cup \{(a, m_T(a)+1)\} \setminus \{(b, m_T(b))\}$, and so $\mathbf{S}(T')$ is adjacent to $\mathbf{S}(T)$. Since ϕ has switching cost s, $\psi_{\mathbf{S}(T)}$ and $\psi_{\mathbf{S}(T')}$ agree on $w - s$ workers. By construction, ψ'_T and $\psi'_{T'}$ must agree on these $w - s$ workers as well, and so it too has switching cost at most s. ◀

In the remainder of the section, we will make the assumption that T is a subset of $[n]$, and we will show how to design an assignment function with switching cost $O(\log w \log n)$ on all pairs of adjacent subsets of $[n]$. By Lemma 5, setting $n = wt$ then implies Theorem 1.

Designing an assignment function as an algorithm. It will be helpful to think of the function we construct for assigning workers to tasks as an algorithm $\mathcal{A}$, which we call the **multi-round balls-to-bins algorithm**. The algorithm $\mathcal{A}$ takes as input a set $T \subseteq [n]$ of tasks with $|T| = w$ and must produce a bijection from the workers $[w]$ to T.

The algorithm constructs this bijection in stages. Each stage is what we call a **partial assignment algorithm**, which takes as input the current sets of workers and tasks that have yet to be matched and assigns some subset of these workers to some subset of the tasks. Formally, we define a partial assignment algorithm to be any function ψ which accepts as input any pair of sets $T \subseteq [n], W \subseteq [w]$ with $|T| = |W|$ and produces a matching between some subset of T and some subset of W. After applying ψ to (T, W), there may remain some unmatched elements $T' \subseteq T$, $W' \subseteq W$. We call (T, W) the **worker-task input** to ψ and (T', W') the **worker-task output**. Since a matching must remove exactly as many elements from T as it does from W, we must also have $|W'| = |T'|$. Consequently, there is a natural notion of the **composition** of two partial assignment algorithms: the composition $\psi' \circ \psi$ applies ψ and then ψ', letting the worker-task output of ψ be the worker-task input to ψ'.

The algorithm. We recall the description of the algorithm $\mathcal{A}$. For each i from 1 to $c \log w$, repeat the following hashing procedure $c \log n$ many times. Initialize a hash table consisting of $w/(1.1)^i$ bins and randomly hash each unassigned worker and each unassigned task into this table. For each bin that contains at least one worker and one task, assign the minimum worker in that bin to the minimum task in that bin.

In more detail, our algorithm $\mathcal{A}$ is the composition of $\log_{1.1} w$ partial-assignment algorithms,

$$\mathcal{A} = \mathcal{A}_1 \circ \mathcal{A}_2 \circ \cdots \circ \mathcal{A}_{\log_{1.1} w}.$$

Let c be a large positive constant. Each $\mathcal{A}_i$ is itself the composition of $c \log n$ partial-assignment algorithms,

$$\mathcal{A}_i = \mathcal{A}_{i,1} \circ \mathcal{A}_{i,2} \circ \cdots \circ \mathcal{A}_{i,c \log n}.$$

Designing the parts. Each $\mathcal{A}_{i,j}$ assigns workers to tasks using what we call a $w/(1.1)^i$**-bin hash**, which we define as follows.

For a given parameter k, a k**-bin hash** selects functions $h_1 : [w] \to [k]$ and $h_2 : [n] \to [k]$ independently and uniformly at random. For each worker $\omega \in [w]$, we say that ω is **assigned** to bin $h_1(\omega)$. Similarly, for each $\tau \in [n]$ we say τ is assigned to $h_2(\tau)$. These functions are then used to construct a partial assignment. Given a worker-task input (W, T), we restrict our attention to only the assignments of workers in W and tasks in T. In each bin $\kappa \in [k]$ with at least one worker and one task assigned, match the smallest such worker to the smallest such task. Importantly, once h_1 and h_2 are fixed, the algorithm $\mathcal{A}_{i,j}$ uses this same pair of hash functions for every worker-task input, which (as we will see later) is what allows it to make very similar assignments for similar inputs and achieve low switching cost.

We set each $\mathcal{A}_{i,j}$ to be an independent random instance of the k-bin hash, where $k = w/(1.1)^i$. Formally, this means that the algorithm $\mathcal{A} = \mathcal{A}_{1,1} \circ \cdots \circ \mathcal{A}_{\log_{1.1} w, c \log n}$ is a random variable whose value is a partial-assignment function. Our task is thus to prove that, with non-zero probability, $\mathcal{A}$ fully assigns all workers to tasks and has small switching cost.

Analyzing the algorithm. In Section 3.2, we show that $\mathcal{A}$ *deterministically* has switching cost $O(\log w \log n)$.

Although $\mathcal{A}$ always has small switching cost, the algorithm is *not* always a legal worker-task assignment function. This is because the algorithm may sometimes act as a *partial* worker-task assignment function, leaving some workers and tasks unassigned.

In Section 3.1, we show that with probability greater than 0 (and, in fact, with probability $1 - 1/\operatorname{poly} n$), the algorithm $\mathcal{A}$ succeeds at fully assigning workers to tasks for *all* worker-task inputs (W, T). Theorem 1 follows by the probabilistic method.

3.1 Bounding the probability of failure

Call a partial-assignment algorithm ψ **fully-assigning** if for every worker/task input (W, T), ψ assigns all of the workers in W to tasks in T. That is, ψ never leaves workers unassigned.

▶ **Proposition 6.** *The multi-round balls-to-bins algorithm $\mathcal{A}$ is fully-assigning with high probability in n. That is, for any polynomial $p(n)$, if the constant c used to define $\mathcal{A}$ is sufficiently large, then $\mathcal{A}$ is fully-assigning with probability at least $1 - O(1/p(n))$.*

Proposition 6 tells us that with high probability in n, $\mathcal{A}$ succeeds at assigning all workers on *all* inputs. We remark that this is a much stronger statement than saying that $\mathcal{A}$ succeeds with high probability in n on a *given* input (W, T).

The key to proving Proposition 6 is to show that each $\mathcal{A}_i$ performs what we call $\big(w/(1.1)^i\big)$**-halving**. A partial-assignment function ψ is said to perform k**-halving** if for every worker/-task input (W, T) of size at most $1.1k$, the worker-task output (W', T') for $\psi(W, T)$ has size at most k.

If every $\mathcal{A}_i$ performs $w/(1.1)^i$-halving, then it follows that

$$\mathcal{A}_1 \circ \cdots \circ \mathcal{A}_{\log_{1.1} w}$$

is a fully-assigning algorithm. Thus our task is to show that each $\mathcal{A}_i$ performs $w/(1.1)^i$-halving with high probability in n.

We begin by analyzing the k-bin hash on a given worker/task input (W, T).

▶ **Lemma 7.** *Let ψ a randomly selected k-bin hash. Let (W, T) be a worker/task input satisfying $|W| = |T| \leq 1.1k$, and let (W', T') be the worker/task output of $\psi(W, T)$. The probability that (W', T') has size k or larger is $2^{-\Omega(k)}$.*

Proof. We may assume that $|W| = |T| \geq k$, else the conclusion is trivially true. Let X be the random variable denoting the number of worker/task assignments made by $\psi(W, T)$. Equivalently, X counts the number of bins to which at least one worker is assigned and at least one task is assigned – call these the **active bins**. We will show that $\Pr[X < \frac{k}{8}] \leq 2^{-\Omega(k)}$. Since $|W| = |T| \leq 1.1k$, this immediately implies that $|W'| = |T'| \leq 1.1k - 0.125k \leq k$ with probability $1 - 2^{-\Omega(k)}$, as desired.

We begin by computing $\mathbb{E}[X]$. For each bin $j \in [k]$, the probability no workers are assigned to bin j is $(1 - 1/k)^{|W|} \leq (1 - 1/k)^k \leq 1/e$. Similarly, the probability that no tasks are assigned to bin j is at most $(1 - 1/k)^{|T|} \leq 1/e$. The probability of bin j being active is therefore at least $1 - 2/e \geq 1/4$. By linearity of expectation, $\mathbb{E}[X] \geq k/4$.

Next we show that the random variable X is tightly concentrated around its mean. Because the bins that are active are not independent of one-another, we cannot apply a Chernoff bound. Instead, we employ McDiarmid's inequality:

▶ **Theorem 8** (McDiarmid '89 [25]). *Let $A_1, \ldots, A_m$ be independent random variables over an arbitrary probability space. Let F be a function mapping $(A_1, \ldots, A_m)$ to $\mathbb{R}$, and suppose F satisfies,*

$$\sup_{a_1, a_2, \ldots, a_m, \overline{a_i}} |F(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m) - F(a_1, a_2, \ldots, a_{i-1}, \overline{a_i}, a_{i+1}, \ldots, a_m)| \leq R,$$

for all $1 \leq i \leq m$. That is, if $A_1, A_2, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m$ are fixed, then the value of A_i can affect the value of $F(A_1, \ldots, A_m)$ by at most R. Then for all $S > 0$,

$$\Pr[|F(A_1, \ldots, A_m) - \mathbb{E}[F(A_1, \ldots, A_m)]| \geq R \cdot S] \leq 2e^{-2S^2/m}.$$

The number of active bins X is a function of at most $2.2 \cdot k$ independent random variables (namely, the hashes $h_1(\omega)$ for each $\omega \in W$ and the hashes $h_2(\tau)$ for each $\tau \in T$). Each of these random variables can individually change the number of active bins by at most one. It follows that we can apply McDiarmid's inequality with $R = 1$ and $m = 2.2k$. Taking $S = k/8$, we obtain

$$\Pr[|X - \mathbb{E}[X]| \geq k/8] \leq e^{-\Omega(k)}.$$

Since $\mathbb{E}[X] \geq k/4$, we have that $\Pr[X < k/8] \leq e^{-\Omega(k)}$, which completes the proof of the lemma. ◀

Our next lemma shows that each $\mathcal{A}_i$ is k-halving with high probability in n, where $k = w/(1.1)^i$.

▶ **Lemma 9.** *Let $\psi_1, \ldots, \psi_{c \log n}$ be independent random k-bin hashes, and let $\psi = \psi_1 \circ \cdots \circ \psi_{c \log n}$. With high probability in n, ψ is k-halving. That is, every worker-task input (W, T) with $|W| = |T| \leq 1.1k$ has a worker task output (W', T') with $|W'| = |T'| \leq k$.*

Proof. Fix an arbitrary worker-task input (W, T) with $|W| = |T| \leq 1.1k$. Let (W_i, T_i) denote the worker-task output after applying the first i rounds, $\psi_1 \circ \cdots \circ \psi_i$. Let p_i denote the probability that $|W_i| = |T_i| > k$.

First, we observe that $p_i \leq e^{-\Omega(k)} p_{i-1}$ for all $i > 1$. Indeed, for $|W_i| = |T_i| > k$, we must necessarily have $|W_{i-1}| = |T_{i-1}| > k$, which occurs with probability p_{i-1}, but in this situation, the probability that ψ_i produces a worker-task output of size greater than k is a further $e^{-\Omega(k)}$ by Lemma 7.

The probability that ψ fails to reduce the size of (W, T) to k or smaller is thus at most

$$p_{c \log n} \leq e^{-\Omega(ck \log n)} \leq n^{-\Omega(ck)}, \tag{1}$$

where c is treated as a parameter.

On the other hand, the number of possibilities for input pairs (W, T) satisfying $|W| = |T| \leq 1.1k$ is

$$\sum_{j=0}^{1.1k} \binom{w}{j} \binom{n}{j} \leq 1.1k \cdot w^{1.1k} n^{1.1k} \leq n^{O(k)}. \tag{2}$$

Combining (1) and (2), the probability that there exists *any* pair (W, T) of size $1.1k$ or smaller which fails to have its size reduced to k or smaller is at most $n^{O(k) - c\Omega(k)}$. If c is selected to be a sufficiently large constant, then it follows that ψ performs k-halving with probability at least $1 - n^{-\Omega(k)}$. ◀

We now prove Proposition 6.

Proof of Proposition 6. By Lemma 9, each algorithm $\mathcal{A}_i$ is $\big(w/(1.1)^i\big)$-halving with high probability in n. By a union bound, it follows that all of $\mathcal{A}_i \in \{\mathcal{A}_1, \ldots, \mathcal{A}_{\log_{1.1} w}\}$ are $\big(w/(1.1)^i\big)$-halving with high probability in n. If this occurs, then

$$\mathcal{A} = \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_{\log_{1.1} w}$$

is fully-assigning, as desired. ◀

3.2 Bounding the switching cost

Recall that two worker/task inputs (W_1, T_1) and (W_2, T_2) are said to be **unit distance** if

$$W_1 \setminus W_2| + |W_2 \setminus W_1| + |T_1 \setminus T_2| + |T_2 \setminus T_1| \leq 2.$$

A partial-assignment algorithm ψ is **s-switching-cost bounded** if for all unit-distance pairs of worker/task inputs (W_1, T_1) and (W_2, T_2), the set of assignments made by $\psi(W_1, T_1)$ deterministically differs from the set of assignments made by $\psi(W_2, T_2)$ by at most s.

In this section, we prove the following proposition.

▶ **Proposition 10.** *The multi-round balls-to-bins algorithm is $O(\log w \log n)$-switching-cost bounded.*

We begin by showing that each of the algorithms $\mathcal{A}_{i,j}$ are $O(1)$-switching-cost bounded.

▶ **Lemma 11.** *For any k, the k-bin hash algorithm is $O(1)$-switching-cost bounded.*

Proof. Let ψ denote the k-bin hash algorithm. Consider unit-distance pairs of worker/task inputs (W_1, T_1) and (W_2, T_2). Changing W_1 to W_2 can change the assignments made by ψ for at most a constant number of bins. Similarly changing T_1 to T_2 can change the assignments made by ψ for at most a constant number of bins. Thus $\psi(W_1, T_1)$ differs from $\psi(W_2, T_2)$ by at most $O(1)$ assignments. ◀

Recall that $\mathcal{A}$ is the composition of the $O(\log w \log n)$ partial-assignment algorithms $\mathcal{A}_{i,j}$'s. The fact that each $\mathcal{A}_{i,j}$ is $O(1)$-switching-cost bounded does not directly imply that $\mathcal{A}$ is $O(\log w \log n)$-switching-cost bounded, however, because switching cost does not necessarily interact well with composition. In order to analyze $\mathcal{A}$, we show that each $\mathcal{A}_{i,j}$ satisfies an additional property that we call being composition-friendly.

A partial-assignment algorithm ψ is **composition-friendly**, if for all unit-distance pairs of worker/task inputs (W_1, T_1) and (W_2, T_2), the corresponding worker/task outputs (W_1', T_1') and (W_2', T_2') are also unit-distance.

Lemma 12 shows that each $\mathcal{A}_{i,j}$ is composition-friendly.

▶ **Lemma 12.** *For any k, the k-bin hash is composition-friendly.*

Proof. Although the algorithm ψ is formally only defined on input (W, T) for which $|W| = |T|$, we will abuse notation here and consider ψ even on worker/task input (W, T) satisfying $|W| \neq |T|$.[4] Define the **difference-score** of a pair of worker/task inputs $I_1 = (W_1, T_1), I_2 = (W_2, T_2)$ to be the quantity

$$d(I_1, I_2) = |W_1 \setminus W_2| + |W_2 \setminus W_1| + |T_1 \setminus T_2| + |T_2 \setminus T_1|.$$

[4] Indeed, the definition of the k-bin hash does not require a worker-task input with $|W| = |T|$. The only reason we require this equality in general is to simplify calculations, as in practice the algorithm will only be run on worker-task inputs of equal size.

We will show the stronger statement that the difference-score $d(O_1, O_2)$ of the corresponding worker/task outputs $O_1 = (W_1', T_1'), O_2 = (W_2', T_2')$ satisfies

$$d(O_1, O_2) \leq d(I_1, I_2). \tag{3}$$

It suffices to consider only two special cases: the case in which $W_2 = W_1 \cup \{\omega\}$ for some worker ω and $T_2 = T_1$; and the case in which $T_2 = T_1 \cup \{\tau\}$ for some task τ and $W_2 = W_1$. Iteratively applying these two cases to transform I_1 into I_2 implies inequality 3.

For this purpose, the roles of W and T are identical, so suppose without loss of generality that $W_2 = W_1 \cup \{\omega\}$ for some worker ω and $T_2 = T_1$. Recall that the assignment of workers and tasks to buckets is determined by some hash functions h_1, h_2 and in particular is the same whether we input W_1 or W_2. We first assign (only) the elements of W_1 and T_1 to their respective buckets, and then look at how including the assignment of ω changes the worker-task output. If h_1 assigns ω to either a bin with no tasks or a bin which already has some lexicographically smaller worker, then we will have $W_2' = W_1' \cup \{w\}$ and $T_2' = T_1'$. If h_1 assigns worker ω to a bin with no other workers and at least one task, we let the smallest such task be τ and see $W_2' = W_1'$ and $T_2' = T_1' \setminus \{\tau\}$. Finally, if h_1 assigns ω to a bin with only larger workers and at least one task, we let the minimal such worker be γ, and we see $W_2' = W_1' \cup \{\gamma\}$ and $T_2' = T_1'$. In all three cases, $d(O_1, O_2) = 1$, as desired. ◀

Next, we will show that composing composition-friendly algorithms has the effect of summing switching costs.

▶ **Lemma 13.** *Suppose that partial-assignment algorithms $\psi_1, \psi_2, \ldots, \psi_k$ are all composition-friendly, and that each ψ_i is s_i-switching-cost bounded. Then $\psi_1 \circ \psi_2 \circ \cdots \circ \psi_k$ is composition-friendly and is $(\sum_i s_i)$-switching-cost-bounded.*

Proof. By induction, it suffices to prove the lemma for $k = 2$. Let $I_1 = (W_1, T_1)$ and $I_2 = (W_2, T_2)$ be unit-distance worker/task inputs.

For $i \in \{1, 2\}$, let $I_i' = (W_i', T_i')$ be the worker/task output for $\psi_1(W_i, T_i)$, and let $I_i'' = (W_i'', T_i'')$ be the worker/task output for $\psi_2(W_i', T_i')$.

Since ψ_1 is composition friendly, its outputs I_1' and I_2' are unit distance. Since I_1' and I_2' are unit distance, and since ψ_2 is composition friendly, the outputs I_1'' and I_2'' of ψ_2 are also unit distance. Thus $\psi_1 \circ \psi_2$ is composition friendly.

Since the inputs I_1 and I_2 to ψ_1 are unit-distance, $\psi_1(I_1)$ and $\psi_1(I_2)$ differ in at most s_1 worker-task assignments. Since the inputs I_1' and I_2' to ψ_2 are also unit distance, $\psi_2(I_1')$ and $\psi_2(I_2')$ differ in at most s_2 worker-task assignments. Thus the composition $\psi_1 \circ \psi_2$ is $(s_1 + s_2)$-switching-cost bounded, as desired. ◀

We can now prove Proposition 10.

Proof of Proposition 10. By Lemma 11, each $\mathcal{A}_{i,j}$ is $O(1)$-switching-cost bounded. By Lemma 12, each $\mathcal{A}_{i,j}$ is composition friendly. Since $\mathcal{A}$ is the composition of the $O(\log w \log n)$ different $\mathcal{A}_{i,j}$'s, it follows by Lemma 13 that $\mathcal{A}$ is $O(\log w \log n)$-switching-cost bounded. ◀

4 Lower bounds on switching cost

Define $s_{w,t}$ to be the optimal switching cost for assignment functions that assign workers $1, 2, \ldots, w$ to multisets of w tasks from the universe $[t]$. The upper bounds in this paper establish that $s_{w,t} \leq O(\log w \log(wt))$. It is natural to wonder whether smaller bounds can be achieved, and in particular, whether a small switching cost that depends only on w can be achieved.

It trivially holds that $s_{w,t} \leq w$. We show that when t is sufficiently large relative to w, there is a matching lower bound of $s_{w,t} \geq w$.

▶ **Theorem 3.** *For every w and $t \geq \mathrm{tow}(\Omega(w))$, every worker-task assignment function has switching cost w.*

Proof. Given any worker-task assignment function ϕ, we can actually find high switching cost between a pair of task subsets, in which all demands are 0 or 1. For each $T \subseteq [t]$ of w tasks, ϕ produces a bijection of workers $[w]$ to tasks T. In order to lower-bound the switching cost, we produce a coloring of the complete w-uniform hypergraph with t vertices. The coloring will be designed so that, if it contains a monochromatic clique on $w+1$ vertices, then the assignment function ϕ must have worst-possible switching cost w. By applying the hypergraph Ramsey theorem, we deduce that, if t is large enough, then the coloring must contain a monochromatic $(w+1)$-clique, completing the lower bound.

Coloring the complete w-uniform hypergraph on t vertices. Let $K_t^{(w)}$ denote the complete w-uniform hypergraph on t vertices. Note that the hyperedges of $K_t^{(w)}$ are just the w-element subsets of $[t]$, which correspond to sets of tasks.

For a task set $T = \{\tau_1, \ldots, \tau_w\}$, where $1 \leq \tau_1 < \cdots < \tau_w \leq t$, we color the hyperedge T with the tuple $\pi = \langle \pi(1), \pi(2), \ldots, \pi(w) \rangle$, where $\tau_{\pi(i)}$ is the task assigned to worker i. One can think of π as a permutation of numbers $\{1, 2, \ldots, w\}$, and thus the coloring consists of at most $w!$ colors.

Monochromatic $(w+1)$-cliques imply high switching cost. The key property of the coloring C is that, if $K_t^{(w)}$ contains a monochromatic $(w+1)$-vertex clique (i.e., $K_{w+1}^{(w)}$), then ϕ must have switching cost w.

Namely, if $K_t^{(w)}$ contains a monochromatic $(w+1)$-clique, then we can find $w+1$ vertices, $\tau_1 < \tau_2 < \cdots < \tau_{w+1}$, such that every w-element subset T of these tasks is assigned the same permutation π as its color. In particular, this means that for the task-set $T_1 = \{\tau_1, \ldots, \tau_w\}$ each worker i is assigned to task $\tau_{\pi(i)}$, but for the task-set $T_2 = \{\tau_2, \ldots, \tau_{w+1}\}$ that same worker i is assigned to a different task $\tau_{\pi(i)+1}$. Thus there is a pair of adjacent task sets T_1, T_2 that exhibit switching cost w.

Finding a monochromatic clique. In order to complete the lower bound, we wish to show that, if t is sufficiently large, then the coloring contains a monochromatic $K_{w+1}^{(w)}$. To do this, we employ the hypergraph Ramsey theorem.

▶ **Theorem 14** (Theorem 1 in [12])**.** *Let $k \geq 2$ and $N \geq n \geq 2$ be positive integers. The hypergraph Ramsey number $R(k, n, N)$ is defined to be the least positive integer M such that for every k-coloring of the hyperedges of $K_M^{(n)}$, the complete n-uniform hypergraph on M vertices, contains a monochromatic copy of $K_M^{(n)}$. This quantity satisfies*

$$R(k, n, N) \leq k^{(k^{n-1})^{(k^{n-2})^{\cdots^{(k^2)^{k(N-n)+1}}}}}.$$

Applying Theorem 14, we see that if $t \geq R(w!, w, w+1)$, then the $(w!)$-coloring of $K_t^{(w)}$ contains a monochromatic $(w+1)$-clique, and the assignment function ϕ must have switching cost w, as desired. By Theorem 14, $R(w!, w, w+1) \leq \mathrm{tow}(O(w))$. which implies that that every worker-task assignment function has switching cost w when $t \geq \mathrm{tow}(\Omega(w))$. This completes the proof of Theorem 3. ◀

Another way of viewing this argument is that a worker-task assignment function with switching cost less than w gives rise to a proper $(w!)$-coloring of a certain graph, with vertex set $\binom{[t]}{w}$ and edges of the form $(\{\tau_1, \ldots, \tau_w\}, \{\tau_2, \ldots, \tau_{w+1}\})$ for $\tau_1 < \tau_2 < \cdots < \tau_{w+1}$. Such graphs are studied under the name of **shift-graphs**, see, e.g., [14, Section 3.4], where the definition and proofs of basic properties are attributed to [11]. In particular, the chromatic number of shift-graphs is known to be $(1 + o(1)) \cdot \log^{(w-1)} t$ (with the superscript denoting iteration). This gives an alternative way to complete the proof of Theorem 3 and it gives the same asymptotic bound on t in terms of w. While the chromatic number lower bound suffices to prove the switching cost bound, the nearly matching upper bound (on chromatic number) suggests that an entirely different technique would be needed in order to asymptotically improve the switching cost bound.

5 Densification into Hamming space

In this section, we apply our results on worker-task assignment to the problem of densification. In particular, we show how to embed sparse high-dimensional binary vectors into dense low-dimensional Hamming space.

Let $\mathcal{H}_n^k$ denote the set of n-dimensional binary vectors with k ones. Let $\mathcal{H}_k(n)$ denote the set of k-dimensional vectors with entries from $[n]$. We show that $\mathcal{H}_n^k$ can be embedded into $\mathcal{H}_k(n)$ with distortion $O(\log n \log k)$.

▶ **Theorem 4.** *There exists a map $\phi : \mathcal{H}_n^k \to \mathcal{H}_k(n)$ such that, for every $\vec{x}, \vec{y} \in \mathcal{H}_n^k$,*

$$\mathrm{Ham}(\vec{x}, \vec{y})/2 \leq \mathrm{Ham}(\phi(\vec{x}), \phi(\vec{y})) \leq O(\log n \log k)\, \mathrm{Ham}(\vec{x}, \vec{y}).$$

Proof. Using Theorem 1, let ψ be a worker-task assignment function mapping workers $1, 2, \ldots, k$ to a task set $T \subseteq [n]$ with switching-cost $O(\log n \log k)$.

For $\vec{x} \in \mathcal{H}_n^k$, define $T(\vec{x}) = \{i \mid \vec{x}_i = 1\}$ to be the task set consisting of the positions in $\vec{x}$ that are 1. Define $\phi(\vec{x})$ to be the k-dimensional vector whose i-th coordinate denotes the task $t \in T(\vec{x})$ to which $\psi(T(\vec{x}))$ assigns worker i. For example, if $k = 3$, $\vec{x} = \langle 0, 1, 0, 1, 1, 0\rangle$, and $\psi(T(\vec{x}))$ assigns workers $1, 2, 3$ to tasks $4, 2, 5$, respectively, then $\phi(\vec{x}) = \langle 4, 2, 5\rangle$.

Since the coordinates of $\phi(\vec{x})$ are a permutation of the positions $T(\vec{x})$ in which $\vec{x}$ is non-zero, it is necessarily the case that

$$\mathrm{Ham}(\phi(\vec{x}), \phi(\vec{y})) \geq |T(\vec{x}) \setminus T(\vec{y})| \geq \mathrm{Ham}(\vec{x}, \vec{y})/2.$$

On the other hand, since ψ has switching cost $O(\log n \log k)$, it is also the case that $\psi(\vec{x})$ and $\psi(\vec{y})$ differ by at most $O(\log n \log k)\, \mathrm{Ham}(\vec{x}, \vec{y})$ assignments, meaning that,

$$\mathrm{Ham}(\phi(\vec{x}), \phi(\vec{y})) \leq O(\log n \log k)\, \mathrm{Ham}(\vec{x}, \vec{y}).$$

This completes the proof of the theorem. ◀

We remark that Theorem 4 can be generalized to allow for the the domain space $\mathcal{H}_n^k$ to have non-binary entries. In particular, if $\mathcal{L}_n^k$ is the set of vectors with non-negative integer entries that sum to k, then there is an embedding $\phi : \mathcal{L}_n^k \to \mathcal{H}_k(n)$ such that, for $x, y \in \mathcal{L}_n^k$,

$$\ell_1(\vec{x}, \vec{y})/2 \leq \mathrm{Ham}(\phi(\vec{x}), \phi(\vec{y})) \leq O(\log n \log k)\ell_1(\vec{x}, \vec{y}).$$

This follows from the same argument as Theorem 4, except that now $T(x)$ is the *multiset* for which each element $i \in [n]$ has multiplicity $\vec{x}_i$, and now ψ is the worker-task assignment mapping workers $1, 2, \ldots, k$ to a task *multiset* $T \subseteq [n]$.

6 Open problems

We leave open the question of closing the gap between upper and lower bounds for the worker-task assignment problem: the upper bound is polylog(wt) and the lower bound is $\log^*(t)$.

One interesting parameter regime is when w and t are comparable in size (say within a polynomial factor of each other). In this regime, no super-constant lower bound is known.

Another interesting direction is the problem of densification into Hamming space. Our upper bound for the worker-task assignment problem implies an upper bound for this problem, but our lower bound does not carry over. We leave open the problem of whether there is a better upper bound or a super-constant lower bound for this problem.

References

1 Alexandr Andoni, Moses S Charikar, Ofer Neiman, and Huy L Nguyen. Near linear lower bound for dimension reduction in ℓ_1. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 315–323. IEEE, 2011.

2 Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 267–276. IEEE, 2011.

3 Radu Berinde, Anna C Gilbert, Piotr Indyk, Howard Karloff, and Martin J Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 798–805. IEEE, 2008.

4 Samuel N Beshers and Jennifer H Fewell. Models of division of labor in social insects. *Annual review of entomology*, 46(1):413–440, 2001.

5 Jean Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

6 Bo Brinkman and Moses Charikar. On the impossibility of dimension reduction in \ell _1. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 514, 2003.

7 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th annual ACM Symposium on Theory of Computing*, pages 712–725, 2016.

8 Moses Charikar, Ofir Geri, Michael P Kim, and William Kuszmaul. On estimating edit distance: Alignment, dimension reduction, and embeddings. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107, page 34, 2018.

9 Moses Charikar and Robert Krauthgamer. Embedding the Ulam metric into ℓ_1. *Theory of Computing*, 2(1):207–224, 2006.

10 Moses Charikar and Amit Sahai. Dimension reduction in the ℓ_1 norm. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 551–560. IEEE, 2002.

11 P Erdős and A Hajnal. On chromatic number of infinite graphs. In *Theory of Graphs (Proc. Colloq., Tihany, 1966)*, pages 83–98. Academic Press, 1968.

12 Paul Erdős and Richard Rado. Combinatorial theorems on classifications of subsets of a given set. *Proceedings of the London mathematical Society*, 3(1):417–439, 1952.

13 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM Symposium on Theory of Computing*, pages 448–455, 2003.

14 Stefan Felsner. *Interval orders: combinatorial structure and algorithms*. PhD thesis, Technische Universität Berlin, 1992. URL: `http://page.math.tu-berlin.de/~felsner/Paper/diss.pdf`.

15 Chryssis Georgiou and Alexander A Shvartsman. Cooperative task-oriented computing: Algorithms and complexity. *Synthesis Lectures on Distributed Computing Theory*, 2(2):1–167, 2011.

16 A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin. Algorithmic linear dimension reduction in the ℓ_1 norm for sparse vectors. In *Allerton 2006 (44th Annual Allerton Conference on Communication, Control, and Computing*, 2006.

17 Anna Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

18 Piotr Indyk. Explicit constructions for compressed sensing of sparse signals. In *Proceedings of the nineteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 30–33, 2008.

19 William B Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

20 Michael JB Krieger, Jean-Bernard Billeter, and Laurent Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406(6799):992, 2000.

21 Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J Matarić. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research*, 25(3):225–241, 2006.

22 Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

23 Kathryn Sarah Macarthur, Ruben Stranders, Sarvapali D Ramchurn, and Nicholas R Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *AAAI*, pages 701–706, 2011.

24 Jiří Matoušek. On variants of the Johnson–Lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

25 Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.

26 James McLurkin and Daniel Yamins. Dynamic task assignment in robot swarms. In *Robotics: Science and Systems*, volume 8. Citeseer, 2005.

27 James Dwight McLurkin. *Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots*. PhD thesis, Massachusetts Institute of Technology, 2004.

28 Raghu Meka, Omer Reingold, and Yuan Zhou. Deterministic Coupon Collection and Better Strong Dispersers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 872–884, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.872.

29 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *Journal of the ACM (JACM)*, 54(5):23–es, 2007.

30 Tsvetomira Radeva, Anna Dornhaus, Nancy Lynch, Radhika Nagpal, and Hsin-Hao Su. Costs of task allocation with local feedback: Effects of colony size and extra workers in social insects and other multi-agent systems. *PLoS computational biology*, 13(12):e1005904, 2017.

31 Gene E Robinson. Regulation of division of labor in insect societies. *Annual review of entomology*, 37(1):637–665, 1992.

32 Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

33 Hsin-Hao Su, Lili Su, Anna Dornhaus, and Nancy Lynch. Ant-inspired dynamic task allocation via gossiping. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 157–171. Springer, 2017.

34 Hsin-Hao Su and Nicole Wein. Lower Bounds for Dynamic Distributed Task Allocation. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168, pages 99:1–99:14, 2020. doi:10.4230/LIPIcs.ICALP.2020.99.

Fully-Dynamic Graph Sparsifiers Against an Adaptive Adversary

Aaron Bernstein
Rutgers University, Piscataway, NJ, USA

Jan van den Brand
Simons Institute, Berkeley, CA, USA
University of California Berkeley, CA, USA

Maximilian Probst Gutenberg
ETH Zürich, Switzerland

Danupon Nanongkai
University of Copenhagen, Denmark
KTH Royal Institute of Technology, Stockholm, Sweden

Thatchaphol Saranurak
University of Michigan, Ann Arbor, MI, USA

Aaron Sidford
Stanford University, CA, USA

He Sun
University of Edinburgh, UK

Abstract

Designing efficient dynamic graph algorithms against an *adaptive* adversary is a major goal in the field of dynamic graph algorithms and has witnessed many exciting recent developments in, e.g., dynamic matching (Wajc STOC'20) and decremental shortest paths (Chuzhoy and Khanna STOC'19). Compared to other graph primitives (e.g. spanning trees and matchings), designing such algorithms for *graph spanners* and (more broadly) *graph sparsifiers* poses a unique challenge since there is no fast deterministic algorithm known for static computation and the lack of a way to adjust the output slowly (known as "small recourse/replacements").

This paper presents the first non-trivial efficient adaptive algorithms for maintaining many sparsifiers against an adaptive adversary. Specifically, we present algorithms that maintain

1. a polylog(n)-spanner of size $\tilde{O}(n)$ in polylog(n) amortized update time,
2. an $O(k)$-approximate cut sparsifier of size $\tilde{O}(n)$ in $\tilde{O}(n^{1/k})$ amortized update time, and
3. a polylog(n)-approximate spectral sparsifier in polylog(n) amortized update time.

Our bounds are the first non-trivial ones even when only the recourse is concerned. Our results hold even against a stronger adversary, who can access the random bits previously used by the algorithms and the amortized update time of all algorithms can be made worst-case by paying sub-polynomial factors. Our spanner result resolves an open question by Ahmed et al. (2019) and our results and techniques imply additional improvements over existing results, including (i) answering open questions about decremental single-source shortest paths by Chuzhoy and Khanna (STOC'19) and Gutenberg and Wulff-Nilsen (SODA'20), implying a nearly-quadratic time algorithm for approximating minimum-cost unit-capacity flow and (ii) de-amortizing a result of Abraham et al. (FOCS'16) for dynamic spectral sparsifiers.

Our results are based on two novel techniques. The first technique is a generic black-box reduction that allows us to assume that the graph is initially an expander with almost uniform-degree and, more importantly, stays as an almost uniform-degree expander while undergoing only edge deletions. The second technique is called *proactive resampling*: here we constantly *re-sample* parts of the input graph so that, independent of an adversary's computational power, a desired structure of the underlying graph can be always maintained. Despite its simplicity, the analysis of this sampling scheme is far from trivial, because the adversary can potentially create dependencies between the random choices used by the algorithm. We believe these two techniques could be useful for developing other adaptive algorithms.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 20; pp. 20:1–20:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases dynamic graph algorithm, adaptive adversary, spanner, sparsifier

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.20

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2004.08432 [24]

Funding *Aaron Bernstein*: Funded by NSF Career grant 1942010.
Jan van den Brand: Funded by ONR BRC grant N00014-18-1-2562 and by the Simons Institute for the Theory of Computing through a Simons-Berkeley Postdoctoral Fellowship. Research partially done at KTH.
Maximilian Probst Gutenberg: Research partially done while at University of Copenhagen.
Danupon Nanongkai: This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 715672. Also partially supported by the Swedish Research Council (Reg. No. 2015-04659 and 2019-05622).
Thatchaphol Saranurak: Research partially done at KTH and supported by the Swedish Research Council (Reg. No. 2015-04659).
Aaron Sidford: Supported in part by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship.
He Sun: Funded by an EPSRC Early Career Fellowship (EP/T00729X/1).

Acknowledgements We thank Julia Chuzhoy and Gramoz Goranci for discussions.

1 Introduction

Dynamic graph algorithms maintain information in an input graph undergoing edge *updates*, which typically take the form of edge insertions and deletions. Many efficient algorithms have been developed in this setting, such as those for maintaining a minimum spanning tree, maximum matching, shortest distances, and sparsifiers. However, many of these algorithms are randomized and, more importantly, make the so-called *oblivious adversary assumption*, which assumes that each update given to the algorithm cannot depend on answers of the algorithm to earlier queries. In other words, the entire update sequence is fixed by some adversary in advance, and then each update is given to the algorithm one by one. This assumption is crucial for many recent advances in the design of efficient randomized algorithms for dynamic problems (e.g. [68, 63, 35, 22, 10, 1, 82, 9, 25]).

The oblivious-adversary assumption significantly limits the use of dynamic algorithms in certain interactive environments and, in particular, the setting where these dynamic algorithms are employed as *subroutines* for other algorithms. For example, the recent partially-dynamic single-source shortest paths algorithm without this assumption [21] has been used to obtain an almost-linear time approximate min-cost flow and balanced separator algorithm in the static setting (see also, e.g., [17, 16, 61, 62, 42, 43], for prior attempts in this direction). In addition, [40] pointed out that their goal of computing a (static) short cycle decomposition could have been achieved easily using existing dynamic spanner algorithms, if such algorithms worked without the oblivious-adversary assumption. Because of this, designing dynamic algorithms *without* the oblivious adversary assumption has become a major goal in the field of dynamic graph algorithms in recent years. We call such algorithms *adaptive* and say that they work against an *adaptive adversary*.

Thanks to the recent efforts in developing adaptive algorithms, such algorithms now exist for maintaining a number of graph primitives, such as minimum spanning trees with bounded worst-case update time [41, 77, 76, 88], partially-dynamic single-source shortest

paths [50, 42, 17, 18, 16, 62, 61, 60], and fully-dynamic matching [26, 27, 28, 29, 87]. This line of research on dynamic graph algorithms has also brought new insights on algorithm design in the static setting (e.g. flow, vertex connectivity, matching, and traveling salesman problem [73, 42, 86, 37, 38]). One very recent exciting application is the use of an adaptive dynamic algorithm called *expander decomposition* to compute maximum-weight matching and related problems in nearly-linear time on moderately dense graphs [86].

Graph sparsifiers

Despite the fast recent progress, very little was known for certain important primitives, like maintaining *graph sparsifiers* against an adaptive adversary. To formalize our discussion, we say that a sparsifier of a graph $G = (V, E)$ is a sparse graph $H = (V, E')$ that approximately preserves properties of G, such as all cuts (*cut sparsifiers*), all-pairs distances (*spanners*), and spectral properties (*spectral sparsifiers*). For any integer $\alpha \geq 1$, an *α-spanner* of graph $G = (V, E)$ is a subgraph H such that for any pair of nodes (u, v), the distance between u and v in H is at most α times their distance in G. An *α-cut sparsifier* of G is a sparse graph H that preserves all cut sizes up to an α factor: that is, $\delta_H(S) \in [\delta_G(S), \alpha\delta_G(S)]$ for every $S \subseteq V$, where $\delta_G(S)$ (respectively $\delta_H(S)$) is the total weight of edges between S and $V \setminus S$ in G (respectively H) for any $S \subset V$. An α-spectral sparsifier is a sparse graph that provides an even stronger guarantee than an α-cut sparsifier (see full version for the definition).

A dynamic algorithm for maintaining a spanner or a cut sparsifier is given a weighted undirected n-node graph G to preprocess, and returns a spanner or cut-sparsifier H of G. After this, it must process a sequence of *updates*, each of which is an edge insertion or deletion of G. After each update, the algorithm outputs edges to be inserted and deleted to H so that the updated H remains an α-spanner or cut-sparsifier of the updated G. The algorithm's performance is measured by the *preprocessing time* (the time to preprocess G initially); the *update time* (the time to process each update); the *stretch* (the value of α); and the *size* of the spanner (the number of edges). The update time is typically categorized into two types: *amortized case* update time and *worst case* update time. The more desirable one is the *worst-case* update time which holds for every single update. This is in contrast to an *amortized* update time which holds "on average"; i.e., for any t, an algorithm is said to have an amortized update time of t if, for any k, the total time it spends to process the first k updates is at most kt.

Spanners and cut sparsifers are fundamental objects that have been studied extensively in various settings (e.g., [5, 79, 11, 78, 44, 58, 57, 15, 53, 3, 4]). In the dynamic setting, they have been actively studied since 2005 (e.g. [7, 51, 8, 48, 23, 10, 31, 20]). A fairly tight algorithm with amortized update time for maintaining dynamic spanners was known in 2008 due to Baswana et al. [10]. For any $k \geq 1$, their algorithm maintains, with high probability, a $(2k-1)$-spanner of size $\tilde{O}(kn^{1+1/k})$ in $O(k^2 \log^2 n)$ amortized update time[1]. The stretch and size tradeoff is almost tight assuming Erdős' girth conjecture, which implies that a $(2k-1)$-spanner must contain $\Omega(n^{1+1/k})$ edges. Recently, Bernstein et al. [20] showed how to "de-amortize" the result of Baswana et al. [10], giving an algorithm that in $O(1)^k \log^3(n)$ *worst-case* update time maintains, w.h.p., a $(2k-1)$-spanner of size $\tilde{O}(n^{1+1/k})$.

We refer the reader to the full version for other related results and clear comparison. For dynamic cut sparsifiers, the only result we are aware of is [1], which maintains a $(1+\epsilon)$-cut sparsifier in polylogarithmic worst-case update time. [1] can also maintain a $(1+\epsilon)$-spectral sparsifier within the same update time, but this holds only for the amortized update time.

[1] Throughout, $\tilde{O}$ hides $O(\text{polylog}(n))$ factors. With high probability (w.h.p.) means with probability at least $1 - 1/n^c$ for any constant $c > 1$.

Similar to other dynamic algorithms, most existing dynamic spanner and cut sparsifier algorithms are *not* adaptive. The exceptions are the algorithms of [7], which can maintain a 3-spanner (respectively a 5-spanner) of size $O(n^{1+1/2})$ (respectively $O(n^{1+1/3})$) in $O(\Delta)$ time, where Δ is the maximum degree. These algorithms are deterministic, and thus work against an adaptive adversary. Since Δ can be as large as $\Omega(n)$, their update time is rather inefficient as typically polylog(n) or $n^{o(1)}$ update times are desired. Designing dynamic algorithms for this low update time is one of the major objectives of our paper.

Challenges

Developing efficient adaptive algorithms for maintaining graph sparsifiers poses great challenges in the general research program towards adaptive dynamic algorithms. First, computing many sparsifiers inherently relies on the use of randomness. Even in the static setting, existing fast algorithms for constructing cut and spectral sparsifiers are all randomized, and known deterministic algorithms require $\Omega(n^4)$ time [12, 89]. In fact, a nearly-linear time deterministic algorithm for a certain cut sparsifier would resolve a major open problem about computing the minimum cut deterministically [70, 56, 75]. Thus, in contrast to other primitives, such as the minimum spanning tree or approximate maximum matching, for which efficient deterministic algorithms exist in the static setting, there is little chance to dynamically maintain sparsifiers *deterministically*. (Deterministic dynamic algorithms always work against adaptive adversaries.)

Secondly, even if we allowed infinite update time and focused on the strictly simpler objective of minimizing the changes in the maintained sparsifier (the so-called *recourse* or *replacements* in online algorithms), it is unclear from existing techniques whether it is possible to maintain such a sparsifier against an adaptive adversary while only making (amortized) polylog(n) changes to the sparsifier per update to the input graph. For example, an $O(\log n)$-spanner of $\tilde{O}(n)$ edges can be easily maintained with $\tilde{O}(n)$ recourse per update by replacing the entire spanner by a new one after every update. Is it possible that an adversary who can see the output spanner can make a few changes to the graph so that a new $O(\log n)$-spanner has to change completely? An answer to this question is unclear. This is in contrast to many dynamic graph primitives where bounding the changes is obvious even against adaptive adversaries. For example, it can be easily shown that maintaining the minimum spanning tree requires at most one edge insertion and one edge deletion after each update to the input graph.

Designing algorithms with low recourse is a prerequisite for fast dynamic algorithms, and there are several graph problems where low-recourse algorithms were the crucial bottleneck, e.g. maximal independent set [33, 6, 36, 13, 74], planar embeddings [65, 66], and topological sorting [19].The lack of recourse-efficient algorithms makes it very challenging to maintain sparsifiers against an adaptive adversary.

1.1 Our Results

We show how to dynamically maintain both spanners, cut sparsifiers, and spectral sparsifiers against an adaptive adversary in *poly-logarithmic* update time and recourse. We summarize these results as follows:

▶ **Theorem 1** (Adaptive Spanner). *There is a randomized* adaptive *algorithm that, given an n-vertex graph undergoing edge insertions and deletions, with high probability, explicitly maintains a* polylog(n)*-spanner of size $\tilde{O}(n)$ using* polylog(n) *amortized update time.*

▶ **Theorem 2** (Adaptive Cut Sparsifier). *There is a randomized* adaptive *algorithm that, given an n-vertex graph undergoing edge insertions and deletions and a parameter $k \geq 1$, with high probability, maintains an $O(k)$-cut sparsifier of size $\tilde{O}(n)$ using $\tilde{O}(n^{1/k})$ amortized update time, which is* $\text{polylog}(n)$ *time when $k = \log n$.*

▶ **Theorem 3** (Adaptive Spectral Sparsifier). *There is a randomized* adaptive *algorithm that, given an n-vertex graph undergoing edge insertions and deletions, with high probability, maintains a* $\text{polylog}(n)$*-spectral sparsifier of size $\tilde{O}(n)$ using* $\text{polylog}(n)$ *amortized update time.*

All results above hold even against a stronger adversary, called *randomness-adaptive* in [76]. This adversary can access the random bits previously used by our algorithms (but not the future random bits). Theorem 1 is the first algorithm with $o(n)$ update time against an adaptive adversary, and answers the open problem in [2]. The only previous adaptive algorithm is by [7] which can take $O(n)$ update time. No non-trivial dynamic adaptive algorithm for cut sparsifiers and spectral sparsifiers is known before Theorems 2 and 3.

Compared to results assuming the oblivious-adversary assumption (e.g. [10, 20, 1]), our bounds are not as tight. For example, Theorem 1 does not achieve the standard $(2k-1)$-spanner with $O(n^{1+1/k})$ edges. One reason for this limitation is that it is not clear if such trade-off is possible even when we focus on the *recourse*, as discussed above. Maintaining spanners or other sparsifiers against adaptive adversaries with tight trade-offs and polylogarithmic recourse is a challenging barrier that is beyond the scope of this paper. Additionally, the sparse-spanner regime studied in this paper is generally the most useful for applications to other problems (see discussion in Section 1.2); getting a sharper trade-off would not lead to significant improvements for most of these applications.

All the above results can be *deamortized*.[2] For example, a $2^{O(\sqrt{\log n}\log\log(n))}$-spanner of size $\tilde{O}(n)$ can be maintained in $2^{O(\sqrt{\log n}\log\log(n))}$ *worst-case* update time. Also, for any k, a $2^{O(k\,\text{polylog}(k))}$-cut sparsifier of size $\tilde{O}(n)$ can be maintained in $\tilde{O}(n^{1/k})$ worst-case time. In particular, we can maintain an $O(\log^* n)$-cut sparsifier and an $O(1)$-cut sparsifier in $n^{o(1)}$ and n^{ϵ} time for any constant ϵ, respectively.

Our deamortization technique also implies, as a side result, the first non-trivial algorithm with *worst-case* update time against an oblivious adversary for maintaining spectral sparsifiers.

▶ **Theorem 4** (Oblivious Spectral Sparsifier). *There is a randomized algorithm against an oblivious adversary that, given an n-vertex graph undergoing edge insertions and deletions and $\epsilon \geq 1/\text{polylog}(n)$, with high probability, maintains a $(1+\epsilon)$-spectral sparsifier of size $n \cdot 2^{O(\sqrt{\log n})}$ using $2^{O(\log^{3/4} n)}$ worst-case update time.*

The previous algorithm by Abraham et al. [1] maintains a $(1+\epsilon)$-spectral sparsifier of size $\tilde{O}(n)$ using $\text{polylog}(n)$ amortize update time. Further [1] asked if the update time can be made worst-case. Theorem 4 answers this open question modulo $n^{o(1)}$ factors.

1.2 Applications

Our results imply several interesting applications. Our first set of applications are for the decremental $(1+\epsilon)$-approximate single-source shortest paths (SSSP) problem. There has been a line of work [17, 16, 18, 62] on fast adaptive algorithms for solving this problem.

[2] To do this, we use, e.g., the sparsification technique [49] and a sophisticated dynamic expander decomposition; see Section 2.3 for an overview.

Although all these algorithms are adaptive, they share a drawback that they cannot return the shortest path itself; they can only maintain distance estimates. Very recently, Chuzhoy and Khanna [42] showed a partial fix to this issue for some algorithms [17, 16] and consequently obtained impressive applications to static flow algorithms. Unfortunately, this fix only applies in a more restricted setting, and moreover it is not clear how the technique from [42] can be used to fix the same issue in other algorithms (e.g. [18, 62]).

We show that our main result from Theorem 1 can be employed in a consistent and simple way, such that the path-reporting issue in all previous algorithms in [17, 16, 18, 62] can be fixed. This resolves an open question posed in multiple papers [16, 62, 42]. We summarize these applications below:

▶ **Corollary 5** (Fixing the path-reporting issue of [18, 62]). *For any decremental unweighted graph $G = (V, E)$, fixed source s, and constant $\epsilon > 0$, there is an adaptive algorithm $\mathcal{B}$ that maintains the $(1+\epsilon)$-approximate distances from vertex s to every vertex $v \in V$* and supports corresponding shortest path queries. *The algorithm $\mathcal{B}$ has expected total update time $mn^{0.5+o(1)}$, distance estimate query time $O(\log\log n)$ and shortest path query time $\tilde{O}(n)$.*

Corollary 5 gives the first adaptive algorithm without the path-reporting issue that can take $o(n^2)$ total update time. The next algorithm works on weighted graphs and is near-optimal on dense graphs:

▶ **Corollary 6** (Fixing the path-reporting issue of [17, 16]). *For any decremental weighted graph $G = (V, E, w)$ with W being the ratio between maximum and minimum edge weight, fixed source s, and $\epsilon > 0$, there is an adaptive algorithm $\mathcal{A}$ that maintains the $(1+\epsilon)$-approximate distances from vertex s to every vertex $v \in V$* and supports corresponding shortest path queries. *The algorithm $\mathcal{A}$ has expected total update time $\tilde{O}(n^2 \log W)$, distance estimate query time $O(\log\log(nW))$ and shortest path query time $\tilde{O}(n \log W)$.*

Corollary 6 can be compared to two previous results [42, 43]. In [42], their algorithm requires slower $n^{2+o(1)} \log W$ total update time, and needs to assume that the input graph undergoes only *vertex* deletions which is more restrictive. So Corollary 6 strictly improves the algorithm by [42]. In [43], they use very different techniques than ours and show an algorithm with $n^{2+o(1)} \log W$ total update time, distance query time $O(\log\log(nW))$, shortest path query time $O(|P| n^{o(1)})$ when a path P is returned, and is deterministic. This algorithm is incomparable to Corollary 6. Our result has slightly faster total update time, but their algorithm is deterministic and guarantees faster shortest path query time.

By plugging Corollary 6 into the standard multiplicative weight update framework (e.g. [55, 52, 42, 73]), we get the following:

▶ **Corollary 7.** *There exist $(1+\epsilon)$-approximate algorithms with expected running time $\tilde{O}(n^2)$ for the following problems:*

1. *minimum-cost maximum s-t flow in undirected vertex-capacitated graphs, and*
2. *minimum-cost maximum s-t flow in undirected unit-edge-capacity graphs.*

Corollary 7 slightly reduces the $n^{2+o(1)}$ run time from [42, 43] to $\tilde{O}(n^2)$.[3]

The second set of applications are faster algorithms for variants of multi-commodity flow problems using Theorem 1. For example, we achieve a static $\tilde{O}((n+k)n)$-time polylog(n)-approximation algorithm for the *congestion minimization* problem with k demand pairs on unweighted vertex-capacitated graphs. This improves the $\tilde{O}((m+k)n)$-time $O(\log(n)/\log\log(n))$-approximation algorithms implied by Karakostas [69] in terms of the running time at the cost of a worse approximation ratio. See the full version for more detail.

[3] We note that the result of [43] is deterministic.

Finally, we apply Theorem 4 to the problem of maintaining *effective resistance*. Durfee et al. [47, 46] presented a dynamic algorithm with $\tilde{O}(n^{6/7})$ amortized update time for $(1+\epsilon)$-approximately maintaining the effective resistance between a fixed pair of nodes. Plugging our result in the algorithm of Durfee et al. leads to an $n^{6/7+o(1)}$ worst-case update time. (Both of these results assume an oblivious adversary.)

1.3 Techniques

To prove the above results, the first key tool is the black-box reduction in Theorem 8 that allows us to focus on *almost-uniform-degree expanders*[4]. Theorem 8 works for a large class of problems satisfying natural properties (defined in Section 2) which includes spanners, cut sparsifiers, and spectral sparsifiers. Hence the theorem uses the term "α-approximate sparsifier" without defining the exact type of sparsifier.

▶ **Theorem 8** (Informal Blackbox Reduction, see full version for the formal statements)**.** *Assume that there is an algorithm $\mathcal{A}$ that can maintain an α-approximate sparsifier on an n-vertex graph G with the following promises:*

- *G undergoes batches of edge deletions*[5] *(isolated nodes are automatically removed),*
- *G is unweighted,*
- *after each batch of deletions, G is an expander graph, and*
- *after each batch of deletions, G has almost uniform degree, i.e. the maximum degree $\Delta_{\max}$ and the minimum degree $\Delta_{\min}$ are within a* polylog(n) *factor.*

Then, there is another algorithm $\mathcal{B}$ with essentially the same amortized update time for maintaining an α-approximate sparsifier of essentially the same size on a general weighted *graph undergoing both edge insertions and deletions. If $\mathcal{A}$ is adaptive or deterministic, then so is $\mathcal{B}$.*

As it is well-known that many problems become much easier on expanders (e.g., [84, 83, 81, 67, 34, 72]), we believe that this reduction will be useful for future developments of dynamic algorithms. For example, if one can come up with an adaptive algorithm for maintaining $(1+\epsilon)$-cut sparsifiers on expanders, then one can immediately obtain the same result on general graphs.

Our second technique is a new sampling scheme called *proactive resampling*: here we constantly re-sample parts of the input graph so that, independent of an adversary's computational power, a desired structure of the underlying graph can be always maintained; see Section 2 for a high-level discussion of this technique. Since there are still few known tools for designing algorithms that work against an adaptive adversary, we expect that our technique will prove useful for the design of other adaptive algorithms in the future.

We further extend the black-box reduction from Theorem 8 to algorithms with worst-case update time, which allows us to deamortize both Theorem 1 and Theorem 2 with slightly worse guarantees. It also easily implies Theorem 4.

1.4 Subsequent Development

Since this paper has appeared in April 2020, there have been exciting subsequent developments based on techniques of this paper.

[4] Expanders are graphs with high conductance (see Section 2). Intuitively, they are "robustly connected" graphs.

[5] That means, in each iteration the algorithm is given a set $D \subset E$ of edges that are to be deleted.

Our dynamic spectral sparsifiers with $(1+\epsilon)$-approximation but large query time (see full version) has been employed as a blackbox subroutine in a line of work on faster algorithms for exact max-flow [86, 85, 54], and also in [39] for making a dynamic polylog(n)-approximate all-pairs max flow algorithm work against an adaptive adversary. In [14], the authors opened the blackbox of our spectral sparsifier and combined our dynamic expander decomposition with our sparsification techniques to obtain the first dynamic algorithms with $o(n)$ update time against an adaptive adversary for $(1+\epsilon)$-approximate global min cuts and all-pairs effective resistances. Our almost-uniform-degree expander decomposition is also used in the context of online algorithms [59].

Our proactive resampling technique has been extended in a follow-up work [30] for maintaining fully dynamic 3-spanners against an adaptive adversary, in contrast with polylog(n)-spanners.

2 Overview

All our algorithms use a common framework based on expanders, which results in a reduction from fully dynamic algorithms on general graphs to the special case of decremental algorithms on expander graphs. The reduction holds for a general class of graph problems that satisfy some criteria. These criteria are satisfied for spectral-sparsifiers, cut-sparsifiers and spanners. In this section, we define the abstract criteria needed for our reduction (See Conditions 1-5 below), so that we only need to prove our algorithm once and apply it to all these types of sparsifiers.

The overview is split into three parts. In Section 2.1 we show the reduction for amortized update time. In Section 2.2 we show how to take advantage of the reduction by designing efficient algorithms on expanders. Finally, in Section 2.3 we finish the overview with a sketch of how to extend the reduction to worst-case update-time algorithms.

2.1 Reduction to Expanders: Amortized Update Time

We now outline our black-box reduction, which can preserve several nice properties of the algorithms. That is, given an algorithm with property x running on expander, we obtain another algorithm with property x with essentially the same running time and approximation guarantee, where the property x can be "deterministic", "randomized against an adaptive adversary", or "worst-case update time". In this subsection, we focus on amortized update time: see Section 2.3 for an overview of how to extend the reduction to apply to worst-case algorithms.

The reduction holds for any graph problem that satisfies a small number of conditions. We formalize a graph problem as a function $\mathcal{H}$ that maps (G, ϵ) for a graph G and parameter $\epsilon > 0$ to a set of graphs. We say a dynamic algorithm $\mathcal{A}$ solves $\mathcal{H}(\epsilon)$ if for every input graph G, algorithm $\mathcal{A}$ maintains/computes a graph $H \in \mathcal{H}(G, \epsilon)$. For example we could define $\mathcal{H}(G, \epsilon)$ to be the set of all $(1+\epsilon)$-cut sparsifiers. So then saying "data structure $\mathcal{A}$ solves $\mathcal{H}(\epsilon)$" means that $\mathcal{A}$ maintains for any input graph an $(1+\epsilon)$-cut sparsifier.

Pertubation Property

The first property required by our reduction allows us to slightly perturb the edges, i.e. scale each edge $\{u, v\}$ by some small factor $f_{u,v}$ bounded by $1 \leq f_{u,v} \leq e^{\epsilon}$. Define $\zeta \cdot G$ to be the graph G with all edge-weights multiplied by ζ.

$$\text{Let } G' \text{ be } G \text{ scaled by up to } e^{\epsilon}, \text{ then } G' \in \mathcal{H}(G, \epsilon) \text{ and } e^{\epsilon} \cdot G \in \mathcal{H}(G', \epsilon). \tag{1}$$

Property (1) implies that $G \in \mathcal{H}(G, \epsilon)$ for all $\epsilon > 0$. For example any graph is a (potentially dense) spectral approximation or spanner of itself. The property is also useful when we want to discretize the edge weights. A common technique is to round edge weights to the nearest power of e^{ϵ} in order to discretize the set of possible edge weights without changing graph properties such as the spectrum or distances too much. Combined with the following *union* property, this also allows us to generalize algorithm for unweighted graphs to support weighted graphs.

Union Property

Say that $G = \bigcup_{i=1}^{k} G_i$ for some k and that $s_1, ..., s_k \in \mathbb{R}$. Then the union property is defined as follows:

$$\text{If } H_i \in \mathcal{H}(G_i, \epsilon) \text{ and } 0 \leq s_i, \text{ then } \bigcup_i s_i \cdot H_i \in \mathcal{H}\left(\bigcup_i s_i \cdot G_i, \epsilon\right). \tag{2}$$

Combining this property with the previous *pertubation* property (1) gives us the following reduction. Given a graph G with real edge weights from $[1, W]$, one can decompose G into graphs $G_1, ..., G_k$, such that each G_i contains edges with weights in $[e^{(i-1)\epsilon}, e^{i\epsilon})$. One can then use any algorithm that solves $\mathcal{H}$ on *unweighted* graphs to obtain $H_i \in \mathcal{H}(G'_i, \epsilon)$ for all $i = 1, ..., k$, where G'_i is the graph G_i when ignoring the edge weights. Then $\bigcup_i e^{i\epsilon} \cdot H_i \in \mathcal{H}(\bigcup_i e^{i\epsilon} \cdot G_i, \epsilon) \subset \mathcal{H}(G, \epsilon)$ by combining property (2) and (1). Thus one obtains an algorithm that solves $\mathcal{H}$ on *weighted* graphs.

Reduction for Amortized Update Time

Loosely speaking, our black-box states the following. Say that we have a data structure $\mathcal{A}_X$ on a graph X that at all times maintains a sparsifier in $\mathcal{H}(X, \epsilon)$ with amortized update time $T(\mathcal{A}_X)$, but assumes the following restricted setting: **1)** Every update to X is an edge *deletion* (no insertion), and **2)** X is always an expander. We claim that $\mathcal{A}_X$ can be converted into a *fully dynamic* algorithm $\mathcal{A}$ that works on any graph G, and has amortized update time $T(\mathcal{A}) = \tilde{O}(T(\mathcal{A}_X))$.

We first outline this black-box under the assumption that we have a dynamic algorithm that maintains a decomposition of $G = \bigcup_i G_i$ into edge disjoint expander graphs $G_1, G_2, ...$ This dynamic algorithm will have the property that whenever the main graph G is updated by an adversarial edge insertion/deletion, each expander G_i receives only edge deletions, though occasionally a new expander G_j is added to the decomposition. Thus one can simply initialize $\mathcal{A}_X$ on the expander G_j to obtain some $H_j \in \mathcal{H}(G, \epsilon)$, when G_j is added to the decomposition. Then whenever an edge deletion is performed to G_j, we simply update the algorithm $\mathcal{A}_X$ to update the graph H_j. By the union property (2) we then have that

$$H := \bigcup_i H_i \in \mathcal{H}\left(\bigcup_i G_i, \epsilon\right) = \mathcal{H}(G, \epsilon).$$

So we obtain an algorithm $\mathcal{A}$ that can maintain a sparsifier H of G. We are left with proving how to obtain this dynamic algorithm for maintaining the expander decomposition of G.

Dynamic Expander Decomposition

The idea is based on the expander decomposition and expander pruning of [80]. Their expander decomposition splits V into disjoint node sets $V_1, V_2, ...$, such that the induced subgraphs $G[V_i]$ on each V_i are expanders, and there are only $o(m)$ edges between these

expanders. In the full version we show that by recursively applying this decomposition on the subgraph induced by the inter-expander edges, we obtain a partition of the *edges* of G into a union of expanders. This means, we can decompose G into subgraphs $G_1, G_2, ...$, where each G_i is an expander and $\bigcup_i G_i = G$. We show in the full version that the time complexity of this decomposition algorithm is $\tilde{O}(m)$.

We now outline how we make this decomposition dynamic in the full version. Assume for now, that we have a decomposition of G into $G = \bigcup_i G^{(i)}$, where for all i, the graph $G^{(i)}$ has at most 2^i edges, but each $G^{(i)}$ is not necessarily an expander. Further, each $G^{(i)}$ is decomposed into expanders $G^{(i)} = \bigcup_j G^{(i)}_j$. To make this decomposition dynamic, we will first consider edge insertions where we use a technique from [64]. Every edge insertion performed by the adversary is fed into the graph $G^{(1)}$. Now, when inserting some edges into some $G^{(i)}$, there are two cases: (i) the number of edges in $G^{(i)}$ remains at most 2^i. In that case recompute the expander decomposition $G^{(i)} = \bigcup_j G^{(i)}_j$ of $G^{(i)}$. Alternatively we have case (ii) where $G^{(i)}$ has more than 2^i edges. In that case we set $G^{(i)}$ to be an empty graph and insert all the edges that previously belonged to $G^{(i)}$ into $G^{(i+1)}$. Note that on average it takes 2^{i-1} adversarial insertions until $G^{(i)}$ is updated, and we might have to pay $\tilde{O}(2^i)$ to recompute its decomposition, so the amortized update time for insertions is simply $\tilde{O}(1)$.

For edge deletions we use the expander pruning technique based on [80] (refined from [77, 76, 88]). An over-simplified description of this technique is that, for each update to the graph, we can repeatedly prune (i.e. remove) some $\tilde{O}(1)$ edges from the graph, such that the remaining part is an expander. So whenever an edge is deleted from G, we remove the edge from its corresponding $G^{(i)}_j$, and remove/prune some $\tilde{O}(1)$ extra edges from $G^{(i)}_j$, so that it stays an expander graph. These pruned edges are immediately re-inserted into $G^{(1)}$ to guarantee that we still have a valid decomposition $G = \bigcup_i G^{(i)}$. In summary, we are able to maintain a decomposition $G = \bigcup_{i,j} G^{(i)}_j$ of G into expander graphs. This decomposition changes only by creating new expanders and removing edges from existing expanders, so we can run the decremental expander algorithm $\mathcal{A}_X$ on each $G^{(i)}_j$.

Contraction Property and Reduction to Uniform Degree Expanders

Many problems are easier to solve on graphs with (near) uniform degree. Thus, we strengthen our reduction to work even if the decremental algorithm $\mathcal{A}_X$ assumes that graph X has near-uniform degree. On its own, the expander decomposition described above is only able to guarantee that for each expander the minimum degree is close to the average degree; the maximum degree could still be quite large. In order to create a (near) uniform degree expander, we split these high degree nodes into many smaller nodes of smaller degree. In order to perform this operation, we need the condition that whichever graph problem $\mathcal{H}$ we are trying to solve must be able to handle the reverse operation, i.e. when we contract many small degree nodes into a single large degree node.

$$\begin{aligned}&\text{When contracting } W \subset V \text{ in both } G \text{ and } H \in \mathcal{H}(G, \epsilon),\\ &\text{let } G' \text{ and } H' \text{ be the resulting graphs, then } H' \in \mathcal{H}(G', \epsilon).\end{aligned} \tag{3}$$

All in all, our black-box reduction shows that in order to solve a sparsification problem $\mathcal{H}$ in the fully dynamic model on general graphs, we need to **1)** show that $\mathcal{H}$ satisfies the perturbation, union, and contraction properties above (Properties 1-3) AND **2)** Design an algorithm $\mathcal{A}_X$ for $\mathcal{H}$ in the simpler setting where the dynamic updates are purely decremental (only edge deletions), and where the dynamic graph G is always guaranteed to be a near-uniform degree expander.

We now present the second main contribution of our paper, which is a new adaptive algorithm $\mathcal{A}_X$ on expanders. We conclude the overview with a discussion of the worst-case reduction (Section 2.3), for which we will need two additional properties of the problem $\mathcal{H}$.

2.2 Adaptive Algorithms on Expanders

We showed above that maintaining a sparsifier in general graphs can be reduced to the same problem in a near-uniform-degree expander. Thus, for the rest of this section we assume that $G = (V, E)$ is *at all times* a ϕ-expander with max degree $\Delta_{\max}$ and min-degree $\Delta_{\min}$, and that G is only subject to edge deletions. Let $n = |V|, m = |E|$. In this overview, we assume that $1/\phi$ and $\Delta_{\max}/\Delta_{\min}$ are $O(\text{polylog}\, n)$, and we assume $\Delta_{\min} \gg 1/\phi$. Define $\textsc{Inc}_G(v)$ to the edges incident to v in G.

We now show how to maintain a $O(\log(n))$-approximate cut-sparsifier H in G against an adaptive adversary; it is not hard to check that H is also a spanner of stretch $\tilde{O}(1/\phi)$, because a cut-sparsifier of a ϕ-expander is itself a $\tilde{\Omega}(\phi)$-expander, and hence has diameter $\tilde{O}(1/\phi)$. See full version for details.

Static Expander Construction

We first show a very simple *static* construction of $H \subseteq G$. Define $\rho = \tilde{\Theta}\left(\frac{\Delta_{\max}}{\Delta_{\min}^2 \phi^2}\right) = \tilde{\Theta}\left(\frac{1}{\Delta_{\min}}\right)$, with a sufficiently large polylog factor. Now, every edge is independently sampled into H with probability ρ, and if sampled, is given weight $1/\rho$. To see that H is a cut sparsifier, consider any cut $X, \bar{X}$, with $|X| \leq n/2$. We clearly have $\mathbb{E}[|E_H(X, \bar{X})|] = \rho |E_G(X, \bar{X})|$, so since every edge in H has weight $1/\rho$, we have the same weight *in expectation*. For a high probability bound, want to show that $\Pr[|E_H(X, \bar{X})| \sim \rho |E_G(X, \bar{X})|] \geq 1 - n^{-2|X|}$; we can then take a union bound over the $O(n^{|X|})$ cuts of size $|X|$.

Since the graph is an expander, we know that $|E_G(X, \bar{X})| \geq \text{vol}_G(X) \cdot \phi \geq |X| \cdot \Delta_{\min} \cdot \phi = \tilde{\Omega}(|X|\Delta_{\min})$. Thus, by our setting of $\rho = \tilde{\Theta}(1/\Delta_{\min})$, we have $\mathbb{E}[|E_H(X, \bar{X})|] \geq |X| \log^2(n)$. Since each edge is sampled independently, a chernoff bound yields the desired concentration bound for $|E_H(X, \bar{X})|$.

Naive Dynamic Algorithms

The most naive dynamic algorithm is: whenever the adversary deletes edge (u, v), resample all edges in $\textsc{Inc}_G(u)$ and $\textsc{Inc}_G(v)$: that is, include each such edge in H with probability ρ. Efficiency aside, the main issue with this protocol is that the adversary can cause some target vertex x to become *isolated* in H, which clearly renders H not a cut sparisifer. To see this, let $y_1, \ldots, y_k$ be the neighbors of x. The adversary then continually deletes arbitrary edges $(y_1, z) \neq (y_1, x)$, which has the effect of resampling edge (x, y_1) each time. With very high probability, the adversary can ensure within $\log(n)$ such deletions (y_1, z) that (x, y_1) is NOT included in H; the adversary then does the same for y_2, then y_3, and so on.

Slightly Less Naive Algorithm

To fix the above issue, we effectively allow vertices u and v to have separate copies of edge (u, v), where u's copy can only be deleted if u itself is resampled. Formally, every vertex v will have a corresponding set of edges S_v and we will always have $H = \bigcup_{v \in V} S_v$, where all edges in H have weight $1/\rho$. We define an operation $\textsc{SampleVertex}(v)$ that *independently* samples each edge in $\textsc{Inc}_G(v)$ into S_v with probability ρ. The naive implementation of $\textsc{SampleVertex}(v)$ takes time $O(\deg_G(v)) = O(\Delta_{\max})$ time, but an existing technique used

in [71, 45, 32] allows us to implement SAMPLEVERTEX(v) in time $O(\rho\Delta_{\max}\log(n)) = \tilde{O}(1)$. (The basic idea is that the sampling can be done in time proportional to the number of edges successfully chosen, rather than the number examined.)

The dynamic algorithm is as follows. At initialization, construct each S_v by calling SAMPLEVERTEX(v), and then set $H = \bigcup_{v\in V} S_v$. Whenever the adversary deletes edge (u, v), replace S_u and S_v with new sets SAMPLEVERTEX(u) and SAMPLEVERTEX(v), and modify $H = \bigcup_{v\in V} S_v$ accordingly. By the above discussion, the update time is clearly $\tilde{O}(1)$. We now show that this algorithm effectively guarantees a good *lower bound* on the weight of each cut in H, but might still lead to an overly high weight. Consider any cut $(X, \bar{X})$. By the expansion of G, the *average* vertex $x \in X$ has $\text{INC}_G(x) \cap E_G(X, \bar{X}) \geq \phi\Delta_{\min}$. For simplicity, let us assume that *every* vertex $x \in X$ has $\text{INC}_G(x) \cap E_G(X, \bar{X}) = \tilde{\Omega}(\phi\Delta_{\min}) = \tilde{\Omega}(\Delta_{\min})$, as we can effectively ignore the small fraction of vertices for which this is false. Now, say that an operation SAMPLEVERTEX(x) succeeds if it results in $|S_x \cap E_G(X, \bar{X})| \sim \rho|\text{INC}_G(x) \cap E_G(X, \bar{X})|$. Because of our setting for ρ and our assumption that $\text{INC}_G(x) \cap E_G(X, \bar{X}) = \tilde{\Omega}(\Delta_{\min})$, a Chernoff bound guarantees that each SAMPLEVERTEX(x) succeeds with probability $1-n^{-10}$. Now, since the adversary makes at most m updates before the graph is empty, each SAMPLEVERTEX(x) is called at most n^2 times, so there is a $1-n^{-8}$ probability that *every* call SAMPLEVERTEX(x) is successful; we call such vertices always-successful. A simple probability calculation shows that $\Pr[\text{at least } |X|/2 \text{ vertices in } X \text{ are always-successful}] \geq 1 - n^{-2|X|}$, which allows us to union bound over all cuts of size X. Thus, at all times, half the vertices in X have $|S_x \cap E_G(X, \bar{X})| \sim \rho|\text{INC}_G(x) \cap E_G(X, \bar{X})|$; assuming for simplicity that this is an "average" half of vertices, i.e. that these vertices have around half of the edges crossing the cut, we have $|E_H(X, \bar{X})| \geq |\bigcup_{x\in X} S_x \cap E_G(X, \bar{X})| \gtrsim \rho|E_G(X, \bar{X})|/2$.

The above idea already implies that we can maintain a sparse graph H where each cut is expanding, i.e. a sparse expander, against an adaptive adversary. As an expander has low diameter, H is a spanner. Therefore, we are done if our goal is a dynamic spanner algorithm.

Unfortunately, this algorithm is not strong enough for maintaining cut sparsifiers, as the algorithm may result in $|E_H(X, \bar{X})| \gg \rho|E_G(X, \bar{X})|$. Let $\Delta_{\max} \sim \sqrt{n}$, and consider the following graph G. There is a set X of size $\sqrt{n}$ such that $G[X]$ is a clique and $G[\bar{X}]$ is $\sqrt{n}$-degree-expander. There is also a $\sqrt{n}$-to-1 matching from X to $\bar{X}$: so every vertex in $y \in \bar{X}$ has *exactly one* edge e_y crossing the cut. It is easy to check that G is an expander. The adversary then does the following. For each $y \in \bar{X}$, it keeps deleting edges in $E(y, \bar{X})$ until e_y is sampled into S_y; with high probability, this occurs within $O(\log(n)/\rho)$ deletions for each vertex y. Thus, at the end, $H \supseteq \bigcup_{y\in\bar{X}} S_y$ contains all of $E_G(X, \bar{X})$.

Better Algorithm via Proactive Sampling

We now show how to modify the above algorithm to ensure that w.h.p., $|E_H(X, \bar{X})| = \tilde{O}(\rho|E_G(X, \bar{X})|)$; we later improve this to $|E_H(X, \bar{X})| = O(\rho\log(n)|E_G(X, \bar{X})|)$. We let time t refer to the tth adversarial update. As before, we always have $H = \bigcup_{v\in V} S_v$, and if the adversary deletes edge (u, v) at time t, the algorithm immediately calls SAMPLEVERTEX(u) and SAMPLEVERTEX(v). The change is that the algorithm also calls SAMPLEVERTEX(u) and SAMPLEVERTEX(v) at times $t+1, t+2, t+4, t+8, t+16, \ldots$; we call this *proactive sampling*. The proof that $|E_H(X, \bar{X})| \gtrsim \rho|E_G(X, \bar{X})|/2$ remains basically the same as before. We now upper bound $|E_H(X, \bar{X})|$.

The formal analysis is somewhat technical, but the crux if the following **key claim:** for any $(u, v) \in G$, we have that after t adversarial updates, $\Pr[(u, v) \in H \text{ at time t}] \leq 2\rho\log(t) \leq 2\rho\log(m)$. We then use the key claim as follows: consider any cut $(X, \bar{X})$. If every edge in $E_G(X, \bar{X})$ was *independently* sampled into H with probability at most $2\rho\log(m)$,

then a Chernoff bound would show that $|E_H(X, \bar{X})| \leq 4\rho \log(m)|E_G(X, \bar{X})|$ with probability at least $1 - n^{-2|X|}$, as desired. Unfortunately, even though every individual edge-sampling occurs with probability ρ, independent of everything that happened before, it is NOT the case that event $e \in H$ is independent from event $e' \in H$: the adversary is adaptive, so its sampling strategy for e' can depend on whether or not e was successfully sampled into H at an earlier time. Nonetheless, we show in the full proof that these dependencies can be disentangled.

Let us now sketch the proof for the key claim. The edge (u, v) can appear in H because it is in S_u or S_v at the time t. Let us bound the probability that $(u, v) \in S_u$ at time t. Let $T_{\text{schedule}}(u)$ be all times before t for which SampleVertex(u) has been scheduled by proactive sampling: so whenever the adversary updates an edge (u, v) at time t', times $t', t' + 1, t' + 2, t' + 4, t' + 8, \ldots$ are added to $T_{\text{schedule}}(u)$. Let $T^{t'}_{\text{schedule}}(u) \subset [t', t]$ be the state of $T_{\text{schedule}}(u)$ at time t'. Now, we say that a call to SampleVertex(u) at time t' is *relevant* if $T^{t'+1}_{\text{schedule}}(u) = \emptyset$. Observe that for (u, v) to be in S_u at time t, it must have been added during some relevant call SampleVertex(u), because every non-relevant call is followed by another call before time t which invokes again SampleVertex(u) and thereby deletes the previously sampled set S_u and replaces it by a new one. We complete the proof by claiming that there are at most $\log(t)$ relevant calls SampleVertex(u). This is because if a relevant call occurs at t', then proactive sampling adds some time t^* to $T_{\text{schedule}}(u)$ such that $(t' + t)/2 \leq t^* \leq t$; thus, there can be no relevant calls in time interval $[t', (t' + t)/2]$. So each relevant call halves the possible time interval for other relevant calls, so there are at most $\log(t)$ relevant calls.

We now briefly point out why this modified algorithm has $|E_H(X, \bar{X})| = \tilde{O}(\rho|E_G(X, \bar{X})|)$, rather than the desired $|E_H(X, \bar{X})| = O(\rho \log(n)|E_G(X, \bar{X})|)$. Consider again the graph G consisting of a vertex set X of size $\sqrt{n}$ such that $G[X]$ is a complete graph, and let $\overline{X}$ be a $\sqrt{n}$-degree expander in $G[\overline{X}]$. Additionally, we have a $\sqrt{n}$-to-one matching, i.e. every vertex in X is matched to $\sqrt{n}$ vertices in $\overline{X}$. The graph is still an expander as argued before.

Now observe that $\Delta_{\max} = 2\sqrt{n}$ and we obtain a first sparsifier H at time 0 of G where we have weight on the cut, i.e. $|E_H(X, \bar{X})|/\rho$, of size $\sim n$ (which is the number of edges crossing). In particular, the weight on edges in $|E_H(X, \bar{X}) \cap \bigcup_{x \in \overline{X}} S_x|/\rho \sim n$, i.e. the vertices in $\overline{X}$ carry half the weight of the cut in the sparsifier. But over the course of the algorithm, the adversary can delete edges in the cut $(X, \overline{X})$ that are in $G \setminus H$. Observe that the resampling events do not affect edges in $E_H(X, \bar{X}) \cap \bigcup_{x \in \overline{X}} S_x$ since none of the deleted edges is incident to any such edge (recall the $\sqrt{n}$-to-one matching). The adversary can continue until the cut only has weight in G of $|X|\Delta_{\min}\phi$ without violating the expander and min-degree guarantees. But then the weight in H on the cut is still $\sim n$ while the weight in G is only $\sim n(\Delta_{\max}/\Delta_{\min})\phi$. Thus, we only obtain a $\sim (\Delta_{\max}/\Delta_{\min})\phi$-approximation (plus a $\log n$-factor from proactive sampling might appear).

Final Algorithm

To resolve the issue above, we would like to ensure that the edges in $E_H(X, \bar{X})$ are resampled when $|E_G(X, \bar{X})|$ changes by a large amount. We achieve this with one last modification to the algorithm: for every $v \in V$, whenever $\deg_G(v)$ decreases by $\zeta = \phi\Delta_{\min}$, we run SampleVertex$(w)$ for every edge $(v, w) \in G$. It is not hard to check that each vertex will only resampled a total of $O(\Delta^2_{\max}/\zeta) = \tilde{O}(\Delta_{\max})$ additional times as a result of this change which is subsumed by $\tilde{O}(m)$ when summing over the vertices. (A naive implementation of the above modification only leads to small *amortized* update time, but this can easily be worst-case by staggering the work over several updates using round-robin scheduling.) We leave the analysis of the approximation ratio for the full version.

By using the convenient lemma which says that any cut sparsifier on an expander is also a spectral sparsifier, we also obtain an adaptive algorithm for spectral sparsifier.

2.3 Reduction to Expanders: Worst-Case Update Time

We now outline how to extend our black-box reduction to work with worst-case update time. We again assume there exists some algorithm $\mathcal{A}_X$ that maintains for any graph G a sparsifier $H \in \mathcal{H}(G, \epsilon)$, provided that G stays a uniform degree expander throughout all updates, all of which are only edge deletions.

The condition that G always remains an expander is too strong, but we can use expander pruning to maintain the property from the perspective of $\mathcal{A}_X$. Consider some deletion in G: although G may not be an expander, we can use pruning to find a subset of edges $P \subset E(G)$, such that $G \setminus P$ is an expander. We then input all edges in P as deletions to $\mathcal{A}_X$, so the graph $G \setminus P$ in question is still an expander: $\mathcal{A}_X$ thus returns $H \in \mathcal{H}(G \setminus P, \epsilon)$. Then based on property (1) and (2) it can be shown that $H \cup P \in \mathcal{H}(G, \epsilon)$. So by taking all the pruned edges together with the sparsifier H of $G \setminus P$ we obtain a sparsifier of G, even when G itself is no longer an expander.

Unfortunately this dynamic sparsifier algorithm has two downsides: (i) The maintained sparsifier is only sparse for a short sequence of updates, as otherwise the set of pruned edges becomes too large and thus the output $H \cup P$ becomes too dense. (ii) The algorithm only works on graphs that are initially an expander.

Extending the algorithm to general graphs

To extend the previous algorithm to work on general graphs, we run the static expander decomposition algorithm. As outlined before, we can decompose G into subgraphs $G_1, G_2, ...$, where each G_i is an expander and $\bigcup_i G_i = G$. We can then run the algorithm, outlined in the previous paragraph, on each of these expanders and the union of all the obtained sparsifiers will be a sparsifier of the original input G.

Similar as before, one downside of this technique is that the size of the sparsifier will increase with each update, because more and more edges will be pruned. Thus, the resulting dynamic algorithm can only maintain a sparsifier for some limited number of updates.

Extending the number of updates

A common technique for dynamic algorithms, which only work for some limited number of updates (say k updates), is to reset the algorithm after k updates. If the algorithm has preprocessing time p and update time u, then one can obtain an algorithm with amortized update time $O(p/k + u)$. However, the worst-case complexity would be quite bad, because once the reset is performed, the old sparsifier (from before the reset) must be replaced by the new one. Listing all edges of the new sparsifier within a single update would be too slow. There is a standard technique for converting such an amortized bound to an equivalent worst-case bound. The idea is to slowly translate from the old sparsifier to the new one, by only listing few edges in each update. For this we require another property for $\mathcal{H}$ that guarantees that the sparsifier stays valid, even when removing a few of its edges.

Transition Property

Consider some $H_1, H_2 \in \mathcal{H}(G, \epsilon)$, and we now want to have a slow transition from H_1 to H_2, by slowly removing edges from H_1 from the output (and slowly inserting edges of H_2). The exact property we require is as follows:

$$\text{Let } H_1, H_2 \in \mathcal{H}(G, \epsilon) \text{ and } H \subset H_1, \text{ then } (e^{\delta} - 1)H \cup H_2 \in \mathcal{H}(G, \epsilon + \delta). \tag{4}$$

Here $H \subset H_1$ represents the remaining to be removed edges (or alternatively $H_1 \setminus H$ are the remaining to be inserted edges). Exploiting this property we are able to obtain a $O(p/k + u)$ worst-case update time.

As the output grows with each update, we must perform the reset after $k = O(n)$ updates, otherwise the output becomes too dense. This is unfortunate as the preprocessing time is $p = \Omega(m)$, because one must read the entire input, which is too slow to obtain a subpolynomial update time. This issue can be fixed via a sparsification technique based on [49], presented in the full version. By using this technique, we can make sure that $m = O(n^{1+o(1)})$ and thus the preprocessing time will be fast enough to allow for $O(n^{o(1)})$ update time.

For this sparsification technique we require the following transitivity property.

Transitivity Property

$$\text{If } H \in \mathcal{H}(G, \epsilon), \text{ then } \mathcal{H}(H, \delta) \subset \mathcal{H}(G, \delta + \epsilon). \tag{5}$$

Intuitively this means that an approximation H of G and an approximation H' of H, then H' is also a (slightly worse) approximation of G.

Properties 1-5 above are precisely the properties required of a graph problem by our black-box reduction for worst-case update time. We show in the full version that the sparsifiers discussed in this paper (spectral sparsifier, cut sparsifier, spanner) satisfy all these properties.

Sparsification Technique

We now outline the sparsification technique, whose formal proof is presented in the full version. Let G be an arbitrary graph. We partition the edges of G into equally sized subgraphs $G_1, G_2, ..., G_d$ for some $d > 1$. Note that, if we have ε-approximate sparsifiers $H_1, ..., H_d$ of $G_1, ..., G_d$, then $\bigcup_i H_i$ is a ε-approximate sparsifier of G by property (2). In addition, if we have a ε-approximate sparsifier H of $\bigcup_i H_i$, then H is a (2ε)-approximate sparsifier of G by property (5). This allows us to obtain a faster algorithm as follows: If G has m edges, then each G_i has only m/d edges, so the dynamic algorithm runs faster on these sparse G_i. Further, since the H_i are sparse (let's say $O(n)$ edges), the graph $\bigcup_i H_i$ has only $O(dn)$ edges and maintaining H is also faster than maintaining a sparsifier of G directly, if $dn = o(m)$. The next idea is to repeat this trick recursively: We repeatedly split each G_i into $d = n^{o(1)}$ graphs, until the graphs have only $O(n^{1+o(1)})$ edges. This means we obtain some tree-like structure rooted at G, where each tree-node G' represents a subgraph of G and its tree-children are the d subgraphs $G'_1, ..., G'_d$ of G'. For the graphs that form leaves of this tree, we run our dynamic sparsifier algorithm. We also obtain a sparsifier H' of any non-leaf tree-node G', by running our dynamic algorithm on $\bigcup_{i=1}^{d} H'_i$, where the H'_i are sparsifiers of the child-tree-nodes G'_i of the tree-node G'. Thus all instances of our dynamic algorithm always run on sparse input graphs. However, there is one downside: When some sparsifier H'_i changes, the sparsifier H' must also change. Let's say some edge is deleted from G, then the edge is deleted from one leaf-node of the tree-structure, and this update will propagate

from the leaf-node all the way to the root of the tree. This can be problematic because when the dynamic algorithm changes some $c > 1$ many edges of the sparsifier for each edge update, then the number of updates grows exponentially with the depth of the tree-like structure. In [49] Eppstein et al. circumvented this issue by assuming an extra property which they call *stability property*, which essentially says that this exponential growth does not occur. Our modified sparsification technique no longer requires this assumption, instead we balance the parameter d carefully to make sure the blow-up of the propagation is only some sub-polynomial factor.

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *FOCS*, pages 335–344, 2016. doi:10.1109/FOCS.2016.44.

2 Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *CoRR*, abs/1909.03152, 2019.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14, 2012.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *APPROX-RANDOM*, pages 1–10, 2013.

5 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993. doi:10.1007/BF02189308.

6 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *STOC*, pages 815–826. ACM, 2018.

7 Giorgio Ausiello, Paolo Giulio Franciosa, and Giuseppe F. Italiano. Small stretch spanners on dynamic graphs. *J. Graph Algorithms Appl.*, 10(2):365–385, 2006. Announced at ESA'05. URL: http://jgaa.info/accepted/2006/AusielloFranciosaItaliano2006.10.2.pdf, doi:10.7155/jgaa.00133.

8 Surender Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008. doi:10.1016/j.ipl.2007.11.001.

9 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $o(\log n)$ update time (corrected version). *SIAM J. Comput.*, 47(3):617–650, 2018. doi:10.1137/16M1106158.

10 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012. Announced at SODA'08. doi:10.1145/2344422.2344425.

11 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007. doi:10.1002/rsa.20130.

12 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Rev.*, 56(2):315–334, 2014.

13 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, pages 382–405. IEEE Computer Society, 2019.

14 Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. Dynamic algorithms against an adaptive adversary: Generic constructions and lower bounds. *arXiv preprint*, 2021. To appear at STOC'22. arXiv:2111.03980.

15 András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

16 Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *ICALP*, volume 80, pages 44:1–44:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.44.

17 Aaron Bernstein and Shiri Chechik. Deterministic decremental single source shortest paths: beyond the $o(mn)$ bound. In *STOC*, pages 389–397, 2016.

18 Aaron Bernstein and Shiri Chechik. Deterministic partially dynamic single source shortest paths for sparse graphs. In *SODA*, pages 453–469, 2017.

19 Aaron Bernstein and Shiri Chechik. Incremental topological sort and cycle detection in expected total time. In *SODA*, pages 21–34. SIAM, 2018.

20 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *SODA*, pages 1899–1918, 2019.

21 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. *arXiv preprint*, 2021. arXiv:2101.07149.

22 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *STOC*, pages 365–376, 2019.

23 Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *SODA*, pages 1355–1365, 2011.

24 Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. *CoRR*, abs/2004.08432, 2020.

25 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *SODA*, pages 1–20, 2018.

26 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *SODA*, 2015.

27 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, pages 398–411, 2016.

28 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *SODA*, pages 470–489, 2017.

29 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $o(1/\epsilon^2)$ amortized update time. In *SODA*, 2019.

30 Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert. Simple dynamic spanners with near-optimal recourse against an adaptive adversary. In submission.

31 Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. In *ESA*, pages 17:1–17:18, 2016. doi:10.4230/LIPIcs.ESA.2016.17.

32 Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. In *ICALP*, pages 133–144. Springer, 2012.

33 Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In *PODC*, pages 217–226. ACM, 2016.

34 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *PODC*, pages 66–73, 2019.

35 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *FOCS*, 2018.

36 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *FOCS*, pages 370–381. IEEE Computer Society, 2019.

37 Chandra Chekuri and Kent Quanrud. Near-linear time approximation schemes for some implicit fractional packing problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 801–820. SIAM, 2017.

38 Chandra Chekuri and Kent Quanrud. Fast approximations for metric-tsp via linear programming. *arXiv preprint*, 2018. arXiv:1802.01242.

39 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *FOCS*, pages 1135–1146. IEEE, 2020.

40 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *FOCS*, pages 361–372, 2018. doi:10.1109/FOCS.2018.00042.

41 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019.

42 Julia Chuzhoy and Sanjeev Khanna. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *STOC*, pages 389–400, 2019.

43 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *SODA*, pages 2478–2496. SIAM, 2021.

44 Bilel Derbel, Mohamed Mosbah, and Akka Zemmari. Sublinear fully distributed partition with applications. *Theory Comput. Syst.*, 47(2):368–404, 2010. doi:10.1007/s00224-009-9190-x.

45 Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

46 David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic effective resistances. *CoRR*, abs/1804.04038, 2018.

47 David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In *STOC*, pages 914–925, 2019.

48 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17, 2011. Announced at ICALP'07. doi:10.1145/1921659.1921666.

49 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

50 Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

51 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *SODA*, pages 745–754, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070537.

52 Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000. announced at FOCS'99. doi:10.1137/S0895480199355754.

53 Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *STOC*, pages 71–80, 2011.

54 Yu Gao, Yang P Liu, and Richard Peng. Fully dynamic electrical flows: sparse maxflow faster than goldberg-rao. *arXiv preprint*, 2021. arXiv:2101.07233.

55 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007. doi:10.1137/S0097539704446232.

56 Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Minimum cut in o(m $\log^2$ n) time. In *ICALP*, volume 168 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

57 Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. In *DISC*, pages 29:1–29:17, 2018. doi:10.4230/LIPIcs.DISC.2018.29.

58 Ofer Grossman and Merav Parter. Improved deterministic distributed construction of spanners. In *DISC*, pages 24:1–24:16, 2017. doi:10.4230/LIPIcs.DISC.2017.24.

59 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Sahil Singla. Online carpooling using expander decompositions. In *FSTTCS*, volume 182 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

60 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Symposium on Theory of Computing*, 2020. arXiv:2001.10751.

61 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In *SODA*, pages 2542–2561, 2020.

62 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *SODA*, pages 2522–2541, 2020.

63 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *J. ACM*, 65(6):36:1–36:40, 2018. announced at FOCS'14.

64 Monika Rauch Henzinger and Valerie King. Maintaining minimum spanning trees in dynamic graphs. In *ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 594–604. Springer, 1997.

65 Jacob Holm and Eva Rotenberg. Dynamic planar embeddings of dynamic graphs. *Theory Comput. Syst.*, 61(4):1054–1083, 2017.

66 Jacob Holm and Eva Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *STOC*, pages 167–180. ACM, 2020.

67 Arun Jambulapati and Aaron Sidford. Efficient $\tilde{O}(n/\epsilon)$ spectral sketches for the laplacian and its pseudoinverse. In *SODA*, pages 2487–2503. SIAM, 2018.

68 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *SODA*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

69 George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Trans. Algorithms*, 4(1):13:1–13:17, 2008. doi:10.1145/1328911.1328924.

70 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic global minimum cut of a simple graph in near-linear time. In *STOC*, pages 665–674. ACM, 2015.

71 Donald Ervin Knuth. Seminumerical algorithms. *The art of computer programming*, 2, 1997.

72 Huan Li, He Sun, and Luca Zanetti. Hermitian Laplacians and a Cheeger inequality for the Max-2-Lin problem. In *ESA*, pages 71:1–71:14, 2019.

73 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *STOC*, pages 121–130, 2010. doi:10.1145/1806689.1806708.

74 Morteza Monemizadeh. Dynamic maximal independent set. *CoRR*, abs/1906.09595, 2019.

75 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: Sequential, cut-query and streaming algorithms. In *STOC*, 2020.

76 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, Las vegas, and $O(n^{1/2-\epsilon})$-time. In *STOC*, pages 1122–1129, 2017. doi:10.1145/3055399.3055447.

77 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, pages 950–961, 2017.

78 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *ICALP*, pages 261–272, 2005. doi:10.1007/11523468_22.

79 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.

80 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *SODA*, pages 2616–2635, 2019.

81 Jonah Sherman. Nearly maximum flows in nearly linear time. In *FOCS*, pages 263–269, 2013.

82 Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43.

83 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90, 2004.

84 Luca Trevisan. Approximation algorithms for unique games. In *FOCS*, pages 197–205. IEEE Computer Society, 2005.

85 Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and l1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 859–869, 2021.

86 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *FOCS*, pages 919–930. IEEE, 2020.

87 David Wajc. Rounding dynamic matchings against an adaptive adversary. *Symposium on Theory of Computing*, 2020. `arXiv:1911.05545`.

88 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *STOC*, pages 1130–1143, 2017. `doi:10.1145/3055399.3055415`.

89 Anastasios Zouzias. A matrix hyperbolic cosine algorithm and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 846–858. Springer, 2012.

Fast Sampling via Spectral Independence Beyond Bounded-Degree Graphs

Ivona Bezáková
Department of Computer Science, Rochester Institute of Technology, NY, USA

Andreas Galanis
Department of Computer Science, University of Oxford, UK

Leslie Ann Goldberg
Department of Computer Science, University of Oxford, UK

Daniel Štefankovič
Department of Computer Science, University of Rochester, Rochester, NY, USA

Abstract

Spectral independence is a recently-developed framework for obtaining sharp bounds on the convergence time of the classical Glauber dynamics. This new framework has yielded optimal $O(n \log n)$ sampling algorithms on bounded-degree graphs for a large class of problems throughout the so-called uniqueness regime, including, for example, the problems of sampling independent sets, matchings, and Ising-model configurations.

Our main contribution is to relax the bounded-degree assumption that has so far been important in establishing and applying spectral independence. Previous methods for avoiding degree bounds rely on using L^p-norms to analyse contraction on graphs with bounded connective constant (Sinclair, Srivastava, Yin; FOCS'13). The non-linearity of L^p-norms is an obstacle to applying these results to bound spectral independence. Our solution is to capture the L^p-analysis recursively by amortising over the subtrees of the recurrence used to analyse contraction. Our method generalises previous analyses that applied only to bounded-degree graphs.

As a main application of our techniques, we consider the random graph $G(n, d/n)$, where the previously known algorithms run in time $n^{O(\log d)}$ or applied only to large d. We refine these algorithmic bounds significantly, and develop fast nearly linear algorithms based on Glauber dynamics that apply to all constant d, throughout the uniqueness regime.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Random walks and Markov chains

Keywords and phrases Hard-core model, Random graphs, Markov chains

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.21

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version* (theorem numbering here matches v2): `arxiv.org/abs/2111.04066`

Funding *Ivona Bezáková*: Supported in part by NSF grant DUE-1819546.
Daniel Štefankovič: Research supported by NSF grant CCF-2007287.

1 Introduction

Spectral independence method was introduced by Anari, Liu, and Oveis Gharan [2] as a framework to obtain polynomial bounds on the mixing time of Glauber dynamics. Originally based on a series of works on high-dimensional expansion [17, 9, 21, 18, 1], it has since then been developed further using entropy decay by Chen, Liu, and Vigoda [8] who obtained optimal $O(n \log n)$ mixing results on graphs of bounded maximum degree Δ whenever the

framework applies. This paper focuses on relaxing the bounded-degree assumption of [8], in sparse graphs where the maximum degree is not the right parameter to capture the density of the graph.

As a running example we will use the problem of sampling (weighted) independent sets, also known as the sampling problem from the hard-core model. For a graph $G = (V, E)$, the hard-core model with parameter $\lambda > 0$ specifies a distribution $\mu_{G,\lambda}$ on the set of independent sets of G, where for an independent set I it holds that $\mu_{G,\lambda}(I) = \lambda^{|I|}/Z_{G,\lambda}$ where $Z_{G,\lambda}$ is the partition function of the model (the normalising factor that makes the probabilities add up to 1). For bounded-degree graphs of maximum degree $d+1$ (where $d \geq 2$ is an integer), it is known that the problems of sampling and approximately counting from this model undergo a computational transition at $\lambda_c(d) = \frac{d^d}{(d-1)^{d+1}}$, the so-called uniqueness threshold [29, 27, 12]: they are poly-time solvable when $\lambda < \lambda_c(d)$, and computationally intractable for $\lambda > \lambda_c(d)$. Despite this clear complexity picture, prior to the introduction of spectral independence, the algorithms for $\lambda < \lambda_c(d)$ were based on elaborate enumeration techniques whose running times scale as $n^{O(\log d)}$ [29, 19, 22, 23]. The analysis of Glauber dynamics[1] using spectral independence in the regime $\lambda < \lambda_c(d)$ yielded initially $n^{O(1)}$ algorithms for any d [2], and then $O(n \log n)$ for bounded-degree graphs [8] (see also [7]). More recently, Chen, Feng, Yin, and Zhang [5] obtained $O(n^2 \log n)$ results for arbitrary graphs $G = (V, E)$ that apply when $\lambda < \lambda_c(\Delta_G - 1)$, where Δ_G is the maximum degree of G (see also [14] for related results when Δ_G grows like $\log n$).

The maximum degree is frequently a bad measure of the density of the graph, especially for graphs with unbounded-degree. One of the most canonical examples is the random graph $G(n, d/n)$ where the maximum degree grows with n but the average degree is d, and therefore one would hope to be able to sample from $\mu_{G,\lambda}$ for λ up to some constant, instead of $\lambda = o(1)$ that the previous results yield. In this direction, [26, 24] obtained an $n^{O(\log d)}$ algorithm based on correlation decay that applies to all $\lambda < \lambda_c(d)$ for all graphs with "connective constant" bounded by d (meaning, roughly, that for all $\ell = \Omega(\log n)$ the number of length-ℓ paths starting from any vertex is bounded by d^ℓ). The result of [24] applies to $G(n, d/n)$ for all $d > 0$. In terms of Glauber dynamics on $G(n, d/n)$, [20] showed an $n^{1+\Omega(1/\log\log n)}$ lower bound on the mixing time in the case of the Ising model; this lower bound actually applies to most well-known models, and in particular rules out $O(n \log n)$ mixing time results for the hard-core model when $\lambda = \Omega(1)$. The mixing-time lower bound on $G(n, d/n)$ has only been matched by complementary fast mixing results in models with strong monotonicity properties, see [20] for the ferromagnetic Ising model and [4] for the random-cluster model. Such monotonicity properties unfortunately do not hold for the hard-core model, and the best known results [10, 11] for Glauber dynamics on $G(n, d/n)$ give an n^C algorithm for $\lambda < 1/d$ and sufficiently large d (where C is a constant depending on d).

Our main contribution is to obtain nearly linear-time algorithms on $G(n, d/n)$, for all of the models considered in [24], i.e., the hard-core model, the monomer-dimer model (weighted matchings), and the antiferromagnetic Ising model. Key to our results are new spectral independence bounds for any $d > 0$ in the regime $\lambda < \lambda_c(d)$ for arbitrary graphs $G = (V, E)$ in terms of their "d-branching value" (which resembles the connective-constant notion of [24]). To state our main theorem for the hard-core model on $G(n, d/n)$, we first extend the definition

[1] Recall, for a graph G, the Glauber dynamics for the hard-core model iteratively maintains a random independent set $(I_t)_{t\geq 0}$, where at each step t a vertex v is chosed u.a.r. and, if $I_t \cup \{v\}$ is independent, it sets $I_{t+1} = I_t \cup \{v\}$ with probability $\frac{\lambda}{\lambda+1}$, otherwise $I_{t+1} = I_t \backslash \{v\}$. The mixing time is the maximum number (over the starting I_0) of steps t needed to get within total variation distance $1/4$ of $\mu_{G,\lambda}$, see Section 4.1 for the precise definitions.

of $\lambda_c(d)$ to all reals $d > 0$ by setting $\lambda_c(d) = \frac{d^d}{(d-1)^{d+1}}$ for $d > 1$, and $\lambda_c(d) = \infty$ for $d \in (0, 1)$. We use the term "*whp* over the choice of $G \sim G(n, d/n)$" as a shorthand for "as n grows large, with probability $1 - o(1)$ over the choice of $G(n, d/n)$". An ε-sample from a distribution μ supported on a finite set Ω is a random $\sigma \in \Omega$ whose distribution ν satisfies $\|\nu - \mu\|_{\mathrm{TV}} \leq \varepsilon$, where $\|\nu - \sigma\|_{\mathrm{TV}} = \frac{1}{2}\sum_{\sigma\in\Omega} |\nu(\sigma) - \mu(\sigma)|$.

▶ **Theorem 1.** *Let $d, \lambda > 0$ be such that $\lambda < \lambda_c(d)$. For any arbitrarily small constant $\theta > 0$, there is an algorithm such that, whp over the choice of $G \sim G(n, d/n)$, when the algorithm is given as input the graph G and an arbitrary rational $\varepsilon > 0$, it outputs an ε-sample from $\mu_{G,\lambda}$ in time $n^{1+\theta} \log \frac{1}{\varepsilon}$.*

The reader might wonder why is there no constant in front of the running time (in Theorems 1, 2, and 3) or why is there no requirement that n is sufficiently large? The assumption that n is sufficiently large is taken care of in the whp condition: there is a function $f_{d,\lambda,\theta} : \mathbb{Z} \to \mathbb{R}$ such that $\lim_{n\to\infty} f_{d,\lambda,\theta}(n) = 0$ and the "whp" means with probability $\geq 1 - f_{d,\lambda,\theta}(n)$ (the function $f_{d,\lambda,\theta}$ will have value ≥ 1 for small n, making the conclusion trivial for such n). Moreover, the family of $O(n^{1+\theta})$ algorithms from Theorem 1 can be turned into an $n^{1+o(1)}$ algorithm as follows. The function $f_{d,\lambda,\theta}$ is computable (and efficiently invertible), see Remark 35 in the full version for a discussion. Let $n_0 = 1$ and $n_k > n_{k-1}$ be such that $f_{d,\lambda,1/k}(n) \leq 1/k$ for all $n \geq n_k$. We are going to run the algorithm of Theorem 1 with $\theta = 1/k$ for $n \in \{n_{k-1}, \ldots, n_k - 1\}$. Note that the "combined" algorithm succeeds with probability $1 - o(1)$ and runs in time $n^{1+o(1)}$.

We further remark here that the algorithm of Theorem 1 (as well as Theorems 2 and 3 below) can also recognise in time $n^{1+o(1)}$ whether the graph $G \sim G(n, d/n)$ is a "good" graph, i.e., we can formulate graph properties that guarantee the success of the algorithm, are satisfied whp, and are also efficiently verifiable, see Section C.4 in the full version for details.

The key to obtaining Theorem 1 is to bound the spectral independence of $G(n, d/n)$. The main strategy that has been applied so far to bound spectral independence is to adapt suitably correlation decay arguments and, therefore, it is tempting to use the correlation decay analysis of [24]. This poses new challenges in our setting since [24] uses an L^p-norm analysis of correlation decay on trees, and the non-linearity of L^p-norms is an obstacle to converting their analysis into spectral independence bounds (in contrast, for bounded-degree graphs, the L^∞-norm is used which can be converted to spectral independence bounds using a purely analytic approach, see [7]). Our solution to work around that is to "linearise" the L^p-analysis by taking into account the structural properties of subtrees. This allows us to amortise over the tree-recurrence using appropriate combinatorial information (the d-branching values) and to bound subsequently spectral independence; details are given in Section 3, see Lemmas 10 and 12 (and equation (2) that is at the heart of the argument). Once the spectral independence bound is in place, further care is needed to obtain the fast nearly linear running time, paying special attention to the distribution of high-degree vertices inside $G(n, d/n)$ and to blend this with the entropy-decay tools developed in [8], see Section 4.2 for this part of the argument.

In addition to our result for the hard-core model, we also obtain similar results for the Ising and the matchings models. The configurations of the Ising model on a graph $G = (V, E)$ are assignments $\sigma \in \{0, 1\}^V$ which assign the spins 0 and 1 to the vertices of G. The Ising model with parameter $\beta > 0$ corresponds to a distribution $\mu_{G,\beta}$ on $\{0, 1\}^V$, where for an assignment $\sigma \in \{0, 1\}^V$, it holds that $\mu_{G,\beta}(\sigma) = \beta^{m(\sigma)}/Z_{G,\beta}$ where $m(\sigma)$ is the number of edges whose endpoints have the same spin assignment under σ, and $Z_{G,\beta}$ is the partition function of the model. The model is antiferromagnetic when $\beta \in (0, 1)$, and ferromagnetic otherwise. For $d \geq 1$, let $\beta_c(d) = \frac{d-1}{d+1}$; for $d \in (0, 1)$, let $\beta_c(d) = 0$. It is known that on

bounded-degree graphs of maximum degree $d+1$ the sampling/counting problem for the antiferromagnetic Ising model undergoes a phase transition at $\beta = \beta_c(d)$, analogous to that for the hard-core model [25, 19, 28, 13].

▶ **Theorem 2.** *Let $d, \beta > 0$ be such that $\beta \in (\beta_c(d), 1)$. For any constant $\theta > 0$, there is an algorithm such that, whp over the choice of $G \sim G(n, d/n)$, when the algorithm is given as input the graph G and an arbitrary rational $\varepsilon > 0$, it outputs an ε-sample from $\mu_{G,\beta}$ in time $n^{1+\theta} \log \frac{1}{\varepsilon}$.*

For a graph $G = (V, E)$, the matchings model with parameter $\gamma > 0$, also known as the monomer-dimer model, corresponds to a distribution $\mu_{G,\gamma}$ on the set of matchings of G, where for a matching M, it holds that $\mu_{G,\gamma}(M) = \gamma^{|M|}/Z_{G,\gamma}$ where $Z_{G,\gamma}$ is the partition function. For general graphs $G = (V, E)$ and $\gamma = O(1)$, [15, 16] gave an $O(n^2 m \log n)$ algorithm (where $n = |V|, m = |E|$), which was improved for bounded-degree graphs to $O(n \log n)$ in [8] using spectral independence. For $G(n, d/n)$, [24] gave an $O(n^{\log d})$ deterministic algorithm using correlation decay, and [14] showed that Glauber dynamics mixes in $n^{2+o(1)}$ steps in the case that $\gamma = 1$.

▶ **Theorem 3.** *Let $d, \gamma > 0$. For any constant $\theta > 0$, there is an algorithm such that, whp over the choice of $G \sim G(n, d/n)$, when the algorithm is given as input the graph G and an arbitrary rational $\varepsilon > 0$ outputs an ε-sample from $\mu_{G,\gamma}$ in time $n^{1+\theta} \log \frac{1}{\varepsilon}$.*

In the next section, we give the main ingredients of our algorithm for the hard-core model and we give the proof of Theorem 1. The proofs of Theorems 2 and 3 build on similar ideas, though there are some modifications needed to obtain the required spectral independence bounds. We give their proofs in Section B.3 of the full version.

Before proceeding let us finally mention that, to go beyond the 2-spin models studied here, the main obstacle is to establish the spectral independence bounds for graphs with potentially unbounded degrees. As it is pointed out in [24, Section 7], their correlation-decay analysis in terms of the connective constant using L^p-norms does not extend to other models in a straightforward manner, and hence it is natural to expect that the same is true for spectral independence as well.

2 Proof outline for Theorem 1

Our algorithm for sampling from the hard-core model on a graph $G = (V, E)$ is an adaptation of Glauber dynamics on an appropriate set of "small-degree" vertices U, the details of the algorithm are given in Figure 1. Henceforth, analogously to the Ising model, it will be convenient to view the hard-core model as a 2-spin model supported on $\Omega \subseteq \{0,1\}^V$, where Ω corresponds to the set of independent sets of G (for an independent set I, we obtain $\sigma \in \{0,1\}^V$ by setting $\sigma_v = 1$ iff $v \in I$).

Note that for general graphs G, implementing Steps 2 and Steps 3 of the algorithm might be difficult. The following lemma exploits the sparse structure of $G(n, d/n)$ and in particular the fact that high-degree vertices are sparsely scattered. We will use this in the proof of our main theorems to show that the algorithm $\textsc{Sample}(G, T)$ can be implemented very efficiently for appropriate D, paying only $O(\log n)$ per loop operation in Step 2 and only $O(n \log n)$ in Step 3. The *tree-excess* of a graph $G = (V, E)$ is defined as $|E| - |V| + 1$.

▶ **Lemma 4.** *Let $d > 0$ be an arbitrary real. There exist constants $D, \ell > 0$ such that the following holds whp over the choice of $G = (V, E) \sim G(n, d/n)$. Each of the connected components of $G[V \setminus U]$, where U is the set of vertices of degree $\leq D$, has size $O(\log n)$ and tree-excess at most ℓ.*

Algorithm SAMPLE(G, T).

Parameters: $D > 0$ (threshold for small/high degree vertices).

Input: Graph $G = (V, E)$, integer $T \geq 1$ (number of iterations).

1. **Initialisation:** Let U be the set of all vertices with degree $\leq D$.
   Let X_0 be the empty independent set on U.
2. **Main loop:** For $t = 1, \ldots, T$,
   - Pick a vertex u uniformly at random from U.
   - For every vertex $v \in U \backslash \{u\}$, set $X_t(v) = X_{t-1}(v)$.
   - Sample the spin $X_t(u)$ according to $\mu_{G,\lambda}\big(\sigma_u \mid \sigma_{U\backslash\{u\}} = X_t(U\backslash\{u\})\big)$, i.e., update u according to the hard-core distribution on the *whole graph* G, conditioned on the spins of $U\backslash\{u\}$.
3. **Finalisation:** Sample $\sigma \sim \mu_{G,\lambda}\big(\cdot \,\big|\, \sigma_U = X_T\big)$, i.e., extend X_T to the whole vertex set of G by sampling from $\mu_{G,\lambda}$ conditioned on the configuration on U.

Figure 1 The SAMPLE(G, T) subroutine for sampling from the hard-core distribution $\mu_{G,\lambda}$. We use the analogue of this algorithm for the Ising model with parameter β (replacing $\mu_{G,\lambda}$ by $\mu_{G,\beta}$). For the monomer-dimer model, the only difference is that the algorithm needs to update (single) edges in F, where F is the set of vertices whose both endpoints lie in U (i.e., degree $\leq D$).

Lemma 4 follows using relatively standard techniques from random graphs and is proved in Section C of the full version. Later, we will establish a more refined version of this property that will allow us to bound the mixing time of the single-site dynamics that we consider (the main loop of SAMPLE(G, T)).

The key ingredient needed to prove our main result is to show that the main loop of our sampling algorithm returns a good sample on the induced hard-core distribution on the set U. More precisely, for a graph $G = (V, E)$ and $U \subseteq V$, we let $\mu_{G,\lambda,U}(\cdot)$ denote the induced distribution on the spins of U, i.e., the marginal distribution $\mu_{G,\lambda}(\sigma_U = \cdot)$.

▶ **Lemma 5.** *Let $d, \lambda > 0$ be constants such that $\lambda < \lambda_c(d)$. For any arbitrarily small constant $\theta > 0$, there is $D > 0$ such that the following holds whp over the choice of $G \sim G(n, d/n)$.*

Let U be the set of vertices in G of degree $\leq D$. Then, for any $\varepsilon > 0$, for $T = \lceil n^{1+\theta/2} \log \frac{1}{\varepsilon} \rceil$, the main loop of SAMPLE(G, T) returns a sample X_T from a distribution which is ε-close to $\mu_{G,\lambda,U}$.

We will prove Lemma 5 in Section 4.2. With these two lemmas we are ready to prove Theorem 1.

Proof of Theorem 1. We give first the details for the more interesting case $d \geq 1$. Consider arbitrarily small $\theta > 0$ and D, ℓ as in Lemmas 4 and 5, so that whp G satisfies the properties therein. Let $\varepsilon > 0$ be the desired accuracy for sampling from $\mu_{G,\lambda}$; it is sufficient to consider $\varepsilon < 1/\mathrm{e}$. Let U be the set of vertices with degree $\leq D$, and set $T = \lceil n^{1+\theta/2} \log \frac{1}{\varepsilon} \rceil$.

By Lemma 5, whp over the choice of G, the main loop of SAMPLE(G, T) returns a configuration $X_T : U \to \{0, 1\}$ that is ε-close to $\mu_{G,\lambda,U}$. Note that each iteration of the main loop of SAMPLE(G, T) can be implemented in $O(\log n)$ time since $G[V\backslash U]$ has components of size $O(\log n)$ and tree excess at most ℓ. In particular, any vertex $u \in U$ can be adjacent to at most D of these components, and therefore the component of u in $G[(V\backslash U) \cup \{u\}]$ has

size $O(\log n)$ and tree excess at most $k = D\lceil \ell \rceil = O(1)$. We can therefore sample the spin of u under $\mu_{G,\lambda}$ conditioned on the spins of $U\backslash\{u\}$ in time $O(4^k \log n) = O(\log n)$.[2] Therefore, the main loop of $\textsc{Sample}(G,T)$ runs in time $O(T \log n)$. Analogously, the finalisation step of $\textsc{Sample}(G,T)$, i.e., extending the configuration X_T on U to a configuration σ on the whole vertex set V, can be implemented in time $O(n \log n)$ by iterating over the vertices in $V\backslash U$ and using the fact that the components of $G[V\backslash U]$ have excess at most ℓ. Therefore, the overall running time of the algorithm is bounded by $O(T \log n) + O(n \log n)$, which is less than $\lceil n^{1+\theta} \log \frac{1}{\varepsilon} \rceil$ for all sufficiently large n. It remains to note that, since X_T is ε-close to the marginal distribution of $\mu_{G,\lambda}$ on U, and the finalisation step is done perfectly conditioned on the configuration on U, the final configuration σ is ε-close to the distribution $\mu_{G,\lambda}$.

For $d < 1$, whp G consists of tree-like components of size $O(\log n)$, and therefore we can obtain a perfect sample from $\mu_{G,\lambda}$ in time $O(n \log n)$ by going through the vertices one by one and, for each vertex, taking $O(\log n)$ time to compute its marginal, conditioned on the spins already sampled. ◀

3 Spectral independence via branching values

We first introduce the notions of spectral independence and pairwise vertex influences, which we will later use to bound the mixing time of the main loop of $\textsc{Sample}(G,T)$, i.e., to prove Lemma 5. We will define the terminology in a general way that will be useful both for our analysis of the hard-core model, and for our later analysis of other models.

Let $q \geq 2$ be an integer indicating the number of spins and let V be a set of size n. We will consider distributions μ supported on a set $\Omega \subseteq [q]^V$.[3] For $S \subseteq V$, let $\Omega_S = \{\tau \in [q]^S \mid \mu(\sigma_S = \tau) > 0\}$ be the set of all partial configurations on $[q]^S$ that have non-zero marginal under μ. For $\tau \in \Omega_S$, let μ_τ be the conditional distribution on Ω induced by τ, i.e., $\mu_\tau(\cdot) = \mu(\cdot \mid \sigma_S = \tau)$. Let $\mu_{\min} = \min_{\sigma\in\Omega} \mu(\sigma)$.

For $S \subseteq V$ and $\tau \in \Omega_S$, the *influence matrix* conditioned on τ is the matrix $\mathbf{\Psi}_\tau$ whose rows and columns are indexed by $\tilde{V}_\tau = \{(v,i) \mid v \in V\backslash S,\ \mu_\tau(\sigma_v = i) > 0\}$, where the entry indexed by $(v,i),(w,k)$ equals $\mu_\tau(\sigma_w = k \mid \sigma_v = i) - \mu_\tau(\sigma_w = k)$ if $v \neq w$, and 0 otherwise. It is a standard fact that the eigenvalues of the matrix $\mathbf{\Psi}$ are all real ([2]), and we denote by $\lambda_1(\mathbf{\Psi})$ its largest eigenvalue.

▶ **Definition 6.** *Let $q \geq 2$ be an integer and V be a set of size $n \geq 2$. Let μ be a distribution supported over $\Omega \subseteq [q]^V$. Let $\eta, b > 0$. We say that μ is* η-spectrally independent *if for all $S \subset V$ and $\tau \in \Omega_S$, it holds that $\lambda_1(\mathbf{\Psi}_\tau) \leq \eta$. We say that μ is* b-marginally bounded *if for all $S \subset V$, $v \in V\backslash S$, $\tau \in \Omega_S$, and $i \in [q]$, it either holds that $\mu_\tau(\sigma_v = i) = 0$ or else $\mu_\tau(\sigma_v = i) \geq b$.*

Following [2, 7], for distributions μ induced by 2-spin systems, we work with the following notion of pairwise vertex-influence, which can be used to bound the spectral independence. For a graph $G = (V,E)$ and $\tau \in \{0,1\}^S$ for some $S \subset V$, for vertices u, v with $u \in V\backslash S$ and $0 < \mu_\tau(\sigma_u = 1) < 1$, we define the *influence* of u on v (under μ_τ) as

$$\mathcal{I}^\tau_G(u \to v) = \mu_\tau(\sigma_v = 1 \mid \sigma_u = 1) - \mu_\tau(\sigma_v = 1 \mid \sigma_u = 0).$$

[2] One "naive" way to do this is by considering a spanning tree and then brute-forcing over all $\leq 4^k$ possibilities for the endpoints of the excess edges (the spins on each edge can be set in at most 4 ways). For each of these, the marginal probability at u and the corresponding partition function can be computed using dynamic programming on the left-over tree.

[3] For an integer $k \geq 1$, we denote by $[k]$ the set $\{0, 1, \ldots, k-1\}$.

For matchings, we will work with an analogous notion from the perspective of edges (see Section B.2 of the full version). For all these models, spectral independence will be bounded by summing the absolute value of the influences of an arbitrary vertex u to the rest of the graph.

In turn, it has been shown in [7] that summing the influences of a vertex u in a graph G reduces to summing the sum of influences on the self-avoiding walk tree emanating from u, see Lemma 22 in the full version. Therefore, we only need to focus on trees arising as self-avoiding walk trees.

3.1 The branching value

We will need the following notion to capture the growth of the self-avoiding walk tree from a vertex.

▶ **Definition 7.** *Let $d > 0$ be a real number and $G = (V, E)$ be a graph. For a vertex v in G, the d-branching value S_v equals $\sum_{\ell \geq 0} N_{v,\ell}/d^\ell$, where $N_{v,\ell}$ is the number of (simple) paths with a total of $\ell + 1$ vertices starting from v (for convenience, we set $N_{v,0} = 1$).*

We will show the following lemma in Section C.1 which bounds the d'-branching value of $G(n, d/n)$ for any $d' > d$.

▶ **Lemma 8.** *Let $d \geq 1$. Then, for every $d' > d$ and $\varepsilon > 0$, whp over the choice of $G \sim G(n, d/n)$, the d'-branching value of every vertex in G is at most $\varepsilon \log n$.*

3.2 Spectral independence for the hard-core model

In this section, we bound the spectral independence of $G(n, d/n)$ in the hard-core model when $\lambda < \lambda_c(d)$. We will need the following technical lemma that can be derived from [24]. The derivation details are similar to an analogous lemma for matchings (cf. Lemma 26 in the full version), which can be found in [3, Lemma 15].

▶ **Lemma 9** ([24])**.** *Let $d > 1$ and $\lambda > 0$ be constants such that $\lambda < \lambda_c(d)$. Let $\chi \in (1, 2)$ be given from $\frac{1}{\chi} = 1 - \frac{d-1}{2} \log\left(1 + \frac{1}{d-1}\right)$ and set $a = \frac{\chi}{\chi - 1}$. Consider also the function $\Phi(y) = \frac{1}{\sqrt{y(1+y)}}$ for $y > 0$. Then, there is a constant $0 < \kappa < 1/d$ such that the following holds for any integer $k \geq 1$.*

Let $x_1, \ldots, x_k$ be reals and $x = \lambda \prod_{i=1}^{k} \frac{1}{1+x_i}$. Then $(\Phi(x))^a \sum_{i=1}^{k} \left(\frac{x}{(1+x_i)\Phi(x_i)}\right)^a \leq \kappa^{a/\chi}$.

We will show the following.

▶ **Lemma 10.** *Let $d > 1$ and $\lambda > 0$ be constants such that $\lambda < \lambda_c(d)$. Then, there is a constant $\chi > 1$ such that the following holds.*

Let $T = (V, E)$ be a tree rooted at ρ, whose d-branching value is $\leq \alpha$ and whose root has k children. Then, for the hard-core distribution on T with parameter λ, any $S \subseteq V \backslash \{\rho\}$ and $\tau \in \Omega_S$ with $0 < \mu_\tau(\sigma_\rho = 1) < 1$, it holds that

$$\sum_{v \in V} \left|\mathcal{I}_T^\tau(\rho \to v)\right| \leq W_k \alpha^{1/\chi},$$

where $W_k > 0$ is a real depending only on the degree k of the root (and the constants d, λ).

Proof. Let $\kappa \in (0, 1/d)$ and $\chi \in (1, 2)$ be the constants from Lemma 9, and $\Phi(x) = \frac{1}{\sqrt{x(1+x)}}$ be also as in Lemma 9.

We may assume without loss of generality that S is empty (and τ is trivial) by truncating the tree T using the following procedure: just remove vertices $u \in S$ with $\tau_u = 0$, and for $u \in S$ with $\tau_u = 1$ remove u and all of its neighbours. Note that for all the removed vertices v it holds that $\mathcal{I}_T^\tau(\rho \to v) = 0$, so the removal procedure does not decrease the sum of the absolute influences, while at the same time decreasing the d-branching value of the tree T. Henceforth, we will drop τ and S from notation.

To prove the lemma, we will work inductively on the depth of the tree. To this end, we first define for each vertex u in T the following values α_u and R_u; the α's capture a rooted analogue of the branching value of internal vertices within T, while the R's the marginals of the vertices in the corresponding subtrees. More precisely, if v is a leaf, set $\alpha_v = 1$ and $R_v = \lambda$; otherwise set $\alpha_v = 1 + \frac{1}{d}\sum_{i=1}^k \alpha_{v_i}$ and $R_v = \lambda \prod_{i=1}^k \frac{1}{1+R_{v_i}}$, where $v_1, \dots, v_k$ are the children of v. Note that for the root ρ we have that $\alpha_\rho = S_\rho \le \alpha$, where S_ρ is the d-branching value of ρ in the tree T. Moreover, if we denote by T_v the subtree of T rooted at v and by u the parent of v in T, then it holds that

$$R_v = \frac{\mu_{T_v,\lambda}(\sigma_v = 1)}{\mu_{T_v,\lambda}(\sigma_v = 0)} \quad \text{and} \quad \mathcal{I}_T(u \to v) = -\frac{R_v}{R_v + 1}. \tag{1}$$

The first equality is fairly standard and can be proved using induction on the height of the tree, while the second one is [7, Lemma 15] (it also follows directly from the definition of influence and the first equality).

For an integer $h \ge 0$, let $L(h)$ be the nodes at distance h from the root ρ. Let $M_k = \sqrt{1 + (1+\lambda)^k/\lambda}$, where recall that k is the degree of the root ρ. We will show that

$$\sum_{v \in L(h)} \left(\frac{\alpha_v}{\alpha_\rho}\right)^{1/\chi} \frac{\left|\mathcal{I}_T(\rho \to v)\right|}{R_v \Phi(R_v)} \le M_k (d\kappa)^{h/\chi}. \tag{2}$$

Since $\alpha_v \ge 1$ for $v \in V$, $\alpha_\rho \le \alpha$ and $R_v\Phi(R_v) \le 1$, (2) yields $\sum_{v \in L(h)} \left|\mathcal{I}_T(\rho \to v)\right| \le M_k \alpha^{1/\chi} (d\kappa)^{h/\chi}$ for all integer $h \ge 0$, and therefore summing over h, we obtain that

$$\sum_{v \in V} \left|\mathcal{I}_T(\rho \to v)\right| \le M_k \alpha^{1/\chi} \sum_{h \ge 0} (d\kappa)^{h/\chi} \le \frac{M_k \alpha^{1/\chi}}{1 - (d\kappa)^{1/\chi}},$$

which proves the result with $W_k = \frac{M_k}{1-(d\kappa)^{1/\chi}}$. So it only remains to prove (2).

We will work inductively. The base case $h = 0$ is equivalent to $M_k \ge 1/(R_\rho \Phi(R_\rho)) = \sqrt{1 + 1/R_\rho}$, which is true since from the recursion for R_ρ we have that $R_\rho \ge \lambda/(1+\lambda)^k$. For the induction step, consider $v \in L(h-1)$ and suppose it has $k_v \ge 0$ children, denoted by v_i for $i \in [k_v]$. Then, for each $i \in [k_v]$, since v is on the unique path joining ρ to v_i, it holds that (see [2, Lemma B.2])

$$\mathcal{I}_T(\rho \to v_i) = \mathcal{I}_T(\rho \to v)\mathcal{I}_T(v \to v_i),$$

so we can write

$$\begin{aligned}\sum_{v \in L(h)} \left(\frac{\alpha_v}{\alpha_\rho}\right)^{1/\chi} \frac{\left|\mathcal{I}_T(\rho \to v)\right|}{R_v \Phi(R_v)} = \\ \sum_{v \in L(h-1)} \left(\frac{\alpha_v}{\alpha_\rho}\right)^{1/\chi} \frac{\left|\mathcal{I}_T(\rho \to v)\right|}{R_v \Phi(R_v)} \sum_{i \in [k_v]} \left(\frac{\alpha_{v_i}}{\alpha_v}\right)^{1/\chi} R_v \Phi(R_v) \frac{\left|\mathcal{I}_T(v \to v_i)\right|}{R_{v_i}\Phi(R_{v_i})}.\end{aligned} \tag{3}$$

Consider an arbitrary $v \in L(h-1)$. Then, since $\frac{1}{\chi} + \frac{1}{a} = 1$, by Hölder's inequality we have that

$$\sum_{i\in[k_v]} \left(\frac{\alpha_{v_i}}{\alpha_v}\right)^{1/\chi} R_v\Phi(R_v)\frac{|\mathcal{I}_T(v \to v_i)|}{R_{v_i}\Phi(R_{v_i})} \leq \left(\sum_{i\in[k_v]} \frac{\alpha_{v_i}}{\alpha_v}\right)^{1/\chi} \left((R_v\Phi(R_v))^a \sum_{i\in[k_v]} \left(\frac{|\mathcal{I}_T(v \to v_i)|}{R_{v_i}\Phi(R_{v_i})}\right)^a\right)^{1/a}. \tag{4}$$

Note that for $x = R_v$ and $x_i = R_{v_i}$, $i \in [k_v]$, we have from (1) that $\frac{|\mathcal{I}_T(v\to v_i)|}{R_{v_i}} = \frac{1}{1+x_i}$ and $x = \lambda\prod_{i\in[k_v]} \frac{1}{1+x_i}$, so by Lemma 9 we have that

$$\left((R_v\Phi(R_v))^a \sum_{i\in[k_v]} \left(\frac{|\mathcal{I}_T(v \to v_i)|}{R_{v_i}\Phi(R_{v_i})}\right)^a\right)^{1/a} \leq \kappa^{1/\chi}.$$

By definition of the d-branching value we also have $\alpha_v = 1 + \frac{1}{d}\sum_{i\in[k_v]} \alpha_{v_i} \geq \frac{1}{d}\sum_{i\in[k_v]} \alpha_{v_i}$, so plugging these back into (4) yields

$$\sum_{i\in[k_v]} \left(\frac{\alpha_{v_i}}{\alpha_v}\right)^{1/\chi} R_v\Phi(R_v)\frac{|\mathcal{I}_T(v \to v_i)|}{R_{v_i}\Phi(R_{v_i})} \leq (d\kappa)^{1/\chi}.$$

In turn, plugging this into (3) and using the induction hypothesis yields (2), finishing the proof. ◀

▶ Remark 11. For simplicity, and since it is not important for our arguments, the constant W_k in the proof depends exponentially on the degree k of the root. With a more careful inductive proof (cf. [7, Proof of Lemma 14]), the dependence on k can be made linear. In either case, because of the high-degree vertices in $G(n, d/n)$, both bounds do not yield sufficiently strong bounds on the spectral independence of the whole distribution $\mu_{G,\lambda}$, and this is one of the reasons that we have to consider the spectral independence on the induced distribution on low-degree vertices.

Recall that for a graph $G = (V, E)$ and $U \subseteq V$, we let $\mu_{G,\lambda,U}(\cdot)$ denote the marginal distribution on the spins of U, i.e., the distribution $\mu_{G,\lambda}(\sigma_U = \cdot)$.

▶ **Lemma 12.** *Let $d \geq 1$ and $\lambda > 0$ be constants such that $\lambda < \lambda_c(d)$. Then, for any constants $D, \varepsilon > 0$, whp over the choice of $G \sim G(n, d/n)$, the marginal hard-core distribution $\mu_{G,\lambda,U}$, where U is the set of vertices in G with degree $\leq D$, is $(\varepsilon \log n)$-spectrally independent.*

Proof. Let $D, \varepsilon > 0$ be arbitrary constants, and let $d' > d$ be such that $\lambda < \lambda_c(d')$; such d' exists because the function $\lambda_c(\cdot)$ is continuous in the interval $(1, \infty)$ and $\lambda_c(d) \to \infty$ for $d \downarrow 1$. Let $\chi \in (1, 2)$ and $W = \max\{W_1, \ldots, W_D\}$ where χ and the W_k's are as in Lemma 10 (corresponding to the constants d', λ). By Lemma 8, whp all of the vertices of the graph $G = (V, E) \sim G(n, d/n)$ have d'-branching value less than $\varepsilon \log n$. We will show that the result holds for all such graphs G (for sufficiently large n).

Let U be the set of vertices in G with degree $\leq D$, and let for convenience $\mu = \mu_{G,\lambda,U}$. Consider arbitrary $S \subset U$ and $\tau \in \Omega_S$. It suffices to bound the largest eigenvalue of the influence matrix $\mathbf{\Psi}_\tau$ by $\varepsilon \log n$. Analogously to [2, 7], we do this by bounding the absolute-value row sums of $\mathbf{\Psi}_\tau$. Recall that the rows and columns of $\mathbf{\Psi}_\tau$ are indexed by $\tilde{V}_\tau = \{(v, i) \mid v \in U\backslash S, \mu_\tau(\sigma_v = i) > 0\}$, where the entry indexed by $(v, i), (w, k)$ equals $\mu_\tau(\sigma_w = k \mid \sigma_v = i) - \mu_\tau(\sigma_w = k)$ if $v \neq w$, and 0 otherwise. Consider arbitrary $(v, i) \in \tilde{V}_\tau$; our goal is to show

$$\sum_{(w,k)\in\tilde{V}_\tau} \big|\mu_\tau(\sigma_w = k \mid \sigma_v = i) - \mu_\tau(\sigma_w = k)\big| \le \varepsilon \log n. \tag{5}$$

Henceforth, we will also assume that $\mu_\tau(\sigma_v = i) < 1$ (in addition to $\mu_\tau(\sigma_v = i) > 0$), otherwise the sum on the l.h.s. is equal to 0. Then, by the law of total probability, for any $(w,k) \in \tilde{V}_\tau$ we have

$$\begin{aligned}\big|\mu_\tau(\sigma_w = k \mid \sigma_v = i) - \mu_\tau(\sigma_w = k)\big| &\le \big|\mu_\tau(\sigma_w = k \mid \sigma_v = 1) - \mu_\tau(\sigma_w = k \mid \sigma_v = 0)\big| \\ &= \big|\mathcal{I}^\tau_G(v \to w)\big|,\end{aligned}$$

where the last equality follows from the fact that μ is the marginal distribution of $\mu_{G,\lambda}$ on U. Therefore, we can bound

$$\sum_{(w,k)\in\tilde{V}_\tau} \big|\mu_\tau(\sigma_w = k \mid \sigma_v = i) - \mu_\tau(\sigma_w = k)\big| \le \sum_{w\in U} \big|\mathcal{I}^\tau_G(v \to w)\big| \le \sum_{w\in V} \big|\mathcal{I}^\tau_G(v \to w)\big|.$$

By Lemma 22 in the full version, for the self-avoiding walk tree $T = (V_T, E_T)$ from v, there is a subset $Z \subseteq V_T \backslash \{\rho\}$ and a configuration $\phi \in \{0,1\}^Z$ such that

$$\sum_{w\in V} \big|\mathcal{I}^\tau_G(v \to w)\big| \le \sum_{w\in V_T} \big|\mathcal{I}^\phi_T(v \to w)\big|,$$

where $\mathcal{I}^\phi_T(v \to \cdot)$ denotes the influence of v on the vertices of T (in the hard-core distribution $\mu_{T,\lambda}$ conditioned on ϕ). Since the d'-branching value of v (and any other vertex of G) is bounded by $\varepsilon \log n$ and the degree of v is $\le D$, by Lemma 10 applied to T, we have that

$$\sum_{w\in V_T} \big|\mathcal{I}^\phi_T(v \to w)\big| \le W(\varepsilon \log n)^{1/\chi}.$$

Since $\chi > 1$, for all sufficiently large n we have that $W(\varepsilon \log n)^{1/\chi} \le \varepsilon \log n$, which proves (5). ◀

We also record the following corollary of the arguments in Lemma 10.

▶ **Corollary 13.** *Let $\lambda > 0$ and $D > 0$ be real numbers. For a graph $G = (V, E)$, let U be the set of vertices in G with degree $\le D$ and suppose that $|U| \ge 2$. Then, the distribution $\mu := \mu_{G,\lambda,U}$ is b-marginally bounded for $b = \frac{\lambda}{\lambda + (1+\lambda)^D}$.*

Proof. By Lemma 22 in the full version, for any vertex $v \in U$ and any boundary condition τ on (a subset of) $U\backslash\{v\}$, there is a corresponding tree T and a boundary condition ϕ on T such that $\mu_\tau(\sigma_v = \cdot) = \nu_\phi(\sigma_v = \cdot)$. Since v has degree $\le D$, from the proof of Lemma 10, see in particular equation (1), we have that $\nu_\phi(\sigma_v = \cdot) \ge b$, where b is as in the lemma statement. ◀

4 Entropy factorisation for bounded-degree vertices

In this section, we show how to convert the spectral independence results of the previous section into fast mixing results for Glauber dynamics on the set of small-degree vertices on $G(n, d/n)$. Our strategy here follows the technique of [8], though to obtain nearly linear results we have to pay attention to the connected components induced by high-degree vertices and how these can connect up small-degree vertices.

4.1 Preliminaries

Entropy factorisation for probability distributions

For a real function f on $\Omega \subseteq [q]^V$, we use $\mathbf{E}_\mu(f)$ for the expectation of f with respect to μ and, for $f : \Omega \to \mathbb{R}_{\geq 0}$, $\mathrm{Ent}_\mu(f) = \mathbf{E}_\mu[f \log f] - \mathbf{E}_\mu(f) \log \mathbf{E}_\mu(f)$, with the convention that $0 \log 0 = 0$. Finally, for $S \subset V$, let $\mathrm{Ent}^S_\mu(f) = \mathbf{E}_{\tau \sim \mu_{V\setminus S}}\big[\mathrm{Ent}_{\mu_\tau}(f)\big]$ i.e., $\mathrm{Ent}^S_\mu(f)$ is the expected value of the conditional entropy of f when the assignment outside of S is chosen according to the marginal distribution $\mu_{V\setminus S}$ (the induced distribution of μ on $V\setminus S$). For convenience, when $S = V$, we define $\mathrm{Ent}^S_\mu(f) = \mathrm{Ent}_\mu(f)$. The following inequality of entropy under tensor product is a special case of Shearer's inequalities.

▶ **Fact 14.** *Let $q, k \geq 2$ be integers and suppose that, for $i \in [k]$, μ_i is a distribution supported over $\Omega_i \subseteq [q]^{V_i}$, where $V_1, \ldots, V_k$ are pairwise disjoint sets. Let $\mu = \mu_1 \otimes \cdots \otimes \mu_k$ be the product distribution on $\Omega = \Omega_1 \times \cdots \times \Omega_k$. Then, for any $f : \Omega \to \mathbb{R}_{\geq 0}$, it holds that* $\mathrm{Ent}_\mu(f) \leq \sum_{i=1}^k \mathrm{Ent}^{V_i}_\mu(f)$.

To bound the mixing time of Markov chains such as the Glauber dynamics, we will be interested in establishing inequalities for factorisation of entropy, defined as follows.

▶ **Definition 15.** *Let $q \geq 2$, $r \geq 1$ be integers and V be a set of size $n \geq r + 1$. Let μ be a distribution supported over $\Omega \subseteq [q]^V$. We say that μ satisfies the r-uniform-block factorisation of entropy with multiplier*[4] *C_r if for all $f : \Omega \to \mathbb{R}_{\geq 0}$ it holds that* $\frac{r}{n}\mathrm{Ent}_\mu(f) \leq C_r \frac{1}{\binom{n}{r}} \sum_{S \in \binom{V}{r}} \mathrm{Ent}^S_\mu(f)$.

The following lemma will be useful to bound the (r-uniform-block) factorisation multiplier for conditional distributions on sets with small cardinality.

▶ **Lemma 16** ([8, Lemma 4.2])**.** *Let $q \geq 2$ be an integer and V be a set of size $n \geq 2$. Let μ be a distribution supported over $\Omega \subseteq [q]^V$ which is b-marginally bounded for some $b > 0$. Then, for any $S \subseteq V$ and $\tau \in \Omega_{V\setminus S}$, for $f : \Omega \to \mathbb{R}_{\geq 0}$, it holds that* $\mathrm{Ent}_{\mu_\tau}(f) \leq \frac{2|S|^2 \log(1/b)}{b^{2|S|+2}} \sum_{v \in S} \mathrm{Ent}^v_{\mu_\tau}(f)$.

The r-uniform-block Glauber dynamics and its mixing time

For an integer $r = 1, \ldots, n$, the r-uniform-block Glauber dynamics for μ is a Markov chain $(X_t)_{t \geq 0}$ where $X_0 \in \Omega$ is an arbitrary configuration and, for $t \geq 1$, X_t is obtained from X_{t-1} by first picking a subset $S \in V$ of size r uniformly at random and updating the configuration on S according to

$$\mu\big(\sigma_S = \cdot \mid \sigma_{V\setminus S} = X_{t-1}(V\setminus S)\big).$$

For $\varepsilon > 0$, the mixing time of the r-uniform-block Glauber dynamics is defined as $T_{\mathrm{mix}}(\varepsilon) = \max_{\sigma \in \Omega} \min\big\{t \,\big|\, X_0 = \sigma,\ \|\nu_t - \mu\|_{\mathrm{TV}} \leq \varepsilon\big\}$, where ν_t denotes the distribution of X_t. Note, the case $r = 1$ corresponds to the single-site dynamics, where at every step the spin of a single vertex, chosen u.a.r., is updated conditioned on the spins of the remaining vertices.

[4] We note that in related works C_r is usually referred to as the "factorisation constant"; we deviate from this terminology since for us C_r will depend on n (cf. Corollary 19 and Lemma 21), and referring to it as a constant could cause confusion.

▶ **Lemma 17** (See, e.g., [8, Lemma 2.6 & Fact 3.5(4)] or [6, Lemma 3.2.6 & Fact 3.4.2])**.** *Let $q \geq 2$, $r \geq 1$ be integers and V be a set of size $n \geq r+1$. Let μ be a distribution supported over $\Omega \subseteq [q]^V$ that satisfies the r-uniform-block factorisation of entropy with multiplier C_r. Then, for any $\varepsilon > 0$, the mixing time of the r-uniform-block Glauber dynamics on μ satisfies*

$$T_{\mathrm{mix}}(\varepsilon) \leq \left\lceil C_r \frac{n}{r}\Big(\log\log\frac{1}{\mu_{\min}} + \log\frac{1}{2\varepsilon^2}\Big)\right\rceil, \text{ where } \mu_{\min} = \min_{\sigma\in\Omega}\mu(\sigma).$$

We remark that to deduce the lemma from [8] or [6], which refer to the so-called "entropy decay constant κ", one needs to use the equality $C_r\kappa = r/n$ from [8, Lemma 2.6] or [6, Lemma 3.2.6].

From spectral independence to r-uniform-block factorisation multipliers

The following theorem is shown in [8]; while the version that we state here cannot be found verbatim in [8], we explain in the appendix how to combine the results therein to obtain it.

▶ **Theorem 18** ([8])**.** *Let $q \geq 2$ be an integer and V be a set of size $n \geq 2$. Let μ be a distribution supported over $\Omega \subseteq [q]^V$ that is η-spectrally independent and b-marginally bounded for $\eta, b > 0$.*

Then, for all integers $r = 1, \ldots, n$, the distribution μ satisfies the r-uniform-block factorisation of entropy with multiplier $C_r = \frac{r}{n}\frac{\sum_{k=0}^{n-1}\Gamma_k}{\sum_{k=n-r}^{n-1}\Gamma_k}$, where $\Gamma_k = \prod_{j=0}^{k-1}\alpha_j$ for $k \in [n]$[5] and $\alpha_k = \max\left\{0, 1 - \frac{4\eta}{b^2(n-1-k)}\right\}$ for $k \in [n-1]$.

4.2 Entropy factorisation for bounded-degree vertices in the hard-core model

The first step of the analysis of Glauber dynamics for the hard-core model on the set of small-degree vertices will be to employ spectral independence results of Section 3.2 to conclude fast mixing for the r-uniform block Glauber dynamics for $r = \theta|U|$ for any arbitrarily small constant θ. This step will follow by applying the recent technology of entropy factorisation described above.

The second step is the more challenging for us. Here we need to conclude fast mixing for $r = 1$, and in particular prove that $C_1/C_r = n^{o(1)}$. This is done roughly by studying the connected components of G that arise when resampling an r-subset of the low-degree vertices; the factorisation multiplier of these components controls the ratio C_1/C_r. While this resembles the approach of [8], there is a key difference here, in that high-degree vertices are not resampled. This can not only cause potentially large components, but also imposes a deterministic lower bound on components sizes (since a component consisting of high-degree vertices will be deterministically present in the percolated graph consisting of the r-subset of low-degree vertices and all of the high-degree vertices). This lower bound on the component sizes is actually more significant than it might initially seem since the relatively straightforward bound of $\Omega(\log n)$ would unfortunately give a relatively large factorisation multiplier of $n^{\Omega(1)}$ (through Lemma 16). Instead, we need to show that components have size $o(\log n)$, which in turn requires more delicate estimates for the distribution of high-degree vertices in connected sets (see Lemma 20 below).

[5] We note that for $k = 0$, the product defining Γ_k is empty and therefore $\Gamma_0 = 1$.

We start with the following corollary of Lemma 12, which converts a spectral independence bound into a bound on the factorisation multiplier for the r-uniform-block Glauber dynamics when r scales linearly with small-degree vertices. This is analogous to [8, Lemma 2.4], where they obtain a $2^{O(\eta/b^2)}$ bound on C_r when $r = \Theta(n)$ via Theorem 18 (where η is the spectral independence bound and b is the bound on the marginals). By restricting to small-degree vertices, we obtain that b is a constant, which combined with the bound $\eta = o(\log n)$ from Lemma 12 gives the bound $C_r = n^{o(1)}$, as detailed below. The proof of the corollary is given for completeness in Section D of the full version.

▶ **Corollary 19.** *Let $d \geq 1$ and $\lambda > 0$ be constants such that $\lambda < \lambda_c(d)$. Then, for any constants $D, \theta > 0$, whp over the choice of $G \sim G(n, d/n)$, the marginal hard-core distribution $\mu_{G,\lambda,U}$, where U is the set of vertices in G with degree $\leq D$, satisfies for any integer $r \in \left[\theta|U|, |U|\right]$ the r-uniform block factorisation of entropy with multiplier $C_r \leq n^{\theta}$.*

Note that the reason that we are able to use the same θ in the bounds for r and C_r is that the bound on C_r is loose (we can obtain a sharper result since we have a bound on the spectral independence of $\varepsilon \log n$ for any $\varepsilon > 0$).

We will now refine the bound of Corollary 19 down to $r = 1$ by exploiting the fact that high-degree vertices are sparsely scattered. In particular, we will need the following lemma which is a refinement of Lemma 4. For a graph $G = (V, E)$, we say that a set $S \subseteq V$ is connected if the induced subgraph $G[S]$ is connected.

▶ **Lemma 20.** *Let $d > 0$ be an arbitrary real. There exists an $L > 0$ such that for any $\delta \in (0, 1)$, the following holds whp over the choice of $G = (V, E) \sim G(n, d/n)$. For $\Delta = 1/(\delta \log \frac{1}{\delta})$, for all integers $k \geq \delta \log n$ and any $v \in V$, there are $\leq (2\mathrm{e})^{\Delta L k}$ connected sets $S \subseteq V$ containing v with $|S| = k$. Moreover, every such set contains $\geq k/2$ vertices with degree less than $L\Delta$.*

The proof of Lemma 20 is given in Section C.3 of the full version. We are now ready to show the following.

▶ **Lemma 21.** *Let $d \geq 1$ and $\lambda > 0$ be constants such that $\lambda < \lambda_c(d)$. For any $\theta > 0$, there is a constant $D > 0$ such that whp over the choice of $G \sim G(n, d/n)$, the marginal hard-core distribution $\mu_{G,\lambda,U}$, where U is the set of vertices in G with degree $\leq D$, satisfies the 1-uniform-block factorisation of entropy with multiplier $C_1 \leq n^{\theta}$.*

Proof. Let $L > 0$ be as in Lemma 20, and consider an arbitrarily small constant $\theta > 0$. Let $\delta \in (0, 1)$ be a sufficiently small constant so that for $D = L/(\delta \log \frac{1}{\delta})$ and $b = \frac{\lambda}{\lambda + (1+\lambda)^D}$ it holds that $\frac{1}{b^{2\delta}} < \mathrm{e}^{\theta/4}$; such a constant exists since $b^{2\delta} \to 1$ as $\delta \downarrow 0$. Let $\Delta = 1/(\delta \log \frac{1}{\delta})$ and $\zeta > 0$ be a small constant so that $2(2\mathrm{e})^{L\Delta}(2\zeta)^{1/2} \leq b^2/2$.

Let U be the vertices in G with degree $\leq D$, and let $r = \lfloor \zeta |U| \rfloor + 1$. Let $\mu = \mu_{G,\lambda,U}$. By Corollary 19, we have that whp over the choice of G, there is $C_r \leq n^{\theta/3}$ such that for every $f : \Omega \to \mathbb{R}_{\geq 0}$ it holds that

$$\frac{r}{|U|}\mathrm{Ent}_{\mu}(f) \leq C_r \frac{1}{\binom{|U|}{r}} \sum_{S \in \binom{U}{r}} \mathrm{Ent}^S_{\mu}(f). \tag{6}$$

For $S \subseteq U$, let $\mathcal{C}'(S)$ denote the collection of the connected components of the graph $G[S \cup (V \backslash U)]$, viewed as vertex sets, and let $\mathcal{C}(S) = \bigcup_{R \in \mathcal{C}'(S)} \{R \cap U\}$ be the restriction of these components to the set U. Note that, for $S \subseteq U$ and $\tau \in \Omega_{U \backslash S}$, μ_τ factorises over the components of $G[S \cup (V \backslash U)]$ and in particular, applying Fact 14, we have that

$$\mathrm{Ent}^S_{\mu}(f) = \mathbf{E}_{\tau \sim \mu_{U \backslash S}}\left[\mathrm{Ent}_{\mu_\tau}(f)\right] \leq \mathbf{E}_{\tau \sim \mu_{U \backslash S}}\left[\sum_{R \in \mathcal{C}(S)} \mathrm{Ent}^R_{\mu_\tau}(f)\right].$$

Using the bound in Lemma 16, we further obtain that

$$\begin{aligned}\mathrm{Ent}_\mu^S(f) &\le \mathbf{E}_{\tau\sim\mu_{U\setminus S}}\Big[\sum_{R\in\mathcal{C}(S)}\frac{2|R|^2\log(1/b)}{b^{2|R|+2}}\sum_{u\in R}\mathrm{Ent}_{\mu_\tau}^u(f)\Big]\\ &= \sum_{R\in\mathcal{C}(S)}\sum_{u\in R}\frac{2|R|^2\log(1/b)}{b^{2|R|+2}}\mathrm{Ent}_\mu^u(f),\end{aligned}$$

where the last equality follows by linearity of expectation and the fact that $\mathbf{E}_{\tau\sim\mu_{U\setminus S}}[\mathrm{Ent}_{\mu_\tau}^u(f)] = \mathrm{Ent}_\mu^u(f)$. Plugging this bound into (6), we obtain that

$$\mathrm{Ent}_\mu(f) \le \frac{2C_r\log(1/b)}{b^2\binom{|U|-1}{r-1}}\sum_{S\in\binom{U}{r}}\sum_{R\in\mathcal{C}(S)}\sum_{u\in U}\frac{|R|^2}{b^{2|R|}}\mathrm{Ent}_\mu^u(f).$$

which yields that

$$\mathrm{Ent}_\mu(f) \le \frac{2C_r\log(1/b)}{b^2}\sum_{u\in U}\mathrm{Ent}_\mu^u(f)\sum_{k=1}^{n}\frac{k^2}{b^{2k}}\Pr[\mathcal{C}_u(S)=k], \tag{7}$$

where $\Pr[\mathcal{C}_u(S)=k]$ denotes the probability that u belongs to a set of size k in the set $\mathcal{C}(S)$, when we pick S uniformly at random from $\{S\in\binom{U}{r} \mid u\in S\}$. Define analogously $\Pr[\mathcal{C}'_u(S)=k]$ to be the probability that u belongs to a connected component of size k in the set $\mathcal{C}'(S)$. By Lemma 20, whp over $G\sim G(n,d/n)$, for all vertices u and any integer $t\ge\delta\log n$, there are at most $(2\mathrm{e})^{L\Delta t}$ connected sets of size t containing a given vertex u, and each of them contains at least $t/2$ vertices from U. In particular, for any integer $k\ge\delta\log n$, it holds that $\Pr[\mathcal{C}_u(S)=k]\le\Pr[k\le\mathcal{C}'_u(S)\le 2k]$. For all $k\le 2|U|$, the probability that a specific subset of $k/2$ vertices of U is present in $G[S\cup(V\setminus U)]$ is at most $\frac{\binom{|U|-\lceil k/2\rceil}{r-\lceil k/2\rceil}}{\binom{|U|-1}{r-1}}\le\left(\frac{r}{|U|}\right)^{k/2}\le(2\zeta)^{k/2}$. Therefore, for all $k\ge\delta\log n$, by a union bound over the connected sets of size k, we have

$$\Pr[\mathcal{C}'_u(S)=k]\le(2\mathrm{e})^{L\Delta k}2^k(2\zeta)^{k/2} = \left(2(2\mathrm{e})^{L\Delta}(2\zeta)^{1/2}\right)^k\le(b^2/2)^k,$$

where in the first inequality the first factor is the number of size-k connected sets T of G containing u, the second factor is an upper bound on the number of size $k/2$ subsets W of U that might be included in T and the final factor is the probability that W is included in S. The last inequality is by the choice of ζ. It follows that

$$\Pr[\mathcal{C}_u(S)=k]\le\Pr[k\le\mathcal{C}'_u(S)\le 2k]\le 2k(b^2/2)^k.$$

From this bound and the inequality $1/b^{2\delta}<\mathrm{e}^{\theta/4}$ by the choice of δ, we can split and bound the rightmost sum in (7) by

$$\sum_{k=1}^{n}\frac{k^2}{b^{2k}}\Pr[\mathcal{C}_u(S)=k]\le\frac{(\delta\log n)^2}{b^{2\delta\log n}}+\sum_{k\ge\delta\log n}\frac{2k^3}{2^k}\le n^{\theta/3},$$

where the last inequality holds for all sufficiently large n. In turn, plugging this into (7), we obtain that μ satisfies the 1-uniform block factorisation of entropy with multiplier $C_1=\frac{2C_r\log(1/b)}{b^2}n^{\theta/3}\le n^\theta$ for all sufficiently large n (since b is a constant and $C_r\le n^{\theta/3}$), as needed. ◀

Lemma 5 now follows easily by combining Lemmas 17 and 21. This was the last ingredient needed in the proof of Theorem 1.

References

1 V. L. Alev and L. C. Lau. Improved analysis of higher order random walks and applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1198–1211, 2020.

2 N. Anari, K. Liu, and S. Oveis Gharan. Spectral independence in high-dimensional expanders and applications to the hardcore model. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1319–1330, 2020.

3 I. Bezáková, A. Galanis, L. A. Goldberg, and D. Štefankovič. The complexity of approximating the matching polynomial in the complex plane. *ACM Trans. Comput. Theory*, 13(2), 2021.

4 A. Blanca and R. Gheissari. Sampling from Potts on random graphs of unbounded degree via random-cluster dynamics. *CoRR*, abs/2107.10246, 2021.

5 X. Chen, W. Feng, Y. Yin, and X. Zhang. Rapid mixing of Glauber dynamics via spectral independence for all degrees. *CoRR*, abs/2105.15005, 2021. `arXiv:2105.15005`.

6 Z. Chen. *Optimal Mixing of Markov Chains for Spin Systems via Spectral Independence*. PhD thesis, Georgia Institute of Technology, 2021.

7 Z. Chen, K. Liu, and E. Vigoda. Rapid mixing of Glauber dynamics up to uniqueness via contraction. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1307–1318, 2020. Full version at `arXiv:2004.09083`.

8 Z. Chen, K. Liu, and E. Vigoda. Optimal mixing of Glauber dynamics: Entropy factorization via high-dimensional expansion. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1537–1550, 2021. Full version at arXiv:2011.02075.

9 I. Dinur and T. Kaufman. High dimensional expanders imply agreement expanders. In *58th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2017*, pages 974–985. IEEE Computer Soc., Los Alamitos, CA, 2017. `doi:10.1109/FOCS.2017.94`.

10 C. Efthymiou. MCMC sampling colourings and independent sets of $G(n, d/n)$ near uniqueness threshold. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 305–316, 2014.

11 C. Efthymiou, T. P. Hayes, D. Štefankovič, and E. Vigoda. Sampling random colorings of sparse random graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1759–1771, 2018.

12 A. Galanis, Q. Ge, D. Štefankovič, E. Vigoda, and L. Yang. Improved inapproximability results for counting independent sets in the hard-core model. *Random Struct. Algorithms*, 45(1):78–110, 2014.

13 A. Galanis, D. Štefankovič, and Eric Vigoda. Inapproximability of the partition function for the antiferromagnetic Ising and hard-core models. *Comb. Probab. Comput.*, 25(4):500–559, 2016.

14 V. Jain, H. T. Pham, and T. D. Vuong. Spectral independence, coupling with the stationary distribution, and the spectral gap of the Glauber dynamics. *CoRR*, abs/2105.01201, 2021. `arXiv:2105.01201`.

15 M. Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Springer Science & Business Media, 2003.

16 M. Jerrum and A.r Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.

17 T. Kaufman and D. Mass. High dimensional random walks and colorful expansion. In *8th Innovations in Theoretical Computer Science Conference*, volume 67 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 4, 27. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2017.

18 T. Kaufman and I. Oppenheim. High order random walks: beyond spectral gap. *Combinatorica*, 40(2):245–281, 2020. `doi:10.1007/s00493-019-3847-0`.

19 L.Li, P. Lu, and Y. Yin. Correlation decay up to uniqueness in spin dystems. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 67–84, 2013.

20 E. Mossel and A. Sly. Exact thresholds for Ising–Gibbs samplers on general graphs. *The Annals of Probability*, 41(1):294–328, 2013. doi:10.1214/11-AOP737.

21 I. Oppenheim. Local spectral expansion approach to high dimensional expanders Part I: Descent of spectral gaps. *Discrete Comput. Geom.*, 59(2):293–330, 2018. doi:10.1007/s00454-017-9948-x.

22 V. Patel and G. Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM J. Comput.*, 46(6):1893–1919, 2017.

23 H. Peters and G. Regts. On a conjecture of Sokal concerning roots of the independence polynomial. *Michigan Mathematical Journal*, 68(1):33–55, 2019.

24 A. Sinclair, P. Srivastava, D. Štefankovič, and Y. Yin. Spatial mixing and the connective constant: optimal bounds. *Probability Theory and Related Fields*, 168(1):153–197, 2017.

25 A. Sinclair, P. Srivastava, and M. Thurley. Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 941–953, 2012.

26 A. Sinclair, P. Srivastava, and Y. Yin. Spatial mixing and approximation algorithms for graphs with bounded connective constant. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 300–309. IEEE Computer Society, 2013.

27 A. Sly. Computational transition at the uniqueness threshold. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS '10*, pages 287–296, 2010.

28 A. Sly and N. Sun. The computational hardness of counting in two-spin models on d-regular graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 361–369, 2012.

29 D. Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC '06*, pages 140–149, 2006.

Deterministic Sensitivity Oracles for Diameter, Eccentricities and All Pairs Distances

Davide Bilò
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy

Keerti Choudhary
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Sarel Cohen
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Israel

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany

Martin Schirneck
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

We construct data structures for extremal and pairwise distances in directed graphs in the presence of transient edge failures. Henzinger et al. [ITCS 2017] initiated the study of fault-tolerant (sensitivity) oracles for the diameter and vertex eccentricities. We extend this with a special focus on space efficiency. We present several new data structures, among them the first fault-tolerant eccentricity oracle for dual failures in subcubic space. We further prove lower bounds that show limits to approximation vs. space and diameter vs. space trade-offs for fault-tolerant oracles. They highlight key differences between data structures for undirected and directed graphs.

Initially, our oracles are randomized leaning on a sampling technique frequently used in sensitivity analysis. Building on the work of Alon, Chechik, and Cohen [ICALP 2019] as well as Karthik and Parter [SODA 2021], we develop a hierarchical framework to derandomize fault-tolerant data structures. We first apply it to our own diameter and eccentricity oracles and then show its versatility by derandomizing algorithms from the literature: the distance sensitivity oracle of Ren [JCSS 2022] and the Single-Source Replacement Path algorithm of Chechik and Magen [ICALP 2020]. This way, we obtain the first deterministic distance sensitivity oracle with subcubic preprocessing time.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Pseudorandomness and derandomization; Mathematics of computing → Graph algorithms

Keywords and phrases derandomization, diameter, eccentricity, fault-tolerant data structure, sensitivity oracle, space lower bound

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.22

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `http://arxiv.org/abs/2204.10679`

1 Introduction

The problems of computing *shortest paths*, *graph diameter*, and *vertex eccentricities* are fundamental in many applications of both theoretical and applied computer science. We address these problems in the setting of *fault tolerance*. The interest in this problem setting stems from the fact that most real-world networks are prone to failures. These are unpredictable but usually small in numbers and transient due to some simultaneous repair process. However, in an error-prone network, it is not always practical to recompute distances

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 22; pp. 22:1–22:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

from scratch even if the number of edge failures is bounded. A commonly adopted solution is that of designing *f-edge fault-tolerant oracles*, that is, compact data structures that can quickly report exact or approximate extremal and pairwise distances in the network after up to f edges failed. These structures are also known as *sensitivity oracles*, where the sensitivity is the maximum number f of supported failures.

Many known fault-tolerant data structures are randomized. The algorithm that preprocesses the underlying network may depend on random bits or the correctness of the oracle's answers is only guaranteed with some probability. Besides the practical difficulties of working with (true) randomness in computing, it is an interesting question to what extend randomness as a resource is needed to obtain efficient fault-tolerant oracles. In this paper, we show that for a wide range of applications randomness can be removed with only a slight loss of performance, or even none at all in some cases. For this, we develop a novel derandomization framework and combine it with known techniques to obtain the following results.

- We present new deterministic f-edge fault-tolerant oracles that report the exact/approximate diameter and vertex eccentricities in directed graphs and we show lower bounds charting the limits of approximation vs. space and diameter vs. space trade-offs.
- We derandomize the single-failure *distance sensitivity oracle* (DSO) of Ren [32] that can report exact distance for any pair of vertices in constant time. Our result gives the first deterministic exact DSO with truly sub-cubic processing time and constant query time.
- We derandomize the algorithm of Chechik and Magen [12] for the *Single-Source Replacement Paths* (SSRP) problem on directed graphs, that is, the task of finding a shortest path from a distinguished source vertex to every target, for every possible edge failure.

We believe that our techniques are of independent interest and can help derandomize also other algorithms and data structures in the fault-tolerant domain. Throughout the paper, the underlying network is modeled by a directed graph $G = (V, E)$, possibly with weights on its edges, where V is the set of n vertices and E the set of m edges.

1.1 Diameter and Eccentricity Oracles in Directed Graphs

In Section 3, we discuss fault-tolerant oracles for the diameter and vertex eccentricities of a directed graph. We abbreviate *f-edge fault-tolerant diameter oracle* as f-FDO and *f-edge fault-tolerant eccentricity oracle* as f-FEO. In case of a single failure, $f = 1$, we shorten this to FDO and FEO, respectively. The problem of designing FDOs was originally raised by Henzinger et al. [26] and recently received some renewed interest by Bilò et al. [6]. Although the major focus of the latter work was on undirected graphs, the authors also showed that, for directed graphs, one can compute, in $\widetilde{O}(mn + n^2/\varepsilon)$ time,[1] an oracle of size[2] $O(m)$ and constant query time that guarantees a *stretch* of $1 + \varepsilon$, that is, it reports an upper bound on the value of the diameter within a factor of $1+\varepsilon$, for any $\varepsilon > 0$.

Bilò et al. [6] also gave a complementary space lower bound showing that any fault-tolerant diameter oracle with a sufficiently small stretch must take $\Omega(m)$ bits of space. However, this is not the full picture in that their construction only holds for diameter 2. We show here that in reality there is a transition happening: the larger the diameter, the more space we can save, up to a point where even $o(m)$-space oracles become possible. We aim at pinpointing this transition, starting with a generalization of the bound in [6] to diameter up to $n/\sqrt{m}$.

[1] For a non-negative function $g = g(n)$, we use $\widetilde{O}(g)$ to denote $O(g \cdot \mathsf{polylog}(n))$.
[2] Unless stated otherwise, we measure the space in the number of $O(\log n)$-bit machine words.

▶ **Theorem 1.** *Let $n, m, D \geqslant 3$ be integers with $D = O(\frac{n}{\sqrt{m}})$. Any FDO with stretch $\sigma < \frac{3}{2} - \frac{1}{D}$ on n-vertex, m-edge unweighted directed graphs of diameter D requires $\Omega(m)$ bits of space, regardless of the query time.*

Given an oracle for the fault-tolerant eccentricities with query time q, one can emulate a diameter oracle with query time nq by taking the maximum over all vertices. The information-theoretic lower bound of Theorem 1 is independent of the query time and therefore every FEO also must have size $\Omega(m)$.

Notably, Theorem 1 implies that, for any $0 < \delta \leqslant 1$ and all digraphs with $n^{1+\delta}$ edges and a relatively small diameter of $O(\sqrt{n^{1-\delta}})$, an FDO of stretch essentially $3/2$ takes $\Omega(n^{1+\delta})$ bits of space. As hinted above, this approximation vs. space trade-off no longer holds when we consider directed graphs with large diameter of $\omega(n^{5/6})$, for which we can design FDOs of quasi-linear (in n) space and negligible stretch.

▶ **Theorem 2.** *Let G be a directed graph with n vertices, m edges, and diameter $D = \omega(n^{5/6})$ and let $\varepsilon = \frac{n^{5/6}}{D} = o(1)$. There is an FDO for G with stretch $1 + \varepsilon$, preprocessing time $\widetilde{O}(mn)$, space $O(n \log^2 n)$, and constant query time.*

The gap between the stretch-size trade-offs provided in Theorem 1 and Theorem 2, respectively, suggests that there must be a threshold between $n/\sqrt{m}$ and $n^{5/6}$ where low-stretch FDOs of sub-linear size and constant query time become possible. We further narrow this gap and aim to find the smallest value for the diameter for which one can design an FDO with $o(m)$ space and constant query time. We show that this is possible for directed graphs with diameter $\omega((n^{4/3} \log n)/\sqrt{m})$. We leave it as an open problem to determine the smallest function g such that directed graphs with diameter $g(n)/\sqrt{m}$ admit an FDO with $o(m)$ space. Our results show that g is of order $\omega(n)$ and $O(n^{4/3} \log n)$.

▶ **Theorem 3.** *Let G be be a directed graph with n vertices, m edges, and diameter $\omega((n^{4/3} \log n)/(\varepsilon\sqrt{m}))$. For any $\varepsilon = \varepsilon(n, m) > 0$, there is an FDO for G with stretch $1 + \varepsilon$, preprocessing time $\widetilde{O}(mn)$, space $o(m)$, and constant query time.*

For the sake of readability, the FDOs in Section 3 are randomized. Later, in Section 4, we describe our derandomization framework and show how to apply it to both FDOs.

We now move our attention to the case multiple edge failures and give bounds in terms of f and n on the minimum space requirement of f-FDOs. Bilò et al. [6] designed an f-FDO for *undirected* graphs of stretch $f + 2$ that takes space $\widetilde{O}(fn)$. The size of this oracle is optimal up to polylogarithmic factors. In the next theorem, we show that such compact oracles are impossible for directed graphs, even when allowing arbitrarily large stretch

▶ **Theorem 4.** *Let n, f be positive integers such that $2^{f/2} = O(n)$. Any f-FDO with an arbitrary finite stretch on n-vertex directed graphs requires $\Omega(2^{f/2} n)$ bits of space, regardless of the query time.*

The lower bound of Theorem 4 marks an exponential-in-f separation between the undirected and directed setting. The directed graph used in the proof is inspired by the lower-bound construction used by Baswana et al. [3] for the *f-edge fault-tolerant Single-Source Reachability* problem. This problem asks to compute the sparsest subgraph H of a directed graph G that preserves reachability from a designated source vertex s, that is, for every vertex v and every set F of $|F| \leqslant f$ edge failures, there is path from s to v in H that avoids every edge in F if and only if there is such a path in G. Baswana et al. [3] provided a class of directed graphs for which any subgraph preserving single-source reachability with sensitivity

f has $\Omega(2^f n)$ edges. Our lower bound requires non-trivial extensions of their construction as it needs to satisfy several additional properties. For example, the directed graph in [3] has unbounded diameter, while any lower bound for FDOs requires strongly connected graphs.

We also consider the design of fault-tolerant eccentricity oracles for general directed graphs as well as directed acyclic graphs (DAGs). For the single-failure case and exact eccentricities, there is a folklore solution using the DSO of Bernstein and Karger [4] that runs in $\widetilde{O}(n^3)$ time. Henzinger et al. [26] showed how to trade stretch for running time and presented an $(1+\varepsilon)$-approximate solution with preprocessing time $\widetilde{O}(mn + n^{3/2}\sqrt{Dm/\varepsilon})$, where D denotes the diameter of the underlying graph. Both oracles build a look-up table of size $O(n^2)$ using the fact that, for any vertex v, only the failure of an edge on a shortest path tree rooted in v can change the eccentricity of v. The table allows for a constant query time but generalizing this to multiple failures $f \geqslant 2$ would take $\Omega(n^{f+1})$ space. We show how to do better than that. We give a meta-theorem that turns any exact or approximate DSO for pairwise distances into an FEO for eccentricities. Plugging in any compact DSO for multiple failures then immediately gives a space improvement for the FEO. In the following, with stretch $\sigma = 1$, we mean exact oracles.

▶ **Theorem 5.** *Let G be a (undirected or directed and possibly edge-weighted) graph with n vertices and m edges. Given access to a DSO for G with sensitivity f, stretch $\sigma \geqslant 1$, preprocessing time P, space S, and query time Q, one can construct an f-FEO for G with stretch $1 + \sigma$, preprocessing time $O(mn + P)$, space $O(n + S)$, and $O(f \cdot Q)$ query time.*

There are multiple distance oracles to choose from, all with different strengths and weaknesses. When using the DSO for of Duan and Pettie [15], we get in polynomial time a 2-approximate 2-FEO with space $O(n^2 \log^3 n)$. To the best of our knowledge, this is the first eccentricity oracle for dual failures in subcubic space. Van den Brand and Saranurak [8] gave a DSO supporting an arbitrary number of failures f. On directed graphs with integer edge weights in the range $[-M, M]$ it has polynomial space and preprocessing time, but a query time that depends both on the sensitivity and the graph size. Let $\omega < 2.37286$ be the matrix multiplication exponent [1]. Plugging the DSO in [8] into our reduction gives an f-FEO with stretch 2, $O(Mn^3)$ space[3], and query time $O(Mnf^{\omega+1})$. On undirected graphs, we can make the query time independent of n by applying the very recent DSO by Duan and Ren [17] with $O(fn^4)$ space and a query time of $f^{O(f)}$. However, the preprocessing of the latter is only polynomial for constant f. Since our reduction also applies to approximate oracles, we get, for any $f = o(\log n / \log\log n)$ and $\varepsilon > 0$, an f-FEO in polynomial time with stretch $(2 + \varepsilon)$, space $O(n^2((\log n)/\varepsilon)^f f)$ and query time $O(f^6 \log n)$ via the DSO by Chechik et al. [11].

As already mentioned above, the $\Omega(2^{f/2} n)$-bits lower bound in Theorem 4 also holds for FEOs. On DAGs, however, we can improve upon this and obtain a space requirement that is reminiscent of the one Bilò et al. [6] gave for *undirected* graphs. Note that in a DAG at most one vertex can have bounded eccentricity.

▶ **Theorem 6.** *Let G be a directed acyclic graph with, m real-weighted edges, n vertices, and a distinguished source vertex s. For any integer f, there is an f-FEO for G with stretch f, preprocessing time $\widetilde{O}(m)$, space $O(nf)$, and $O(f)$ query time.*

All the results for f-FDOs and f-FEOs are presented for edge failures. However, they also hold for vertex failures using well-known transformation techniques for directed graphs.[4]

---

[3] In [8], the space of the DSO is phrased as $O(Mn^3 \log n)$ *bits*.

[4] Indeed, we can transform the directed graph G into some graph G' with $2n$ vertices. We represent each vertex v of G with an edge (v^-, v^+) in G', and replace each edge (u, v) of G with the edge (u^+, v^-) in G'. For edge-weighted G, the weight of the new vertex-edge is set to 0 keeping eccentricities. For unweighted G, the eccentricity of v in any subgraph H of G is half the eccentricity of v^- in $H' \subseteq G'$.

1.2 Derandomization Technique

We now turn to the derandomization of fault-tolerant data structures. In Section 4, we develop the *Hierarchical Double Pivots Hitting Sets* (HDPH) algorithm as the center piece of a framework to derandomize known replacement paths algorithms and oracles. The aim of the HDPH algorithm is to compute a sequence of sets $B_1, \dots, B_{\log n} \subseteq V(G)$ such that each B_i has size $\widetilde{O}(n/2^i)$ and hits a set $\mathcal{P}_i$ of (replacement) paths each of length $\Omega(2^i)$. Unfortunately, the paths $\mathcal{P}_i$ that need to be hit by B_i are not known in advance. Our algorithm fixes this issue by iteratively computing the set of paths $\mathcal{P}_i$ using the previous sets $B_0, \dots, B_{i-1}$. The algorithm relies on the ability of the oracle we want to derandomize to be *path-reporting*, that is, to report a path representing the exact or approximate distance between the queried vertices for DSOs, the diameter for FDOs, or the vertex eccentricity for FEOs. We show how to implement the HDPH algorithm to derandomize the FDOs in Theorems 2 and 3, the DSO of Ren [32] for directed graphs with integer edge weights in the range $[1, M]$, and the algorithm of Chechik and Magen [12] for the SSRP problem in directed graphs.

Distance Sensitivity Oracles. The concept of DSOs was introduced by Demetrescu et al. [13] who showed how to compute an exact DSO of size $O(n^2 \log n)$ and constant query time in $\widetilde{O}(mn^2)$ time. Later, Bernstein and Karger [4] improved the preprocessing time to $\widetilde{O}(mn)$ and Duan and Zhang [18] reduced the space to $O(n^2)$, which is asymptotically optimal. Algebraic algorithms are known to further improve the preprocessing times, if one is willing to employ fast matrix multiplication, see [10, 23] and the references therein. For more results on approximate DSOs for both single and multiple failures, see [11, 14, 15].

We combine the HDPH framework with a recent breakthrough result by Karthik and Parter [28] to derandomize the path-reporting DSO of Ren [32] for directed graphs with integer edge weights in the range $[1, M]$. This was the first DSO that achieved a constant query time with a randomized subcubic preprocessing time of $O(Mn^{2.7233})$. On undirected graphs, the preprocessing improves to $\widetilde{O}(Mn^{(\omega+3)/2}) = O(Mn^{2.6865})$. Our derandomization of Ren's DSOs in both settings incurs a slight loss of efficiency. Nevertheless, we obtain the first deterministic DSO with constant query time and truly sub-cubic preprocessing. This improves significantly over the result by Alon, Chechik, and Cohen [2] who designed a DSO with $O(mn^{4-\alpha})$ preprocessing time and $\widetilde{O}(n^{2\alpha})$ query time, for any $\alpha \in (0, 1)$.

▶ **Theorem 7.** *For any n-vertex directed graph G with integer edge weights in the range $[1, M]$, there exists a deterministic path-reporting DSO with $O(Mn^{2.8068})$ preprocessing time and constant query time. If G is undirected, the preprocessing time decreases to $\widetilde{O}(Mn^{(\omega+6)/3}) = O(Mn^{2.7910})$.*

Recently, Gu and Ren [23] presented a new randomized DSO with a preprocessing time of $O(Mn^{2.5794})$. Unfortunately, our HDHP algorithm cannot be used to derandomize it for the following two reasons. First, the DSO of Gu and Ren is not path-reporting. Secondly, it internally relies on probabilistic polynomial identity testing. It is a long-standing open question how to derandomize this, far beyond the field of fault-tolerant data structures.

Single Source Replacement Paths Problem. In the SSRP problem we want to compute replacement paths from a designated source to each destination vertex, under each possible edge failure. Grandoni and Vassilevska Williams [21, 22] first developed an algorithm for both directed and undirected graphs with integer edge weights in the range $[1, M]$ that uses fast matrix multiplication and runs in $\widetilde{O}(Mn^{\omega})$ time. Chechik and Cohen [9] presented an $\widetilde{O}(m\sqrt{n} + n^2)$ time SSRP algorithm for *undirected* graphs that was later simplified and generalized to deal with multiple sources by Gupta et al. [24]. In this paper we use our

HDPH framework to derandomize the recent $\widetilde{O}(m\sqrt{n}+n^2)$ time randomized algorithm for *directed* graphs developed by Chechik and Magen [12], without any loss in the time complexity. Specifically, we prove the following result.

▶ **Theorem 8.** *There exists a deterministic algorithm for the Single Source Replacement Path problem in unweighted directed graphs running in time* $\widetilde{O}(m\sqrt{n}+n^2)$.

2 Preliminaries

We let $G=(V,E)$ denote a directed graph on n vertices and m edges, potentially edge-weighted by some function $w\colon E\to\mathbb{R}$. We tacitly assume that G is strongly connected, in particular, $m=\Omega(n)$. For any (weighted) directed graph H (possibly different from G), we denote by $V(H)$ and $E(H)$ the set of its vertices and edges, respectively. Let P be a path in H from $s\in V(H)$ to $t\in V(H)$, we say that P is an *s-t-path* in H. We denote by $|P|=\sum_{e\in E(P)}w(e)$ the *length* of P, that is, its total weight. If H is unweighted, we let $|P|=|E(P)|$ denote the number of its edges. For $u,v\in V(P)$, we let $P[u..v]$ denote the subpath of P from u to v. For $s,t\in V(H)$, the *distance* $d_H(s,t)$ is the minimum length of any s-t-path in H; if s and t are disconnected, we set $d_H(s,t)=+\infty$. When talking about the base graph G, we drop the subscripts if this does not create any ambiguities. The *eccentricity* of a vertex $s\in V(H)$ is $\mathrm{ecc}_H(s)=\max_{t\in V(H)}d_H(s,t)$, the *diameter* is $\mathrm{diam}(H)=\max_{s\in V(H)}\mathrm{ecc}_H(s)$. For a set $F\subseteq E(H)$ of edges, let $H-F$ be the graph obtained from H by removing all edges in F. A *replacement path* $P_H(s,t,F)$ is a shortest path from s to t in $H-F$. Its length $d_H(s,t,F)=|P_H(s,t,F)|$ is the *replacement distance*. The *fault-tolerant eccentricity* of a vertex $s\in V$ of the base graph with respect to F is $\mathrm{ecc}_{G-F}(s)$, the *fault-tolerant diameter* is $\mathrm{diam}(G-F)$.

For a positive integer f, an *f-edge fault-tolerant eccentricity oracle* (f-FEO) for G reports, upon query (s,F) with $|F|\leqslant f$, the value $\mathrm{ecc}_{G-F}(s)$. An *f-edge fault-tolerant diameter oracle* returns $\mathrm{diam}(G-F)$ upon query F. For a single edge failure, we write FEO for 1-FEO and abbreviate $F=\{e\}$ to e. For any real number $\sigma=\sigma(n,m,f)\geqslant 1$, an f-FEO is said to have *stretch* σ, or be *σ-approximate*, if the returned value $\widehat{\mathrm{ecc}}(s,F)$ on query (s,F) satisfies $\mathrm{ecc}_{G-F}(s)\leqslant\widehat{\mathrm{ecc}}(s,F)\leqslant\sigma\cdot\mathrm{ecc}_{G-F}(s)$, analogously for f-FDOs. The *preprocessing time* is the time needed to compute the data structure, its *query time* is the time needed to return an answer. For weighted graphs, we assume the weight function being such that all distances can be stored in a single word on $O(\log n)$ bits. Unless stated otherwise, we measure the *space* of the oracles in the number of words. The oracles cannot access features of graph G except those stored during preprocessing. The size of the input does not count against the space of the data structures.

3 Diameter and Eccentricity Oracles

This section discusses fault-tolerant oracles for the diameter and vertex eccentricity in directed graphs. We start by presenting space lower bounds for FDOs that guarantee a certain stretch when supporting single or multiple edge failures, respectively. In the single-failure case, the bound depends on the diameter of the graph. Roughly speaking, if the base graph has low diameter, we cannot save much space over just storing all edges. The picture changes if the diameter grows larger. We show that then we can obtain FDOs with $o(m)$ space, or even $\widetilde{O}(n)$. We then turn the discussing to eccentricity oracles for dual and multiple failures. Note that there we need to report not only one value per graph $G-F$, but one per vertex in each of those, so special techniques are needed to handle the space increase.

3.1 Space Lower Bounds for Diameter Oracles

Bilò et al. [6] showed that any FDO with a stretch $\sigma < 3/2$ on *undirected* m-edge graphs must take $\Omega(m)$ bits of space. In particular, any data structure that can distinguish between a fault-tolerant diameter of 2 and 3 has this size. Their construction transfers to directed graphs (by merely doubling each undirected edge into two directed ones). However, they do not parameterize the graphs by their diameter, namely, if the FDO has to distinguishing between diameter D and $3D/2$ for some $D \geqslant 3$. We generalize their result by showing that there is an intermediate range of D where the $\Omega(m)$-bit bound still applies. However, here the situation is more intricate in that large values of D do allow for significant space reductions.

The construction[5] by Bilò et al. [6] cannot be extended to $D \geqslant 3$. The proof of the next lemma had to be deferred to the full version due to space reasons.

▶ **Lemma 9.** *Let n, m, D be integers such that $n^2 \geqslant m \geqslant n \geqslant 4$, and $n/\sqrt{m} > D \geqslant 3$. There exists a family $\mathcal{G}$ of n-vertex directed graphs with diameter D and $\Theta(m)$ edges such that any data structure for graphs in $\mathcal{G}$ that decides whether the fault-tolerant diameter remains at D or increases to $(3D-1)/2$ for odd D (or $(3D/2)-1$ for even D) requires $\Omega(m)$ bits of space.*

It is now easy to obtain the $\Omega(m)$-bit lower bound of Theorem 1 since any FDO of stretch $\sigma < \frac{3}{2} - \frac{1}{D}$ must tell the two cases apart.

We now turn to diameter oracles that support more than one edge failure, $f > 1$. Theorem 4 states that they require space that is exponential in f, even if we allow the stretch and query time to be arbitrarily large (but finite). It follows from the next lemma together with the observation that such f-FDOs have to detect whether the edge failures disconnect the graph.

▶ **Lemma 10.** *Any data structure for n-vertex digraphs that decides for at most $2f = O(\log n)$ edge failures whether the fault-tolerant diameter is finite requires $\Omega(2^f n)$ bits of space.*

3.2 Improved Upper Bounds

The above discussion shows that for graphs with *small* diameter, there is no hope to obtain an FDO whose space is much smaller than what is needed to store the full graph. At least not while retaining good stretch at the same time. The lower bound in Theorem 1, however, breaks down for a *large* diameter. Indeed, we show next that in this regime we can do much better in terms of space, without sacrificing stretch or query time.

Theorems 2 and 3 will follow from the same construction. The initial way we present it in Lemma 12 uses randomization in the form of a well-known sampling lemma, see [22, 33]. We will later discuss how to derandomize the oracles.

▶ **Lemma 11** (Sampling Lemma)**.** *Let H be a n-vertex directed graph, $c > 0$ a positive constant, and $L \geqslant c \ln n$. Define a random set $B \subseteq V(H)$ by sampling each vertex of H independently with probability $(c \ln n)/L$. With probability at least $1 - \frac{1}{n^c}$, the cardinality of B is $O((n \log n)/L)$. Let further $\mathcal{P}$ be a set of ℓ simple paths in H, each of which spans L vertices. With probability at least $1 - \frac{\ell}{n^c}$, we have $V(P) \cap B \neq \emptyset$ for every $P \in \mathcal{P}$.*

▶ **Lemma 12.** *For any n-vertex, m-edge unweighted directed graph G with diameter $D = \omega(\log n)$ and any $\varepsilon = \varepsilon(n, m, D) > 0$, we can compute in time $\widetilde{O}(mn + n^4/(\varepsilon^3 D^3))$ an FDO with $1+\varepsilon$ stretch, $O(n + (n^{8/3} \log^2 n)/(\varepsilon^2 D^2))$ space, and constant query time.*

[5] The graph used in the proof of [6, Lemma 12] has the property that failing any edge can increase the diameter by at most 1.

Proof. Let $D = \mathrm{diam}(G)$, $b = n/(\varepsilon D)$, and $c > 0$ a sufficiently large constant. We sample a set $B \subseteq V$ of *pivots* by including each vertex independently with probability $(2bc \ln n)/n$. By Lemma 11 with $L = n/2b = \varepsilon D/2$, there are $O(b \log n)$ many pivots w.h.p.

For the graph G, compute in $\widetilde{O}(mn)$ time the $O(1)$-query time distance sensitivity oracle of Bernstein and Karger [4]. We further compute a subgraph H of G that is just the union of $|B|$ shortest-path trees, one rooted at each pivot. We iterate over the edges of H and compute the collection $\mathcal{X}$ of all those $e \in E(H)$ such that $d(b_1, b_2, e) > d(b_1, b_2)$ for some pair $(b_1, b_2) \in B \times B$. The time to compute $\mathcal{X}$ is $O(n|B|^3) = \widetilde{O}(n^4/(\varepsilon^3 D^3))$ since processing an edge in H requires $|B|^2$ calls to the DSO. Observe that any subgraph of G that exactly preserves distances between all pairs in $B \times B$ must contain all the edges of $\mathcal{X}$. Bodwin [7] showed that there are distance-preserving subgraphs with respect to $B \times B$ with at most $O(n + n^{2/3}|B|^2)$ edges. Thus, the size of $\mathcal{X}$ is bounded by $O(n + n^{2/3}|B|^2)$.

Next, we build a dictionary $\mathcal{D}_{\mathcal{X}}$ in which we store the the edges in $\mathcal{X}$ together with the maximum distance between any pair of pivots if the edge fails (or $\mathrm{diam}(G)$ if this is larger). In other words, for each $e \in \mathcal{X}$, we store $\phi(e) = \max\{\max_{b_1,b_2 \in B} d(b_1, b_2, e), \mathrm{diam}(G)\}$. Let $\mathcal{Y}$ be the set of all edges in E such that $G - e$ is no longer strongly connected. We build a dictionary $\mathcal{D}_{\mathcal{Y}}$ in which we store information about the edges $\mathcal{Y}$. It is well-known that $\mathcal{Y}$ contains $O(n)$ edges and can be computed in time $O(m)$ [27].

Recall that $b = n/(\varepsilon \,\mathrm{diam}(G))$. The oracle's output $\widehat{D}(e)$ is defined as follows: if $e \in \mathcal{Y}$, then $\widehat{D}(e) = \infty$; if $e \in \mathcal{X}$, $\widehat{D}(e) = \phi(e) + n/b$; otherwise, the oracle outputs $\widehat{D}(e) = \mathrm{diam}(G) + n/b = (1 + \varepsilon)\,\mathrm{diam}(G)$.

Evidently, the oracle is correct for all $e \in \mathcal{Y}$. It is also easy to verify that all outputs are at most $\phi(e) + n/b \leqslant \mathrm{diam}(G - e) + n/b = \mathrm{diam}(G - e) + \varepsilon\,\mathrm{diam}(G) \leqslant (1 + \varepsilon)\,\mathrm{diam}(G - e)$. To prove that they are also at least $\mathrm{diam}(G - e)$, consider a vertex pair $(u, v) \in V \times V$ such that $d(u, v, e) = \mathrm{diam}(G - e) < \infty$. With high probability[6] by Lemma 11, there exists a shortest u-v-path in $G - e$ and two pivots $b_u, b_v \in B$ on that path such that $d(u, b_u, e), d(b_v, v, e) \leqslant L = n/2b$. We have $\mathrm{diam}(G-e) = d(u, b_u, e)+d(b_u, b_v, e)+d(b_v, v, e)$. Suppose $e \notin \mathcal{X}$. Then, $d(b_u, b_v, e) = d(b_u, b_v) \leqslant \mathrm{diam}(G)$ holds and therefore $\mathrm{diam}(G - e) \leqslant \mathrm{diam}(G) + n/b = \widehat{D}(e)$. If $e \in \mathcal{X}$, then $d(b_u, b_v, e) \leqslant \phi(e)$ and $\mathrm{diam}(G - e) \leqslant \phi(e) + n/b = \widehat{D}(e)$.

There are k-element dictionaries of size $O(k)$ and $O(1)$ query time computable in time $\widetilde{O}(k)$ [25]. The dictionaries have total size $O(n+n^{2/3}|B|^2) = O(n+(n^{8/3} \log^2 n)/(\varepsilon^2 D^2))$. ◀

The oracle in Lemma 12 can also be extended to handle vertex failures. The only modification required is to add to set $\mathcal{X}$ those vertices $v \in V$ that satisfy $d(b_1, b_2, v) > d(b_1, b_2)$ for some $(b_1, b_2) \in B \times B$, and to add to $\mathcal{Y}$ to be those vertices v for which $G - v$ is not strongly connected. Suppose $D = \omega(n^{5/6})$, inserting any $\varepsilon \geqslant n^{5/6}/D = o(1)$ above gives an FDO with near linear space and $1 + o(1)$ stretch that is computable in time $\widetilde{O}(mn)$, which proves Theorem 2. Furthermore, for graphs with diameter $\omega((n^{4/3} \log n)/(\varepsilon\sqrt{m}))$, we obtain in $\widetilde{O}(mn)$ time an FDO with constant query time and $o(m)$ space (Theorem 3).

3.3 Eccentricity Oracles

We now prove Theorem 5 that constructs an f-edge fault-tolerant eccentricity oracle from a DSO supporting f failures. The improved f-FEO for DAGs can be found in the full version.

[6] We say an event occurs *with high probability* (w.h.p.) if it has success probability $1 - n^{-c}$ for some constant $c > 0$ that can be made arbitrarily large.

Let $\mathcal{D}$ be a DSO with sensitivity f and stretch σ that, on (un-)directed possibly weighted graphs, can be computed in time P, uses S space, and has a query time of Q. For any given source $s \in V$ and query set F of $|F| \leqslant f$ edges, our oracle reports an $(1+\sigma)$-approximation of the eccentricity of s in $G - F$. We simply store $\mathcal{D}$ and, for each $x \in V$, the value $ecc_G(x)$. All eccentricities in the base graph G can be obtained with a BFS from each vertex in $O(mn)$.

Upon query $(s, F = \{(x_1, y_1), \ldots, (x_f, y_f)\})$, we use $\mathcal{D}$ to compute $d(s, y_i, F)$, for all $1 \leqslant i \leqslant f$. Our estimate is $\widehat{\text{ecc}}_{G-F}(s) = \text{ecc}_G(s) + \max_{1 \leqslant i \leqslant f} d(s, y_i, F)$. The time taken to compute $\widehat{\text{ecc}}_{G-F}(s)$ is $O(f \cdot Q)$ and the space requirement of the oracle is $O(n + S)$.

Now we show that $\widehat{\text{ecc}}_{G-F}(s)$ is a $(1 + \sigma)$-approximation of $\text{ecc}_{G-F}(s)$. Let F_0 be the subset of F consisting of those edges in F that lie on some shortest-path tree T rooted in s. If F_0 is empty, we immediately get $\text{ecc}_{G-F}(s) = \text{ecc}_G(s) \leqslant \widehat{\text{ecc}}_{G-F}(s)$. Otherwise, for any $v \in V$, either $d(s, v) = d(s, v, F)$ or there exists an $(x, y) \in F_0$ such that y is an ancestor of v in T. In this latter case $d(y, v, F) \leqslant \text{ecc}_G(s)$. This proves that $d(s, v, F) \leqslant d(s, y, F) + d(y, v, F) \leqslant d(s, y, F) + \text{ecc}_G(s) \leqslant \widehat{\text{ecc}}_{G-F}(s)$. Thus, $\text{ecc}_{G-F}(s) \leqslant \widehat{\text{ecc}}_{G-F}(s)$. Next observe that $\text{ecc}_G(s) \leqslant \text{ecc}_{G-F}(s)$ and $\max_{1 \leqslant i \leqslant f} d(s, y_i, F) \leqslant \sigma \cdot \text{ecc}_{G-F}(s)$, which proves that $\widehat{\text{ecc}}_{G-F}(s) \leqslant (1 + \sigma) \cdot \text{ecc}_{G-F}(s)$.

4 Derandomization Framework

The fault-tolerant diameter oracles in Theorems 2 and 3 are randomized. They both follow from Lemma 12 which in turn relies on a random hitting set to intersect all replacement paths of a certain length. In fact, many more data structures and algorithms in the fault-tolerant setting follow a sampling-based approach similar to Lemma 11, see e.g. [9, 12, 22, 32, 33, 35]. It is an interesting question whether these algorithms can be derandomized efficiently. Currently there is no single approach to derandomize Lemma 11 in the same $O(n)$ time it uses to go through all vertices. Therefore, the literature focuses on the specific applications. The goal is to replace the sampling step by a deterministic construction of the hitting set that, while taking $\omega(n)$ time, does not (or only marginally) increase the asymptotic running time of the whole algorithm. Recently, there was some progress on notable special cases. Karthik and Parter [28] gave a derandomization for the algebraic version of the distance sensitivity oracle of Weimann and Yuster [35] with a slightly higher running time (for a detailed discussion see Lemma 15 below). Bilò et al. [5] derandomized the SSRP algorithms of Grandoni and Vassilevska Williams [22] as well as Chechik and Cohen [9]. Their derandomization succeeds in the same time bounds as the original randomized algorithm, but the technique only works for undirected graphs. Here, we develop a framework for directed graphs. We first apply it to our own FDOs and then show its versatility by also derandomizing the DSO of Ren [32] and the SSRP algorithm of Chechik and Magen [12].

We build on the work of Alon, Chechik, and Cohen [2]. We first review some technical details of their result and then describe our additions. For now, we assume the base graph G to be unweighted and only later (in Section 5) incorporate positive integer edge weights. For concreteness, consider the task in Lemma 12 of finding a set $B \subseteq V$, the pivots, such that for all $s, t \in V$ and edge $e \in E$ with replacement distance $d(s, t, e)$ at least $L = \varepsilon \operatorname{diam}(G)/2$, there exists some replacement path $P(s, t, e)$ that contains a pivot $x \in B$. Other fault-tolerant algorithms pose similar requirements on B. The technique in [2] consists of computing a small set of *critical* paths, much smaller than the set of all $O(mn^2)$ replacement paths. Once we have those, a hitting set can be computed with the folklore greedy algorithm, called GreedyPivotSelection in [2], that always selects a vertex that is contained in the most unhit paths.[7] Alternatively, one can use the blocker set algorithm of King [29].

[7] To achieve the performance of Lemma 13, one has to truncate all paths by selecting L vertices arbitrarily from each $P \in \mathcal{P}$. This is non-issue for us as, by construction, all our paths will have length $\Theta(L)$.

▶ **Lemma 13** (Alon, Chechik, and Cohen [2]). *Let $1 \leqslant L \leqslant n$ and $1 \leqslant q = \mathsf{poly}(n)$ be two integers. Let $P_1, \ldots, P_q \subseteq V$ be sets of vertices that, for every $1 \leqslant k \leqslant q$, satisfy $|P_k| \geqslant L$. The algorithm* GreedyPivotSelection *computes in time $\widetilde{O}(qL + n^2/L)$ a set $B \subseteq V$ of $|B| = O((n \log q)/L) = \widetilde{O}(n/L)$ pivots such that, for every index k, it holds that $B \cap P_k \neq \emptyset$.*

The crucial part is to quickly find the paths P_k such that hitting them is sufficient to hit all long replacement path. Of course, this could be done by computing all-pairs shortest paths in each graph $G-e$ in total time $\widetilde{O}(m^2 n)$ using Dijkstra's algorithm (or $\widetilde{O}(mn^{2.5302})$ if one is willing to use fast rectangular matrix multiplication [20, 36]). However, this is much more than the $\widetilde{O}(mn + n^4/(\varepsilon^3 \operatorname{diam}(G)^3))$ time bound we had in Lemma 12. For the applications in [2], a single set of paths and therefore a single hitting set was sufficient. Bilò et al. [5] (with slightly different requirements on the set B) were able to make do with three sets, exploiting the undirectedness of the underlying graph.

We extend this to directed graphs using a hierarchical approach to find the critical paths. Observe how the length parameter L in Lemma 13 serves two roles. The longer the paths, the longer it takes to compute B, but the *fewer* vertices suffice to intersect all paths. Additionally, we have to compute the set of critical paths which takes (at least) linear time in their length. So L has to fall just in the right range for the computation to be fast. To achieve this, we use an exponentially growing sequence of lengths $L_1, L_2, \ldots, L_{O(\log n)}$ and, instead of a single set, compute a sequence $B_1, B_2, \ldots$ of exponentially shrinking sets such that, in the i-th stage, B_i hits, again for all $s, t \in V$ and $e \in E$, some replacement path of length at least L_i. However, this poses some new difficulties because now the collection of critical paths has to be computed step by step. Imagine in the i-th stage, we have already obtained the all the subsets $\mathcal{P}_j$, $j < i$, of paths with respective lengths L_j. The key observation is that the hitting sets B_j from the previous rounds carry valuable information that can be harnessed to find the new set $\mathcal{P}_i$ faster, this then offsets the run time penalty incurred by the greater length of the new paths.

The HDPH Algorithm. We now describe the Hierarchical Double Pivots Hitting Sets (HDPH) algorithm that makes these ideas concrete. It can be seen as a "reference implementation" of the framework. For a specific application, one still has to adapt the details. The algorithm is more general than what is needed for diameter oracles in Theorems 2 and 3. For example, it also pertains to vertex failures. Later, in Section 5, we show an example how to modify the algorithm for other problems (more are found in the full version).

Let $C \geqslant 3/2$ be a constant. The aim of the HDPH algorithm is to compute a sequence of sets $B_1, \ldots, B_{\lceil \log_C n \rceil} \subseteq V$ of size $|B_i| = \widetilde{O}(n/C^i)$ such that for all vertices $s, t \in V$ and failure $f \in E \cup V$ with $d(s, t, f) \in (C^i, C^{i+1}]$ there exists a replacement path $P(s, t, f)$ that contains a pivot $z \in B_i$. It assumes access to the "APSP data" of the original graph G, that is, the distance $d(s, t)$ for all s, t and a corresponding shortest path $P(s, t)$. Also, it requires a deterministic path-reporting distance sensitivity oracle with constant query time (both for the distance and each reported edge) as a black box.

The HDPH algorithm is sketched in Algorithm 1. In lines 1 and 2 it initializes $B_i = V$ for $i \leqslant 2$. In lines 3-12, for $3 \leqslant i \leqslant \lceil \log_C n \rceil$, we iteratively compute the hitting sets B_i by using the hitting sets from the previous 3 iterations to obtain a set of shortest and replacement paths $\mathcal{P}_i$ of length $\Theta(n/C^i)$ that one needs to hit, and then use the greedy algorithm GreedyPivotSelection to compute the set of pivots B_i which hits this set of paths $\mathcal{P}_i$. The paths are defined as follows. First, in line 4 we add to $\mathcal{P}_i$ shortest paths $P(x, y)$ whose length is in the range $(C^i, C^{i+1}]$ such that $x, y \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$ are pivots from the last 3 iterations. Then, in lines 5-11, for every pair of pivots $x, y \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$ whose shortest path

Algorithm 1 Hierarchical Double Pivots Hitting Sets (HDPH) Algorithm.

Input: APSP data and a deterministic path-reporting DSO with $O(1)$ query time.
Output: The hitting sets $B_0, \dots, B_{\lceil \log_C n \rceil}$.

1 **for** $i \in [0, 2]$ **do**
2 $B_i \leftarrow V$
3 **for** $i \in [3, \lceil \log_C n \rceil]$ **do**
4 Let $\mathcal{P}_i = \{P(x,y) \mid x, y \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$ such that $d(x,y) \in (C^{i-6}, C^{i+1}]\}$
5 **for** $x, y \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$ **do**
6 **if** $d(x,y) \leqslant C^{i+1}$ **then**
7 **for** $f \in E(P(x,y)) \cup V(P(x,y))$ **do**
8 query the DSO for $d(x,y,f)$
9 **if** $d(x,y,f) \in (C^{i-6}, C^{i+1}]$ **then**
10 query the DSO for $P(x,y,f)$
11 $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup \{P(x,y,f)\}$
12 $B_i \leftarrow$ GreedyPivotSelection$(\mathcal{P}_i)$
13 **return** $B_0, \dots, B_{\lceil \log_C n \rceil}$

$P(x,y)$ is of length at most C^{i+1}, and for every edge or every $f \in E(P(x,y)) \cup V(P(x,y))$ we query the DSO with (x,y,f) to compute the distance $d(x,y,f)$. If $d(x,y,f) \in (C^{i-6}, C^{i+1}]$ then we use the DSO to also report a replacement path $P(x,y,f)$ and add it to $\mathcal{P}_i$.

The next lemma proves the properties of the resulting hitting sets and the run time.

▶ **Lemma 14.** *Given the APSP data and a deterministic path-reporting DSO with $O(1)$ query time, the HDPH algorithm deterministically computes, in $\widetilde{O}(n^2)$ time, all the hitting sets B_i, with $0 \leqslant i \leqslant \lceil \log_C n \rceil$. For every $0 \leqslant i \leqslant \lceil \log_C n \rceil$, it holds that $|B_i| = \widetilde{O}(n/C^i)$. For every pair of vertices $s, t \in V$ and for every failing edge or vertex $f \in E \cup V$ such that $d(s,t,f) \in (C^i, C^{i+1}]$ there exists a pivot $z \in B_i$ such that $d(s,t,f) = d(s,z,f) + d(z,t,f)$. Finally, for every pair of vertices $s, t \in V$ such that $d(s,t) \in (C^i, C^{i+1}]$, there exists a pivot $z \in B_i$ such that $d(s,t) = d(s,z) + d(z,t)$.*

Proof. We first prove by induction that for every $i \in [0, \lceil \log_C n \rceil]$ it holds that $|B_i| = \widetilde{O}(n/C^i)$. The claim trivially holds for $i \leqslant 2$ as $B_0 = B_1 = B_2 = V$. For the inductive step, we assume that $|B_j| = \widetilde{O}(n/C^j)$ for every $j < i$. We show that the set of paths $\mathcal{P}_i$ contains $\widetilde{O}(n^2/C^i)$ paths, each of length $\Theta(C^i)$, and thus the result of the greedy algorithm $B_i \leftarrow$ GreedyPivotSelection$(\mathcal{P}_i)$ contains, by Lemma 13, at most $\widetilde{O}(n/C^i)$ vertices. Moreover, the runtime of the GreedyPivotSelection procedure is $\widetilde{O}(n^2)$.

For every $s, t \in V$, let $P(s,t)$ denote the shortest s-t-path in the APSP data. There are two places where paths are added to $\mathcal{P}_i$. In line 4, the algorithm adds shortest paths between vertices $x, y \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$ whenever $d(x,y) = |P(x,y)| \in (C^{i-6}, C^{i+1}]$, and by the induction hypothesis there are $\widetilde{O}(n^2/C^{2(i-1)}) = \widetilde{O}(n^2/C^i)$ such pairs of vertices (since $|B_j| = \widetilde{O}(n/C^j)$ for every $j < i$). Thus, the claim holds for the paths in line 4. In line 11, the algorithm adds paths $P(x,y,f)$ to $\mathcal{P}_i$ only for pairs $x, y \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$ and edges or vertices $f \in E(P(x,y)) \cup V(P(x,y))$ with $d(x,y) \leqslant C^{i+1}$, there are $\widetilde{O}(C^{i+1} \cdot (n^2/C^{2(i-1)})) = \widetilde{O}(n^2/C^i)$ such triples (x,y,f). The only paths added there are such that $d(x,y,f) \in (C^{i-6}, C^{i+1}]$ (due to the condition in line 9) and thus the length of $P(x,y,f)$ is $\Theta(C^i)$. So the claim holds here as well.

Next, we prove that the runtime of the algorithm is $\widetilde{O}(n^2)$. We show that a single iteration of the for loop in lines 4-16 takes $\widetilde{O}(n^2)$ time, and as there are $O(\log n)$ iterations for $i \in [3, \lceil \log_C n \rceil]$. The number of pairs $x, y \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$ is $\widetilde{O}(n^2/C^{2(i-1)})$. The inner loop in lines 7-11 is executed only if $d(x,y) \leqslant C^{i+1}$, therefore the number of edges $e \in P(x,y)$ is bounded by C^{i+1} and hence the loop is executed at most $O(C^i)$ times. Each iteration of this loop uses the black-box DSO to compute $d(x,y,f)$ in $O(1)$, and only if $d(x,y,f) \in (C^{i-6}, C^{i+1}]$ then we use the DSO to actually obtain the path $P(x,y,f)$ in $O(|P(x,y,f)|) = O(C^i)$ time. This gives $\widetilde{O}(n^2)$ for the second-most outer loop. We have already seen that computing $\mathsf{GreedyPivotSelection}(\mathcal{P}_i)$ in line 12 takes $\widetilde{O}(n^2)$ time as well.

We claim that for all $s, t \in V$ and every edge or vertex $f \in E \cup V$ such that $d(s,t,f) \in (C^i, C^{i+1}]$, there exists a pivot $z \in B_i$ such that $d(s,t,f) = d(s,z,f) + d(z,t,f)$. That means, there is some s-t-replacement path that contains z. This is clear for $i \leqslant 2$. Let $3 \leqslant i \leqslant \lceil \log_C n \rceil$ and suppose the claim holds for every $j < i$. Let $P(s,t,f) = (v_0 = s, v_1, \ldots, v_k = t)$ be an replacement path with $k = d(s,t,f) \in (C^i, C^{i+1}]$. We define the prefix $P_1 = P(s,t,f)[s..v_{\lceil k/C^3 \rceil}]$ and suffix $P_2 = P(s,t,f)[v_{\lceil (1-1/C^3)k \rceil}..t]$. Both subpaths have length in $(C^{i-3}, C^{i-2}]$. It follows that there are pivots $x_1, x_2 \in B_{i-3}$ with $x_1 \in V(P_1)$, $x_2 \in V(P_2)$. (Strictly speaking, we are merely guaranteed *some* s-$v_{\lceil k/C^3 \rceil}$-replacement path that contains x_1, but we can choose $P(s,t,f)$ so that its prefix is that path; same with P_2.)

Let $P(x_1, x_2, f)$ be the replacement paths returned by the DSO on query (x_1, x_2, f). We claim that it is added to $\mathcal{P}_i$. Observe that $d(x_1, x_2, f) \geqslant d(s,t,f) - |P_1| - |P_2| \geqslant (1 - \frac{2}{C^2})C^i > C^{i-6}$, where we used the assumption $C \geqslant 3/2$ and thus $1 - \frac{2}{C^2} > C^{-6}$. Also, we have $d(x_1, x_2) \leqslant d(x_1, x_2, f) \leqslant d(s,t,f) \leqslant C^{i+1}$. If $d(x_1, x_2, f) = d(x_1, x_2)$, we may assume $P(x_1, x_2, f) = P(x_1, x_2)$, whence it was added in line 4. Otherwise, the failure $f \in V(P(x_1, x_2))$ is on the path. Since $x_1, x_2 \in B_{i-3} \cup B_{i-2} \cup B_{i-1}$, $d(x_1, x_2) \leqslant C^{i+1}$, and $d(x_1, x_2, f) \in (C^{i-6}, C^{i+1}]$ the path $P(x_1, x_2, f)$ is indeed added to $\mathcal{P}_i$ in line 11. Due to $B_i \leftarrow \mathsf{GreedyPivotSelection}(\mathcal{P}_i)$, there exists a vertex $z \in B_i$ such that z is on the path $P(x_1, x_2, f) \subseteq P(s,t,f)$ and thus $d(s,t,f) = d(s,z,f) + d(z,t,f)$.

The proof that for all $s, t \in V$ with $d(s,t) \in (C^i, C^{i+1}]$, there exists a pivot $z \in B_i$ such that $d(s,t) = d(s,z) + d(z,t)$ follows the same argument but is somewhat simpler because the subpaths P_1 and P_2 are guaranteed to be added in line 4. ◀

Derandomizing Theorems 2 and 3. Recall that the oracle in Lemma 12 has preprocessing time $\widetilde{O}(mn + n^4/(\varepsilon^3 \operatorname{diam}(G)^3))$. For its derandomization, and that of Theorems 2 and 3, it is enough to choose $C = 2$, compute APSP *only* in the original graph G, and preprocess the DSO of Bernstein and Karger[8] [4], which takes $\widetilde{O}(mn)$ time. Let i^* be the largest integer i such that $2^i < L = \varepsilon \operatorname{diam}(G)/2$. The set B_{i^*} then hits, for all $s, t \in V$ and $e \in E$ with $d(s,t,e) = \Theta(L)$, some replacement path $P(s,t,e)$, and it has the desired cardinality $\widetilde{O}(n/L)$.

5 Derandomizing Existing Sensitivity Oracles and Algorithms

We now show how the HDPH algorithm can be adapted to derandomize existing sensitivity oracles. In addition to our own technique, we also extensively use a recent breakthrough by Karthik and Parter [28] in the derandomization of fault-tolerant algorithms. We combine both tools and apply them to the distance sensitivity oracle of Ren [32] and the SSRP algorithm of Chechik and Magen [12] In the main part, we concentrate on the DSO because we think that it is a good illustration of the combination of our work and that of Karthik and Parter [28]. The treatment of the SSRP algorithm had to be deferred to the full version.

[8] Bernstein and Karger [4] derandomized their own DSO using a technique by King [29].

5.1 The Distance Sensitivity Oracle of Ren

We start with the oracle of Ren [32]. Recall that, for any two vertices $s, t \in V$ and edge $e \in E$, the replacement distance $d(s, t, e)$ is the length of a shortest s-t-path in $G - e$. A distance sensitivity oracle (DSO) is a data structure that answers query (s, t, e) with $d(s, t, e)$. Ren [32] presented an algebraic DSO with a randomized preprocessing time of $O(Mn^{2.7233})$ on graphs with positive integer edge weights in the range $[1, M]$ and $\widetilde{O}(Mn^{(\omega+3)/2}) = O(Mn^{2.6865})$ time on undirected graphs. Notably, this was the first DSO with both constant query time and subcubic preprocessing, improving over previous work [2, 4, 22, 35]. We derandomize it with a slight increase in running time and obtain a deterministic DSO in time $O(Mn^{2.8068})$ on directed graphs and $\widetilde{O}(Mn^{(\omega+6)/3}) = O(Mn^{2.7910})$ on undirected graphs.

The construction starts with a Core oracle that only reports very small distances, this is then grown iteratively to cover longer paths until the distance between all pairs of vertices are correctly determined. More formally, for a positive real r, let an *r-truncated DSO* report, upon query (s, t, e), the value $d(s, t, e)$ if it is at most r, and $+\infty$ otherwise. The Core is an n^α-truncated DSO for some carefully chosen exponent $\alpha \in (0, 1)$. Each iteration invokes the procedure Extend to turn an r-truncated DSO into an $(3/2)r$-truncated DSO. Note that we can assume $M = \widetilde{O}(n^{(3-\omega)/2})$ as otherwise the deterministic oracle in [4] with an $\widetilde{O}(mn)$ preprocessing time already achieves $\widetilde{O}(Mn^{(\omega+3)/2})$, even on directed weighted graphs. Hence, $\log_{3/2}(Mn) = O(\log n)$ rounds of growing suffice to built the full oracle.

The iterative approach has the advantage that r-truncated DSOs for small r can be computed fast. A bridging-set idea, see [36], is used for the extension. This significantly increases the query time as the oracle has to cycle through the whole bridging set to compute the distance. Ren [32] uses a clever observation, there attributed to Bernstein and Karger [4], to reduce the query time of the extended DSO back to a constant, called the Fast procedure.

Randomness is employed at two points. First, the Core uses a series random subgraphs of G. Secondly, Extend randomly samples a set of pivots to hit all replacement paths of length between r and $(3/2)r$. The subsequent reduction in query time is deterministic.[9] The Core can be derandomized using a recent result by Karthik and Parter [28]. To derandomize Extend, we adapt our technique introduced above. The key differences are that we now have to take care of the edge weights, that is, the number of vertices of a path may be much smaller than its length. Also, due to the iterative approach of not only the derandomization but the actual construction via truncated DSOs, we cannot assume to have access to all relevant paths right from the beginning. Instead, we have to make sure that all intermediary oracles are *path-reporting* and that for the construction of the current hitting set we only use paths of length at most r. The deterministic Core oracle hinges on the following lemma.

▶ **Lemma 15** (Karthik and Parter [28]). *Given a (possibly weighted) graph G on n vertices and a positive real $r = n^\alpha$ for some $\alpha \in (0, 1)$, there is a deterministic algorithm computing $k = O(r^2)$ spanning subgraphs $G_1, \dots, G_k$ of G in time $\widetilde{O}(kn^2)$ such that for any pair of vertices $s, t \in V$, edge $e \in E$, and replacement path $P(s, t, e)$ on at most r edges, there exists an index i such that G_i does not contain the edge e but all edges of $P(s, t, e)$.*

This derandomizes a construction by Weimann and Yuster [35] with the crucial difference that the latter is only required to produce subgraphs such that for all pairs of vertices s, t and edges e that admit possibly multiple replacement paths on at most r edges *at least one* (instead of all) of them survives in one of the graphs G_i in which e was removed. This

[9] The relevant [32, Section 3] is phrased as randomized, but based on the derandomizable algorithm in [4].

relaxed condition is actually enough to build an r-truncated DSO and allows one to make do with only $\widetilde{O}(r)$ random subgraphs, while we have $O(r^2)$ deterministic graphs. See also the discussion in Section 1.3 of [28]. This is the sole reason for the increased running time compared to the original result of Ren [32].

Given a graph G with integer edge weights in the range $[1, M]$, we invoke Lemma 15 to obtain the subgraphs G_i. Recall that $r = n^\alpha$ and let $\omega(1-\alpha)$ be the infimum over all w such that rectangular integer matrices with dimensions $n \times n^{1-\alpha}$ and $n^{1-\alpha} \times n$ can be multiplied using $O(n^w)$ arithmetic operations, $\omega = \omega(1)$ is the usual square matrix multiplication coefficient. Using a variant of Zwick's algorithm [36],[10] we compute APSP restricted to paths on at most r edges in time $\widetilde{O}(Mn^{\omega(1-\alpha)}r)$ per subgraph. If G is undirected, then this can be done faster, namely, in $\widetilde{O}(Mn^\omega)$ per graph with the algorithm of Shoshan and Zwick [34]. Both algorithms in [34, 36] can be adjusted to also compute the actual paths, represented as predecessor trees, which increases the running time only by logarithmic factors.

To answer a query (s, t, e) we cycle through the G_i and, in case the edge e is missing, retrieve the distance $d_{G_i}(s, t)$. By Lemma 15, the minimum over all retrieved distances is the correct replacement distance $d(s, t, e)$. If this minimum is larger than r or no distance has been retrieved at all (as the paths take more than r edges), we return $+\infty$. Since the edge weights are positive integers, every path of *length* at most r uses at most r edges, so we indeed obtain an r-truncated DSO. If an actual replacement path is requested, we return a shortest s-t-path in one of the G_i that attain the minimum. The resulting oracle has query time $\widetilde{O}(r^2)$ and a $\widetilde{O}(n^2r^2 + Mn^{\omega(1-\alpha)}r^3) = \widetilde{O}(Mn^{\omega(1-\alpha)}r^3)$ preprocessing time on directed graphs (using $\omega(1-\alpha) \geqslant 2$). On undirected graphs, this improves to $\widetilde{O}(n^2r^2 + Mn^\omega r^2) = \widetilde{O}(Mn^\omega r^2)$.

As a technical subtlety, the Fast procedure needed to reduce the query time requires *unique* shortest paths[11] of the original graph G. They can be computed in time $\widetilde{O}(M^{1/2}n^{(\omega+3)/2})$ [16, 32]. We will see later that this is not the bottleneck of the preprocessing.

▶ **Lemma 16** (Ren [32]). *From a directed graph G with integer edge weights in $[1, M]$, unique shortest paths, and an r-truncated DSO with preprocessing time P and query time Q, one can built in deterministic time $P + \widetilde{O}(n^2) \cdot Q$ a r-truncated DSO for G with $O(1)$ query time.*

Without access to unique paths, the running time increases to $P + \widetilde{O}(Mn^2) \cdot Q$, see [31]. If the oracle with query time Q (for the distance) is path-reporting (in $O(1)$ time per edge), then the new oracle is path-reporting with $O(1)$ query time (for distances and edges) [32].

We now turn to the main part, where we derandomize the Extend procedure that turns an r-truncated DSOs into $(3/2)r$-truncated DSOs. We adapt our technique to the iterative manner of construction and to the integer weights on the edges. In each stage, we only have access to a truncated DSO. Still, we show how to deterministically compute a sequence $B_1, B_2, \ldots$ of smaller and smaller sets, where B_i is used to derandomize the i-th application of Extend. Again, the construction of B_i depends on the previous sets of pivots, namely, on B_{i-2}. We first describe how to obtain the B_i satisfying certain useful properties and afterwards verify that these properties indeed suffice to make Extend deterministic.

▶ **Lemma 17.** *Let $r_1 \geqslant 1$ be a real number and define $r_{i+1} = (3/2)r_i$. For a graph G with integer edge weights in $[1, M]$, let $\{B_i\}_{i \geqslant 1}$ be a family of subsets of V, such that, for each i,*

(i) $|B_i| = \widetilde{O}(Mn/r_i)$;

[10] The algorithm in [36] is also phrased as randomized, in the same work it is explained how to derandomize it, increasing the running time only by polylogarithmic factors. The same holds for [34].

[11] By *unique shortest paths*, we mean a collection $\mathcal{P}$ containing one shortest path for each pair of vertices such that, for all $s, t \in V$, if $P(s, t) \in \mathcal{P}$ is the shortest path from s to t, then for every vertex u on $P(s, t)$, the path $P(s, u) \in \mathcal{P}$ is the prefix $P(s, t)[s..u]$ and $P(u, t) \in \mathcal{P}$ is the suffix $P(s, t)[u..t]$.

(ii) *for every pair of* $s,t \in V$ *and* $e \in E$ *with* $r_i/2 - M \leqslant d(s,t,e) \leqslant r_i$*, there exists a replacement path* $P(s,t,e)$ *that contains a vertex of* B_i*.*

With access to the shortest paths, a path-reporting r_i*-truncated DSO with* $O(1)$ *query time, and the sets* B_j *with* $j < i$*, one can compute each* B_i *deterministically in time* $\widetilde{O}(M^2n^2 + n^2r_1^2)$*.*

Proof. The proof is by induction over i. For the construction of B_i, we use the previous set B_{i-2}. We set $B_{-1} = B_0 = V$ to unify notation. Following the outline of the derandomization technique, we first assemble a set $\mathcal{P}$ of paths and then greedily compute a hitting set.

For each pair of vertices $x, y \in B_{i-2}$, we check whether the x-y-distance in the base graph G is at most r_i and, if so, retrieve a shortest path $P(x,y)$. If $P(x,y)$ additionally has length at least $r_i/18$, we add it to $\mathcal{P}$. For each edge e on $P(x,y)$ (regardless of the path being added to $\mathcal{P}$), we query the r_i-truncated DSO whether the replacement distance is $d(x,y,e) \in [\frac{r_i}{18}, r_i]$. If so, we request a corresponding replacement path $P(x,y,e)$ to add it to $\mathcal{P}$.

Due to the positive weights, those paths have at most r_i edges and can be obtained in time $O(r_i)$. Assembling $\mathcal{P}$ thus takes time $O(|B_{i-2}|^2 r_i^2)$. If $i \leqslant 2$, this is $O(n^2 r_1^2)$ since $r_2 = (3/2)r_1$. For $i \geqslant 3$, we get $\widetilde{O}((Mn/r_{i-2})^2 \cdot r_i^2) = \widetilde{O}(M^2n^2)$ instead.

We deterministically compute a hitting set B_i for $\mathcal{P}$. Since $\mathcal{P}$ contains at most $|B_{i-2}|^2 \cdot r_i$ paths with at least $r_i/(18M)$ edges each, whence $\Omega(r_i/M)$ vertices, the set B_i has $\widetilde{O}(n/(r_i/M)) = \widetilde{O}(Mn/r_i)$ vertices and is computable in time $\widetilde{O}(|\mathcal{P}| \cdot (r_i/M))$. As before, for $i \leqslant 2$, this is $\widetilde{O}(n^2r_1^2/M)$; and $\widetilde{O}(Mn^2)$ otherwise. We get a running time of $\widetilde{O}(M^2n^2 + n^2r_1^2)$.

It is left to prove that B_i indeed hits at least one replacement path for all $s,t \in V$ and $e \in E$ that satisfy $d(s,t,e) \in [\frac{r_i}{2} - M, r_i]$. Let $P(s,t,e)$ be such a path and define u to be the first vertex on $P(s,t,e)$ (starting from s) such that $d(s,u,e) \geqslant (2/9)r_i - M$. If $i \geqslant 3$, then $r_{i-2} = (4/9)r_i$, whence $d(s,u,e) \in [\frac{r_{i-2}}{2} - M, \frac{r_{i-2}}{2})$ and the induction hypothesis implies that there is *some* replacement path P' from s to u avoiding the edge e such that B_{i-2} contains one vertex of P'. Otherwise, if $i \leqslant 2$, the same fact simply follows from $B_{i-2} = V$.

The path P' is not necessarily equal to the prefix of $P(s,t,e)[s..u]$ (but they have the same length $d(s,u,e)$). Replacing $P(s,t,e)[s..u]$ with P' gives a new replacement path that now has a pivot $x \in B_{i-2}$ on its prefix. Slightly abusing notation, we use $P(s,t,e)$ to denote also the updated path. Let v be the last vertex on $P(s,t,e)$ with $d(v,t,e) \geqslant (2/9)r_i - M$. By the same argument, we can assume that the suffix of $P(s,t,e)[v..t]$ contains a pivot $y \in B_{i-2}$. In the remainder, we show that there is some replacement path $P(x,y,e)$ that is hit by a vertex in B_i. If so, replacing the middle part $P(s,t,e)[x..y]$ with $P(x,y,e)$ finally proves the existence of a replacement path from s to t avoiding e and containing a vertex of B_i.

By the choice of the pivots x, y and the assumption $d(s,t,e) \in [\frac{r_i}{2} - M, r_i]$, the replacement distance $d(x,y,e)$ satisfies

$$r_i \geqslant d(s,t,e) \geqslant d(x,y,e) \geqslant d(s,t,e) - d(s,u,e) - d(v,t,e) \geqslant d(s,t,e) - 2\left(\frac{2r_i}{9} - M\right) \geqslant \frac{r_i}{18}.$$

First, assume that the shortest path $P(x,y)$ in the base graph G does not contain the edge e. Then, $P(x,y)$ can serve as the replacement path. It has length $d(x,y) = d(x,y,e)$ between $r_i/18$ and r_i, and we added it to $\mathcal{P}$. Otherwise, it holds that $e \in P(x,y)$. Observe that $d(x,y) \leqslant d(x,y,e) \leqslant r_i$ remains true. Therefore, we have queried the r_i-truncated DSO with the triple (x,y,e). Due to $d(x,y,e) \geqslant r_i/18$, we received a replacement path $P(x,y,e)$, which we added to $\mathcal{P}$. In both cases, some replacement path is hit by B_i, as desired. ◀

At first glance, it looks like the quadratic dependence on M is too high to be used in the derandomization. However, recall that we can assume $M = \widetilde{O}(n^{(3-\omega)/2})$. Over the $O(\log n)$ rounds with $i \geqslant 3$ and with access to the appropriately truncated DSOs, we can compute the sets $B_3, B_4, \dots$ in total time $\widetilde{O}(M^2n^2) = \widetilde{O}(Mn^{(7-\omega)/2}) = \widetilde{O}(Mn^{2.5})$ even if $\omega = 2$.

The next lemma is the last tool we need to construct the deterministic DSO.

▶ **Lemma 18.** *Let G be a graph with integer edge weights in the range $[1, M]$, $r \geqslant 1$ a real number, and $B \subseteq V$ a set of $\widetilde{O}(Mn/r)$ vertices such that for every pair of $s, t \in V$ and $e \in E$ with $r/2 - M \leqslant d(s,t,e) \leqslant r$, there exists a replacement path $P(s,t,e)$ that contains a vertex of B. Given an r-truncated DSO for G with $O(1)$ query time and the set B, one can, without further preprocessing, construct an $(3/2)r$-truncated DSO with query time $\widetilde{O}(Mn/r)$. Moreover, if the r-truncated DSO is path-reporting, so is the $(3/2)r$-truncated one.*

Proof. For any query (s,t,e), let $D(s,t,e)$ denote the returned value by the r-truncated DSO. If $D(s,t,e) \neq +\infty$, we also take this as the answer of the $(3/2)r$-truncated DSO. Otherwise, define $\ell = \min_{z \in B}\{D(s,z,e) + D(z,t,e)\}$. If $\ell \leqslant (3/2)r$, we return ℓ, and $+\infty$ else. Path queries are handled in the same fashion. In the case of $D(s,t,e) \neq +\infty$, we pass on the path $P(s,t,e)$ returned by the r-truncated DSO. If $\ell \leqslant (3/2)r$, we return the concatenation of $P(s,z,e)$ and $P(z,t,e)$ for some pivot $z \in B$ that attains the minimum ℓ. The query time is $O(|B|) = \widetilde{O}(Mn/r)$ for the distance, after which the path can be returned in $O(1)$ per edge.

It is clear that the query algorithm is correct whenever $d(s,t,e) \leqslant r$ as those queries are entirely handled by the given truncated DSO. Moreover, even if $d(s,t,e) > r$, then ℓ is an upper bound for $d(s,t,e)$ because all sums $D(s,z,e) + D(z,t,e)$ correspond to some path from s to t avoiding e, but not necessarily a shortest path.

Let $P = P(s,t,e)$ be a replacement path of length between r and $(3/2)r$, u the first vertex on P (seen from s) with $d(u,t,e) \leqslant r$, and v the last vertex on P with $d(s,v,e) \leqslant r$. Note that v lies between u and t on the path, whence $d(u,v,e) \leqslant r$. We further have

$$d(u,v,e) \geqslant d(s,t,e) - d(s,u,e) - d(v,t,e) = d(u,t,e) + d(s,v,e) - d(s,t,e) \geqslant 2r - \frac{3}{2}r = \frac{r}{2}.$$

By the properties of B, there exists some replacement path from u to v avoiding e that contains a pivot $z \in B$. With the usual argument of swapping parts of the path, we can assume z lies on the middle section of $P(s,t,e)$ between u and v. By construction, we have $\max\{d(s,z,e), d(z,t,e)\} \leqslant r$ so both distances (and corresponding paths) are correctly determined by the r-truncated DSO. In summary, we get $\ell \leqslant d(s,z,e) + d(z,t,e) = d(s,t,e)$ and the returned value ℓ is indeed the correct replacement distance. ◀

We are left to prove the final running time of the construction. Let $r = n^\alpha$ be the cut-off point for the distances at which we start the iterative growing. We build the Core DSO using the $O(r^2)$ subgraphs, compute unique shortest paths in G, followed by $O(\log n)$ iterations of Extend and Fast invocations, including the computation of the B_i. First, suppose the graph G is undirected. The total time is then

$$\begin{aligned}
&\widetilde{O}(Mn^\omega r^2) + \widetilde{O}(M^{1/2} n^{(\omega+3)/2}) + \widetilde{O}(Mn^{2.5} + n^2 r^2) + \widetilde{O}(n^2) \cdot \sum_{i=1}^{O(\log n)} \widetilde{O}\left(\frac{Mn}{(3/2)^i r}\right) \\
&= \widetilde{O}\left(M^{1/2} n^{(\omega+3)/2} + Mn^{2.5} + Mn^\omega r^2 + \frac{Mn^3}{r}\right) \\
&= \widetilde{O}\left(M^{1/2} n^{(\omega+3)/2} + Mn^{\max\{2.5,\ \omega+2\alpha,\ 3-\alpha\}}\right).
\end{aligned}$$

This is minimum for $\alpha = 1 - (\omega/3)$, where we get a running time of $\widetilde{O}(Mn^{(\omega+6)/3})$.

For directed graphs, determining the best α is a bit more involved. Recall that $O(n^{\omega(1-\alpha)})$ is the time needed to multiply a $n \times n^{1-\alpha}$ matrix with an $n^{1-\alpha} \times n$ matrix. Computing the Core oracle takes time $\widetilde{O}(Mn^{\omega(1-\alpha)}r^3) = \widetilde{O}(Mn^{\omega(1-\alpha)+3\alpha})$. With a similar calculation as above, we obtain a total preprocessing time of $\widetilde{O}\big(M^{1/2}n^{(\omega+3)/2} + Mn^{\max\{2.5,\ \omega(1-\alpha)+3\alpha,\ 3-\alpha\}}\big)$. This is minimized if α solves the equation $\omega(1-\alpha) = 3-4\alpha$. Le Gall and Urrutia [19] gave the current-best estimates for the values of the function ω. This shows that $1-\alpha$ is between 0.8 and 0.85, and we have $\omega(0.8) \leqslant 2.222256$ as well as $\omega(0.85) \leqslant 2.258317$. We exploit the fact that ω is convex [30], giving

$$\omega(1-\alpha) \leqslant \frac{(\alpha-0.15)\omega(0.8) + (0.2-\alpha)\omega(0.85)}{0.85-0.8} \leqslant 2.3665 - 0.72122\alpha$$

Equating the latter with $3-4\alpha$ yields the estimate $\alpha \leqslant 0.193212$, which in turn implies a preprocessing time of $\widetilde{O}(Mn^{2.806788})$.

References

1 Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.

2 Noga Alon, Shiri Chechik, and Sarel Cohen. Deterministic Combinatorial Replacement Paths and Distance Sensitivity Oracles. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 12:1–12:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.12.

3 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault-tolerant subgraph for single-source reachability: General and optimal. *SIAM Journal on Computing*, 47:80–95, 2018. doi:10.1137/16M1087643.

4 Aaron Bernstein and David R. Karger. A Nearly Optimal Oracle for Avoiding Failed Vertices and Edges. In *Proceedings of the 41st Symposium on Theory of Computing (STOC)*, pages 101–110, 2009. doi:10.1145/1536414.1536431.

5 Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Near-Optimal Deterministic Single-Source Distance Sensitivity Oracles. In *Proceedings of the 29th European Symposium on Algorithms (ESA)*, pages 18:1–18:17, 2021. doi:10.4230/LIPIcs.ESA.2021.18.

6 Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Space-Efficient Fault-Tolerant Diameter Oracles. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 18:1–18:16, 2021. doi:10.4230/LIPIcs.MFCS.2021.18.

7 Greg Bodwin. Linear Size Distance Preservers. In *Proceedings of the 28th Symposium on Discrete Algorithms (SODA)*, pages 600–615, 2017. URL: http://dl.acm.org/citation.cfm?id=3039686.3039725.

8 Jan van den Brand and Thatchaphol Saranurak. Sensitive Distance and Reachability Oracles for Large Batch Updates. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, pages 424–435, 2019. doi:10.1109/FOCS.2019.00034.

9 Shiri Chechik and Sarel Cohen. Near Optimal Algorithms for the Single Source Replacement Paths Problem. In *Proceedings of the 30th Symposium on Discrete Algorithms (SODA)*, pages 2090–2109, 2019. doi:10.1137/1.9781611975482.126.

10 Shiri Chechik and Sarel Cohen. Distance Sensitivity Oracles with Subcubic Preprocessing Time and Fast Query Time. In *Proccedings of the 52nd Symposium on Theory of Computing (STOC)*, pages 1375–1388, 2020. doi:10.1145/3357713.3384253.

11 Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1+\epsilon)$-Approximate f-Sensitive Distance Oracles. In *Proceedings of the 28th Symposium on Discrete Algorithms (SODA)*, pages 1479–1496, 2017. doi:10.1137/1.9781611974782.96.

12 Shiri Chechik and Ofer Magen. Near Optimal Algorithm for the Directed Single Source Replacement Paths Problem. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 81:1–81:17, 2020. doi:10.4230/LIPIcs.ICALP.2020.81.

13 Camil Demetrescu, Mikkel Thorup, Rezaul A. Chowdhury, and Vijaya Ramachandran. Oracles for Distances Avoiding a Failed Node or Link. *SIAM Journal on Computing*, 37:1299–1318, 2008. doi:10.1137/S0097539705429847.

14 Ran Duan, Yong Gu, and Hanlin Ren. Approximate Distance Oracles Subject to Multiple Vertex Failures. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*, pages 2497–2516, 2021. doi:10.1137/1.9781611976465.148.

15 Ran Duan and Seth Pettie. Dual-Failure Distance and Connectivity Oracles. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*, pages 506–515, 2009. URL: https://dl.acm.org/citation.cfm?id=1496770.1496826.

16 Ran Duan and Seth Pettie. Fast Algorithms for (max,min)-Matrix Multiplication and Bottleneck Shortest Paths. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*, pages 384–391, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496813.

17 Ran Duan and Hanlin Ren. Maintaining Exact Distances under Multiple Edge Failures. In *Proceedings of the 54th Symposium on Theory of Computing (STOC)*, 2022. To appear.

18 Ran Duan and Tianyi Zhang. Improved Distance Sensitivity Oracles via Tree Partitioning. In *Proceedings of the 15th International Symposium on Algorithms and Data Structures (WADS)*, pages 349–360, 2017. doi:10.1007/978-3-319-62127-2_30.

19 François Le Gall and Florent Urrutia. Improved Rectangular Matrix Multiplication using Powers of the Coppersmith-Winograd Tensor. In *Proceedings of the 29th Symposium on Discrete Algorithms (SODA)*, pages 1029–1046, 2018. doi:10.1137/1.9781611975031.67.

20 François Le Gall. Faster Algorithms for Rectangular Matrix Multiplication. In *Proceedings of the 53rd Symposium on Foundations of Computer Science (FOCS)*, pages 514–523, 2012. doi:10.1109/FOCS.2012.80.

21 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved Distance Sensitivity Oracles via Fast Single-Source Replacement Paths. In *Proceedings of the 53rd Symposium on Foundations of Computer Science (FOCS)*, pages 748–757, 2012. doi:10.1109/FOCS.2012.17.

22 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster Replacement Paths and Distance Sensitivity Oracles. *ACM Transaction on Algorithms*, 16:15:1–15:25, 2020. doi:10.1145/3365835.

23 Yong Gu and Hanlin Ren. Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 76:1–76:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.76.

24 Manoj Gupta, Rahul Jain, and Nitiksha Modi. Multiple Source Replacement Path Problem. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 339–348, 2020. doi:10.1145/3382734.3405714.

25 Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh. Deterministic Dictionaries. *Journal of Algorithms*, 41:69–85, 2001. doi:10.1006/jagm.2001.1171.

26 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional Hardness for Sensitivity Problems. In *Proceedings of the 8th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 26:1–26:31, 2017. doi:10.4230/LIPIcs.ITCS.2017.26.

27 Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Finding Strong Bridges and Strong Articulation Points in Linear Time. *Theoretical Computer Science*, 447:74–84, 2012. doi:10.1016/j.tcs.2011.11.011.

28 Karthik C.S. and Merav Parter. Deterministic Replacement Path Covering. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*, pages 704–723, 2021. doi:10.1137/1.9781611976465.44.

29 Valerie King. Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999. doi:10.1109/SFFCS.1999.814580.

30 Grazia Lotti and Francesco Romani. On the Asymptotic Complexity of Rectangular Matrix Multiplication. *Theoretical Computer Science*, 23:171–185, 1983. doi:10.1016/0304-3975(83)90054-3.

31 Hanlin Ren. Improved Distance Sensitivity Oracles with Subcubic Preprocessing Time. In *Proceedings of the 28th European Symposium on Algorithms (ESA)*, pages 79:1–79:13, 2020. doi:10.4230/LIPIcs.ESA.2020.79.

32 Hanlin Ren. Improved Distance Sensitivity Oracles with Subcubic Preprocessing Time. *Journal of Computer and System Sciences*, 123:159–170, 2022. Journal version of [31]. doi:10.1016/j.jcss.2021.08.005.

33 Liam Roditty and Uri Zwick. Replacement Paths and k Simple Shortest Paths in Unweighted Directed Graphs. *ACM Transaction on Algorithms*, 8:33:1–33:11, 2012. doi:10.1145/2344422.2344423.

34 Avi Shoshan and Uri Zwick. All Pairs Shortest Paths in Undirected Graphs with Integer Weights. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 605–615, 1999. doi:10.1109/SFFCS.1999.814635.

35 Oren Weimann and Raphael Yuster. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Transactions on Algorithms*, 9:14:1–14:13, 2013. doi:10.1145/2438645.2438646.

36 Uri Zwick. All Pairs Shortest Paths Using Bridging Sets and Rectangular Matrix Multiplication. *Journal of the ACM*, 49:289–317, 2002. doi:10.1145/567112.567114.

Hodge Decomposition and General Laplacian Solvers for Embedded Simplicial Complexes

Mitchell Black
School of Electrical Engineering and Computer Science,
Oregon State University, Corvallis, OR, USA

Amir Nayyeri
School of Electrical Engineering and Computer Science,
Oregon State University, Corvallis, OR, USA

Abstract

We describe a nearly-linear time algorithm to solve the linear system $L_1 x = b$ parameterized by the first Betti number of the complex, where L_1 is the 1-Laplacian of a simplicial complex K that is a subcomplex of a collapsible complex X linearly embedded in $\mathbb{R}^3$. Our algorithm generalizes the work of Black et al. [SODA2022] that solved the same problem but required that K have trivial first homology. Our algorithm works for complexes K with arbitrary first homology with running time that is nearly-linear with respect to the size of the complex and polynomial with respect to the first Betti number. The key to our solver is a new algorithm for computing the Hodge decomposition of 1-chains of K in nearly-linear time. Additionally, our algorithm implies a nearly quadratic solver and nearly quadratic Hodge decomposition for the 1-Laplacian of any simplicial complex K embedded in $\mathbb{R}^3$, as K can always be expanded to a collapsible embedded complex of quadratic complexity.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Algebraic topology; Theory of computation → Design and analysis of algorithms; Mathematics of computing → Numerical analysis

Keywords and phrases Computational Topology, Laplacian solvers, Combinatorial Laplacian, Hodge decomposition, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.23

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2205.02134`

Funding The authors were supported in part by NSF grants CCF-1941086 and CCF-1816442.

Acknowledgements The authors would like to thank the reviewers for their helpful comments, especially for an observation that improved the dependence on β in the runtime.

1 Introduction

The dth combinatorial Laplacian of a simplicial complex K is a linear operator that acts on vectors of real numbers associated to the d-simplices of K. The dth combinatorial Laplacian is defined as

$$L_d = \partial_d^T \partial_d + \partial_{d+1} \partial_{d+1}^T,$$

where $\partial_d : C_d(K) \to C_{d-1}(K)$ is the dth boundary map of K, and $C_d(K)$ is the dth chain group of K. The dth Laplacian encodes the incidence of $(d-1)$-, d- and $(d+1)$-simplices. In particular, the 0th Laplacian L_0 is composed of a constant map $\partial_0^T \partial_0$, and the well-known graph Laplacian $\partial_1 \partial_1^T$. The graph Laplacian matrix and its algebraic properties have been extensively studied in algebraic and spectral graph theory, a topic that has flourished into a rich field with many applications in computer science such as graph clustering [25, 27], graph sparsification [29], and max flow solvers [9] (see Spielman's book and references therein [28]).

A highlight of recent advances in algorithmic spectral graph theory is nearly-linear time solvers for linear systems on the graph Laplacian that emerged as a result of decades of research [2, 4, 11, 19, 21–23, 30, 34]. These results imply nearly-linear time solvers for the more general class of symmetric diagonally dominant matrices. They also have triggered research to find out which classes of linear systems admit nearly-linear time solvers [24]. Moreover, these solvers are used for different application areas such as approximation algorithm design and numerical analysis [5, 9].

Recent work has attempted to extend the success of graph Laplacian solvers to higher dimensional Laplacians. Cohen et al. initiated this line of work by introducing a nearly-linear solver for the 1-Laplacian of collapsible complexes embedded in $\mathbb{R}^3$ [10]. Black et al. continued this work by considering complexes with trivial first homology that were subcomplexes of collapsible complexes embedded in $\mathbb{R}^3$ [3]. The solver of Black et al. implies a nearly quadratic solver for any complex with trivial first homology embedded in $\mathbb{R}^3$; they show that a complex embedded in $\mathbb{R}^3$ can be extended to a collapsible embedded complex with at most quadratic complexity.

In this paper, we extend the work of Cohen et al. and Black et al. to any subcomplex of a collapsible complex embedded in $\mathbb{R}^3$, regardless of the rank of its first homology group. The running time of our solver is nearly-linear with respect to the size of the collapsible complex, and polynomial with respect to the rank of its first homology group. The main tool in our paper is a new algorithm for computing the Hodge Decomposition of a 1-chain.

Computing the Hodge decomposition is a problem of independent interest since the Hodge decomposition has found a myriad of applications in topological data analysis, numerical analysis, and computer graphics among other areas [1, 8, 12, 13, 16, 20, 32, 33, 35]. The Hodge decomposition can be computed exactly in $O(n^\omega)$ time by solving a constant number of systems of linear equations, where ω is the matrix multiplication constant. (Approximately) computing the Hodge decomposition in nearly-linear time has been an open question with many possible applications.

Cohen et al. describe nearly-linear projection operators into the coboundary space and cycle space, which implies Hodge decomposition for complexes with trivial homology as the boundary and cycle spaces are identical in this case. In this paper, we describe projection operators into the boundary and harmonic spaces for an arbitrary subcomplex of a collapsible simplicial complex embedded in $\mathbb{R}^3$. Our boundary projection operator is key to our solver. Our results imply 1-Laplacian solvers and projection operators for any simplicial complex embedded in $\mathbb{R}^3$ that are quadratic in the size of the complex and polynomial in the first Betti number; these follow from the fact that any complex in $\mathbb{R}^3$ can be extended to a collapsible complex in $\mathbb{R}^3$ with a quadratic number of simplices [3, Corollary 3.3].

While this paper presents a positive result on extending graph Laplacian solvers to a more general class of Laplacians, a recent work by Ding et al. [15] shows that solving linear equations in arbitrary 1-Laplacians (and therefore arbitrary d-Laplacians) is as hard as solving arbitrary sparse linear equations with bounded integer entries and bounded condition number. An interesting open question is whether or not there exist fast solvers for other classes of simplicial complexes.

1.1 Our Results

Let X be a collapsible simplicial complex with a known collapsing sequence embedded in $\mathbb{R}^3$, and let K be a subcomplex of X. The first result of this paper is a 1-Laplacian solver for K. Recall that $L_1 = \partial_2\partial_2^T + \partial_1^T\partial_1$. We define $L_1^{up} = \partial_2\partial_2^T$ and $L_1^{down} = \partial_1^T\partial_1$. We refer to L_1^{up} and L_1^{down} as the up-Laplacian and down-Laplacian, respectively.

▶ **Theorem 1.** *Let X be a collapsible simplicial complex with a known collapsing sequence linearly embedded in $\mathbb{R}^3$, and let $K \subset X$ be a subcomplex of X. For any $\varepsilon > 0$, there is an operator $LaplacianSolver(X, K, \varepsilon)$ such that*

$$(1-\varepsilon)(L_1[K])^+ \preceq LaplacianSolver(X, K, \varepsilon) \preceq (L_1[K])^+.$$

where $(L_1[K])^+$ is the pseudoinverse of the 1-Laplacian $L_1[K]$. Further, for any $x \in C_1$, $LaplacianSolver(X, K, \varepsilon) \cdot x$ can be computed in $\widetilde{O}\left(\beta^3 \cdot n \cdot \log n \cdot \log(n/(\lambda_{\min}(K) \cdot \lambda_{\min}(X) \cdot \varepsilon))\right)$ [1] time, where n is the total number of simplices in X, $\lambda_{\min}(K)$ and $\lambda_{\min}(X)$ are the smallest nonzero eigenvalues of $L_1^{up}(K)$ and $L_1^{up}(X)$ respectively, and β is the rank of the first homology group of K.

This result is a generalization of Theorem 1.1 of Black et al. [3] that requires K to have trivial first homology. Their running time depends on $\log(n\kappa/\varepsilon)$, with κ being the condition number of $L_1^{up}(K)$ within the boundary space. The condition number is defined $\kappa = \lambda_{\max}(K)/\lambda_{\min}(K)$, where $\lambda_{\max}(K)$ is the largest eigenvalue of $L_1^{up}(K)$, and $\lambda_{\min}(K)$ is the smallest *nonzero* eigenvalue of $L_1^{up}(K)$. We observe that $\lambda_{\max}(K)$ is polynomially bounded with respect to the size of the complex (Lemma 29 in the full paper.) Therefore, the log dependence of the running time of Black et al.'s solver can be simplified to $\log(n/(\lambda_{\min}(K)\cdot\varepsilon))$. The running time of Theorem 1, in contrast, has an extra dependence to $\lambda_{\min}(X)$ within the log, in addition to a polynomial dependence to β. For the special case that $\beta = 0$, we can eliminate the dependence on $\lambda_{\min}(X)$ with a more careful analysis and match the running time of Black et al.

The new ingredient that makes Theorem 1 possible is an approximate projection operator onto the boundary space. Lacking this operator, the previous papers had to assume that K has trivial homology and use a projection into the cycle space instead.

▶ **Lemma 2.** *Let K be a simplicial complex linearly embedded in a collapsible complex X with a known collapsing sequence that is embedded in $\mathbb{R}^3$, and let Π_{bd} be the orthogonal projection operator into the space of boundary 1-chains in K. For any $\varepsilon > 0$, there is an operator $\widetilde{\Pi}_{bd}(\varepsilon)$, such that*

$$(1-\varepsilon)\Pi_{bd} \preceq \widetilde{\Pi}_{bd}(\varepsilon) \preceq (1+\varepsilon)\Pi_{bd}.$$

Further, for any 1-chain x, $\widetilde{\Pi}_{bd}(\varepsilon) \cdot x$ can be computed in $\tilde{O}(\beta^3 \cdot n \cdot \log n \cdot \log(\frac{n}{\lambda_{\min}(X)\cdot\varepsilon}))$ time, where β is the rank of the first homology group of K, n is the total number of simplices in X, and $\lambda_{\min}(X)$ is the smallest nonzero eigenvalue of $L_1^{up}(X)$.

A key technical challenge to achieve our projection operator onto the boundary space is computing a projection into the space of harmonic chains, formalized in part (ii) of the following lemma. Note that our approximation guarantee for projection into the harmonic space is weaker than the one for projection into the boundary space (more on this in the overview).

▶ **Lemma 3.** *Let K be a subcomplex of a collapsible simplicial complex X with a known collapsing sequence that is linearly embedded in $\mathbb{R}^3$. Let β be the rank of the first homology group of K, n be the total number of simplices in X, and $\lambda_{\min}(X)$ be the smallest nonzero eigenvalue of $L_1^{up}(X)$.*

[1] The $\tilde{O}(\cdot)$ notations hides a factor of $\log\log n$.

(i) *For any $\varepsilon > 0$, there is an $\tilde{O}(\beta^2 \cdot n \cdot \log n \cdot \log(\frac{n}{\lambda_{min}(X)\cdot\varepsilon}))$ time algorithm to compute an orthonormal set of vectors $\{\widetilde{g}_1, \ldots, \widetilde{g}_\beta\}$ such that there exists an orthonormal harmonic basis $\{g_1, \ldots, g_\beta\}$ with $\|g_i - \widetilde{g}_i\| \leq \varepsilon$ for all $1 \leq i \leq \beta$.*

(ii) *For any $\varepsilon > 0$, there exists a symmetric matrix $\widetilde{\Pi}_{hr}(\varepsilon)$ such that,*

$$\Pi_{hr} - \varepsilon I \preceq \widetilde{\Pi}_{hr}(\varepsilon) \preceq \Pi_{hr} + \varepsilon I,$$

where Π_{hr} is the orthogonal projection into the harmonic space. Moreover, for any 1-chain x, $\widetilde{\Pi}_{hr}(\varepsilon) \cdot x$ can be computed in $\tilde{O}(\beta^2 \cdot n \cdot \log n \cdot \log(\frac{n}{\lambda_{min}(X)\cdot\varepsilon}))$ time.

Our projection operators into the harmonic and boundary space, along with the projection operator of Cohen et al. [10] into the coboundary space, give all the projection operators needed to compute the Hodge decomposition of 1-chains in K.

Our harmonic projection operator is built using an orthonormal approximate harmonic basis (part (i) of Lemma 3). Dey [14] describes a nearly-linear time algorithm for computing a homology basis for a complex linearly embedded in $\mathbb{R}^3$. Black et al. [3] describe a nearly-linear time algorithm for computing a cohomology basis for subcomplexes of collapsible complexes embedded in $\mathbb{R}^3$. Our harmonic basis, though approximate, can be viewed as a complement to these two results.

1.2 Paper organization

In addition to this introduction, the main body of this paper is a background and overview section. To simplify the presentation, the bulk of the technical details are left for the full version of this paper, and the overview provides a high-level description of our approaches as well as the technical challenges and contribution of this paper. In the overview, we included references to the technical lemmas to enable easy access to the full paper.

The background section introduces standard definitions of the concepts used in this paper. We hope this section provides easy lookup for the reader while reading the overview section as well as the technical part of the paper.

2 Background

In this section, we review basic definitions from linear algebra and algebraic and combinatorial topology that are used in this paper; see references [7, 17, 18, 31] for further background.

2.1 Linear Algebra

Span, Basis. Let $V = \{v_1, \ldots, v_k\}$ be a set of vectors in $\mathbb{R}^n$. The **span** of V, denoted $\text{span}(V)$, is the subspace of $\mathbb{R}^n$ of all linear combinations of V. In particular, V spans $\mathbb{R}^n$ if any vector in $\mathbb{R}^n$ is a linear combination of the vectors in V. We say that V is a **basis** for its span if the dimension of its span equals the cardinality of V.

Linear map, projection, inverse. Let $A : \mathbb{R}^n \to \mathbb{R}^m$ be a linear map, represented by an $m \times n$ matrix. Typically, we don't make a distinction between a linear map and its matrix representation and denote both as A. The **kernel** of A is $\ker(A) := \{x \in \mathbb{R}^n : Ax = 0\}$, and the **image** of A is $\text{im}(A) = \{Ax : x \in \mathbb{R}^m\}$. The **rank** of a linear map is the dimension of its image.

We say that U and V **orthogonally decompose** W, denoted $W = U \oplus V$, if (i) any vector in U is orthogonal to any vector in V, and (ii) any vector in $x \in W$ is a unique sum of vectors in $x_U \in U$ and $x_V \in V$, i.e. $x = x_U + x_V$. The **fundamental theorem of linear**

algebra states that $\mathbb{R}^n = \text{im}(A^T) \oplus \ker(A)$ and $\mathbb{R}^m = \text{im}(A) \oplus \ker(A^T)$, where A^T is the transpose of A obtained by flipping A over its diagonal. In particular, if $A : \mathbb{R}^n \to \mathbb{R}^n$ is symmetric (i.e., $A = A^T$), then $\mathbb{R}^n = \text{im}(A) \oplus \ker(A)$.

A linear map $A : \mathbb{R}^n \to \mathbb{R}^n$ is a **projection** if it is the identity for the vectors in its image, or equivalently, $AA = A$. The map A is an **orthogonal projection** if it maps each point of $\mathbb{R}^n$ to its closest point in $\text{im}(A)$, or equivalently, $A^T = A = AA$. Note for any subspace U of $\mathbb{R}^n$ there is a unique orthogonal projection into U, denoted Π_U. If $\{u_1, \ldots, u_k\}$ is an orthonormal basis for U, the orthogonal projection into U is the linear map $\Pi_U = \sum_{i=1}^{k} u_i u_i^T$.

If a linear map $A : \mathbb{R}^n \to \mathbb{R}^m$ is bijective, it has a well-defined **inverse** denoted $A^{-1} : \mathbb{R}^m \to \mathbb{R}^n$ where $Ax = b \iff A^{-1}b = x$. More generally, the **pseudoinverse** of A is the unique linear map $A^+ : \mathbb{R}^m \to \mathbb{R}^n$ with the following properties: (i) $AA^+A = A$, (ii) $A^+AA^+ = A^+$, (iii) $(AA^+)^T = AA^+$, and (iv) $(A^+A)^T = A^+A$. Admittedly, the definition of the pseudoinverse is not very intuitive. A more intuitive description is that A^+ is the unique linear map with the following properties: (1) A^+ maps any vector $y \in \text{im}(A)$ to the unique vector $x \in \text{im}(A^T)$ such that $Ax = y$, and (2) A^+ maps any vector $y \in \ker(A^T)$ to 0. While it is not true in general that $(A + B)^+ = B^+ + A^+$ for linear maps A and B, this is true if $A^TB = B^TA = 0$; see Campbell [7], Theorem 3.1.1.

Matrix norm, singular values, Loewner order. A symmetric matrix A is **positive semi-definite** if $x^TAx \geq 0$ for each $x \in \mathbb{R}^n$. The **Loewner Order** is a partial order on the set of $n \times n$ symmetric matrices. For symmetric matrices A and B, we say $A \preceq B$ if $B - A$ is positive semidefinite.

Let $x \in \mathbb{R}^n$. Let p be a positive integer. The **p-norm** of x is $\|x\|_p = (\sum_{i=1}^{n} |x[i]|^p)^{\frac{1}{p}}$. We use the 1-norm and 2-norm in this paper. An important fact we will use throughout this paper is that $\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$. For any norm $\|\cdot\|$ on $\mathbb{R}^n$, there is an accompanying **operator norm** of a matrix A defined $\|A\| = \max_{x:\|x\|=1} \|Ax\|$, or equivalently, $\|A\| = \max_{x \neq 0} (\|Ax\|/\|x\|)$. Unless otherwise specified, all norms in this paper will be the 2-norm.

The **singular value decomposition** of $A : \mathbb{R}^n \to \mathbb{R}^m$ for $m \geq n$ (resp. $m \leq n$) is a set of n (resp. m) orthornomal vectors $\{u_1, \ldots, u_n\} \subset \mathbb{R}^m$ called **left singular vectors**, n (resp. m) orthornomal vectors $\{v_1, \ldots, v_n\} \subset \mathbb{R}^n$ called **right singular vectors**, and n (resp m) real numbers $\{\sigma_1, \ldots, \sigma_n\} \subset \mathbb{R}$ called **singular values** such that $A = \sum_{i=1}^{n} \sigma_i u_i v_i^T$. The **condition number** of a linear map $A : \mathbb{R}^n \to \mathbb{R}^n$ is $\kappa(A) = |\sigma_{\max}(A)|/|\sigma_{\min}(A)|$, where $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ are the largest and smallest non-zero singular values of A.

The **eigenvectors** and **eigenvalues** of a matrix $A : \mathbb{R}^n \to \mathbb{R}^n$ are n vectors $\{v_1, \ldots, v_n\} \subset \mathbb{R}^n$ and n real numbers $\{\lambda_1, \ldots, \lambda_n\}$ such that $Av_i = \lambda_i v_i$. The singular values and right singular vectors (resp. left singular values) of a matrix $A : \mathbb{R}^n \to \mathbb{R}^n$ are the square roots of the eigenvalues and eigenvectors of A^TA (resp. AA^T). If a matrix A is symmetric, the eigenvectors of A are orthogonal, and the eigenvectors and eigenvalues of A are the left and right singular vectors and the singular values.

Determinant, Cramer's rule, unimodularity. For any $1 \leq i \leq n$, the **determinant** of an $n \times n$ matrix $A = [a_{i,j}]_{1 \leq i,j \leq n}$ can be defined via its **Laplace expansion** as $\det(A) = \sum_{j=1}^{n} \left((-1)^{i+j} \cdot a_{i,j} \cdot \det(A_{i,j})\right)$, where $A_{i,j}$ is the $(n-1) \times (n-1)$ matrix obtained by removing the ith row and jth column of A. It is well known that $\det(A) \neq 0$ if and only if A is bijective. In that case, **Cramer's rule** give an explicit formula for the solution of the linear system $Ax = b$, which is $x[i] = \det(A_i)/\det(A)$ where A_i is the matrix obtained by replacing the i^{th} column of A with b.

An $n \times n$ matrix A is **unimodular** if $\det(A) \in \{-1, +1\}$. By Cramer's rule, $Ax = b$ has an integer solution if A is unimodular and A and b have integer coefficients. An $n \times m$ matrix B is **totally unimodular** if for any square submatrix A of B, $\det(A) \in \{-1, 0, +1\}$. The 1-boundary matrix of a simplicial complex (defined below) is totally unimodular [26].

2.2 Topology

Simplicial complexes. A **simplicial complex** K is a set of finite sets such that if $\tau \in K$ and $\sigma \subset \tau$, then $\sigma \in K$. A **subcomplex** of K is a subset $L \subset K$ such that L is a simplicial complex. The **vertices** of K is the set $\cup_{\sigma \in K} \sigma$. We assume there is a fixed but arbitrary order $(v_1, \ldots, v_n)$ on the vertices of K.

An element $\sigma \in K$ with $|\sigma| = d + 1$ is a **d-simplex**. A 0-simplex is a **vertex**, a 1-simplex is an **edge**, a 2-simplex is a **triangle**, and a 3-simplex is a **tetrahedron**. The set of all d-simplices in K is denoted K_d. For two simplices $\tau \subset \sigma$, we say that τ is a **face** of σ.

Hodge decomposition, homology, cohomology. The $\mathbf{d^{th}}$ **chain group** of a simplicial complex K is the vector space $C_d(K)$ over $\mathbb{R}$ with orthonormal basis K_d, and an element of $C_d(K)$ is a **d-chain**. The $\mathbf{d^{th}}$ **boundary map** is the linear map $\partial_d : C_d(K) \to C_{d-1}(K)$ defined $\partial_d \sigma = \sum_{i=0}^{d} (-1)^i (\sigma \setminus \{v_{k_i}\})$ for each simplex $\sigma = \{v_{k_0}, \ldots, v_{k_d}\} \in K_d$, where we assume $v_{k_i} < v_{k_j}$ for $i < j$. The $\mathbf{d^{th}}$ **coboundary map** is $\partial_{d+1}^T : C_d(K) \to C_{d+1}(K)$. Elements of $\ker \partial_d$ (resp. $\ker \partial_{d+1}^T$) are **cycles** (resp. **cocyles**), and elements of $\operatorname{im} \partial_{d+1}$ (resp. $\operatorname{im} \partial_d^T$) are **boundaries** or **null-homologous cycles** (resp. **coboundaries**.) Two cycles (resp. cocycles) γ_1 and γ_2 are **homologous** (resp. **cohomologous**) if their difference $\gamma_1 - \gamma_2$ is a boundary (resp. coboundary.)

The $\mathbf{d^{th}}$ **Laplacian** is the linear map $L_d : C_d(K) \to C_d(K)$ defined $L_d = \partial_d^T \partial_d + \partial_{d+1} \partial_{d+1}^T$. The $\mathbf{d^{th}}$ **up-Laplacian** is the linear map $L_d^{up} = \partial_{d+1} \partial_{d+1}^T$, and the $\mathbf{d^{th}}$ **down-Laplacian** is the linear map $L_d^{down} = \partial_d^T \partial_d$.

A key fact of algebraic topology is that $\partial_d \partial_{d+1} = 0$, hence $\operatorname{im} \partial_{d+1} \subset \ker \partial_d$, and $\operatorname{im} \partial_d^T \subset \ker \partial_{d+1}^T$. The $\mathbf{d^{th}}$ **homology group** is the quotient group $H_d(K) = \ker \partial_d / \operatorname{im} \partial_{d+1}$, and the $\mathbf{d^{th}}$ **cohomology group** is the quotient group $H^d(K) = \ker \partial_{d+1}^T / \operatorname{im} \partial_d^T$. Since $\operatorname{im} \partial_d^T \oplus \ker \partial_d$ and $\operatorname{im} \partial_{d+1} \oplus \ker \partial_{d+1}^T$ are two orthogonal decompositions of the d-chain space, the d^{th} homology group and the d^{th} cohomology group have the same rank, which is the $\mathbf{d^{th}}$ **Betti number** of the complex, denoted $\beta_d(K)$. We say two cycles are **homologous** (resp. **cohomologous**) if they are in the same homology (resp. cohomology) class, or equivalently, if their difference is a boundary (resp. coboundary.)

The **Hodge Decomposition** is the orthogonal decomposition of the d^{th} chain group into $C_d(K) = \operatorname{im}(\partial_{d+1}) \oplus \ker(L_d) \oplus \operatorname{im}(\partial_d^T)$. The subspace $\ker(L_d)$ are the **harmonic chains**. Thus, any chain $x \in C_d(K)$ can be uniquely written as the sum $x = x_{bd} + x_{hr} + x_{cbd}$ where $x_{bd} \in \operatorname{im}(\partial_{d+1})$, $x_{hr} \in \ker(L_d)$, and $x_{cbd} \in \operatorname{im}(\partial_d^T)$.

A **d-boundary basis**, **d-coboundary basis** and **d-harmonic basis** are bases for the boundary, coboundary and harmonic spaces. A **d-homology basis** is a maximal set of cycles such that no linear combination of these cycles is a boundary. Similarly, a **d-cohomology basis** is a maximal set of cocycles such that no linear combination of these cocycles is a coboundary. We have the following fact.

▶ **Fact 4.** *A set of cycles (resp. cocycles) is a homology (resp. cohomology) basis if and only if their projection into the harmonic space is a harmonic basis.*

Two cycles (resp. cocycles) are homologous (resp. cohomologous) if they have the same harmonic part, as then their difference is a boundary (resp. coboundary). Accordingly, the previous fact implies that for any cycle (resp. cocycle) x and any homology basis

(resp. cohomology basis) Γ, there is a unique linear combination of the elements of Γ that is homologous (resp. cohomologous) to x; this is the linear combination of Γ with the same harmonic component as x.

A useful property of cohomology bases is they can be used to tell when two cycles are homologous, as described by the following fact.

▶ **Fact 5** (Busaryev et al. [6]). *Let x and y be cycles (resp. cocycles), and let P be a cohomology basis (resp. homology basis.) Then y is homologous (resp. cohomologous) to x if and only if $x \cdot p = y \cdot p$ for all $p \in P$.*

Collapsibility. Let K be a simplicial complex, σ a d-simplex of K, and τ a $(d-1)$-simplex of K that is a face of σ. If τ is not the face of any other simplex, we say that K **collapses** into $K \backslash \{\sigma, \tau\}$; we refer to (σ, τ) as a **collapse pair**. Moreover, we say that a complex collapses to itself. Inductively, we say that a complex K **collapses** into a complex K' if there is a complex K'' such that K collapses to K'' and K'' collapses to K'. We say that a complex K is **collapsible** if it collapses to a single vertex.

When a complex K collapses to a complex K', we obtain a sequence of complexes $K = K_0 \supset K_1 \supset \ldots \supset K_t = K'$, where for each $1 \leq i \leq t$, K_i can be obtained from K_{i-1} by removing one collapse pair. We refer to this sequence as a **collapsing sequence**. The complexes K and K' are homotopy equivalent if one collapses to the other, thus, K and K' have isomorphic homology group. In particular, a collapsible complex has trivial homology groups in every nonzero dimension.

Embeddability. A d-dimensional simplicial complex K is **embedded** if $K \subset R$ for R a triangulation of $\mathbb{R}^{d+1}$. Furthermore, K is **linearly embedded** if there is a homeomorphism from the underlying space $|R|$ to $\mathbb{R}^{d+1}$ that is linear on each simplex, i.e. each 1-simplex is mapped to a line segment, each 2-simplex is mapped to a triangle, etc. All embedded complexes in this paper will be linearly embedded.

We will make use of the dual graph of an embedded complex. Informally, the **dual graph** of an embedded complex is the graph K^* with vertices that are the connected components of $R \setminus K$ and edges between two vertices if there is a d-simplex in K incident to both connected components. Alternatively, the dual graph can be defined with vertices corresponding to a generating set of d-cycles of K. For this construction, see the definition of **Lefschetz set** in the paper [3].

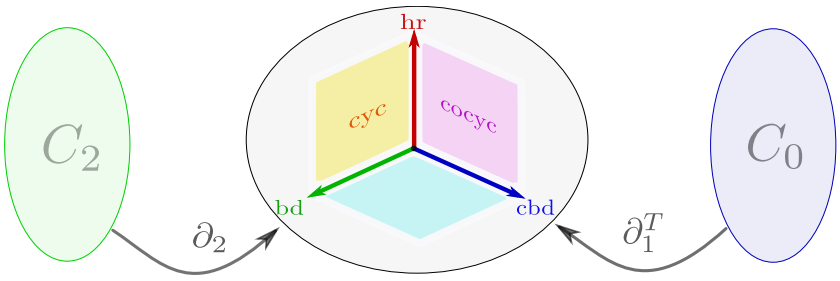
3 Overview

Let X be a collapsible simplicial complex embedded in $\mathbb{R}^3$, and let $K \subseteq X$ be a subcomplex of X. We study two closely related problems: (i) computing the Hodge decomposition of the 1-chains of K, and (ii) solving a linear system $L_1 x = b$, where L_1 is the 1-Laplacian of K (in the overview section, all the operators are with respect to K unless mentioned otherwise.) These two problems are related, as our approximate Laplacian solver uses an approximate Hodge decomposition of the input vector x. More generally, understanding the Hodge decomposition is key to understanding this paper as many proofs rely on some property of the Hodge decomposition. Therefore, we begin our overview with an introduction to the Hodge decomposition.

3.1 The Hodge Decomposition

The Hodge decomposition is a decomposition of the chain group $C_d(K)$ in terms of the kernels and images of the boundary operators ∂_d and ∂_{d+1} and their transposes. Specifically, the problems in this paper consider the first chain group $C_1(K)$, the two boundary operators ∂_2 and ∂_1, and their corresponding coboundary operators ∂_2^T and ∂_1^T. The boundary operator ∂_2 maps each (oriented) triangle to the edges in its boundary; similarly, ∂_1 maps each edge to its two endpoints. A key fact is that $\partial_1\partial_2 = 0$, or equivalently, $\text{im}(\partial_2) \subseteq \ker(\partial_1)$. This implies $\text{im}(\partial_2)$ is orthogonal to $\text{im}(\partial_1^T)$. The former subspace $\text{im}(\partial_2)$ is called the **boundary subspace**, and the latter subspace $\text{im}(\partial_1^T)$ is called the **coboundary subspace**. If K has trivial 1-homology, then $\text{im}(\partial_2) = \ker(\partial_1)$, and the boundary and coboundary spaces give a full orthogonal decomposition of $C_1(K)$, called the **Helmholtz decomposition**. Otherwise, there is a third subspace orthogonal to both the boundary and coboundary subspaces, called the **harmonic subspace**. The harmonic subspace is exactly $\ker(L_1) = \ker(\partial_1) \cap \ker(\partial_2^T)$.

The boundary, coboundary, and harmonic subspaces give a full orthogonal decomposition of $C_1(K)$ called the **Hodge decomposition**, which generalizes the Helmholtz decomposition. Thus, we can express any 1-chain x as $x = x_{cbd} + x_{bd} + x_{hr}$, where x_{cbd}, x_{bd} and x_{hr} are the coboundary, boundary and harmonic part of x and are pairwise orthogonal. The chains $x_{bd} + x_{hr}$ and $x_{cbd} + x_{hr}$ are called the **cyclic** and **cocyclic** parts of x respectively. Similarly, the space spanned by harmonic and boundary chains is called the **cycle space**, and the space spanned by harmonic and coboundary chains is called the **cocycle space**. It is implied by $\partial_1\partial_2 = 0$ that the cycle space and cocycle space are the kernels of ∂_1 and ∂_2^T, respectively. The following figure is an illustration of the Hodge decomposition. Boundary, coboundary, harmonic, cycle, and cocycle spaces are shown using the abbreviations bd, cbd, hr, cyc, and cocyc respectively.



To compute the Hodge decomposition, one seeks orthogonal projection operators into the coboundary, boundary and harmonic subspaces. Let Π_{cbd}, Π_{bd}, and Π_{hr} denote these projection operators. Cohen et al. show that for any 1-chain x, its projection into the coboundary space, $\Pi_{cbd}x$, and cycle space, $\Pi_{cyc}x$, can be approximated quickly with operators $\widetilde{\Pi}_{cbd}$ and $\widetilde{\Pi}_{cyc}$. These projection operators are a key ingredient of their 1-Laplacian solver, as well as the more recent 1-Laplacian solver described by Black et al.; however, both papers are restricted to cases where the first homology group $H_1(K) = 0$. In this paper, we show that for any x, its projection into the boundary space, $\Pi_{bd}x$, can also be approximated quickly. This new projection operator will allow us to generalize the 1-Laplacian solver of Black et al. to complexes with arbitrary first homology. We also give an approximate projection operator into the harmonic space, but our approximation guarantee for this projection operator is weaker (more below).

3.2 Laplacian Solvers

The 1-Laplacian matrix is defined $L_1 = \partial_2\partial_2^T + \partial_1^T\partial_1$. To solve a linear system $L_1 x = b$, one seeks to approximate L_1^+, the pseudoinverse of L_1. As the images of $\partial_2\partial_2^T$ and $\partial_1^T\partial_1$ are orthogonal, then $L_1^+ = (\partial_2\partial_2^T)^+ + (\partial_1^T\partial_1)^+$ (see Campbell [7, Theorem 3.1.1]). Therefore, one can approximate L_1^+ by approximating $(\partial_2\partial_2^T)^+$ and $(\partial_1^T\partial_1)^+$ individually. Computing $(\partial_1^T\partial_1)^+$ is purely a graph problem as ∂_1 is only defined with respect to the vertices and edges of a complex. Cohen et al. show how to approximate $(\partial_1^T\partial_1)^+$ for general complexes [10, Lemma 3.2]. Approximating $(\partial_2\partial_2^T)^+$ is a more challenging problem that requires taking into account the relationship between triangles and the edges. Our algorithm for approximating $(\partial_2\partial_2^T)^+$ relies on our new boundary projection operator, the collapsibility of X, and the embedding of X in $\mathbb{R}^3$.

Cohen et al. show how to approximate $(\partial_2\partial_2^T)^+$ for collapsible complexes embedded in $\mathbb{R}^3$. Black et al. generalize their work to obtain an approximate solver for a subcomplex of a collapsible complex in $\mathbb{R}^3$ provided the subcomplex has trivial homology. Their solver is based on the following general lemma regarding approximations of $(BB^T)^+$ for a general matrix B.

▶ **Lemma 6** (Black et al. [3], Lemma 4.1). *Let B be a linear operator, let $0 \le \varepsilon < 1$, and let $\widetilde{\Pi}_{\mathrm{im}(B)}$ and $\widetilde{\Pi}_{\ker^\perp(B)}$ be symmetric matrices such that $(1-\varepsilon)\Pi_{\mathrm{im}(B)} \preceq \widetilde{\Pi}_{\mathrm{im}(B)} \preceq \Pi_{\mathrm{im}(B)}$, and $(1-\varepsilon)\Pi_{\ker^\perp(B)} \preceq \widetilde{\Pi}_{\ker^\perp(B)} \preceq \Pi_{\ker^\perp(B)}$. Also, let U be a linear map such that for any $y \in \mathrm{im}(B)$, $BUy = y$. We have*

$$(1-(2\kappa+1)\varepsilon)(BB^T)^+ \preceq \widetilde{\Pi}_{\mathrm{im}(B)}U^T\widetilde{\Pi}_{\ker^\perp(B)}U\widetilde{\Pi}_{\mathrm{im}(B)} \preceq (1+\kappa\varepsilon)(BB^T)^+,$$

where κ is the condition number of BB^T within the image of B.

This lemma shows the following linear operators are sufficient for approximating $(\partial_2\partial_2^T)^+$.

- **(i)** An operator U that for 1-boundary $y \in \mathrm{im}(\partial_2)$ returns a 2-chain $x = Uy$ such that $\partial_2 x = y$. For other vectors $z \notin \mathrm{im}(\partial_2)$, U can return anything as long as U is still linear.
- **(ii)** An approximate orthogonal projection operator into $\mathrm{im}(\partial_2^T)$, the coboundary space of 2-chains.
- **(iii)** An approximate orthogonal projection operator into $\mathrm{im}(\partial_2)$, the boundary space of 1-chains.

Black et al. describe an algorithm for computing U that uses the collapsibility and embedding of the supercomplex X. Cohen et al. show that the 2-coboundary space of embedded complexes is dual to the 1-cycle space of the dual graph, hence projection into this space can be approximated using $\widetilde{\Pi}_{cyc}$. Finally, lacking an approximate projection into the boundary space of 1-chains, they needed to assume that their complex has trivial first homology (i.e. that $\mathrm{im}(\partial_2) = \ker(\partial_1)$) so that they can instead use the projection operator into the cycle space of Cohen et al. The boundary projection operator described in this paper allow us to remove that assumption to obtain a solver for any subcomplex K of X. The running time of our new solver polynomially depends on the rank of the homology group and nearly-linearly depends on the size of the complex. We give a complete analysis of our solver in Appendix D of the full paper.

In the rest of this section, we sketch the high level ideas for computing our approximate projection operators. But before we can do that, we need to explain the two notions of approximations that are used in this paper.

3.3 Loewner order approximation

We use the Loewner order on positive semidefinite matrices to specify the approximation quality of our projection and pseudoinverse operators. We see two types of approximation guarantees in this paper for an operator A: ***input-relative error bounds*** of the form $-\varepsilon I \preceq A - \widetilde{A} \preceq \varepsilon I$ and ***output-relative error bounds*** of the form $-\varepsilon A \preceq A - \widetilde{A} \preceq \varepsilon A$. Note for any vector x, an input relative error bound implies $\|(A - \widetilde{A})x\| \leq \varepsilon\|x\|$ – the error is bounded relative to the size of the input x – while an output-relative error bound implies $\|(A - \widetilde{A})x\| \leq \varepsilon\|Ax\|$ – the error is bounded relative to the size of the output Ax. An approximate operator with a small input-relative error can have arbitrarily large output-relative error, for example when x is in the kernel of A. Further, output-relative error bounds are stronger if the norm of $\|A\|$ is at most one, i.e. $\|Ax\| \leq \|x\|$, which is the case for the orthogonal projection operators of the Hodge decomposition.

We achieve an output-relative error bound for our approximation of $(L_1[K])^+$. Further, we achieve an output-relative error bound for our approximation $\widetilde{\Pi}_{bd}$ of Π_{bd}, but an input-error bound for our approximation $\widetilde{\Pi}_{hr}$ of Π_{hr}:

$$-\varepsilon I \preceq -\varepsilon\Pi_{bd} \preceq \Pi_{bd} - \widetilde{\Pi}_{bd}(\varepsilon) \preceq \varepsilon\Pi_{bd} \preceq \varepsilon I, \tag{1}$$

and

$$-\varepsilon I \preceq \Pi_{hr} - \widetilde{\Pi}_{hr}(\varepsilon) \preceq \varepsilon I. \tag{2}$$

Previously, Cohen et al. had shown approximation operators $\widetilde{\Pi}_{cbd}$ and $\widetilde{\Pi}_{cyc}$ for projecting into the coboundary and cycle spaces with output-relative error bounds:

$$-\varepsilon I \preceq -\varepsilon\Pi_{cbd} \preceq \Pi_{cbd} - \widetilde{\Pi}_{cbd}(\varepsilon) \preceq \varepsilon\Pi_{cbd} \preceq \varepsilon I, \tag{3}$$

and

$$-\varepsilon I \preceq -\varepsilon\Pi_{cyc} \preceq \Pi_{cyc} - \widetilde{\Pi}_{cyc}(\varepsilon) \preceq \varepsilon\Pi_{cyc} \preceq \varepsilon I. \tag{4}$$

We use these operators multiple times in our algorithms. For simplification, we drop the explicit mention of the parameter ε when it is clear from the context.

3.4 Projection operators

We first describe our algorithm for computing $\widetilde{\Pi}_{hr}$ (an overview of Appendices A and B of the full version of the paper). Based on that and the operator $\widetilde{\Pi}_{cbd}$ of Equation (3), we show how to compute $\widetilde{\Pi}_{bd}$ (an overview of Appendix C of the full version of the paper).

3.4.1 Harmonic projection

We compute our approximate harmonic projection operator $\widetilde{\Pi}_{hr}$ by computing an approximate orthonormal basis $\widetilde{G} = \{\widetilde{g}_1, \ldots, \widetilde{g}_\beta\}$ of the harmonic space. We then define the approximate projection into the harmonic space to be the linear map $\widetilde{\Pi}_{hr} = \sum_{i=1}^{\beta} \tilde{g}_i \tilde{g}_i{}^T$.

To compute $\widetilde{G}$, our algorithm starts with a cohomology basis $P = \{p_1, \ldots, p_\beta\}$; the algorithm for computing P is given at the end of this section. From P, it computes $\widetilde{H} = \{\widetilde{h}_1, \ldots, \widetilde{h}_\beta\}$, where $\widetilde{h}_i = \widetilde{\Pi}_{cyc} p_i$ and $\widetilde{\Pi}_{cyc}$ is the projection operator of Equation (4). The set $\widetilde{H}$ is an approximate harmonic basis, but it is not orthonormal. Next, we normalize $\widetilde{H}$ to obtain $\widetilde{N} = \{\widetilde{h}_1/\|\widetilde{h}_1\|, \ldots, \widetilde{h}_\beta/\|\widetilde{h}_\beta\|\}$. Finally, we run Gram-Schmidt on $\widetilde{N}$ to obtain $\widetilde{G}$.

To see why $\widetilde{G}$ is an approximate basis for the harmonic space, let us consider a much easier analysis assuming we can use the exact projection in the cycle space Π_{cyc} instead of the approximate projection $\widetilde{\Pi}_{cyc}$. Instead of $\widetilde{H}$, $\widetilde{N}$ and $\widetilde{G}$, let $H = \{h_1, \ldots, h_\beta\}$, $N = \{h_1/\|h_1\|, \ldots, h_\beta/\|h_\beta\|\}$ and $G = \{g_1, \ldots, g_\beta\}$ be the sets of vectors we obtain when we use the exact projection operator. In that case, $h_i = \Pi_{cyc} p_i$ is the harmonic part of p_i; this is because p_i is a cocycle, so projecting it into the cycle space is the same as projecting it into the harmonic space. It follows from Fact 4 in Section 2 that G is an exact orthonormal basis of the harmonic space, thus it defines an exact projection operator into the harmonic space.

In the real scenario where we work with the approximate projection operator $\widetilde{\Pi}_{cyc}$, two undesirable things can happen. First, we can no longer guarantee that the vectors in $\widetilde{N}$ are purely harmonic, as the error introduced by the approximate operator $\widetilde{\Pi}_{cyc}$ may be part boundary. However, this is not an issue, as we can make the boundary components of $\widetilde{N}$ sufficiently small by approximating $\widetilde{\Pi}_{cyc}$ more accurately. Second, and more importantly for our application, the spaces spanned by N and $\widetilde{N}$ can be very different, even if the vectors N and $\widetilde{N}$ are pairwise close. As an example, imagine that we have two pairs of vectors $N = \{\eta_1, \eta_2\}$ and $\widetilde{N} = \{\widetilde{\eta}_1, \widetilde{\eta}_2\}$ such that $\|\eta_i - \widetilde{\eta}_i\| < \varepsilon$ for $i = 1, 2$. We might guess that the two spaces spanned by N and $\widetilde{N}$ are similar as the vector are close, but if η_1 and η_2 are also close, then the two vectors spaces can be drastically different. Figure 1 gives an illustration of this, where N is the set of blue vectors and $\widetilde{N}$ is the set of red vectors. As illustrated in the figure, the space spanned by N and the space spanned by $\widetilde{N}$ can be drastically different.

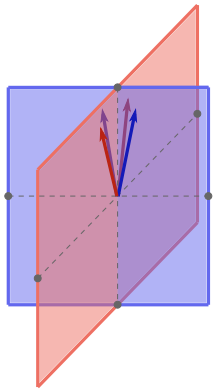


Figure 1 Pairwise closeness between a set of vectors N and $\widetilde{N}$ is not enough to guarantee the spaces spanned by N and $\widetilde{N}$ are close! The red and blue vectors are pairwise close, but the spaces they span are very different.

We can remedy this if we approximate $\widetilde{N}$ within a sufficiently small error ε of N, but this new error bound needs to take into account the similarity of the vectors in N. The question is how accurately we need to approximate $\widetilde{\Pi}_{cyc}$ to obtain a sufficently small approximation error for $\widetilde{N}$. To answer this question, we define a measure of linear independence of N called its δ-independence. Formally, we say that N is **δ-independent** if each vector $h_i/\|h_i\| \in N$ is at distance at least δ from the span of the other vectors of N. Intuitively, larger δ means N is more independent, in the sense that the elements are well-separated. The smaller the δ, the more accurately we need to approximate $\widetilde{\Pi}_{cyc}$ to ensure that N and $\widetilde{N}$ will span similar spaces. This intuition is summarized by the following lemma, showing the error in projection into N as a function of δ, ε, and β, where ε bounds the difference between N and $\widetilde{N}$.

▶ **Corollary 7.** *Let $0 < \delta < 1$, and let $0 < \varepsilon < \left(\frac{\delta}{8\beta}\right)^{\beta}$. Let $N = \{\eta_1, \ldots, \eta_\beta\}$ be a set of δ-linearly independent unit vectors, and let $\widetilde{N} = \{\widetilde{\eta}_1, \ldots, \widetilde{\eta}_\beta\}$ be a set of unit vectors such that $\|\eta_i - \widetilde{\eta}_i\| < \varepsilon$. Let $G = \{g_1, \ldots, g_\beta\}$ be the orthonormal basis that is the output of running Gram-Schmidt on N, and let $\widetilde{G} = \{\widetilde{g}_1, \ldots, \widetilde{g}_\beta\}$ be the output of running Gram-Schmidt on $\widetilde{N}$. Then $\left\|\Pi_{\text{span } N} - \Pi_{\text{span } \widetilde{N}}\right\| = \left\|\sum_{i=1}^{\beta} g_i g_i^T - \widetilde{g}_i \widetilde{g}_i^T\right\| < 2 \cdot \beta \cdot \left(\frac{8\beta}{\delta}\right)^{\beta} \varepsilon$.*

The difficulty here is actually determining a lower bound on the δ-independence of N. We have access to the cohomology basis P, but we need the (normalized) harmonic parts of P to be δ-independent. Note that P can be composed of vectors that are very strongly independent, yet their harmonic parts may only be weakly independent, for example, when the vectors of P have similar harmonic parts but very different coboundary parts.

We show that if P is composed of integer vectors with maximum length $p_{\max}$, then P being linearly independent implies that H is δ-independent for a $\delta \sim 1/(p_{\max} \cdot n_1)^{\beta}$, where n_1 is the number of edges in K. In addition to the properties of P, our proof of Lemma 8 relies on the total unimodularity of ∂_1.

▶ **Lemma 8.** *Let K be a simplicial complex with n_1 edges such that $H_1(K) = \beta$. Let $\{p_1, \ldots, p_\beta\}$ be a 1-cohomology basis for K such that each p_i is an integer vector with maximum Euclidean norm $p_{\max}$. Let h_i be the harmonic part of p_i for $1 \le i \le \beta$. Then*

(i) *$\|h_i\| \ge 1/(\sqrt{n_1} \cdot p_{\max})^{\beta}$ for each $1 \le i \le \beta$, and*

(ii) *$\{h_1/\|h_1\|, \ldots, h_\beta/\|h_\beta\|\}$ is $\left(1/(\sqrt{n_1} \cdot p_{\max})^{\beta}\right)$-independent.*

Computing a Cohomology Basis. The question remains of how to find the cohomology basis P. For this, we use an algorithm on Black et al. Dey [14] describes a nearly-linear time algorithm for computing a homology basis composed of vectors with coordinates in $\{-1, 0, +1\}$. Black et al. [3] describe an operator $C(X, K)$ that when applied to a homology basis returns a cohomology basis. We use the cohomology basis P obtained by applying the operator C to Dey's homology basis.

▶ **Lemma 9** (Dey, Lemma [14])**.** *For a 2-dimensional simplicial complex K linearly embedded in $\mathbb{R}^3$, there exists an algorithm computing a basis for $H_1(K, \mathbb{R})$ in $O(n \log n + n\beta_1)$ time, where n is the complexity of K. Further, the basis is composed of vectors with all coordinates from $\{-1, 0, +1\}$.*

▶ **Lemma 10** (Black et al., Lemma 1.1 [3])**.** *Let X be a collapsible simplicial complex in $\mathbb{R}^3$, and let $K \subset X$ be a subcomplex of X. Let β be the rank of $H_1(K)$ and let n be the total number of simplices of X. Let $\Gamma = \{\gamma_1, \ldots, \gamma_\beta\}$ be a homology basis for K. There is a linear operator $C(X, K)$ such that the set $C_\Gamma = \{C(X, K) \cdot \gamma_1, \ldots, C(X, K) \cdot \gamma_\beta\}$ is a cohomology basis for K. Furthermore, C_Γ can be computed in $O(\beta \cdot n)$ time.*

While Black et al. introduced the algorithm for computing the cohomology basis P, we bound the length of the vectors in P. We bound the length of P by combining a bound on the length of the homology basis with the following bound on the operator norm $\|C(X, K)\|$.

▶ **Lemma 11.** *There is a constant α such that $\|C(X, K)\| \le \alpha \cdot n_1^2 n_2^4 / \left(\lambda_{\min}(L_1^{up}[X])\right)$, where $\lambda_{\min}(L_1(X))$ is the smallest non-zero eigenvalue of $L_1(X)$ and n_1 and n_2 are the number of edges and triangles in X respectively.*

Note that the vectors in Dey's homology basis have bounded length as they have coefficients $\{-1, 0, 1\}$. By combining this observation, Lemma 8, and Lemma 11, we prove that the harmonic part of P are δ-independent for an appropriate value of δ. This is summarized in the following corollary.

▶ **Corollary 12.** *Let X be a collapsible complex embedded in $\mathbb{R}^3$ with a known collapsing sequence and let $K \subset X$ be a subcomplex of X. Let β be the rank of $H_1(K)$ and let n be the total number of simplices in X. There is an $O(n \log n + \beta n)$ time algorithm for computing a cohomology basis $\{p_1, \ldots, p_\beta\}$ of K with harmonic parts $\{h_1, \ldots, h_\beta\}$ such that*

(i) *each h_i has length at least δ, and*

(ii) *the set $\{h_1/\|h_1\|, \ldots, h_\beta/\|h_\beta\|\}$ is δ-linearly independent,*

where $\delta = (\lambda_{\min}(L_2^{up}(X))/(\alpha \cdot n_1^3 n_2^4))^\beta$ for a constant α.

Passing this cohomology basis P to the algorithm above, we obtain $\widetilde{H}$, $\widetilde{N}$, $\widetilde{G}$, $\widetilde{\Pi}_{hr}$ as desired. The exact approximation quality of the approximate harmonic basis and approximate harmonic projection are given in Lemma 3 in the introduction.

3.4.2 Boundary projection

It follows from the Hodge decomposition that the the projection into the boundary space can be written $\Pi_{bd} = I - \Pi_{cbd} - \Pi_{hr}$. We have approximate projections $\widetilde{\Pi}_{cbd}$ and $\widetilde{\Pi}_{hr}$ with input-relative error bounds (Equations (3) and (2) respectively), so we immediately obtain a boundary projection $\overline{\Pi}_{bd}$ with input-relative error bound defined

$$\overline{\Pi}_{bd} = I - \widetilde{\Pi}_{cbd} - \widetilde{\Pi}_{hr} \implies -\varepsilon I \preceq \Pi_{bd} - \overline{\Pi}_{bd} \preceq \varepsilon I.$$

However, we need a boundary projection operator with an output-relative bound for our solver. Unfortunately, the operator $\overline{\Pi}_{bd}$ can have arbitrarily bad output-relative error. Specifically, for any vector x that is orthogonal to the boundary space, this operator has unbounded output-relative error as $\Pi_{bd} x = 0$.

We instead use $\overline{\Pi}_{bd}$ as a starting point for a projection operator with bounded output-relative error. To that end, let's revisit the issue of input vs. output relative error. Let $x = x_{bd} + x_{cocyc}$ be any vector decomposed into its boundary and cocycle parts. The input-relative error bound of $\overline{\Pi}_{bd}$ is proportional to $\|x\| = \|x_{bd} + x_{cocyc}\|$, while for an output-relative error bound, we need the bound to be proportional to $\|\Pi_{bd} x\| = \|x_{bd}\|$. Therefore, a problem arises if x_{cocyc} is much larger than x_{bd}; provided a bound on $\|x_{cocyc}\|/\|x_{bd}\|$, we can accordingly modify the accuracy of our projection operators $\widetilde{\Pi}_{cbd}(\varepsilon)$ and $\widetilde{\Pi}_{hr}(\varepsilon)$ to ensure $\overline{\Pi}_{bd}$ has small output-relative error for x. Unfortunately, $\|x_{cocyc}\|/\|x_{bd}\|$ can be unbounded. To counteract this, we show that we can map x to a different vector x' before passing it to $\overline{\Pi}_{bd}$ such that (1) x' has the same boundary component as x (so $\Pi_{bd} \cdot x = \Pi_{bd} \cdot x'$), and (2) $\|x'_{cocyc}\|/\|x'_{bd}\|$ is bounded.

Specifically, our boundary projection operator is defined $\widetilde{\Pi}_{bd} = (I - P_\Gamma)(I - P_T)\overline{\Pi}_{bd}(I - P_T)^T(I - P_\Gamma)^T$, defined based on two operators P_T and P_Γ. The former was introduced by Cohen et al. to obtain $\widetilde{\Pi}_{cyc}$, and the latter is introduced in this paper; we sketch the ideas of both in this overview. The operator $(I - P_T)^T(I - P_\Gamma)^T$ behaves as we need: it maps x to a chain x' with the same boundary component as x and a relatively bounded cocycle part. We now describe P_T and P_Γ.

Let T be any spanning tree of the 1-skeleton of K. P_T is the operator that maps any 1-chain to the unique 1-chain with the same boundary in T. In particular, for any 1-chain x, $(I - P_T)x$ is a cycle.

Next, let $\Gamma = \{\gamma_1, \ldots, \gamma_\beta\}$ be a 1-homology basis in K. P_Γ is the operator that maps any 1-cycle to the unique linear combination of Γ that is in the same homology class. In particular, for any 1-cycle x, $(I - P_\Gamma)x$ is a boundary.

Now let $F = (I - P_T)^T(I - P_\Gamma)^T$, so $F^T = (I - P_\Gamma)(I - P_T)$. Consider any vector $x = x_{bd} + x_{cbd} + x_{hr}$. We investigate what F does to each of the three constituents of x; what can we say about Fx_{bd}, Fx_{cbd} and Fx_{hr}? In what follows, we frequently use the fact that for any linear map A, $\ker(A)$ and $\mathrm{im}(A^T)$ orthogonally decompose the domain of A.

$(I - P_T)$ maps any 1-chain to a cycle and $(I - P_\Gamma)$ maps any cycle to a boundary cycle; thus, $\mathrm{im}(F^T)$ is a subset of the boundary space. It follows that $\ker(F)$ is a superset of the orthogonal complement of the boundary space, which is the cocycle space. So, F maps any cocycle to zero, in particular, $Fx_{cbd} = 0$ and $Fx_{hr} = 0$. It remains to investigate Fx_{bd}.

P_T maps any cycle to zero, so $\ker(P_T)$ includes the cycle space; hence, $\mathrm{im}(P_T^T)$ is a subset of the orthogonal complement of the cycle space, which is the coboundary space. In particular, $\mathrm{im}(P_T^T)$ is a subset of the cocycle space. In addition, P_Γ maps all boundary cycles to zero, so $\ker(P_\Gamma)$ includes the boundary space; hence, $\mathrm{im}(P_\Gamma^T)$ is within the orthogonal complement of the boundary space, which is the cocycle space. Now consider

$$Fx_{bd} = (I - P_T)^T(I - P_\Gamma)^T x_{bd} = Ix_{bd} - P_T^T(I - P_\Gamma)^T x_{bd} - P_\Gamma^T x_{bd} = x_{bd} + x'_{cocyc},$$

and observe that x'_{cocyc} is indeed in the cocycle space as $\mathrm{im}(P_T^T)$ and $\mathrm{im}(P_\Gamma^T)$ are both within this space.

Overall, $Fx_{cbd} = 0$, $Fx_{hr} = 0$ and $Fx_{bd} = x_{bd} + x'_{cocyc}$, so Fx has the same boundary part as x. Moreover, the norm of the cocyclic part of Fx, $\|x'_{cocyc}\|$, can now be bounded by $\|F\| \cdot \|x_{bd}\|$, as it is produced by applying F to x. The proof of Lemma 3.2 in Cohen et al. and Corollary 14 of this paper provide bounds on $\|I - P_T\|$ and $\|I - P_\Gamma\|$ that are dependent on the number of simplices of X, the smallest non-zero eigenvalue of the up-Laplacian of X, and the first Betti number of K; these bounds in turn provide a bound on $\|F\|$. (Note that $\|A\| = \sqrt{\|AA^T\|}$ for any linear operator A.)

▶ **Lemma 13** (Cohen et al., Proof of Lemma 3.2 [10]). *Let K be any simplicial complex, and let T be a spanning tree of the 1-skeleton of K. Let P_T be the operator that maps any 1-chain x to the unique 1-chain on T with the same boundary, that is (i) $P_T \cdot x \in C_1(T)$, and (ii) $\partial_1 x = \partial_1 P_T \cdot x$. We have $(I - P_T)(I - P_T)^T \preceq n_1^2 I$, where n_1 is the number of edges of K. Further, for any x, $P_T \cdot x$ can be computed in $O(n_1)$ time.*

▶ **Corollary 14.** *Let X, K as defined. Let $\Gamma = \{\gamma_1, \ldots, \gamma_\beta\}$ be the homology basis of Lemma 9, and let P_Γ be the operator that for any cycle α returns the unique linear combination of the cycles of Γ that is homologous to α. We have*

$$(1 - P_\Gamma)(1 - P_\Gamma)^T \preceq \varepsilon \cdot I,$$

for $\varepsilon = (n_1 n_2 / \lambda_{\min}(X))^{c \cdot \beta}$, where $\lambda_{\min}(X)$ is the smallest non-zero eigenvalue of $L_1^{up}(X)$ and c is a constant. Further, for any vector v, $P_\Gamma \cdot v$ can be computed in $O(\beta^2 n_1 + \beta^\omega)$ time.

The accuracy and time complexity of the approximate boundary solver $\widetilde{\Pi}_{bd}$ are described in Lemma 2 in the introduction.

References

1 Douglas Arnold, Richard Falk, and Ragnar Winther. Finite element exterior calculus: from hodge theory to numerical stability. *Bulletin of the American Mathematical Society*, 47(2):281–354, January 2010. doi:10.1090/s0273-0979-10-01278-4.

2 Marshall Bern, John R. Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo. Support-graph preconditioners. *SIAM J. Matrix Anal. Appl.*, 27(4):930–951, December 2005. doi:10.1137/S0895479801384019.

3 Mitchell Black, William Maxwell, Amir Nayyeri, and Eli Winkelman. Computational topology in a collapsing universe: Laplacians, homology, cohomology. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 226–251, 2022. doi:10.1137/1.9781611977073.12.

4 Erik G. Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM J. Matrix Anal. Appl.*, 25(3):694–717, March 2003. doi:10.1137/S0895479801390637.

5 Erik G. Boman, Bruce Hendrickson, and Stephen Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM J. Numer. Anal.*, 46(6):3264–3284, October 2008. doi:10.1137/040611781.

6 Oleksiy Busaryev, Sergio Cabello, Chao Chen, Tamal K. Dey, and Yusu Wang. Annotating simplices with a homology basis and its applications. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory – SWAT 2012*, pages 189–200, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

7 S. L. (Stephen La Vern) Campbell. *Generalized inverses of linear transformations*. Surveys and reference works in mathematics. Pitman, London, 1979.

8 Ozan Candogan, Ishai Menache, Asuman E. Ozdaglar, and Pablo A. Parrilo. Flows and decompositions of games: harmonic and potential games. *Math. Oper. Res.*, 36(3):474–503, 2011. doi:10.1287/moor.1110.0500.

9 Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 273–282, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1993636.1993674.

10 Michael B. Cohen, Brittany Terese Fasy, Gary L. Miller, Amir Nayyeri, Richard Peng, and Noel Walkington. Solving 1-laplacians in nearly linear time: collapsing and expanding a topological ball. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 204–216, USA, 2014. Society for Industrial and Applied Mathematics.

11 Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly mlog1/2n time. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 343–352, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591833.

12 Keenan Crane, Mathieu Desbrun Fernando de Goes, and Peter Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*, SIGGRAPH '13, New York, NY, USA, 2013. ACM.

13 Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discret. Comput. Geom.*, 45(4):737–759, 2011. doi:10.1007/s00454-011-9344-x.

14 Tamal K. Dey. Computing height persistence and homology generators in R^3 efficiently. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2649–2662. SIAM, 2019. doi:10.1137/1.9781611975482.164.

15 Ming Ding, Maximillian Probst Gutenberg, Rasmus Kyng, and Peng Zhang. Hardness Results for Laplacians of Simplicial Complexes via Sparse-Linear Equation Complete Gadgets. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

16 Joel Friedman. Computing betti numbers via combinatorial laplacians. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 386–391, New York, NY, USA, 1996. ACM. doi:10.1145/237814.237985.

17 Allen Hatcher. *Algebraic topology*. Cambridge Univ. Press, Cambridge, 2000. URL: https://cds.cern.ch/record/478079.

18 Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, USA, 2nd edition, 2012.

19 Arun Jambulapati and Aaron Sidford. Ultrasparse ultrasparsifiers and faster laplacian system solvers. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 540–559, USA, 2021.

20 Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial hodge theory. *Math. Program.*, 127(1):203–244, March 2011. doi:10.1007/s10107-010-0419-x.

21 Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 911–920, New York, NY, USA, 2013. ACM. doi:10.1145/2488608.2488724.

22 Ioannis Koutis and Richard Miller, Gary L. and Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 235–244, Washington, DC, USA, 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.29.

23 Ioannis Koutis and Richard Miller, Gary L. and Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society. doi:10.1109/FOCS.2011.85.

24 Rasmus Kyng and Peng Zhang. Hardness results for structured linear systems. *SIAM J. Comput.*, 49(4), 2020. doi:10.1137/17M1161774.

25 Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002. URL: https://proceedings.neurips.cc/paper/2001/file/801272ee79cfde7fa5960571fee36b9b-Paper.pdf.

26 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., USA, 1998.

27 Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi:10.1109/34.868688.

28 Daniel Spielman. Spectral and algebraic graph theory. Available at http://cs-www.cs.yale.edu/homes/spielman/sagt/sagt.pdf (2021/12/01).

29 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 563–568, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374456.

30 Shang-Hua Spielman, Daniel A. and Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90. ACM, 2004.

31 John Stillwell. *Classical Topology and Combinatorial Group Theory*, volume 72 of *Graduate Texts in Mathematics*. Springer, second edition, 1993. URL: http://www.springer.com/mathematics/algebra/book/978-0-387-97970-0.

32 Alireza Tahbaz-Salehi and Ali Jadbabaie. Distributed coverage verification in sensor networks without location information. *IEEE Transactions on Automatic Control*, 55(8):1837–1849, 2010. doi:10.1109/TAC.2010.2047541.

33 Yiying Tong, Santiago Lombeyda, Anil N. Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM Trans. Graph.*, 22(3):445–452, July 2003. doi:10.1145/882262.882290.

34 Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Workshop Talk at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991. Minneapolis, MN.

35 Qianqian Xu, Qingming Huang, Tingting Jiang, Bowei Yan, Weisi Lin, and Yuan Yao. Hodgerank on random graphs for subjective video quality assessment. *IEEE Transactions on Multimedia*, 14(3):844–857, 2012. doi:10.1109/TMM.2012.2190924.

Reconstructing Decision Trees

Guy Blanc
Stanford University, CA, USA

Jane Lange
MIT, Cambridge, MA, USA

Li-Yang Tan
Stanford University, CA, USA

Abstract

We give the first *reconstruction algorithm* for decision trees: given queries to a function f that is opt-close to a size-s decision tree, our algorithm provides query access to a decision tree T where:

- T has size $S := s^{O((\log s)^2/\varepsilon^3)}$;
- $\mathrm{dist}(f, T) \leq O(\mathrm{opt}) + \varepsilon$;
- Every query to T is answered with $\mathrm{poly}((\log s)/\varepsilon) \cdot \log n$ queries to f and in $\mathrm{poly}((\log s)/\varepsilon) \cdot n \log n$ time.

This yields a *tolerant tester* that distinguishes functions that are close to size-s decision trees from those that are far from size-S decision trees. The polylogarithmic dependence on s in the efficiency of our tester is exponentially smaller than that of existing testers.

Since decision tree complexity is well known to be related to numerous other boolean function properties, our results also provide a new algorithm for reconstructing and testing these properties.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Property reconstruction, property testing, tolerant testing, decision trees

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.24

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2012.08735`

Funding *Guy Blanc*: Supported by NSF CAREER Award 1942123.
Jane Lange: Supported in part by NSF Award CCF-2006664, Big George Fellowship, Akamai Presidential Fellowship and Google.
Li-Yang Tan: Supported by NSF CAREER Award 1942123.

Acknowledgements We would like to thank the anonymous reviews of ICALP 2022 for their helpful feedback.

1 Introduction

We study the problem of *reconstructing* decision trees: given queries to a function f that is close to a size-s decision tree, provide fast query access to a decision tree, ideally one of size not much larger than s, that is close to f. This can be viewed as an "on the fly" variant of the problem of properly and agnostically learning decision trees, where the goal there is to output the entire decision tree hypothesis. More broadly, reconstruction algorithms, introduced by Ailon, Chazelle, Comandur, and Liu [2], can be viewed as sublinear algorithms that restore structure – in our case, that of a decision tree – in a function that has been lost due to noise.

Decision trees have long been a popular and effective model in machine learning, and relatedly, they are among the most intensively studied concept classes in learning theory. The literature on learning decision trees is vast, spanning three decades and studying the


problem in a variety of models and from a variety of perspectives [24, 41, 9, 26, 14, 8, 27, 36, 29, 39, 34, 31, 28, 22, 13, 7]. In contrast, the problem of reconstructing decision trees has thus far been surprisingly understudied.

1.1 Our contributions

We give the first reconstruction algorithm for decision trees. Our algorithm achieves a *polylogarithmic* dependence on s in its query and time complexities, exponentially smaller than the information-theoretic minimum required to learn.

▶ **Theorem 1** (Main result). *There is a randomized algorithm which, given queries to $f : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $s \in \mathbb{N}$ and $\varepsilon \in (0, 1)$, provides query access to a fixed decision tree T where*

- *T has size $s^{O((\log s)^2/\varepsilon^3)}$;*
- *$\mathrm{dist}(T, f) \leq O(\mathrm{opt}_s) + \varepsilon$ w.h.p., where opt_s denotes the distance of f to the closest size-s decision tree;*
- *Every query to T is answered with $\mathrm{poly}((\log s)/\varepsilon) \cdot \log n$ queries to f and in $\mathrm{poly}((\log s)/\varepsilon) \cdot n \log n$ time.*

Notably, in the standard setting where $s = \mathrm{poly}(n)$, the query and time complexities of our algorithm are $\mathrm{polylog}(n)$ and $\tilde{O}(n)$ respectively. Previously, the only known approach was to simply properly and agnostically learn f; the current fastest such algorithm has query and time complexities $n^{O(\log \log n)}$ [7].

Our reconstruction algorithm is furthermore *local* in the sense of Saks and Seshadhri [43], allowing queries to be answered in parallel assuming a shared random string. In particular, once f, s, ε and the random string are fixed, all queries are answered consistently with a single decision tree.

1.1.1 Implications of Theorem 1 and further results

By a standard reduction, Theorem 1 gives a *tolerant tester* for decision trees:

▶ **Corollary 2** (Tolerant testing of decision trees). *There is a randomized algorithm which, given queries to $f : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $s \in \mathbb{N}$ and $\varepsilon \in (0, 1)$,*

- *Makes $\mathrm{poly}((\log s)/\varepsilon) \cdot \log n$ queries to f, runs in $\mathrm{poly}((\log s)/\varepsilon) \cdot n \log n$ time, and*
- *Accepts w.h.p. if f is ε-close to a size-s decision tree;*
- *Rejects w.h.p. if f is $\Omega(\varepsilon)$-far from size-$s^{O((\log s)^2/\varepsilon^3)}$ decision trees.*

This adds to a long line of work on testing decision trees [33, 23, 19, 5, 15]. We give an overview of prior testers in Section 1.2, mentioning for now that they all have (at least) an exponentially larger dependence on s in their query and time complexities.

1.1.1.1 A new connection between tolerant testing and learning

It would be preferable if our tester can be improved to reject all f's that are far from size-s decision trees – or more strongly, if our reconstructor can be improved to provide query access to a size-s decision tree.

We show that such a tester, even one that is considerably less efficient than ours, would yield the first polynomial-time algorithm for properly learning decision trees:

▶ **Theorem 3** (Tolerant testing $\Longrightarrow$ Proper learning). *Suppose there is an algorithm which, given query access to $f : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $s \in \mathbb{N}$ and $\varepsilon \in (0, 1)$,*

- *Makes $\mathrm{poly}(s, n, 1/\varepsilon)$ queries to f, runs in $\mathrm{poly}(s, n, 1/\varepsilon)$ time, and*
- *Accepts w.h.p. if f is ε-close to a size-s decision tree;*
- *Rejects w.h.p. if f is $\Omega(\varepsilon)$-far from size-s decision trees.*

Then there is a $\mathrm{poly}(s, n, 1/\varepsilon)$-time membership query algorithm for properly learning size-s decision trees with respect to the uniform distribution.

This would represent a breakthrough on a central open problem in learning theory. Recent work of Blanc, Lange, Qiao, and Tan [7] gives a $\mathrm{poly}(n) \cdot s^{O(\log \log s)}$ time algorithm, improving on the prior state of the art of $n^{O(\log s)}$ [24]. Neither [24]'s nor [7]'s algorithm goes through testing.

It is well known and easy to see that proper learning algorithms yield comparably efficient testers [25]. Theorem 3 provides an example of a converse; we find the existence of such a converse surprising, and are not aware of any previous examples.

1.1.1.2 Reconstructors and testers for other properties

Decision tree complexity is quantitatively related to numerous other complexity measures of boolean functions: Fourier degree, approximate degree, randomized and quantum query complexities, certificate complexity, block sensitivity, sensitivity, etc. Our results therefore immediately yield new reconstructors and tolerant testers for these properties. For example, we have the following:

▶ **Corollary 4** (Reconstruction of low Fourier degree functions). *There is a randomized algorithm which, given queries to $f : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $d \in \mathbb{N}$ and $\varepsilon \in (0, 1)$, provides query access to a fixed function $g : \{\pm 1\}^n \to \{\pm 1\}$ where*

- *g has Fourier degree $O(d^7/\varepsilon^2)$,*
- *$\mathrm{dist}(f, g) \leq O(\mathrm{opt}_d) + \varepsilon$ w.h.p., where opt_d denotes the distance of f to closest $h : \{\pm 1\}^n \to \{\pm 1\}$ of Fourier degree d.*
- *Every query to g is answered in $\mathrm{poly}(d, 1/\varepsilon) \cdot n \log n$ time and with $\mathrm{poly}(d, 1/\varepsilon) \cdot \log n$ queries to f.*

This in turn yields a tolerant tester for Fourier degree. As in the case for decision trees, all prior testers for low Fourier degree [23, 19, 20, 5, 12, 15] have an exponential dependence on d in their query and time complexities.

Table 1 lists examples of measures for which we obtain new reconstruction algorithms, each of which in turn give new tolerant testers.

1.2 Background and comparison with prior work

As already mentioned, Theorem 1 gives the first reconstruction algorithm for decision trees. The problem of testing decision trees, on the other hand, has been intensively studied.

Testing decision trees. Recent work of Bshouty [15] gives an algorithm, running in $\mathrm{poly}(s^s, 1/\varepsilon) \cdot n$ time and using $O((s \log s)/\varepsilon)$ queries, that distinguishes between size-s decision trees from functions that are ε-far from size-s decision trees. Prior to [15], Chakraborty, García-Soriano, and Matsliah [19] gave an $O((s \log s)/\varepsilon^2)$-query algorithm, and before that Diakonikolas, Lee, Matulef, Onak, Rubinfeld, Servedio, and Wan [23] gave an $\tilde{O}(s^4/\varepsilon^2)$-query algorithm. Like [15]'s algorithm, the algorithms of [19, 23] also run in $\mathrm{poly}(s^s, 1/\varepsilon) \cdot n$ time.[1]

[1] All these testers enjoy a weak form of tolerance: they are in fact able to distinguish between functions that are $O(\mathrm{poly}(\varepsilon/s))$-close to size-$s$ decision trees from those that are ε-far from size-s decision trees. (Briefly, this is because their queries, while correlated, are each uniformly distributed.)

Table 1 Performance guarantees of our reconstruction algorithms for various complexity measures. In each row, opt_d denotes the distance from f to the closest function h such that the complexity measure of that row for h is bounded by d. In all cases, every query to g is answered in $\mathrm{poly}(d, 1/\varepsilon) \cdot n \log n$ time with $\mathrm{poly}(d, 1/\varepsilon) \cdot \log n$ queries to f.

Complexity measure	*Assumption* Query access to f that is opt_d-close to h where:	*Guarantee* Query access to g that is $O(\mathrm{opt}_d + \varepsilon)$-close to f where:
Fourier degree	$\deg(h) \leq d$	$\deg(g) \leq O(d^7/\varepsilon^2)$
Approximate degree	$\widetilde{\deg}(h) \leq d$	$\widetilde{\deg}(g) \leq O(d^9/\varepsilon^2)$
Randomized query complexity	$\mathrm{R}(h) \leq d$	$\mathrm{R}(g) \leq O(d^7/\varepsilon^2)$
Quantum query complexity	$\mathrm{Q}(h) \leq d$	$\mathrm{Q}(g) \leq O(d^{10}/\varepsilon^2)$
Certificate complexity	$\mathrm{C}(h) \leq d$	$\mathrm{C}(g) \leq O(d^5/\varepsilon^2)$
Block sensitivity	$\mathrm{bs}(h) \leq d$	$\mathrm{bs}(g) \leq O(d^8/\varepsilon^2)$
Sensitivity	$\mathrm{s}(h) \leq d$	$\mathrm{s}(g) \leq O(d^{13}/\varepsilon^2)$

Compared to these algorithms, our algorithm in Corollary 2 solves an incomparable problem with efficiency parameters that compare rather favorably with theirs. Notably, our time and query complexities both depend *polylogarithmically* on s instead of exponentially and super-linearly respectively.

Turning to the parameterized setting, Kearns and Ron [33] gave a tester with time and query complexities $\mathrm{poly}(n^n, (\log s)^n)$ that distinguishes size-s decision trees over $[0,1]^n$ from functions that are $(\frac{1}{2} - n^{-\Theta(n)})$-far from size-$\mathrm{poly}(2^n, s)$ decision trees. The parameters of this result are such that one should think of the dimension "n" as being a constant rather than an asymptotic parameter.

Property reconstruction

Property reconstruction was introduced by Ailon, Chazelle, Comandur, and Liu [2]. (See also the work of Austin and Tao [3], who termed such algorithms "repair algorithms".) Reconstruction has since been studied for a number of properties, including monotone functions [2, 43, 4], hypergraph properties [3], convexity [21], expanders [32], Lipschitz functions [30], graph connectivity and diameter [17], and error correcting codes [18]. Property reconstruction falls within the *local computation algorithms* framework of Rubinfeld, Tamir, Vardi, and Xie [42].

The paper of Blanc, Gupta, Lange, and Tan [6] designs a decision tree learning algorithm that is amenable to *learnability estimation* [35, 10]: given a training set S of *unlabeled* examples, the performance of this algorithm $\mathcal{A}$ trained on S – that is, the generalization error of the hypothesis that $\mathcal{A}$ would construct if we were to label all of S and train $\mathcal{A}$ on it – can be accurately estimated by labeling only a small number of the examples in S. Their

techniques can be used to derive a reconstruction algorithm that achieves guarantees similar to those in Theorem 1, but only for *monotone* functions f. This limitation is inherent: as noted in [6], their algorithm is fails for non-monotone functions.

1.2.1 The work of Bhsouty and Haddad-Zaknoon

Subsequent to the posting of our work to the ArXiv, Bshouty and Haddad-Zaknoon [16] have given a tester that is closely related, but incomparable, to Corollary 2. Their tester:

- Makes $\mathrm{poly}(s, 1/\varepsilon)$ queries to f, runs in $\mathrm{poly}(n, 1/\varepsilon)$ time, and
- Accepts w.h.p. if f is exactly a size-s decision tree;
- Rejects w.h.p. if f is ε-far from size-$(s/\varepsilon)^{O(\log(s/\varepsilon))}$ decision trees.

Comparing [16]'s tester to ours, their query complexity is independent of n (whereas ours has a $\log n$ dependence), and the size of decision trees in their reject condition is only $(s/\varepsilon)^{O(\log(s/\varepsilon))}$ (whereas we require $s^{O((\log s)^2/\varepsilon^3)}$).

On the other hand, our tester is tolerant and has query complexity that achieves a polylogarithmic instead of polynomial dependence on s. Furthermore, [16] does not give a reconstruction algorithm, while that is the main contribution of our work.

1.3 Future directions

We list a few concrete avenues for future work suggested by our results:

- *Tighter connections between testing and learning:* Our tester rejects functions that are $\Omega(\varepsilon)$-far from quasipoly(s) decision trees, and Theorem 3 shows that a tester that rejects functions that are $\Omega(\varepsilon)$-far from size-s decision trees would yield a comparably efficient algorithm for properly learning decision trees. A concrete avenue for future work is to narrow this gap between quasipoly(s) and s, with the ultimate goal of getting them to match.

  There are also other ways in which Theorem 3 could be strengthened: Do *non-tolerant* testers for decision trees yield proper learning algorithms? Do tolerant testers yield proper learning algorithms with *agnostic* guarantees?
- *Improved reconstruction algorithms and testers for other properties:* The reconstruction algorithms that we obtain for the properties listed in Table 1 follow by combining Theorem 1 with known relationships between these measures and decision tree complexity. It would be interesting to obtain improved parameters by designing reconstruction algorithms that are tailored to each of these properties, without going through decision trees.

  The same questions can be asked of property testers, and about properties that are not known to be quantitatively related to decision tree size. Can we achieve similar exponential improvements in the time and query complexities of non-parameterized testers by relaxing to the parameterized setting? Theorem 3 could be viewed as suggesting that for certain properties, efficient algorithms may only be possible in the parameterized setting.

Finally, we mention that there remains a large gap in the known bounds on the query complexity of non-tolerant testing of decision trees in the non-parameterized setting: the current best upper bound is $\tilde{O}(s)$ [15, 19] whereas the current best lower bound is $\Omega(\log s)$ [23, 5]. It would be interesting to explore whether our techniques could be useful in closing this exponential gap.

Notation

All probabilities and expectations are with respect to the uniform distribution unless otherwise stated; we use boldface (e.g. $\boldsymbol{x}$) to denote random variables. For two functions $f, g : \{\pm 1\}^n \to \{\pm 1\}$, we write $\mathrm{dist}(f, g)$ to denote the quantity $\Pr[f(\boldsymbol{x}) \neq g(\boldsymbol{x})]$. We say that f and g are *ε-close* if $\Pr[f(\boldsymbol{x}) \neq g(\boldsymbol{x})] \leq \varepsilon$, and *$\varepsilon$-far* otherwise.

For a function $f : \{\pm 1\}^n \to \{\pm 1\}$, a decision tree T over the same variables as f, and a node v in T, we write f_v to denote the subfunction of f obtained by restricting f according to the root-to-v path in T. We write $|v|$ to denote the depth of v within T, and so the probability that a uniform random $\boldsymbol{x} \sim \{\pm 1\}^n$ reaches v is $2^{-|v|}$.

2 Proofs of Theorem 1 and Theorem 2

Our proof of Theorem 1 has two main components:

- A structural lemma about functions f that are opt_s-close to a size-s decision tree $T^\star$. While we have no information about the structure of this tree $T^\star$ that f is opt_s-close to, we will show that f is $O(\mathrm{opt}_s + \varepsilon)$-close to a tree $T^\diamond$ of size $S = S(s, \varepsilon)$ with a very specific structure.
- An algorithmic component that leverages this specific structure of $T^\diamond$ to show that for any input $x \in \{\pm 1\}^n$, the value of $T^\diamond(x)$ can be computed with only $\log S \cdot \log n$ queries to f.

Section 2.1 will be devoted to the structural lemma and Section 2.2 to the algorithmic component. We prove Theorem 1 in Section 2.2.2, and we derive Corollary 2 as a simple consequence of Theorem 1 in Section 2.3.

2.1 Structural component of Theorem 1

▶ **Definition 5** (Noise sensitivity). *The noise sensitivity of $f : \{\pm 1\}^n \to \{\pm 1\}$ at noise rate p is the quantity*

$$\mathrm{NS}_p(f) \coloneqq \Pr[f(\boldsymbol{x}) \neq f(\boldsymbol{y})],$$

where $\boldsymbol{x} \sim \{\pm 1\}^n$ is uniform random and $\boldsymbol{y} \sim_p \boldsymbol{x}$ is a p-noisy copy of $\boldsymbol{x}$, obtained from $\boldsymbol{x}$ by independently rerandomizing each coordinate with probability p.

We assign each coordinate $i \in [n]$ of a function f a *score*, which measures the expected decrease in the noise sensitivity of f if x_i is queried:

▶ **Definition 6** (Score of a variable). *Given a function $f : \{\pm 1\}^n \to \{\pm 1\}$, noise rate $p \in (0, 1)$, and coordinate $i \in [n]$, the score of x_i is defined as*

$$\mathrm{Score}_i(f, p) = \mathrm{NS}_p(f) - \mathop{\mathbb{E}}_{\boldsymbol{b} \in \{\pm 1\}} \left[\mathrm{NS}_p(f_{x_i = \boldsymbol{b}})\right].$$

(Our notion of score is equivalent, up to scaling factors depending on p, to the notion of "noisy influence" as in defined in O'Donnell's monograph [38]. We use our definition of score as it simplifies our presentation.) We are now ready to define the tree $T^\diamond$ described at the beginning of this section and state our structural lemma.

▶ **Definition 7.** *For a function $f : \{\pm 1\}^n \to \{\pm 1\}$, parameters $d \in \mathbb{N}$ and $p \in (0,1)$, we write $T_f^{d,p}$ to denote the complete decision tree of depth d defined as follows:*

- *At every internal node v, query x_i where $i \in [n]$ maximizes $\mathrm{Score}_i(f_v, p)$.*[2]
- *Label every leaf ℓ with $\mathrm{sign}(\mathbb{E}[f_\ell])$.*

▶ **Lemma 8** (Structural lemma). *Let $f : \{\pm 1\}^n \to \{\pm 1\}$ be opt_s-close to a size-s decision tree. Then for $d = O((\log s)^3/\varepsilon^3)$ and $p = \varepsilon/(\log s)$, we have $\mathrm{dist}(f, T_f^{d,p}) \le O(\mathrm{opt}_s) + \varepsilon$.*

While Lemma 8 covers the main essence of our structural result, we'll need a slightly more robust version.

▶ **Lemma 9** (Robust version of Lemma 8). *Let $f : \{\pm 1\}^n \to \{\pm 1\}$ be opt_s-close to a size-s decision tree. For $d = O((\log s)^3/\varepsilon^3)$, $p = \varepsilon/(\log s)$, and $\tau = O(\varepsilon^3/(\log s)^3)$, let T be any complete decision tree of depth d satisfying:*

- *At every internal node v, the variable x_i that is queried at this node satisfies:*

$$\mathrm{Score}_i(f_v, p) \ge \max_{j \in [n]} \{\mathrm{Score}_j(f_v, p)\} - \tau.$$

- *Every leaf ℓ such that $|\mathbb{E}[f_\ell]| > \varepsilon$ is labeled $\mathrm{sign}(\mathbb{E}[f_\ell])$.*

Then $\mathrm{dist}(f, T) \le O(\mathrm{opt}_s + \varepsilon)$.

The proofs of Lemmas 8 and 9 are in the full version of this paper.

2.2 Algorithmic component of Theorem 1

2.2.1 Query-efficient simultaneous score estimation

We begin by designing a query-efficient subroutine that simultaneously estimates the scores of all n variables of a function f. The fact that we are able to do so with $O(\log n)$ queries, as opposed to $\Omega(n)$ as would be required by a naive approach, will be a key component in the query efficiency of our reconstructor.

▶ **Theorem 10** (Score estimator). *There is an algorithm which, given query access to a function $f : \{\pm 1\}^n \to \{\pm 1\}$, noise rate $p \in (0,1)$, accuracy parameter $\tau \in (0,1)$, and confidence parameter $\delta \in (0,1)$, for*

$$q = O\left(\frac{\log n + \log(1/\delta)}{\tau^2}\right)$$

makes $O(q)$ queries, runs in $O(qn)$ time, and returns estimates $\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n$ such that, with probability at least $1 - \delta$, satisfies

$$\left|\boldsymbol{\eta}_i - \mathrm{Score}_i(f, p)\right| < \tau \quad \text{for all } i \in [n].$$

We prove Theorem 10 by first giving a 2-query algorithm, UnbiasedEstimator (Figure 1), that runs in $O(n)$ time and outputs unbiased estimates of all n scores. The algorithm of Theorem 10 takes the mean of multiple runs of that unbiased estimator, with its guarantees following from a simple concentration bound.

▶ **Lemma 11** (Analysis of UnbiasedEstimator). *For any $f : \{\pm 1\}^n \to \{\pm 1\}$ and $p \in (0,1)$, let $\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n$ be the outputs of UnbiasedEstimator(f, p). Then*

$$\mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}}[\boldsymbol{\eta}_i] = \mathrm{Score}_i(f, p) \quad \text{for all } i \in [n].$$

[2] Ties are arbitrarily broken; our results hold regardless of how ties are broken.

UnbiasedEstimator(f, p):

Input: Query access to a function $f : \{\pm 1\}^n \to \{\pm 1\}$ and a noise rate $p \in (0, 1)$.
Output: Unbiased estimates of $\text{Score}_i(f, p)$ for all $i \in [n]$.

1. Choose $\boldsymbol{x} \in \{\pm 1\}^n$ uniformly at random and generate a p-noisy copy $\boldsymbol{y}$ of $\boldsymbol{x}$.
2. For each $i \in [n]$, return the estimate

$$\boldsymbol{\eta}_i = \mathbb{1}\big[f(\boldsymbol{x}) \neq f(\boldsymbol{y})\big] \cdot \left(1 - \frac{1}{1 - \frac{p}{2}} \cdot \mathbb{1}[\boldsymbol{x}_i = \boldsymbol{y}_i]\right).$$

Figure 1 UnbiasedEstimator computes unbiased estimates of the scores of all variables of a function f.

Proof. We first note that $\Pr[f(\boldsymbol{x}) \neq f(\boldsymbol{y})]$ is $\text{NS}_p(f)$ by definition. Therefore, it is enough for us to prove that

$$\mathop{\mathbb{E}}_{\boldsymbol{b} \in \{\pm 1\}} \big[\text{NS}_p(f_{x_i = \boldsymbol{b}})\big] = \frac{1}{1 - \frac{p}{2}} \cdot \Pr_{\boldsymbol{x}, \boldsymbol{y}} \left[f(\boldsymbol{x}) \neq f(\boldsymbol{y}) \text{ and } \boldsymbol{x}_i = \boldsymbol{y}_i\right]. \tag{1}$$

Given the above equation, the desired result holds by linearity of expectation and the definition of score. Consider the distribution over $(\boldsymbol{x}, \boldsymbol{y})$ conditioned on the event that $b = \boldsymbol{x}_i = \boldsymbol{y}_i$. That distribution is equivalent to if we picked $\boldsymbol{x}$ randomly from the domain of $f_{x_i = b}$ and selected $\boldsymbol{y}$ by rerandomizing each coordinate in that domain with probability p. Therefore,

$$\begin{aligned}
\text{NS}_p(f_{x_i = b}) &= \Pr_{\boldsymbol{x}, \boldsymbol{y}}[f(\boldsymbol{x}) \neq f(\boldsymbol{y}) \,|\, b = \boldsymbol{x}_i = \boldsymbol{y}_i] \\
&= \frac{1}{\Pr_{\boldsymbol{x}, \boldsymbol{y}}[b = \boldsymbol{x}_i = \boldsymbol{y}_i]} \cdot \Pr_{\boldsymbol{x}, \boldsymbol{y}}[f(\boldsymbol{x}) \neq f(\boldsymbol{y}) \text{ and } b = \boldsymbol{x}_i = \boldsymbol{y}_i].
\end{aligned}$$

We now prove Equation (1):

$$\begin{aligned}
\mathop{\mathbb{E}}_{\boldsymbol{b} \in \{\pm 1\}} \big[\text{NS}_p(f_{x_i = \boldsymbol{b}})\big] &= \mathop{\mathbb{E}}_{\boldsymbol{b} \in \{\pm 1\}} \left[\frac{1}{\Pr_{\boldsymbol{x}, \boldsymbol{y}}[\boldsymbol{b} = \boldsymbol{x}_i = \boldsymbol{y}_i]} \cdot \Pr_{\boldsymbol{x}, \boldsymbol{y}}[f(\boldsymbol{x}) \neq f(\boldsymbol{y}) \text{ and } \boldsymbol{b} = \boldsymbol{x}_i = \boldsymbol{y}_i]\right] \\
&= \frac{1}{\frac{1}{2} \cdot \Pr_{\boldsymbol{x}, \boldsymbol{y}}[\boldsymbol{x}_i = \boldsymbol{y}_i]} \mathop{\mathbb{E}}_{\boldsymbol{b} \in \{\pm 1\}} \left[\Pr_{\boldsymbol{x}, \boldsymbol{y}}[f(\boldsymbol{x}) \neq f(\boldsymbol{y}) \text{ and } \boldsymbol{b} = \boldsymbol{x}_i = \boldsymbol{y}_i]\right] \\
&= \frac{1}{\frac{1}{2} \cdot (1 - \frac{p}{2})} \cdot \frac{1}{2} \cdot \Pr_{\boldsymbol{x}, \boldsymbol{y}} \left[f(\boldsymbol{x}) \neq f(\boldsymbol{y}) \text{ and } \boldsymbol{x}_i = \boldsymbol{y}_i\right] \\
&= \frac{1}{1 - \frac{p}{2}} \cdot \Pr_{\boldsymbol{x}, \boldsymbol{y}} \left[f(\boldsymbol{x}) \neq f(\boldsymbol{y}) \text{ and } \boldsymbol{x}_i = \boldsymbol{y}_i\right].
\end{aligned}$$

Lemma 11 then holds by linearity of expectation. ◀

We now prove Theorem 10.

Proof of Theorem 10. The algorithm runs UnbiasedEstimator(f, p) q times and then outputs the means of each returned estimates. Each estimate from UnbiasedEstimator is bounded between -1 and 1. By Hoeffding's inequality, for any $i \in [n]$,

$$\Pr\left[\left|\boldsymbol{\eta}_i - \text{Score}_i(f, p)\right| \geq \tau\right] \leq -\exp_e\left(-\frac{q \cdot \tau^2}{2}\right).$$

For q as in Theorem 10, the above probability is at most δ/n. By union bound, all estimates are accurate within $\pm\tau$ with probability at least $1-\delta$.

Finally, this algorithm uses only $2q = O(q)$ queries. Each run of UnbiasedEstimator estimator takes $O(n)$ time to construct the query and compute all the estimates, so the entire algorithm takes $O(qn)$ time. ◀

2.2.2 Proof of Theorem 1

We prove Theorem 1 by providing an algorithm, Reconstructor (Figure 2), which assumes query access to a function $f : \{\pm1\}^n \to \{\pm1\}$ and provides fast query access to a tree T meeting the criteria of Theorem 1. We build off a simple observation that also underlies [6]: to determine the output of a decision tree T on a particular input z, it suffices to build the root-to-leaf path corresponding to z, which can be exponentially faster than building the entire tree. Our algorithm is different from [6]'s; as mentioned in the introduction their algorithm is tailored to monotone functions, and is known to fail for non-monotone ones. We on the other hand leverage the specific structure of T established in Section 2.1 together with the query-efficient score estimator from Section 2.2.1 in our design and analysis of Reconstructor.

Reconstructor maintains a partial tree T° containing all the root-to-leaf paths in T corresponding to queries received so far. In the pseudocode for Reconstructor, we use the notation $T^\circ_{\text{internal}}(\alpha) \in [n] \cup \{\varnothing\}$ to indicate the variable queried in $[n]$ at internal node α of the partial tree T°, or $\varnothing$ if that node has not yet been built. Similarly, $T^\circ_{\text{leaf}}(\alpha) \in \{-1, 1, \varnothing\}$ indicates the value at leaf α in T°, or $\varnothing$ if that value has not yet been decided.

Theorem 1 follows from the following two lemmas, showing the correctness and efficiency of Reconstructor respectively.

▶ **Lemma 12** (Correctness of Reconstructor). *For any $f : \{\pm1\}^n \to \{\pm1\}$, $s \in \mathbb{N}$, $\varepsilon \in (0, \frac{1}{2})$, $\delta \in (0,1)$, and sequence of inputs $z^{(1)}, \dots, z^{(m)} \in \{\pm1\}^n$, the outputs of Reconstructor are consistent with some decision tree T where*

- *T has size $s^{O((\log s)^2/\varepsilon^3)}$,*
- *$\operatorname{dist}(T, f) \le O(\operatorname{opt}_s) + \varepsilon$ with probability at least $1-\delta$.*

Proof. The outputs of Reconstructor are always consistent with T° and the depth of T° is always capped at d. Let T be the tree that T° would be if every $x \in \{\pm1\}^n$ were given as an input to Reconstructor. Then, T has size at most $2^d = s^{O((\log s)^2/\varepsilon^3)}$, and every output is consistent with T.

If all score estimates in Step 3(b)i are accurate to $\pm\frac{\tau}{2}$ and expectation estimates is Step 3c are accurate to $\pm\frac{\varepsilon}{4}$, then T meets the criteria of Lemma 9 and therefore $\operatorname{dist}(T, f) \le O(\operatorname{opt}_s) + \varepsilon$. The number of time scores are estimated in Step 3(b)i is at most the number of internal nodes of T, which is $2^d - 1$. Similarly, the number of expectation estimates in Step 3(b)i is at most the number of leaves of T, which is 2^d. By union bound over the possible failures, we see that the failure probability is at most δ. ◀

▶ **Lemma 13** (Efficiency of Reconstructor). *For any $f : \{\pm1\}^n \to \{\pm1\}$, $s \in \mathbb{N}$, $\varepsilon \in (0, \frac{1}{2})$, $\delta \in (0,1)$, particular input $z \in \{\pm1\}^n$, and*

$$q = O\left(\frac{(\log s)^9 \cdot (\log n) \cdot \log(1/\delta)}{\varepsilon^9}\right),$$

upon receiving z as input, Reconstructor$(f, s, \varepsilon, \delta)$ uses $O(q)$ queries and $O(qn)$ time to return an output.

RECONSTRUCTOR$(f, s, \varepsilon, \delta)$:

Input: Query access to a function $f : \{\pm 1\}^n \to \{\pm 1\}$, size parameter s, error parameter ε, and failure probability δ.

Output: Query access to a decision tree T that satisfies $\text{dist}(f, T) \leq O(\text{opt}_s) + \varepsilon$ with probability at least $1 - \delta$.

1. Set parameters d, p, and τ as in Lemma 9.
2. Initialize T° to be the empty partial tree.
3. Upon receiving an input $z \in \{\pm 1\}^n$:
   a. Initialize α to be the root of T°.
   b. Repeat d times.
      i. If $T^\circ_{\text{internal}}(\alpha)$ is $\varnothing$ use the estimator from Theorem 10 to compute estimates of $\text{Score}_i(f_\alpha, p)$ with additive accuracy $\pm\frac{\tau}{2}$ and failure probability $O(\frac{\delta}{2^d})$ for all $i \in [n]$ and set $T^\circ_{\text{internal}}(\alpha)$ to the variable with highest estimated score.
      ii. For $i = T^\circ_{\text{internal}}(\alpha)$, If $\overline{z}_i$ is 1, set α to its right child. Otherwise, set α to its left child.
   c. If $T^\circ_{\text{leaf}}(\alpha)$ is $\varnothing$, use random samples to estimate $\mathbb{E}[f_\ell]$ to additive accuracy $\pm\frac{\varepsilon}{4}$ with failure probability $O(\frac{\delta}{2^d})$ and set $T^\circ_{\text{leaf}}(\alpha)$ to whichever of $\{\pm 1\}$ that estimate is closer to.
   d. Output $T^\circ_{\text{leaf}}(\alpha)$.

Figure 2 RECONSTRUCTOR gives efficient query access to a decision tree is close to f with high probability.

Proof. On each input, the estimator from Theorem 10 is used up to d times. Each uses

$$q_{\text{inner}} \coloneqq O\left(\frac{\log n + \log(2^d/\delta)}{\tau^2}\right) = O\left(\frac{\log n + d + \log(1/\delta)}{\tau^2}\right)$$

queries and $O(q_{\text{inner}} n)$ time. By Hoeffding's inequality, it is sufficient to take

$$q_{\text{leaf}} \coloneqq O\left(\frac{\log(2^d/\delta)}{\varepsilon^2}\right) = O\left(\frac{d + \log(1/\delta)}{\varepsilon^2}\right)$$

random samples in Step 3c. Therefore, the total number of queries used is

$$\begin{aligned} q &= q_{\text{inner}} + q_{\text{leaf}} \\ &= O\left(\frac{\log n + d + \log(1/\delta)}{\tau^2}\right) + O\left(\frac{d + \log(1/\delta)}{\varepsilon^2}\right) \\ &= O\left(\frac{\log n + ((\log s)^3/\varepsilon^3) + \log(1/\delta)}{\varepsilon^6/(\log s)^6} + \frac{((\log s)^3/\varepsilon^2) + \log(1/\delta)}{\varepsilon^2}\right) \\ &= O\left(\frac{(\log s)^9 \cdot (\log n) \cdot \log(1/\delta)}{\varepsilon^9}\right). \end{aligned}$$

The time to prepare all queries is $O(qn)$, and all other computation is asymptotically faster. ◀

▶ Remark 14 (Local reconstruction). We remark that our reconstruction algorithm can be made local in the sense of [43]. They define a reconstruction algorithm, $\mathcal{A}$, to be local, if the output of $\mathcal{A}$ on some input z is a *deterministic* and easy to compute function of z and some small random string ρ. This allows queries to the reconstructor to be answered in parallel, as long as the random string ρ is shared. To make our reconstructor local, we note that the only place randomness is used is in generating samples consistent with some restriction α. We can set ρ to be n bits per sample the constructor might wish to generate. Since the total number of samples the reconstructor needs per input is $\mathrm{poly}(\log s, 1/\varepsilon, \log(1/\delta)) \cdot \log n$, we have

$$|\rho| = \mathrm{poly}(\log s, 1/\varepsilon, \log(1/\delta)) \cdot n \log n$$

On a particular input, the local reconstructor starts with T° being the empty tree. Whenever it wishes to produce a random sample consistent with α, it sets the $\boldsymbol{x} \in \{\pm 1\}^n$ to be next n bits of ρ and then uses $\boldsymbol{x}_\alpha$ for the sample. It's easy to see that this algorithm will keep T° consistent between different runs because it will always compute the same variable as having the highest score given some restriction. Furthermore, the analysis goes through without issue. The only difference between this analysis and one where fresh random bits are used to for each sample is that the queries of different paths may be correlated. In our proof of Lemma 12, we use a union bound to ensure all estimates obtained through sampling are accurate, and that union bound holds regardless of whether those estimates are independent.

2.3 Proof of Corollary 2

In this section we derive Corollary 2 as a simple consequence of Theorem 1. The connection between reconstruction and tolerant testing has been noted in other works (see e.g. [17, 11]); we provide a proof here for completeness.

▶ **Theorem 15** (Corollary 2 restated). *There is an algorithm which, given query access to* $f : \{\pm 1\}^n \to \{\pm 1\}$ *and parameters* $s \in \mathbb{N}$ *and* $\varepsilon, \delta \in (0, 1)$*, runs in* $\mathrm{poly}(\log s, 1/\varepsilon) \cdot n \log n \cdot \log(1/\delta)$ *time, makes* $\mathrm{poly}(\log s, 1/\varepsilon) \cdot \log n \cdot \log(1/\delta)$ *queries to* f*, and*

- *Accepts w.p. at least* $1 - \delta$ *if* f *is* ε*-close to a size-*s *decision tree;*
- *Rejects w.p. at least* $1 - \delta$ *if* f *is* $\Omega(\varepsilon)$*-far from size-*$s^{O((\log s)^2/\varepsilon^3)}$ *decision trees.*

Proof. The algorithm chooses m uniform random inputs, $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)} \sim \{\pm 1\}^n$ where $m = O(\log(1/\delta)/\varepsilon^2)$. Let $\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(m)} \in \{\pm 1\}$ be the output of $\textsc{Reconstructor}(f, s, \varepsilon, \delta)$. The tester rejects if $\mathbb{E}_{\boldsymbol{i} \in [m]}[f(\boldsymbol{x}^{(\boldsymbol{i})}) \neq \boldsymbol{b}^{(\boldsymbol{i})}] > \Omega(\varepsilon)$ and accepts otherwise.

First, we consider the case where f is ε-close to a size-s decision tree (i.e. $\mathrm{opt}_s \leq \varepsilon$). By Lemma 12, with probability at least $1 - \delta$ the outputs of $\textsc{Reconstructor}$ are consistent with a tree, T, satisfying $\mathrm{dist}(T, f) \leq O(\varepsilon)$. By Hoeffding's inequality,

$$\Pr_{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}} \left[\mathop{\mathbb{E}}_{\boldsymbol{i} \in [m]} \left[f(\boldsymbol{x}^{(\boldsymbol{i})}) \neq \boldsymbol{b}^{(\boldsymbol{i})} \right] > \Omega(\varepsilon) \right] \leq \exp(-2m\varepsilon^2) \leq \delta.$$

By a union bound, the tester rejects with probability at most $\delta + \delta = 2\delta$.

We next consider the case where f is $\Omega(\varepsilon)$-far from size-$s^{O((\log s)^2/\varepsilon^3)}$ decision trees. By Lemma 12 it is guaranteed to be consistent. A similar argument to the first case shows that the probability of acceptance is at most $\delta + \exp(-2m\varepsilon^2) = 2\delta$. Finally, the efficiency of this tester is a consequence of Lemma 13 and our choice of $m = O(\log(1/\delta)/\varepsilon^2)$. ◀

3 Proof of Corollary 4

We first restate Theorem 1 with decision tree *depth* instead of *size* as the complexity measure:

▶ **Theorem 16** (Theorem 1 in terms of decision tree depth). *There is a randomized algorithm which, given query access to $f : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $d \in \mathbb{N}$ and $\varepsilon \in (0,1)$, provides query access to a fixed decision tree T where*

- *T has depth $O(d^3/\varepsilon^2)$,*
- *$\mathrm{dist}(T, f) \le O(\mathrm{opt}_d) + \varepsilon$ w.h.p., where opt_d denotes the distance of f to the closest depth-d decision tree.*

Every query to T is answered in $\mathrm{poly}(d, 1/\varepsilon) \cdot n \log n$ time and with $\mathrm{poly}(d, 1/\varepsilon) \cdot \log n$ queries to f.

To see that our proof of Theorem 1 also establishes Theorem 16, we use the fact that every depth-d decision tree has size $\le 2^d$, and recall that the tree T that the algorithm of Theorem 1 provides query access to is a complete tree and hence has depth logarithmic in its size.

Decision tree depth and Fourier degree of boolean functions are known to be polynomially related:

▶ **Fact 17** (Decision tree depth vs. Fourier degree [37, 44]). *For $g : \{\pm 1\}^n \to \{\pm 1\}$ let $\deg(g)$ denote g's Fourier degree and $D(g)$ denote the depth of the shallowest decision tree that computes g. Then $\deg(g) \le D(g)$ and $D(g) \le \deg(g)^3$.*

We first observe Theorem 16 and Fact 17 already gives a quantitatively weaker version of Corollary 4 where g has degree $O(d^9/\varepsilon^2)$. To see this $f : \{\pm 1\}^n \to \{\pm 1\}$ be opt_d-close to a degree-d function $h : \{\pm 1\}^n \to \{\pm 1\}$. By Fact 17, $D(h) \le \deg(h)^3$, and so the algorithm of Theorem 16 provides query access to a decision tree $T : \{\pm 1\}^n \to \{\pm 1\}$ that is $(O(\mathrm{opt}_d) + \varepsilon)$-close to f and where the depth of T is $O(D(h)^3/\varepsilon^2) = O(\deg(h)^9/\varepsilon^2)$. Applying Fact 17 again, we conclude that $\deg(T) \le D(T) \le O(\deg(h)^9/\varepsilon^2)$.

To obtain the sharper bound of $O(\deg(h)^7/\varepsilon^2)$, we observe that the proof of Lemma 8 in fact bounds the depth of T by $O(D(h)^2 \mathrm{Inf}(h)/\varepsilon^2)$. Influence and degree of boolean functions are related via the following basic fact (see e.g. [38, Theorem 37]):

▶ **Fact 18.** *For all $h : \{\pm 1\}^n \to \{\pm 1\}$, we have $\mathrm{Inf}(h) \le \deg(h)$.*

Therefore, we can bound the degree of T by $O(D(h)^2 \mathrm{Inf}(h)/\varepsilon^2) \le O(\deg(h)^7/\varepsilon^2)$.

Guarantees for the other measures listed in Table 1 follow from similar calculations and known quantitative relationships between these measures and decision tree complexity; the current best bounds are summarized in Table 1 of [1].

4 Proof of Theorem 3

In this section, we prove the following theorem:

▶ **Theorem 19** (Tolerant testing of DTs ⇒ Proper learning of DTs). *Let $c > 0$ be an absolute constant and $\mathcal{A}$ be an algorithm with the following guarantee. Given query access to $f : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $s \in \mathbb{N}$ and $\varepsilon \in (0,1)$, the algorithm $\mathcal{A}$:*

- *Accepts w.h.p. if f is ε-close to a size-s decision tree;*
- *Rejects w.h.p. if f is $(c\varepsilon)$-far from all size-s decision trees.*

Then there is an algorithm $\mathcal{B}$ with the following guarantee. Given parameters $s' \in \mathbb{N}$ and $\varepsilon' \in (0,1)$, and query access to a function $g : \{\pm 1\}^n \to \{\pm 1\}$ that is computed by a size-s' decision tree, $\mathcal{B}$ makes $\mathrm{poly}(s', n, 1/\varepsilon')$ calls to $\mathcal{A}$, each with parameters $s \leq s'$ and $\varepsilon \geq \mathrm{poly}(1/s', \varepsilon')$, and produces a decision tree which is ε'-close to g with high probability. Furthermore, the auxiliary computation that g does takes time $\mathrm{poly}(n, s', 1/\varepsilon')$.

Theorem 3 follows as a special case of Theorem 19.
We prove Theorem 19 in two steps:

1. A tolerant tester implies an algorithm for estimating the distance of any function to the class of size-s decision trees. This is well known [40] and applies to any function class, not just decision trees.
2. An algorithm for estimating distance to decision trees implies a proper learner for decision trees. Here, we take advantage of the structure of decision trees.

For a function $g : \{\pm 1\}^n \to \{\pm 1\}$ and $s \in \mathbb{N}$, we write $\mathrm{opt}_s(g)$ to denote the distance of g to the closest size-s decision tree.

▶ **Lemma 20** (Tolerant testing ⇒ distance estimation [40]). *Let c and $\mathcal{A}$ be as in Theorem 19. There exists an estimator $\mathcal{E}$ with the following guarantee. Given query access to $g : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $s' \in \mathbb{N}$ and $\gamma \in (0,1)$, the estimator $\mathcal{E}$ makes c/γ calls to $\mathcal{A}$ and returns an $\boldsymbol{\eta}$ that with high probability satisfies*

$$\boldsymbol{\eta} \leq \mathrm{opt}_s(g) \leq c \cdot \boldsymbol{\eta} + \gamma.$$

Furthermore, the auxiliary computation of g takes time $O(c/\gamma)$.

Proof. The algorithm $\mathcal{E}$ runs $\mathcal{A}$ with $\varepsilon = \frac{\gamma}{c}, \frac{2\gamma}{c}, \frac{3\gamma}{c}, \ldots, 1$, and sets $\boldsymbol{\eta}$ to be the largest ε for which $\mathcal{A}(g, s, \varepsilon)$ rejects. Since $\mathcal{A}(g, s, \boldsymbol{\eta})$ rejected,

$$\boldsymbol{\eta} < \mathrm{opt}_s(g)$$

with high probability. Furthermore, since $\mathcal{A}(g, s, \boldsymbol{\eta} + \frac{\gamma}{c})$ accepted,

$$\begin{aligned}\mathrm{opt}_s(g) &< c \cdot \left(\boldsymbol{\eta} + \tfrac{\gamma}{c}\right) \\ &= c \cdot \boldsymbol{\eta} + \gamma\end{aligned}$$

with high probability. Finally, we note that $\mathcal{E}$ indeed makes c/γ calls to $\mathcal{A}$, and aside from those calls, it only needs to make a single pass over the output of those calls and return the largest ε that led to a rejection, which takes time $O(c/\gamma)$. ◀

We are now ready to state our algorithm, BuildDT (Figure 3), for properly learning size-s' decision trees. BuildDT will additionally take in a depth parameter d that will facilitate our analysis of it (looking ahead, d will be chosen to be $O(\log(s'/\varepsilon'))$ in our proof of Theorem 19).

▶ **Lemma 21** (Error of BuildDT). *For all functions $f : \{\pm 1\}^n \to \{\pm 1\}$ and parameters $s, d \in \mathbb{N}$ and $\gamma \in (0,1)$, the algorithm $\textsc{BuildDT}(f, s, d, \gamma)$ outputs a decision tree T satisfying*

$$\mathrm{dist}(T, f) \leq c^d \cdot \mathrm{opt}_s(f) + \gamma \cdot \frac{c^d - 1}{c - 1} + \frac{s}{2^{d+2}}. \tag{2}$$

BuildDT(f, s, d, γ):

Input: Query access to $f : \{\pm 1\}^n \to \{\pm 1\}$, parameters $s, d \in \mathbb{N}$ and $\gamma \in (0, 1)$.
Output: A size-s depth-d decision tree T.

1. If $s = 1$ or $d = 0$, return $\text{sign}(\mathbb{E}[f])$.
2. For each $i \in [n]$ and integers $s_0, s_1 \geq 1$ satisfying $s_0 + s_1 = s$:
   a. Use $\mathcal{E}$ from Lemma 20 to obtain estimates $\boldsymbol{\eta}(x_i = 0, s_0)$ and $\boldsymbol{\eta}(x_i = 1, s_1)$ that satisfy:
   $$\boldsymbol{\eta}(x_i = 0, s_0) \leq \text{opt}_{s_0}(f_{x_i=0}) \leq c \cdot \boldsymbol{\eta}(x_i = 0, s_0) + \gamma;$$
   $$\boldsymbol{\eta}(x_i = 1, s_1) \leq \text{opt}_{s_1}(f_{x_i=1}) \leq c \cdot \boldsymbol{\eta}(x_i = 1, s_1) + \gamma.$$
   b. Store $\text{error}(i, s_1, s_2) \leftarrow \frac{1}{2}\big(\boldsymbol{\eta}(x_i = 0, s_0) + \boldsymbol{\eta}(x_i = 1, s_1)\big)$.
3. Let $(i^\star, s_0^\star, s_1^\star)$ be the tuple that minimizes $\text{error}(i, s_0, s_1)$. Output the tree with $x_{i^\star}$ as its root, BuildDT$(f_{x_{i^\star}=0}, s_0^\star, d-1, \gamma)$ as its left subtree, and BuildDT$(f_{x_{i^\star}=1}, s_1^\star, d-1, \gamma)$ as its right subtree.

Figure 3 BuildDT computes a size-s depth-d decision tree that approximates a target function $f : \{\pm 1\}^n \to \{\pm 1\}$.

Proof. We proceed by induction on s and d. If $s = 1$, then in Step 1, BuildDT outputs the best decision tree of size 1. Therefore, $\text{dist}(T, f) \leq \text{opt}_s(f)$, satisfying Equation (2). If $d = 0$ and $s \geq 2$, then $\frac{s}{2^{d+2}} \geq \frac{2}{4} = \frac{1}{2}$. Furthermore, in Step 1, BuildDT always outputs a tree with error at most $\frac{1}{2}$. Therefore,

$$\text{dist}(T, f) \leq \frac{s}{2^{d+2}} \leq c^d \cdot \text{opt}_s(f) + \gamma \cdot \frac{c^d - 1}{c - 1} + \frac{s}{2^{d+2}}.$$

Finally, we consider the case where $d \geq 1$ and $s \geq 2$. Let T_{opt} be the size-s decision tree that is $\text{opt}_s(f)$ close to f. Let $x_{i_{\text{opt}}}$ the root of T_{opt}, and $s_{0,\text{opt}}$, $s_{1,\text{opt}}$ the sizes of the left and right subtrees of T_{opt} respectively. Since the estimates computed in Step 2a are underestimates of or equal to the true error (i.e. $\text{error}(i_{\text{opt}}, s_{0,\text{opt}}, s_{1,\text{opt}}) \leq \text{opt}_s(f)$), and since $i^\star, s_0^\star, s_1^\star$ are chosen in Step 3 to minimize the estimated error, we have

$$\text{error}(i^\star, s_0^\star, s_1^\star) \leq \text{error}(i_{\text{opt}}, s_{0,\text{opt}}, s_{1,\text{opt}}) \leq \text{opt}_s(f).$$

Finally, we bound $\text{dist}(T, f)$. Let T_0 and T_1 be the left and right subtrees of T. Then,

$$\begin{aligned}
\text{dist}(T, f) &= \tfrac{1}{2}\big(\text{dist}(T_0, f_{x_{i^\star}=0}) + \text{dist}(T_1, f_{x_{i^\star}=1})\big) \\
&\leq \tfrac{1}{2}\Big(c^{d-1} \cdot \text{opt}_{s_0^\star}(f_{x_{i^\star}=0}) + \gamma \cdot \frac{c^{d-1} - 1}{c - 1} + \frac{s_0^\star}{2^{d+1}} \\
&\quad + c^{d-1} \cdot \text{opt}_{s_1^\star}(f_{x_{i^\star}=1}) + \gamma \cdot \frac{c^{d-1} - 1}{c - 1} + \frac{s_1^\star}{2^{d+1}}\Big) \qquad \text{(Inductive hypothesis)} \\
&= c^{d-1} \cdot \tfrac{1}{2}\left(\text{opt}_{s_0^\star}(f_{x_{i^\star}=0}) + \text{opt}_{s_1^\star}(f_{x_{i^\star}=1})\right) + \gamma \cdot \frac{c^{d-1} - 1}{c - 1} + \frac{s_0^\star + s_1^\star}{2^{d+2}} \\
&\leq c^{d-1} \cdot \tfrac{1}{2}\big((c \cdot \boldsymbol{\eta}(x_{i^\star} = 0, s_0^\star) + \gamma) + (c \cdot \boldsymbol{\eta}(x_{i^\star} = 1, s_1^\star) + \gamma)\big) + \gamma \cdot \frac{c^{d-1} - 1}{c - 1} + \frac{s}{2^{d+2}} \\
&= c^d \cdot \tfrac{1}{2}\left(\boldsymbol{\eta}(x_{i^\star} = 0, s_0^\star) + \boldsymbol{\eta}(x_{i^\star} = 1, s_1^\star)\right) + c^{d-1} \cdot \gamma + \gamma \cdot \frac{c^{d-1} - 1}{c - 1} + \frac{s}{2^{d+2}}
\end{aligned}$$

$$= c^d \cdot \mathrm{error}(i^\star, s_0^\star, s_1^\star) + \gamma \cdot \frac{c^{d-1}(c-1) + c^{d-1} - 1}{c-1} + \frac{s}{2^{d+2}}$$
$$\leq c^d \cdot \mathrm{opt}_s(f) + \gamma \cdot \left(\frac{c^d - 1}{c-1}\right) + \frac{s}{2^{d+2}}.$$

The desired result holds by induction. ◀

For readability, Lemma 21 assumes that BuildDT is able to compute round($\mathbb{E}[f]$) in Step 1. To make BuildDT efficient, we would only estimate $\mathbb{E}[f]$ by querying f on uniform random inputs $\boldsymbol{x} \in \{\pm 1\}^n$. If those estimates are computed to accuracy ε', then each leaf of our tree can have up to ε' additional error. This is not an issue since it increases the total error of T, which is simply the average of the error at each leaf, by only ε'.

Finally, we prove Theorem 19:

Proof of Theorem 19. Our goal is to properly learn a size-s' decision tree $g : \{\pm 1\}^n \to \{\pm 1\}$ to accuracy ε'. To do so, we run BuildDT(g, s', d, γ), with d set to

$$d = \log(s'/\varepsilon') - 1,$$

and γ set to

$$\gamma = \frac{\varepsilon'}{2 \cdot \max(2,c)^d} = \frac{\varepsilon'}{2} \cdot (2^{-d})^{\max(1,\log c)} = \frac{\varepsilon'}{2} \cdot \left(\frac{\varepsilon'}{s'}\right)^{\max(1,\log c)}.$$

By Lemma 21, for T the tree BuildDT outputs,

$$\begin{aligned}\mathrm{dist}(T,g) &\leq c^d \cdot \mathrm{opt}_{s'}(g) + \gamma \cdot \frac{c^d - 1}{c-1} + \frac{s'}{2^{d+2}} \\ &\leq 0 + \frac{\varepsilon'}{2 \cdot \max(2,c)^d} \cdot \max(2,c)^d + \frac{s'}{2^{\log(s'/\varepsilon')+1}} \\ &\leq \frac{\varepsilon'}{2} + \frac{\varepsilon'}{2} = \varepsilon'.\end{aligned}$$

Hence, BuildDT produces the desired output. We next argue that it is efficient. During the recursion, BuildDT is called at most s' times in total. Each such call makes $O(ns')$ calls to $\mathcal{E}$. By Lemma 20, those calls to $\mathcal{E}$ each make c/ε calls to $\mathcal{A}$. Hence, the total number of calls to $\mathcal{A}$ is

$$O\left(\frac{n(s')^2}{\gamma}\right) = O\left(\frac{n(s')^2}{\varepsilon'} \cdot \left(\frac{s'}{\varepsilon'}\right)^{\max(1,\log c)}\right) = \mathrm{poly}(n, s', 1/\varepsilon').$$

The total auxiliary computation of BuildDT is bounded by the same quantity. Finally, each call to $\mathcal{A}$ is made with parameters s and ε where $s \leq s'$ and $\varepsilon = \frac{\varepsilon'}{2} \cdot \left(\frac{\varepsilon'}{s'}\right)^{\max(1,\log c)} \geq \mathrm{poly}(1/s', \varepsilon')$. ◀

References

1. Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of huang's sensitivity theorem. *arXiv preprint*, abs/2010.12629, 2020. `arXiv:2010.12629`.
2. Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008.
3. Tim Austin and Terence Tao. Testability and repair of hereditary hypergraph properties. *Random Structures & Algorithms*, 36(4):373–463, 2010.

4 Arnab Bhattacharyya, Elena Grigorescu, Madhav Jha, Kyomin Jung, Sofya Raskhodnikova, and David Woodruff. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. *SIAM Journal on Discrete Mathematics*, 26(2):618–646, 2012.

5 Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *computational complexity*, 21(2):311–358, 2012.

6 Guy Blanc, Neha Gupta, Jane Lange, and Li-Yang Tan. Estimating decision tree learnability with polylogarithmic sample complexity. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

7 Guy Blanc, Jane Lange, Mingda Qiao, and Li-Yang Tan. Properly learning decision trees in almost polynomial time. In *Proceedings of the 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, 2021.

8 Avirm Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–262, 1994.

9 Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Inform. Process. Lett.*, 42(4):183–185, 1992. doi:10.1016/0020-0190(92)90237-P.

10 Avrim Blum and Lunjia Hu. Active tolerant testing. In *Proceedings of the 31st Conference On Learning Theory (COLT)*, volume 75, pages 474–497, 2018.

11 Zvika Brakerski. Local property restoring, 2008. Manuscript.

12 Joshua Brody and Pooya Hatami. Distance-sensitive property testing lower bounds. *CoRR*, abs/1304.6685, 2013. arXiv:1304.6685.

13 Alon Brutzkus, Amit Daniely, and Eran Malach. ID3 learns juntas for smoothed product distributions. In *Proceedings of the 33rd Annual Conference on Learning Theory (COLT)*, pages 902–915, 2020.

14 Nader Bshouty. Exact learning via the monotone theory. In *Proceedings of 34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 302–311, 1993.

15 Nader Bshouty. Almost optimal testers for concise representations. In *Proceedings of the 24th International Conference on Randomization and Computation (RANDOM)*, pages 5:1–5:20, 2020.

16 Nader H. Bshouty and Catherine A. Haddad-Zaknoon. On learning and testing decision tree. *ArXiv preprint*, abs/2108.04587, 2021. arXiv:2108.04587.

17 Andrea Campagna, Alan Guo, and Ronitt Rubinfeld. Local reconstructors and tolerant testers for connectivity and diameter. In *Proceedings of the 17th International Workshop on Randomization and Computation (RANDOM)*, pages 411–424, 2013.

18 Sourav Chakraborty, Eldar Fischer, and Arie Matsliah. Query complexity lower bounds for reconstruction of codes. *Theory of Computing*, 10(19):515–533, 2014. doi:10.4086/toc.2014.v010a019.

19 Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Efficient sample extractors for juntas with applications. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 545–556, 2011.

20 Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Nearly tight bounds for testing function isomorphism. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1683–1702, 2011.

21 Bernard Chazelle and C Seshadhri. Online geometric reconstruction. In *Proceedings of the 22nd Annual Symposium on Computational Geometry (SoCG)*, pages 386–394, 2006.

22 Sitan Chen and Ankur Moitra. Beyond the low-degree algorithm: mixtures of subcubes and their applications. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 869–880, 2019.

23 Ilias Diakonikolas, Homin Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco Servedio, and Andrew Wan. Testing for concise representations. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–558, 2007.

24 Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

25 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998.

26 Thomas Hancock. Learning $k\mu$ decision trees on the uniform distribution. In *Proceedings of the 6th Annual Conference on Computational Learning Theory (COT)*, pages 352–360, 1993.

27 Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122, 1996.

28 Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach. *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

29 Jeffrey C. Jackson and Rocco A. Servedio. On learning random dnf formulas under the uniform distribution. *Theory of Computing*, 2(8):147–172, 2006. doi:10.4086/toc.2006.v002a008.

30 Madhav Jha and Sofya Raskhodnikova. Testing and reconstruction of lipschitz functions with applications to data privacy. *SIAM Journal on Computing*, 42(2):700–731, 2013.

31 Adam Kalai, Alex Samorodnitsky, and Shang-Hua Teng. Learning and smoothed analysis. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 395–404, 2009.

32 Satyen Kale, Yuval Peres, and Comandur Seshadhri. Noise tolerance of expanders and sublinear expansion reconstruction. *SIAM Journal on Computing*, 42(1):305–323, 2013.

33 Michael Kearns and Dana Ron. Testing problems with sublearning sample complexity. *Journal of Computer and System Sciences*, 61(3):428–456, 2000.

34 Adam Klivans and Rocco Servedio. Toward attribute efficient learning of decision lists and parities. *Journal of Machine Learning Research*, 7(Apr):587–602, 2006.

35 Weihao Kong and Gregory Valiant. Estimating learnability in the sublinear data regime. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 5460–5469, 2018.

36 Dinesh Mehta and Vijay Raghavan. Decision tree approximations of boolean functions. *Theoretical Computer Science*, 270(1-2):609–623, 2002.

37 Gatis Midrijānis. Exact quantum query complexity for total boolean functions. *arXiv preprint*, quant-ph/0403168, 2004. arXiv:quant-ph/0403168.

38 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

39 Ryan O'Donnell and Rocco Servedio. Learning monotone decision trees in polynomial time. *SIAM Journal on Computing*, 37(3):827–844, 2007.

40 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.

41 Ronald Rivest. Learning decision lists. *Machine learning*, 2(3):229–246, 1987.

42 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.

43 Michael Saks and Comandur Seshadhri. Local monotonicity reconstruction. *SIAM Journal on Computing*, 39(7):2897–2926, 2010.

44 Avishay Tal. Properties and applications of boolean function composition. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 441–454, 2013.

Sublinear-Round Parallel Matroid Intersection

Joakim Blikstad ✉
KTH Royal Institute of Technology, Sweden

Abstract

Despite a lot of recent progress in obtaining faster sequential matroid intersection algorithms, the fastest *parallel* poly(n)-query algorithm was still the straightforward $O(n)$-round parallel implementation of Edmonds' augmenting paths algorithm from the 1960s.

Very recently, Chakrabarty-Chen-Khanna [FOCS'21] showed the lower bound that any, possibly randomized, parallel matroid intersection algorithm making poly(n) rank-queries requires $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity. They ask, as an open question, if the lower bound can be improved to $\tilde{\Omega}(n)$, or if there can be sublinear-round, poly(n)-query algorithms for matroid intersection.

We resolve this open problem by presenting the first sublinear-round parallel matroid intersection algorithms. Perhaps surprisingly, we do not only break the $\tilde{O}(n)$-barrier in the rank-oracle model, but also in the weaker independence-oracle model. Our rank-query algorithm guarantees $O(n^{3/4})$ rounds of adaptivity, while the independence-query algorithm uses $O(n^{7/8})$ rounds of adaptivity, both making a total of poly(n) queries.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Parallel computing models; Theory of computation → Approximation algorithms analysis; Mathematics of computing → Matroids and greedoids

Keywords and phrases Matroid Intersection, Combinatorial Optimization, Parallel Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.25

Category Track A: Algorithms, Complexity and Games

Funding This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 71567. The author is also supported by the Swedish Research Council (Reg. No. 2019-05622).

Acknowledgements I thank Danupon Nanongkai and Sagnik Mukhopadhyay for insightful discussions and their valuable comments throughout the development of this work. Part of this work was done while the author visited BARC and the University of Copenhagen.

1 Introduction

Matroid intersection. Given two matroids[1] $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same n-element ground set V (but with different notions of independence $\mathcal{I}_1, \mathcal{I}_2$), the *matroid intersection problem* is to find the largest common independent set $S^* \subseteq \mathcal{I}_1 \cap \mathcal{I}_2$. This is a fundamental discrete optimization problem that has been studied for over half a century. Matroid intersection can be used to model many important combinatorial optimization problems, such as bipartite matching, finding arborescences, spanning tree packing, etc. As such, matroid intersection is a natural avenue to study all these problems simultaneously.

Oracle access. There are two standard ways to access the matroids – *independence oracles* and *rank oracles* – and we study both in this work. In an *independence-query* we may ask if $S \subseteq V$ is independent in one of the matroids, i.e. a query of the form "Is $S \in \mathcal{I}_1$?" or "Is $S \in \mathcal{I}_2$?" In a *rank-query* we instead ask for the *rank* of $S \subseteq V$ in one of the matroids. The

[1] Matroids are a well-studied combinatorial structure which can be though of as a generalization of the notion of linear independence in vector spaces. For a formally definition, see Definition 4.

rank $\mathrm{rk}_1(S)$ with respect to the matroid $\mathcal{M}_1$ (similarly rk_2 for $\mathcal{M}_2$) is the size of the largest (or, equivalently, any maximal) independent set, w.r.t. $\mathcal{I}_1$, contained in S. Note that the rank oracle is strictly more powerful than the independence oracle, since $S \in \mathcal{I}_1$ if and only if $\mathrm{rk}_1(S) = |S|$.

Parallel matroid intersection. A parallel matroid intersection algorithm accesses the oracle in rounds. In each round, a number of queries – that may only depend on the answers to queries made in previous rounds – can be issued in parallel. There is certainly a trade-off between (1) *adaptivity*, usually measured by the number of rounds, and (2) the total number of queries. When constructing parallel algorithms the goal is often to have as few rounds of adaptivity as possible while making only polynomially many queries in total.

Previous work. Edmonds [10] showed the first polynomial algorithm for matroid intersection in the 1960s, using $O(n^3)$ independence-queries, and there has been a long line of research since then e.g. [1, 2, 3, 5, 6, 8, 9, 10, 18, 19, 21]. Many of these are based on Edmonds' framework of finding *augmenting paths* in the *exchange graph*. In the sequential setting, only recently was the quadratic $O(n^2)$-query-barrier broken, first for rank-queries by Chakrabarty-Lee-Sidford-Singla-Wong in FOCS 2019 [5] and subsequently also for independence-queries by Blikstad-v.d.Brand-Mukhopadhyay-Nanongkai in STOC 2021 [3]. The current state-of-the-art in the sequential setting are the[2] $\tilde{O}(n\sqrt{n})$ rank-query algorithm by [5] and the $\tilde{O}(n^{7/4})$ independence-query algorithm by [2].

When it comes to the *parallel* setting, there is a straightforward $O(n)$-round, $\mathrm{poly}(n)$ independence-query implementation of Edmonds' algorithm: find the (up to $O(n)$ many) augmenting paths one-by-one. Each augmenting path can be found in a single round by querying all the potential edges in the exchange graph. In some special cases of matroid intersection we can do much better: a sequence of work has shown that both bipartite matching [11, 16, 20] and subsequently linear matroid intersection [13, 20] are in RNC[3] and quasi-NC.[4]

Another line of relevant work is showing that, in the parallel setting, the *search*-problem (finding a largest common independent set S) and the *decision*-problem (just finding the size of the answer) are "equivalent" (with only $O(\mathrm{polylog}(n))$ overhead). This is not at all obvious in the parallel setting, however, a recent work from SODA 2022 by Ghosh-Gurjar-Ray [12] shows that this is indeed the case for *weighted* matroid intersection, with rank-oracle access.

In FOCS 1985, Karp-Upfal-Wigderson [15] showed that any *independence-query* algorithm, possibly randomized, that finds a maximum independent set (basis) in a *single* matroid must use $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity if it makes $\mathrm{poly}(n)$ queries. They also show algorithms to find a basis of a (single) matroid in $O(\sqrt{n})$ rounds of independence-queries or a single round of the more powerful rank-queries. Arguably, this polynomial gap between the independence-query ($\tilde{\Omega}(n^{1/3})$ rounds) and rank-query ($O(1)$ rounds) for the seemingly easy problem to find a basis of a matroid illustrates that the independence-query is much weaker than the rank-query when used in parallel algorithms.

Nevertheless, a recent result from FOCS 2021 by Chakrabarty-Chen-Khanna [4] shows that even *rank-query* algorithms require a polynomial number of rounds to solve matroid intersection. In particular, they show a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds of adaptivity for any, possibly randomized, $\mathrm{poly}(n)$ rank-query matroid intersection algorithm.

---

[2] We use the usual convention of hiding $\mathrm{polylog}(n)$-factors with $\tilde{O}$ and $\tilde{\Omega}$ throughout the paper.
[3] Randomized $\mathrm{polylog}(n)$ rounds of adaptivity with $\mathrm{poly}(n)$ total work.
[4] Deterministic $\mathrm{polylog}(n)$ rounds of adaptivity with $n^{O(\log n)}$ total work.

Despite efficient algorithms for some special cases of matroid intersection, the trivial $O(n)$-round algorithm has remained unbeaten in the general case. The major open question (asked, for example, by [4]) is then whether it is possible to beat the $O(n)$-round barrier, or if matroid intersection is inherently very sequential and requires $\tilde{\Omega}(n)$ rounds of adaptivity.

Our results. We answer the above question by showing the first sublinear-round parallel matroid intersection algorithms, both in the rank-oracle and independence-oracle models. In particular, we obtain the following theorem.

▶ **Theorem 1** (Sublinear-round Matroid Intersection). *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(n^{3/4})$ rounds of polynomially many rank-queries, or*
- *$O(n^{7/8})$ rounds of polynomially many independence-queries.*

Our results, together with the lower bounds of [4, 15], imply that the true adaptivity of matroid intersection is somewhere between $n^{1/3}$ and $n^{3/4}$ (or $n^{7/8}$ for independence queries).

▶ Remark 2. Although we focus on the query-complexity in this paper, we note that the rounds and work in our algorithms are dominated, up to log-factors, by the oracle queries.

1.1 Technical Overview

The exchange graph and augmenting paths. Like many matroid intersection algorithms, we work in Edmonds' framework of finding *augmenting paths* in the *exchange graph*. The *exchange graph* $G(S)$ with respect to a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a directed bipartite graph, where finding a shortest (s, t)-path – called an *augmenting path* – means that we can increase the size of S by one. In a single round of $O(n^2)$ independence (or rank) queries, we can learn the entire exchange graph, and can thus find an augmenting path if one exists. This immediately gives a straightforward $O(n)$-round algorithm: *find the (up to $O(n)$ many) augmenting paths one-by-one.*

The exchange graph depends on the current common independent set S, and changes after each augmentation. In fact, if we have two disjoint augmenting paths p_1 and p_2 in $G(S)$, it is **not** necessarily the case that we can augment along both of these: augmenting along p_1 might destroy the path p_2 even if they were disjoint.[5] This forms the main difficulty in trying to beat the $O(n)$-round barrier, and illustrates the need in finding several "compatible" augmenting paths which can all be augmented along simultaneously.

Blocking flow. Cunningham [6] was the first to introduce *blocking flow* algorithms to matroid intersection, similar to Hopcroft-Karp's [14] bipartite matching or Dinitz's [7] max-flow algorithms. The idea is to run in phases, where after each phase the length of a shortest augmenting path in the exchange graph has increased. This is done by finding a maximal collection of compatible shortest augmenting paths. Both of the current state-of-the-art sequential $O(n\sqrt{n})$-rank-query [5] and $O(n^{7/4})$-independence-query [2] algorithms are based on versions of these blocking flow ideas. The $O(n\sqrt{n})$-rank-query algorithm still finds the augmenting paths in a sequential way, so it does unfortunately not seem to parallelize well.

[5] This is unlike the case of augmenting path algorithms for bipartite matching or maximum flow, where one can indeed augment along disjoint paths simultaneously.

The $O(n^{7/4})$-independence-query algorithm, on the other hand, is based on a recent notion of *augmenting sets* introduced by Chakrabarty-Lee-Sidford-Singla-Wong [5]. This notion of *augmenting sets* precisely captures what a collection of "compatible" shortest augmenting paths looks like. The authors of [5] also present an algorithm to find such augmenting sets, using independence-queries.

Our contribution is to show that a modified version of the augmenting sets algorithm of [5, Section 6] (which was later improved by [2]) can be implemented in parallel when combined with the parallel matroid-basis finding algorithms of Karp-Upfal-Wigderson [15]. Previous to this work, augmenting sets algorithms have before only been used in the sequential setting, and only in the independence-oracle model. Nevertheless, augmenting sets are what allows us to break the $O(n)$-round barrier also with rank-queries.

2 Preliminaries

We use the standard definitions of *matroid* $\mathcal{M} = (V, \mathcal{I})$; *rank* $\mathrm{rk}(X)$ for any $X \subseteq V$; *exchange graph* $G(S)$ for a *common independent set* $S \in \mathcal{I}_1 \cap \mathcal{I}_2$; and *augmenting paths* in $G(S)$ throughout this paper. For completeness, we define them below. We also need the notions of *augmenting sets* introduced by [5], which we also define in later this section.

▶ **Definition 3** (Set notation)**.** We will use $A + x$ and $A - x$ to denote $A \cup \{x\}$ respectively $A \setminus \{x\}$, as is usual in matroid intersection literature. We will also use $A + B := A \cup B$, and $A - B := A \setminus B$.

Matroids

▶ **Definition 4** (Matroid)**.** A *matroid* is a tuple $\mathcal{M} = (V, \mathcal{I})$ of a *ground set* V of n elements, and non-empty family $\mathcal{I} \subseteq 2^V$ of *independent sets* satisfying:
Downward closure: if $S \in \mathcal{I}$, then $S' \in \mathcal{I}$ for all $S' \subseteq S$.
Exchange property: if $S, S' \in \mathcal{I}$, $|S| > |S'|$, then there exists $x \in S \setminus S'$ such that $S' + x \in \mathcal{I}$.

▶ **Definition 5** (Matroid rank)**.** The *rank* of $A \subseteq V$, denoted by $\mathrm{rk}(A)$, is the size of the largest (or, equivalently, any maximal) independent set contained in A. It is well-known that the rank-function is submodular, i.e. $\mathrm{rk}(A + x) - \mathrm{rk}(A) \geq \mathrm{rk}(B + x) - \mathrm{rk}(B)$ whenever $A \subseteq B \subseteq V$ and $x \in V \setminus B$. Note that $\mathrm{rk}(A) = |A|$ if and only if $A \subseteq \mathcal{I}$.

▶ **Definition 6** (Matroid Intersection)**.** Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set V, a *common independent set* S is a set in $\mathcal{I}_1 \cap \mathcal{I}_2$. The *matroid intersection problem* asks us to find the largest common independent set. We use rk_1 and rk_2 to denote the rank functions of the corresponding matroids, and $n = |V|$ to be the size of the ground set.

The Exchange Graph

▶ **Definition 7** (Exchange graph)**.** Given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ over the same ground set, and a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the *exchange graph* $G(S)$ is a directed bipartite graph on vertex set $V \cup \{s, t\}$ with the following arcs (or directed edges):
1. (s, b) for $b \in V \setminus S$ when $S + b \in \mathcal{I}_1$.
2. (b, t) for $b \in V \setminus S$ when $S + b \in \mathcal{I}_2$.
3. (a, b) for $b \in V \setminus S, a \in S$ when $S - a + b \in \mathcal{I}_1$.
4. (b, a) for $b \in V \setminus S, a \in S$ when $S - a + b \in \mathcal{I}_2$.

We will denote the set of elements at distance k from s by the distance-layer D_k. Note that $D_k \subseteq V \setminus S$ when k is odd and $D_k \subseteq S$ when k is even.

▶ **Definition 8** (Shortest augmenting path)**.** A shortest (s,t)-path $p = (s, b_1, a_2, b_3, a_4, \ldots, a_{\ell-1}, b_\ell, t)$ (with $b_i \in V \setminus S$ and $a_i \in S$) in $G(S)$ is called a *shortest augmenting path*. We can *augment* S along the path p to obtain $S' = S + b_1 - a_2 + b_3 - a_4 \ldots + b_\ell$, which is well-known to also be a common independent set (with $|S'| = |S| + 1$). Conversely, there must exist a shortest augmenting path whenever S is not a largest common independent set.

Augmenting Sets

Augmenting Sets is a notion capturing a "blocking flow" in the exchange graph, and was introduced by [5], and also subsequently used in the algorithms of [2,3]. In order to efficiently find "good" augmenting sets, the algorithm works with a relaxed form of them instead, called *partial* augmenting sets. The following definitions and key properties of (partial) augmenting sets are copied from [5] where one can find the corresponding proofs.

▶ **Definition 9** (Augmenting Sets, from [5, Definition 24])**.** Let $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and $G(S)$ be the corresponding exchange graph with shortest (s,t)-path of length $\ell + 1$ (ℓ must be odd) and distance layers $D_1, D_2, \ldots, D_\ell$. A collection of sets $\Pi_\ell := (B_1, A_2, B_3, A_4, \ldots, A_{\ell-1}, B_\ell)$[6] form an *augmenting set* in $G(S)$ if the following conditions are satisfied:

(a) $A_k \subseteq D_k$ for even k, and $B_k \subseteq D_k$ for odd k.
(b) $|B_1| = |A_2| = |B_3| = \cdots = |B_\ell|$
(c) $S + B_1 \in \mathcal{I}_1$
(d) $S + B_\ell \in \mathcal{I}_2$
(e) For all even $1 \leq k \leq \ell$, we have $S - A_k + B_{k+1} \in \mathcal{I}_1$
(f) For all odd $1 \leq k \leq \ell$, we have $S - A_{k+1} + B_k \in \mathcal{I}_2$

▶ **Definition 10** (Partial Augmenting Sets, from [5, Definition 37])**.** We say that $\Phi_\ell := (B_1, A_2, B_3, A_4, \ldots, A_{\ell-1}, B_\ell)$ forms a *partial augmenting set* if it satisfies the conditions (a), (c),[7] and (e) of an *augmenting set*, plus the following two relaxed conditions :

(b) $|B_1| \geq |A_2| \geq |B_3| \geq \cdots \geq |B_\ell|$.
(f) For all odd $1 \leq k \leq \ell$, we have $\mathrm{rk}_2(S - A_{k+1} + B_k) = \mathrm{rk}_2(S)$.

▶ **Theorem 11** (from [5, Theorem 25])**.** *Let $\Pi_\ell := (B_1, A_2, B_3, A_4, \cdots, A_{\ell-1}, B_\ell)$ be the an augmenting set in the exchange graph $G(S)$. Then the set $S' := S \oplus \Pi_\ell := S + B_1 - A_2 + B_3 - \cdots - A_{\ell-1} + B_\ell$ is a common independent set.*[8]

We also need the notion of *maximal* augmenting sets, which naturally correspond to a maximal ordered collection of shortest augmenting paths, where, after augmentation, the (s,t)-distance must have increased. Together with a lemma from [6] (Lemma 14), we can see, on a high-level, how to obtain $(1-\varepsilon)$-approximation algorithms: *find "blocking flows" (i.e. maximal augmenting sets) until the (s,t)-distance is $\Omega(1/\varepsilon)$.*

▶ **Definition 12** (Maximal Augmenting Sets, from [5, Definition 35])**.** Let $\Pi_\ell = (B_1, A_2, B_3, \cdots, A_{\ell-1}, B_\ell)$ and $\tilde{\Pi}_\ell = (\tilde{B}_1, \tilde{A}_2, \tilde{B}_3, \cdots, \tilde{A}_{\ell-1}, \tilde{B}_\ell)$ be two augmenting sets in $G(S)$. We say $\tilde{\Pi}_\ell$ *contains* Π_ℓ if $B_k \subseteq \tilde{B}_k$ and $A_k \subseteq \tilde{A}_k$, for all k. An augmenting set Π_ℓ is called *maximal* if there exists no other augmenting set $\tilde{\Pi}_\ell$ containing Π_ℓ.

[6] Our indexing of the sets differ a bit from [2,5].
[7] Note that we intentionally skip item (d), unlike [5] which includes it in the definition, however they do not always maintain this property in their algorithms.
[8] Note that $|S'| = |S| + |B_1|$. In particular, an augmenting set with $|B_1| = 1$ is exactly an augmenting path. [5] shows that augmenting sets correspond exactly to a sequence of consecutive shortest augmenting paths.

▶ **Lemma 13** (from [5, Theorem 36]). *An augmenting set Π_ℓ is maximal if and only if there is no augmenting path of length at most $\ell+1$ in $G(S \oplus \Pi_\ell)$.*

▶ **Lemma 14** (Cunningham [6]). *If the length of the shortest (s,t)-path in $G(S)$ is at least $2\ell+1$, then $|S| \geq (1-O(1/\ell))r$, where r is the size of the largest common independent set.*

3 Warm-up: Finding a *Maximal* Common Independent Set

Consider first the easier problem of finding a *maximal* (instead of *maximum*) common independent set: that is we want to find a set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ such that there is no $x \in V \setminus S$ for which $S + x \in \mathcal{I}_1 \cap \mathcal{I}_2$. It is well-known that a maximal common independent set is also a $\frac{1}{2}$-approximation for the matroid intersection problem, and indeed our algorithm for a general $(1-\varepsilon)$-approximation (Section 4) will use similar ideas as our algorithm to find a *maximal* common independent set in this section.

In the sequential setting there is a very easy $O(n)$-query greedy algorithm: *Start with $S = \emptyset$ and go through all elements $x \in V$ and add them to S if $S + x$ is independent in both matroids.* However, this greedy algorithm is inherently very sequential and does not seem to adapt well to the parallel setting. Instead, we must somehow try to find several xs "in parallel" which we can all add to S simultaneously without breaking independence.

3.1 One Matroid

Let us start even simpler, and consider how to find a *maximal* independent set[9] S in a single matroid $\mathcal{M} = (V, \mathcal{I})$. It turns out that in our final matroid intersection algorithm we will many times, as a subroutinue, need to do exactly this.

Karp-Upfal-Wigderson [15] provides some simple parallel algorithms (both for rank- and independence-oracle access), whose results we present in Lemma 15. We briefly sketch their algorithms below, more details and full proofs can be found in [15, 17].

Rank Oracle. The rank-query algorithm only needs a single round. Let $\{v_1, v_2, \ldots, v_n\} = V$ be the elements of the ground set, and let $V_i = \{v_1, v_2, \ldots, v_{i-1}, v_i\}$ (so that $V_0 = \emptyset$ and $V_n = V$). Now query $\mathrm{rk}(V_i)$ for all i, and return $S = \{v_i : \mathrm{rk}(V_i) > \mathrm{rk}(V_{i-1})\}$. Intuitively, we can imagine that we go through all elements v_i one-by-one and add them to S if and only if the rank goes up.

Independence Oracle. The independence-query algorithm will need $O(\sqrt{n})$ rounds of $O(n)$ queries per round. Partition the elements of V into $\sqrt{n}$ different groups of (almost) equal size $F_1, F_2, \ldots F_{\sqrt{n}}$. If any group is independent (say F_i), then we select it, and consider the contracted matroid $\mathcal{M}/F_i$. Note that this can only happen $\sqrt{n}$ times. On the other hand, if all F_i are dependent, then we will find one element per group (that is $\sqrt{n}$ in total) which we can safely discard: If $\{v_1, v_2, \ldots, v_k\} = F_i$ are the elements of F_i, we query all prefixes, i.e. "Is $\{v_1, v_2, \ldots, v_j\} \in \mathcal{I}$?" for all j, and discard the first element v_j for which the answer is "No".

[9] Such a set S is usually called a *basis* of the matroid, and due to the exchange-property all the maximal independent sets must have the same size.

▶ **Lemma 15** (Parallel basis algorithm, [15]). *There is a deterministic parallel algorithm which given a matroids $\mathcal{M} = (V, \mathcal{I})$ finds a **maximal** independent set $S \in \mathcal{I}$ using either*

- *$O(1)$ round of $O(n)$ many rank-queries, or*
- *$O(\sqrt{n})$ rounds of $O(n)$ many independence-queries.*[10]

▶ Remark 16. Note that if we have a set $X \in \mathcal{I}$ and $Y \subseteq V \setminus X$, we can with the same algorithms as above find a maximal $Y' \subseteq Y$ such that $X + Y' \in \mathcal{I}$, even though the above algorithms are only stated as if $X = \emptyset$ and $Y = V$. This is since we can consider the contracted and restricted matroid $\mathcal{M}' = (\mathcal{M} \,/\, X) \setminus (V \setminus Y)$; and an independence/rank-query on $\mathcal{M}'$ can be simulated with the corresponding query on $\mathcal{M}$.

3.2 Two Matroids

Now we return to our problem of finding a maximal common independent set S of two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$. Suppose we already have some common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$. We will try to add more elements to S until it becomes maximal.

Firstly, let us concentrate on the first matroid and pick a maximal set $B \subseteq V$ such that $S + B \in \mathcal{I}_1$ using Lemma 15. However, $S + B$ is not necessarily independent in the second matroid, so we would need to fix this: let $B' \subseteq B$ be a maximal subset such that $S + B' \in \mathcal{I}_2$, which we again can find using Lemma 15. Now we know $S + B' \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a common independent set, so we set $S \leftarrow S + B'$, and we have made some progress (unless $B' = \emptyset$ of course).

At this point we can make a crucial observation: *we can safely discard the elements $x \in B \setminus B'$, since now $S + x \notin \mathcal{I}_2$.* Hence, for each element in B we have either (i) added it to our common independent set or (ii) discarded it. As long as $|B|$ is relatively large (say $\approx \sqrt{n}$), we have made significant progress.

On the other hand, if $|B|$ is small, we may resort to a different strategy. By the exchange property of matroids, we know that any $A \subseteq V$ such that $S + A \in \mathcal{I}_1$ has size $|A| \leq |B|$. So we can add at most $|B|$ more elements to our common independent set S before it becomes maximal. We can thus simply find these remaining (up to $|B|$ many) elements one-by-one, using one round each.

We present this two-stage strategy below in Algorithm 1, which is parametrized by the cut-off threshold p for when to consider $|B|$ small. The optimal choice of p differs depending on the oracle access (independence or rank) we have.

Adaptivity. The first stage of Algorithm 1 runs in $O(n/p \cdot \mathcal{T}_{\text{basis}})$ rounds if $\mathcal{T}_{\text{basis}}$ is the number of rounds needed to find a maximal independent set for a single matroid (Lemma 15 gives $\mathcal{T}^{\text{rank}}_{\text{basis}} = O(1)$ and $\mathcal{T}^{\text{indep}}_{\text{basis}} = O(\sqrt{n})$). This is since the size of F will decrease by $|B| \geq p$ each time the while-loop is run, which can happen at most n/p times. The second stage of Algorithm 1 runs in $O(p)$ rounds, both for independence and rank-oracle. Picking p optimally gives: $O(\sqrt{n})$ rounds of rank-queries (with $p = \sqrt{n}$); or $O(n^{3/4})$ rounds of independence-queries (with $p = n^{3/4}$). This proves Theorem 17, stated below.

▶ **Theorem 17.** *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a **maximal** common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(\sqrt{n})$ rounds of polynomially many rank-queries, or*
- *$O(n^{3/4})$ rounds of polynomially many independence-queries.*

[10] KUW [15] also provides a lower bound of $\tilde{\Omega}(n^{1/3})$ rounds for any independence-query algorithm which uses only polynomial number of queries per round, even if randomization is allowed. It remains an open problem to close this gap between $\tilde{\Omega}(n^{1/3})$ and $O(\sqrt{n})$.

Algorithm 1 Maximal Common Independent Set.

Let $S = \emptyset$ and $F = V$.
while true **do** ▷ Stage 1
  Find a maximal $B \subseteq F$ such that $S + B \in \mathcal{I}_1$ (using Lemma 15).
  if $|B| \leq p$ **then break**
  Find a maximal $B' \subseteq B$ such that $S + B' \in \mathcal{I}_2$ (using Lemma 15).
  Update $S \leftarrow S + B'$ and $F \leftarrow F - B$.
while true **do** ▷ Stage 2
  Query "Is $S + x \in \mathcal{I}_1$" and "Is $S + x \in \mathcal{I}_2$?" for all $x \in F$ in parallel.
  Pick an arbitrary $x \in F$ such that $S + x \in \mathcal{I}_1$ and $S + x \in \mathcal{I}_2$.
  if no such x exists **then break**
  Update $S \leftarrow S + x$ and $F \leftarrow F - x$.

4 Finding a *Maximum* Common Independent Set

In this section we present our sublinear-round matroid intersection algorithm.

The algorithm consists of two steps: first it finds an $(1-\varepsilon)$-approximation, and then it finds the remaining εn (which is sublinear if $1/\varepsilon$ is polynomially large in n) augmenting paths one-by-one. Each such remaining augmenting path can be found in a single round of n^2 independence (or rank) queries: in parallel query each possible edge of the exchange graph, and then see if there was an augmenting path. Indeed, when we know all the edges of the exchange graph, we do not need any more (rounds of) queries to figure out if there was an path.[11] If we skipped the $(1-\varepsilon)$-approximation step and just found all the augmenting paths one-by-one we obtain the straightforward $O(n)$-round algorithm.

The difficult part of the algorithm is how to find the $(1-\varepsilon)$-approximation in sublinear number of rounds (even when $1/\varepsilon$ is polynomially large). To do this, we would need to find many augmenting paths simultaneously, and indeed this is our strategy. Our main result of this section is this approximation algorithm which we summarize in Theorem 18 below.

▶ **Theorem 18** (Sublinear-round $(1-\varepsilon)$-approximation). *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ of size $|S| \geq (1-\varepsilon)r$, where r is the size of the largest common independent set, using either*

- *$O(\sqrt{n}/\varepsilon)$ rounds of polynomially many rank-queries, or*
- *$O(n^{3/4}/\varepsilon)$ rounds of polynomially many independence-queries.*

Exact algorithm. By an appropriate choice of ε ($\varepsilon = n^{-1/4}$ for rank-oracle and $\varepsilon = n^{-1/8}$ for independence-oracle), together with our discussion above, the main result (Theorem 1, restated below) of the paper – the sublinear-round exact algorithm – follows immediately from Theorem 18. The remainder of this paper will go towards proving Theorem 18, i.e. the $(1-\varepsilon)$-approximation algorithm.

▶ **Theorem 1** (Sublinear-round Matroid Intersection). *There is a deterministic parallel algorithm which given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$ on the same ground set V, finds a largest common independent set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ using either*

- *$O(n^{3/4})$ rounds of polynomially many rank-queries, or*
- *$O(n^{7/8})$ rounds of polynomially many independence-queries.*

[11] Note that if we also care about the number of rounds of work of the algorithm (and not just the rounds of *queries*), we can find the augmenting path in the exchange graph in just poly(n) rounds, as s,t-reachability is well-known to be in NC.

4.1 Blocking Flow

The approximation algorithm maintains a common independent set S and runs in $O(1/\varepsilon)$ phases, where in the i'th phase it eliminates all augmenting paths of length $2i$ by finding a blocking flow, similar to the Hopcroft-Karp's [14] bipartite matching algorithm and Dinitz's [7] max-flow algorithm. By *blocking flow* we mean a set of compatible shortest augmenting paths after which augmenting along them the (s,t)-distance in the exchange graph has increased. At the end, by Lemma 14, we will have found a common independent set S which is a $(1-\varepsilon)$-approximation, since the shortest augmenting path will have length $O(1/\varepsilon)$.

This idea of applying blocking flow algorithms to matroid intersection originates from the algorithm of Cunningham [6], but has since been improved by Chakrabarty-Lee-Sidford-Singla-Wong [5] and subsequently Blikstad [2], and it is the framework of these two later algorithms which we will follow.

▶ Remark 19. In the first phase we will eliminate all augmenting paths of length 2. This corresponds exactly to finding a *maximal* common independent set, like we did in Section 3. In general, we will show that we can implement any phase in the same round-complexity as the first phase, using similar ideas.

Beginning of a phase. In each phase, we consider a layered graph, where we let the *distance-layer* D_i denotes all the elements of distance i from the source node s in the exchange graph $G(S)$. At the beginning of a phase, the algorithm will use a single round (of $O(n^2)$ queries) to find these distance layers: simply query all potential edges of the exchange graph.

Unfortunately, knowing all the edges in the exchange graph is not sufficient to find a blocking flow, since a set of disjoint augmenting paths might not be compatible with each other. The exchange graph $G(S)$ does not capture the full structure of the matroid intersection problem, and this is where the difficulty in obtaining sublinear-round matroid intersection algorithms comes from. There is a need to be able to find many compatible augmenting paths "in parallel".

Augmenting sets. The notion of a collection of *compatible*[12] augmenting paths is captured by *augmenting sets*, as defined in Definition 9. So our goal in a phase is to find a *maximal* augmenting set (see Definition 12), which is what we formally mean by "blocking flow". After augmenting along a maximal augmenting set, Lemma 13 implies that the (s,t)-distance has increased, and we can move on to the next phase.

Our algorithm will follow the framework of [5] and [2], which are the state-of-the-art sequential independence-query approximation algorithms. The overall idea can be seen as a generalization of the warm-up *maximal* common independent set algorithm from Section 3. Instead of working with just a single distance layer in the exchange graph we now have up to $O(1/\varepsilon)$ many layers. Fortunately, layers far apart from each other can be handled relatively well in parallel, and we will see that the final adaptivity of our algorithm to find a blocking flow in a phase will not depend on the number of layers.

We start by, on a high level, summarizing how the algorithm of [2, 5] implements a phase in two stages. Our main result is how we can implement this algorithm efficiently in a parallel.

[12] That is they can all be augmented along simultaneously without breaking independence in either matroid.

1. The first stage keeps track of a *partial* augmenting set (Definition 10) which it keeps *refining* by a series of operations on adjacent distance layers in the exchange graph, to make it closer to a *maximal* augmenting set.
2. When we are "close enough" to a maximal augmenting set, the progress we make in the first stage slows down. Then we fall back to the second stage in which we find the relatively few remaining augmenting paths individually one at a time.

4.2 First Stage: Refining

The basic refining ideas and procedures in this section are the same as in [2,5]; our contribution is to show how they can be implemented in a parallel fashion.

Say we are in the phase where the (s,t)-distance is $\ell+1$, that is we have ℓ layers in our exchange graph. The algorithm keeps track of a partial augmenting set $\Phi_\ell = (B_1, A_2, B_3, \ldots, A_{\ell-1}, B_\ell)$ (see Definition 10), which it makes local improvements to, called *refining*. Essentially Φ_ℓ looks like a stair-case: B_{k+1} is a set which can be "matched" to some subset of the previous layer A_k; and similarly A_{i+1} can be "matched" to a subset of B_i. As long as Φ_ℓ is "far" from being a maximal augmenting set, the refinement procedures make significant progress. When Φ_ℓ becomes "close" to being a maximal augmenting set we move on to the second stage.

We maintain three types of elements in each layer D_k in the exchange graph.
Selected. Sets A_k and B_k form the partial augmenting set Φ_ℓ.
Removed. Sets R_k contain the discarded elements which we have deemed useless.
Fresh. Sets F_k contain the elements which are neither selected nor removed.

All elements are initially fresh, and for convenience we also define "imaginary" empty boundary layers $D_0 = D_{\ell+1} = \emptyset$, with corresponding sets $A_0, R_0, F_0, A_{\ell+1}, R_{\ell+1}, F_{\ell+1}$. Note that (A_k, R_k, F_k) forms a partition of $D_k \subseteq S$ when k is even, and that (B_k, R_k, F_k) partitions $D_k \subseteq V \setminus S$ when k is odd.

The idea of the refinement procedures is to make some local improvements to adjacent distance-layers. While doing this, we make sure that elements only change their types from *fresh* $\to$ *selected* $\to$ *removed*, but never in the other direction. In order to formalize that the removed elements are actually useless, we maintain the following *phase invariants*.

▶ **Definition 20** (Phase Invariants, from [5, Section 6.3.2])**.** The *phase invariants* are:
(a-b) $\Phi_\ell = (B_1, A_2, B_3, \ldots, A_{\ell-1}, B_\ell)$ forms a partial augmenting set.[13]
(c) For all even k, $\mathrm{rk}_1(W - R_k) = \mathrm{rk}_1(W) - |R_k|$ where $W = S - A_k + (D_{k+1} - R_{k+1})$.[14]
(d) For all odd k, $\mathrm{rk}_2(W + R_k) = \mathrm{rk}_2(W)$ where $W = S - (D_{k+1} - R_{k+1}) + B_k$.

▶ Remark 21. Invariant (c) and (d) essentially says that if R_{k+1} is useless, then so is R_k, for both even and odd layers, and thus, by induction, all removed elements are indeed useless. For example, (d) says that any element $x \in R_k$ does not increase the rank, even if we take away all non-useless elements $(D_{k+1} - R_{k+1})$ in the next layer. Hence such an x cannot be "matched" to any non-useless element in the next layer, so it is safe to discard it, since we will never be able to add it to B_k while maintaining that $(\ldots, B_k, A_{k+1}, \ldots)$ form an partial augmenting set.

[13] The naming of this invariant as (a-b) is to be consistent with [5] where this condition is split up into two separate items (a) and (b).
[14] This invariant differs from [5], where it was written in the following equivalent form: For $1 \le k \le \ell/2$, for any $X \subseteq B_{2k+1} + F_{2k+1} = D_{2k+1} - R_{2k+1}$, if $S - (A_{2k} + R_{2k}) + X \in \mathcal{I}_1$ then $S - A_{2k} + X \in \mathcal{I}_1$.

4.2.1 Refining Locally

We now present the basic refinement procedures from [5], which are operations on two neighboring layers. We note that [2] improves upon the algorithm of [5] (in the sequential setting) by considering refinement operations on **three** consecutive layers instead. Unfortunately, the three-layer refinement procedures of [2] does not seem to work efficiently in the parallel setting.

Intuitively, for an even k, `RefineAB`(k) tries to extend B_{k+1} as much as possible while it still can be "matched" from A_k in the previous layer (i.e. while $S - A_k + B_{k+1} \in \mathcal{I}_1$). After this, if $|A_k| > |B_{k+1}|$, we can remove elements from A_k and argue that they are useless (if they were useful, then it should have been possible to "match" them to something more in the next layer, but this is not the case since B_{k+1} could not be extended more). So `RefineAB`(k) extends B_{k+1} and shrinks A_k so that they are the same size. Doing so, $|A_k^{old}| - |B_{k+1}^{old}|$ elements have changed types, and this crucial observation is what allows us to measure progress. For an odd k, `RefineBA`(k) works very similarly, but now between the consecutive layers (B_k, A_{k+1}).

Algorithm 2 `RefineAB`(k) for even k. (called `Refine1` in [5, Algorithm 9])

1: Find a maximal $B \subseteq F_{k+1}$ s.t. $S - A_k + B_{k+1} + B \in \mathcal{I}_1$
2: $B_{k+1} \leftarrow B_{k+1} + B, \quad F_{k+1} \leftarrow F_{k+1} - B$
3: Find a maximal $A \subseteq A_k$ s.t. $S - A_k + B_{k+1} + A \in \mathcal{I}_1$
4: $A_k \leftarrow A_k - A, \quad R_k \leftarrow R_k + A$

Algorithm 3 `RefineBA`(k) for odd k. (called `Refine2` in [5, Algorithm 10])

1: Find a maximal $B \subseteq B_k$ s.t. $S - (D_{k+1} - R_{k+1}) + B \in \mathcal{I}_2$
2: $R_k \leftarrow R_k + B_k - B, \quad B_k \leftarrow B$
3: Find a maximal $A \subseteq F_{k+1}$ s.t. $S - (D_{k+1} - R_{k+1}) + B_k + A \in \mathcal{I}_2$
4: $A_{k+1} \leftarrow A_{k+1} + F_k - A, \quad F_k \leftarrow A$

▶ Remark 22. When we are in the first phase, that is when there is only a single layer between s and t in the graph, running `RefineAB`(0) and `RefineBA`(1) corresponds to our warm-up algorithm to find a *maximal* common independent set from Section 3. In particular `RefineAB`(0) finds a maximal B_1 such that $S + B_1 \in \mathcal{I}_1$, and `RefineAB`$(1)$ shrinks B_1 such that $S + B_1 \in \mathcal{I}_2$ too.

▶ **Lemma 23.** *`RefineAB` and `RefineBA` can each be implemented in either:*

- $O(1)$ *rounds of polynomially many rank-queries, or*
- $O(\sqrt{n})$ *rounds of polynomially many independence-queries.*

Proof. The refine procedures only need to find a maximal independent set (for a single matroid) twice, so we can apply Lemma 15. ◀

The following properties are proven in [5].

▶ **Lemma 24** (from [5, Lemmas 40-42])**.** *Both `RefineAB` and `RefineBA` preserve the phase invariants. Also: after `RefineAB`(k) is run, we have $|A_k| = |B_{k+1}|$ (unless $k = 0$). After `RefineBA`(k) is run, we have $|B_k| = |A_{k+1}|$ (unless $k = \ell$).*

▶ **Observation 25.** *Lemma 24 can be used to messure progress. In particular, after running* `RefineAB`(k)*,* $|A_k| = |B_{k+1}|$*, so a total of* $|A_k^{old}| - |B_{k+1}^{old}|$ *elements must have changed types (*$x \in A_k^{old}$ *might have been* removed*, while a* $x \in F_{k+1}^{old}$ *might have been* selected*). Similarly* `RefineBA`(k) *will change types of* $|B_k^{old}| - |A_{k+1}^{old}|$ *elements. Note that each element can only change type at most twice (from* fresh *to* selected *to* removed*), so this observation can be used to measure progress.*

4.2.2 Refining Globally

In the sequential algorithms of [2, 5], a refinement pass consists of running `RefineAB`(k) and `RefineAB`(k) for all k in sequence. However, in the parallel setting we can do better. Since `RefineAB` and `RefineBA` only change things locally in two adjacent layers, we observe that we can perform several of these refinement operations in parallel.

Algorithm 4 `Refine`().

1: In parallel, run `RefineAB`(k) for all even $0 \leq k \leq \ell$.
2: In parallel, run `RefineBA`(k) for all odd $0 \leq k \leq \ell$.

▶ **Lemma 26.** *`Refine` can be implemented in either:*
- $O(1)$ *rounds of polynomially many rank-queries, or*
- $O(\sqrt{n})$ *rounds of polynomially many independence-queries.*

The following Lemma 27 will be useful to bound the number of `Refine` calls needed in our final algorithm, and is similar to [5, Corollary 43].

▶ **Lemma 27.** *Suppose that* $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$ *and that* $S + B_\ell \in \mathcal{I}_2$*, before `Refine` is run. After `Refine` is run we have:*

(i) $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$*, still.*
(ii) $S + B_\ell \in \mathcal{I}_2$*, still.*
(iii) $|B_1'| - |B_\ell|$ *elements have changed their type, where* B_1' *is any maximal subset of* $(D_1 - R_1)$ *such that* $S + B_1' \in \mathcal{I}_1$*.*

Proof. Property (i) is true, since it is true just after we run `RefineBA`(k), by Lemma 27, and this is what is done in the last step of `Refine`. Similarly property (ii) is true, since `RefineBA`(ℓ) ensures this when "shrinking" B_ℓ (see how B is picked in Algorithm 3 line 1).

What remains is to prove property (iii). Let $(B_1^{old}, A_2^{old}, \ldots)$ be the sets before `Refine` is run. In the first line of Algorithm 4, we run `RefineAB`(2), `RefineAB`(4), `RefineAB`(6), …, which according to Lemma 24 and Observation 25, has incurred a total of $\sum |A_k^{old}| - |B_{k+1}^{old}| = |B_1^{old}| - |B_\ell^{old}|$ type-changes (the sum telescopes since we assume $|B_{k-1}^{old}| = |A_k^{old}|$).

Also note that we run `RefineAB`(0), which extends B_1 until it is a maximal subset of $D_1 \setminus R_1$ such that $S + B_1 \in \mathcal{I}_1$ (line 1 of Algorithm 2). This means that an additional $|B_1'| - |B_1^{old}|$ elements have changed their type – from *fresh* to *selected* – in the first layer, where B_1' is the value of B_1 we get after running `RefineAB`(0). Note that this B_1' is a maximal subset of $(D_1 - R_1)$ such that $S + B_1' \in \mathcal{I}_1$ (see line 1 of Algorithm 2).

Hence, in the first line of Algorithm 4, $|B_1'| - |B_\ell|$ types have changed. We might additionally change types of more elements when running the second line of Algorithm 4. ◀

▶ Remark 28. We measure progress in terms of Lemma 27. Since each element can only change types twice (from *fresh* ↦ *selected* ↦ *removed*), there will be in total $O(n)$ type-changes. If we just run `Refine`, we might need to do so $O(n)$ times (in the case when $|B_1'| - |B_\ell|$ is

constant), so this is not good enough to obtain a sublinear round parallel algorithm. Like we did in the easier case of *maximal* common independent set, we must swap to a different strategy when the progress of refining stagnates, i.e. when $|B'_1| - |B_\ell|$ is relatively small.

4.3 Second Stage: Finding the Remaining Augmenting Paths

When $|B'_1| - |B_{\ell+1}|$ is relatively small, we fall back to finding a special kind of augmenting paths one-by-one. We will show that we only need to find $|B'_1| - |B_{\ell+1}|$ many such paths before we get stuck and have found our *maximal* augmenting set (i.e. the desired blocking flow). These special kind of augmenting paths we consider are essentially augmenting paths in the exchange graph *with respect to our partial augmenting set*. They were first introduced by [2], and are called *valid paths*.

▶ **Definition 29** (Valid path, from [2, Definition 31]). $(b_i, a_{i+1}, b_{i+2}, \ldots, a_{\ell-1}, b_\ell, t)$ is a *valid path* (w.r.t. our partial augmenting set) *starting at* b_i if for all $k \geq i$:

(a) $a_k \in F_k$ for even k and $b_k \in F_k$ for odd k.
(b) $S - A_{i-1} + B_i + b_i \in \mathcal{I}_1$.
(c) $S + B_\ell + b_\ell \in \mathcal{I}_2$.
(d) $S - A_{k+1} + B_k - a_{k+1} + b_k \in \mathcal{I}_2$ for odd k.
(e) $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$ for even k.

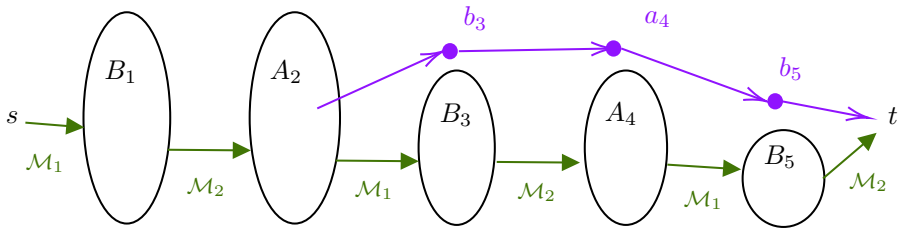


Figure 1 An example of a valid path (b_3, a_4, b_5, t) starting at b_3.

▶ Remark 30. Compare the definition of valid paths to the edges in the exchange graph from Definition 7. Essentially, items (b-e) corresponds to edges of the exchange graph $G(S + B_1 - A_2 + B_3 - A_4 + \cdots + B_\ell)$ of S *after* augmenting along our partial augmenting set. Note also that item (b) can only hold when $|A_{i-1}| > |B_i|$ (or, when $i = 1$ and $A_0 = \emptyset$ is an "imaginary" boundary set).

▶ **Lemma 31** (Augmenting along a valid path [2, Lemma 33]). *Suppose that* $|B_k| = |A_{k+1}|$ *for all odd* $1 \leq k \leq \ell - 1$ *and that* $S + B_\ell \in \mathcal{I}_2$.[15] *If* $(b_i, a_{i+1}, b_{i+2}, \ldots, b_\ell, t)$ *is a valid path starting at* b_i*, then* $(B_1, A_2, \ldots, B_{i-2}, A_{i-1}, B_i + b_i, A_{i+1} + a_{i+1}, \ldots, B_\ell + b_\ell)$ *is a partial augmenting set satisfying the phase invariants and with* $S + B_\ell + b_\ell \in \mathcal{I}_2$.

Proof sketch. It is easy to verify that all the properties in the definition of a partial augmenting set are still satisfied after the augmentation. Moreover, the phase invariants are also true, since the sets B_k and A_k are only extended by the augmentation (so an element deemed useless before remains useless). ◀

[15] These conditions are actually redundant, since they are covered by items (d) and (c) in the definition of the valid paths. However, they make the intuition slightly easier, and our algorithm maintains them.

▶ **Lemma 32.** *We can find a valid path, if one exists, in a single round of $O(n^2)$ queries.*

Proof. Finding a valid path in a single round of queries is not very different from finding a normal augmenting path. In a single round we query all potential "directed edges", that is all potential (a_k, b_{k+1}) or (b_k, a_{k+1}) pairs satisfying the items of Definition 29 (valid paths). Then we can combine these edges to form a valid path, or else determine that no valid path exist. ◀

▶ Remark 33. After augmenting along a valid path, $|B_\ell|$ increases by one. Let B_1' be any any maximal subset of $(D_1 - R_1)$ such that $S + B_1' \in \mathcal{I}_1$. Note that we always know that $|B_1'| \geq |B_1| \geq |B_\ell|$. Hence, we need to only find at most $|B_1'| - |B_\ell|$ many valid paths to augment along after we finished the first stage.

We also need the following lemma saying that if, for an element $x \in F_k$, there is no "partial" valid path $(x, \ldots, a_{\ell-1}, b_\ell, t)$ (satisfying all items of a valid path, except maybe item (b) of Definition 29), then it is safe to delete x. We prove this by showing that if x has no "out-edges" (of the form of items (c-e) in Definition 29), then it can be removed.

▶ **Lemma 34.** *Suppose that $|B_k| = |A_{k+1}|$ for all odd $1 \leq k \leq \ell - 1$ and that $S + B_\ell \in \mathcal{I}_2$. Then:*

- *If $b_\ell \in F_k$ is such that $S + B_\ell + b_\ell \notin \mathcal{I}_2$, we can safely remove it.*
- *For a given $b_k \in F_k$, if there exist no $a_{k+1} \in F_{k+1}$ such that $S - A_{k+1} + B_k - a_{k+1} + b_k \in \mathcal{I}_2$, then it is safe to remove b_k.*
- *For a given $a_k \in F_k$, if there exist no $b_{k+1} \in F_{k+1}$ such that $S - A_k + B_{k+1} - a_k + b_{k+1} \in \mathcal{I}_1$, then it is safe to remove a_k.*

Proof sketch. We must argue that the phase invariants are preserved when the elements are removed. It is straightforward to verify that phase invariants (c) and (d) hold in all these three cases. As a black-box intuition, we can imagine temporarily selecting the element x we want to remove, and then running either `RefineAB` or `RefineBA` and note that this procedure can immediately remove x again (and the refine-procedures preserves the invariants). ◀

4.4 Combining the Stages

Now we present the full algorithm of a phase, whose goal is to find a maximal augmenting set, that is a "blocking flow". Pseudo-code can be found in Algorithm 5, which is parametrized by a cut-off threshold p (which will be different for rank- and independence-query) for when to move from the first to second stage.

▶ Remark 35. After the two stages, we will have some partial augmenting set $(B_1, A_2, \ldots, B_\ell)$ such that there are no more valid paths. However, it is not yet an actual augmenting set, for instance it can be the case that $|A_k| > |B_{k+1}|$ for some k. Still, we can argue that $(B_1, A_2, \ldots, B_\ell)$ contains some *maximal augmenting set* $(\tilde{B}_1, \tilde{A}_2, \ldots, \tilde{B}_\ell)$ with $\tilde{B}_\ell = B_\ell$. So we will need a short extra clean-up step to reduce our partial augmenting set to such a maximal augmenting set.

Note that it is possible to show that we actually can directly augment along our partial augmenting set $(B_1, A_2, \ldots, B_\ell)$ which the algorithm finds (this relies on the extra properties that $|B_k| = |A_{k+1}|$ and $S + B_\ell \in \mathcal{I}_2$). That is $S' = S + B_1 - A_2 + B_2 - \cdots + B_\ell \in \mathcal{I}_1 \cap \mathcal{I}_2$ is a common independent set. Additionally $|S'| = |S| + |B_\ell|$, so we have increased the size of S as much as we would have if we found the the maximal augmenting set $(\tilde{B}_1, \tilde{A}_2, \ldots, \tilde{B}_\ell)$ instead. However, there is a critical problem with this approach: *there can be short augmenting paths in $G(S')$*. This means that such an approach will have failed to eliminate all (s,t)-paths of length $\leq \ell$. Hence the clean-up step is actually necessary.

Algorithm 5 Implementation of phase $(\ell+1)/2$.

1: In parallel query all potential edges of the exchange graph $G(S)$ (see Definition 7).
2: Find the distance layers $D_1, D_2, \dots D_\ell$.
3: Initialize $B_k = \emptyset$, $A_k = \emptyset$, $R_k = \emptyset$ and $F_k = D_k$ for all k.

4: **while** true **do** ▷ Stage 1
5: Find a maximal $B' \subseteq D_1 - R_1$ such that $S + B' \in \mathcal{I}_1$ (using Lemma 15).
6: **if** $|B'| - |B_\ell| \leq p$ **then break**
7: Call `Refine`() (Algorithm 4).

8: **while** true **do** ▷ Stage 2
9: In a single round, find a valid path if one exists (Lemma 32).
10: **if** no valid path exists **then break**
11: Augment the partial augmenting set along the found valid path (Lemma 31).

▷ Clean-up
12: Remove all elements which do not have any partial valid path from them (see Lemma 34).
13: Sequentially, call $\texttt{RefineBA}(\ell), \texttt{RefineAB}(\ell-1), \texttt{RefineBA}(\ell-2), \dots \texttt{RefineAB}(0)$
14: Augment along the *maximal augmenting set* $\Phi = (B_1, A_2, \dots, B_\ell)$:
15: that is, update $S \leftarrow S + B_1 - A_2 + B_3 + \dots + B_\ell$.

Correctness. In the beginning of Algorithm 5 the following hold: (i) the phase invariants (Definition 20); (ii) $|B_k| = |A_{k+1}|$ for all odd $1 \leq k \leq \ell - 1$; and (iii) $S + B_\ell \in \mathcal{I}_2$.

In the first stage, whenever we call `Refine`, the above properties (i-iii) are all preserved according to Lemma 27. Similarly, in the second stage, whenever we augment along a valid path, the above properties (i-iii) are also preserved, by Lemma 31.

What remains to be shown is that after the clean-up phase, $\Phi = (B_1, A_2, \dots, B_\ell)$ is a maximal augmenting set. We prove this by showing that these refine calls cannot *select* any new elements, that is they do not add any elements to the sets B_k or A_k. If we show this, then we know that $|B_\ell| = |A_{\ell-1}| = \dots = |B_1|$ after all these refine calls, as $\texttt{RefineAB}(\ell-1)$ reduced $|A_{\ell-1}|$ to match $|B_\ell|$; $\texttt{RefineBA}(\ell-2)$ reduces $|B_{\ell-2}|$ to match $|A_{\ell-1}|$; etc.

To argue that $\texttt{RefineAB}(k)$ does not select any new elements, we note that if it added b_k to B_k, it meant that $S - A_{k+1} + B_k + b_k \in \mathcal{I}_1$. However, since b_k was not removed in line 12 of the algorithm, there must have been a valid path starting from b_k, which is a contradiction. The argument for $\texttt{RefineBA}(k)$ is the same. Note that since we only remove elements, no new valid paths can occur.

Now, after $|B_1| = |A_2| = \dots = |B_\ell|$, $(B_1, A_2, \dots, B_\ell)$ forms a *maximal* augmenting set. If it did not, there must have been some path $(b_1, a_2, \dots, b_\ell)$ which we can add to it, but this is impossible, since this path would have been a valid path (starting at b_1).

Rounds of adaptivity. The first stage of Algorithm 5 runs in $O(n/p \cdot \mathcal{T}_{\texttt{Refine}})$ rounds if $\mathcal{T}_{\texttt{Refine}}$ is the number of rounds needed to run `Refine`() once (Lemma 26 gives $\mathcal{T}^{\text{rank}}_{\texttt{Refine}} = O(1)$ and $\mathcal{T}^{\text{indep}}_{\texttt{Refine}} = O(\sqrt{n})$). This is since each time we run refine we will have at least p type-changes (Lemma 27), and in total each of the n elements can change types at most twice.

The second stage of Algorithm 5 runs in $O(p)$ rounds, both for independence and rank-oracle, by Lemma 32. Picking p optimally gives: $O(\sqrt{n})$ rounds of rank-queries (with $p = \sqrt{n}$) or $O(n^{3/4})$ rounds of independence-queries (with $p = n^{3/4}$) for the first and second stages combined.

Finally the clean-up stage runs, sequentially, with $O(\ell)$ refinement operations. This takes $O(\ell)$ rounds of rank-queries or $O(\ell\sqrt{n})$ rounds of independence queries, so this depends on the number of layers. However, we argue that we can ignore this term for the interesting range of ℓ. This is because when ℓ is too large ($> \sqrt{n}$ for rank-queries and $> n^{1/4}$ for independence-queries), we know by Lemma 14 that there are only $O(1/\ell)$ many augmenting paths left in total, and we can instead find them one-by-one in at most $O(\sqrt{n})$ rounds for rank-queries or $O(n^{3/4})$ rounds for independence queries.

Concluding, we have argued that we can implement a blocking-flow phase in $O(\sqrt{n})$ rounds of rank-queries or $O(n^{3/4})$ rounds of independence-queries.

Approximation algorithm. Running Algorithm 5 for $O(1/\varepsilon)$ phases eliminates all paths in the exchange graph of length $O(1/\varepsilon)$ (Lemma 13), so by Lemma 14 we know that the common independent set S we end up with is a $(1-\varepsilon)$-approximation. The adaptivity is thus $O(\sqrt{n}/\varepsilon)$ rounds of rank-queries or $O(n^{3/4}/\varepsilon)$ rounds of independence-queries. Hence we have shown a $(1-\varepsilon)$-approximation algorithm using $O(\sqrt{n}/\varepsilon)$ rounds of (polynomially many) rank-queries or $O(n^{3/4}/\varepsilon)$ rounds of (polynomially many) independence-queries, which proves Theorem 18.

References

1 Martin Aigner and Thomas A. Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971.

2 Joakim Blikstad. Breaking o(nr) for matroid intersection. In *ICALP*, volume 198 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.31.

3 Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *STOC*, pages 421–432. ACM, 2021. doi:10.1145/3406325.3451092.

4 Deeparnab Chakrabarty, Yu Chen, and Sanjeev Khanna. A polynomial lower bound on the number of rounds for parallel submodular function minimization and matroid intersection. In *FOCS*, pages 37–48. IEEE Computer Society, 2021. doi:10.1109/FOCS52979.2021.00013.

5 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *FOCS*, pages 1146–1168. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00072.

6 William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM J. Comput.*, 15(4):948–957, 1986. doi:10.1137/0215066.

7 Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

8 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial structures and their applications*, pages 69–87. Gordon and Breach, 1970.

9 Jack Edmonds. Matroid intersection. In *Annals of discrete Mathematics*, volume 4, pages 39–49. Elsevier, 1979.

10 Jack Edmonds, GB Dantzig, AF Veinott, and M Jünger. Matroid partition. *50 Years of Integer Programming 1958–2008*, page 199, 1968.

11 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. *SIAM J. Comput.*, 50(3), 2021. doi:10.1137/16M1097870.

12 Sumanta Ghosh, Rohit Gurjar, and Roshan Raj. A deterministic parallel reduction from weighted matroid intersection search to decision. In *SODA*, pages 1013–1035. SIAM, 2022. doi:10.1137/1.9781611977073.44.

13 Rohit Gurjar and Thomas Thierauf. Linear matroid intersection is in quasi-nc. *Comput. Complex.*, 29(2):9, 2020. doi:10.1007/s00037-020-00200-z.

14 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

15 Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel computation on matroids. In *FOCS*, pages 541–550. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.57.

16 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Comb.*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

17 Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel search. *J. Comput. Syst. Sci.*, 36(2):225–253, 1988. doi:10.1016/0022-0000(88)90027-X.

18 Eugene L. Lawler. Matroid intersection algorithms. *Math. Program.*, 9(1):31–56, 1975. doi:10.1007/BF01681329.

19 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *FOCS*, pages 1049–1065. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.68.

20 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574. Akademie-Verlag, Berlin, 1979.

21 Huy L. Nguyen. A note on cunningham's algorithm for matroid intersection. *CoRR*, abs/1904.04129, 2019. arXiv:1904.04129.

Privately Estimating Graph Parameters in Sublinear Time

Jeremiah Blocki
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Elena Grigorescu
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Tamalika Mukherjee
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Abstract

We initiate a systematic study of algorithms that are both differentially-private and run in sublinear time for several problems in which the goal is to estimate natural graph parameters. Our main result is a differentially-private $(1+\rho)$-approximation algorithm for the problem of computing the average degree of a graph, for every $\rho > 0$. The running time of the algorithm is roughly the same (for sparse graphs) as its non-private version proposed by Goldreich and Ron (Sublinear Algorithms, 2005). We also obtain the first differentially-private sublinear-time approximation algorithms for the maximum matching size and the minimum vertex cover size of a graph.

An overarching technique we employ is the notion of *coupled global sensitivity* of randomized algorithms. Related variants of this notion of sensitivity have been used in the literature in ad-hoc ways. Here we formalize the notion and develop it as a unifying framework for privacy analysis of randomized approximation algorithms.

2012 ACM Subject Classification Security and privacy → Privacy-preserving protocols; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases differential privacy, sublinear time, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.26

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2202.05776`

Funding *Jeremiah Blocki*: supported in part by NSF CNS-1931443 and NSF CCF-1910659.
Elena Grigorescu: supported in part by NSF CCF-1910659 and NSF CCF-1910411.
Tamalika Mukherjee: supported in part by NSF CCF-1910659 and NSF CCF-1910411.

Acknowledgements We thank several anonymous reviewers for their valuable feedback on a preliminary version of this work. We thank Soheil Behnezhad for bringing to our attention the subtlety in the analysis of [26], first discovered by [10], and finally resolved in his recent work [3]. We also thank Jakub Tetek for bringing up several points for clarification, which we have incorporated in the current version.

1 Introduction

Graphs are frequently used to model massive data sets (e.g., social networks) where the users are the nodes, and their relationships are the edges of the graphs. These relationships often consist of sensitive information, which drives the need for privacy in this setting.

Differential Privacy (DP) [12] has become the gold standard in privacy-preserving data analysis due to its compelling privacy guarantees and mathematically rigorous definition. Informally, a randomized function computed on a graph is *differentially private* if the distribution of the function's output does not change significantly with the presence or

absence of an individual edge (or node). See [13] for a comprehensive tutorial on differential privacy.

▶ **Definition 1** (Differential-privacy). *Let $\mathcal{G}_n$ denote the set of all n-node graphs. An algorithm $\mathcal{A}$ is (ε, δ) node-DP (resp. edge-DP) if for every pair of node-neighboring (resp. edge-neighboring)*[1] *graphs $G_1, G_2 \in \mathcal{G}_n$, and for all sets $\mathcal{S}$ of possible outputs, we have that $\Pr[\mathcal{A}(G_1) \in \mathcal{S}] \leqslant e^{\varepsilon} \Pr[\mathcal{A}(G_2) \in \mathcal{S}] + \delta$. When $\delta = 0$ we simply say that the algorithm is ε-DP.*

Since the graphs appearing in modern applications are massive, it is also often desirable to design *sublinear-time* algorithms that approximate natural combinatorial properties of the graph, such as the average degree, the number of connected components, the cost of a minimum spanning tree, the number of triangles, the size of a maximum matching, the size of a minimum vertex cover, etc. For an excellent survey on sublinear-time algorithms for approximating graph parameters, we refer the reader to [29].

There has been a lot of work in developing differentially-private algorithms for estimating graph parameters in polynomial-time, with respect to *edge differential privacy*, i.e., neighboring graphs that differ by a single edge in Definition 1. Nissim, Raskhodnikova, and Smith [25] demonstrated the first edge-differentially private graph algorithms. They showed how to estimate the cost of a minimum spanning tree and the number of triangles in a graph by calibrating noise to a local variant of sensitivity called *smooth sensitivity*. Subsequent works in designing edge differentially-private algorithms for computing graph statistics include [21, 19, 23, 36]. Gupta, Ligett, McSherry, Roth and Talwar [18] gave the first edge differentially-private algorithms for classical graph optimization problems, such as vertex cover, and minimum s-t cut, by making clever use of the exponential mechanism in existing non-private algorithms that solve the same problem.

An even more desirable notion of privacy in graphs is the notion of *node differential privacy* i.e., neighboring graphs that differ by a single node and edges incident to it in Definition 1. The concept of node differentially-private algorithms for 1-dimensional functions (functions that output a single real value) on graphs was first rigorously studied independently by Kasiviswanathan, Nissim, Raskhodnikova and Smith [22], as well as, Blocki, Blum, Datta, and Sheffet [4], and Chen and Zhou [9]. Their techniques were later extended to higher-dimensional functions on graphs [28, 6]. Subsequent works have focused on developing node differentially-private algorithms for a family of network models: stochastic block models and graphons [7, 30]. A more recent line of work has focused on the continual release of graph statistics such as degree-distributions and subgraph counts in an online setting [33, 15]. Gehrke, Lui, and Pass [16] introduce a more robust notion of differential privacy called Zero-Knowledge Differential Privacy (ZKDP), which tackles the problem of auxiliary information in social networks. This work uses existing results from sublinear-time algorithms as a building block to achieve ZKDP for several graph problems. However, it is important to note that the final ZKDP mechanisms are not computable in sublinear-time.

The literature on designing differentially-private algorithms for estimating graph parameters in sublinear time is far less developed. The only paper we are aware of is due to Sivasubramaniam, Li and He [32], who give the first sublinear-time differentially-private algorithm for approximating the average degree of a graph. Our work addresses this gap

[1] Graphs $G_1 = (V, E_1)$, $G_2 = (V, E_2)$ are *node-neighboring*, denoted by $G_1 \sim_v G_2$, if there exists a vertex $v \in V$ such that $E_1(V \setminus \{v\}) = E_2(V \setminus \{v\})$. Graphs G_1 and G_2 are *edge-neighboring* i.e., $G_1 \sim_e G_2$ if there exists an edge e such that $E_1 \setminus \{e\} = E_2 \setminus \{e\}$.

by initiating a systematic study of differentially-private sublinear-time algorithms for the problems of estimating the following graph parameters: (1) the average-degree of a graph, (2) the size of a maximum matching, and (3) the size of a minimum vertex cover. As an overarching technique, we formally introduce the notion of *Coupled Global Sensitivity* and use it to analyze the privacy of our randomized approximation algorithms.

1.1 Our Results

1.1.1 Privately Approximating the Average Degree

We obtain a differentially-private sublinear-time algorithm for estimating the average degree $\bar{d}_G = \frac{\sum_{v \in V} \deg(v)}{|V|}$, of a graph $G = (V, E)$, with respect to edge-differential privacy, which achieves a multiplicative approximation of $(1+\rho)$, for any constant $\rho > 0$. Specifically, our algorithm outputs a value $\tilde{d}$ such that w.h.p. we have $(1-\rho)\bar{d}_G \leqslant \tilde{d} \leqslant (1+\rho)\bar{d}_G$, for graphs with $\bar{d}_G = \Omega(1)$. Throughout the paper we denote $|V| = n$.

We work in the *neighbor-query* model, in which we are given oracle access to a simple graph $G = (V, E)$, where the algorithm can obtain the identity of the i-th neighbor of a vertex $v \in V$ in constant time. If $i > \deg(v)$ for a particular vertex v, then $\perp$ is returned. The algorithm may also perform *degree queries*, namely for any $v \in V$ it can obtain $\deg(v)$ in constant time.

▶ **Theorem 2.** *There is an ε-edge differentially-private $(1+\rho)$-approximation algorithm for estimating the average degree $\bar{d}_G \geqslant 1$ of a graph G on n vertices that runs in time*[2] *$O(\sqrt{n} \cdot poly(\log(n)/\rho) \cdot poly(1/\varepsilon))$ where $\varepsilon^{-1} = o(\log^{1/4}(n))$.*

The problem of estimating the average degree of a graph was first studied by Feige [14], who gave a sublinear time $(2+\rho)$-approximation (multiplicative) for any constant $\rho > 0$, making $O(\sqrt{n/d_0})$ many degree queries, where d_0 is a lower bound on the number of queries. He also notes that $\Omega(\sqrt{n/d_0})$ queries are necessary for a $2 - o(1)$-approximation, and hence, for the interesting cases when we may assume $d_0 \geqslant 1$, $\Omega(n)$ degree queries are necessary[3]. Goldreich and Ron [17] subsequently gave a $(1+\rho)$-approximation using both degree and neighbor queries, running in time $\tilde{O}((n/\sqrt{m}) \cdot \mathrm{poly}(1/\rho))$. This bound is also tight, since every constant-factor approximation algorithm must make $\Omega(n/\sqrt{\rho m})$ degree and neighbor queries [17]. A simpler analysis achieving the same bounds was given by Seshadri [31]. Further, Dasgupta, Kumar and Sarlós [11] studied this problem in the model where access to the graph is via samples, in the context of massive networks where the number of nodes may not be known. They obtain a $(1+\rho)$-approximation that uses roughly $O(\log d_U \cdot \log\log d_U)$ samples where d_U is an upper bound on the maximum degree of the graph.

In recent work, Sivasubramaniam, Li and He [32] gave a sublinear-time differentially-private algorithm for approximating the average degree of a graph using Feige's [14] algorithm. Their algorithm achieves a $(2+\rho+o(1))$-approximation for every constant $\rho > 0$. They achieve this by calculating a tight bound for the global sensitivity of the final estimate of Feige's algorithm and adding Laplace noise with respect to this quantity appropriately. By contrast, we achieve a $(1+\rho)$-approximation for any constant $\rho > 0$ – assuming that the privacy parameter is $\varepsilon^{-1} = o(\log^{1/4} n)$.

[2] from here on, we use running time and number of queries interchangeably.

[3] Observe that for $\bar{d}_G = o(1)$ a multiplicative approximation algorithm that can distinguish between two graphs on n vertices, one with 0 edges, and another with, say 1 edge, must sample $\Omega(n)$ vertices, and hence cannot be running in sublinear time.

1.1.2 Privately Approximating the Size of a Maximum Matching and Minimum Vertex Cover

Given an undirected graph, a set of vertex-disjoint edges is called a *matching*. A matching M is *maximal* if M is not properly contained in another matching. A matching M is *maximum* if for any other matching M', $|M| \geqslant |M'|$. A *vertex cover* of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. A *minimum* vertex cover is a vertex cover of the smallest possible size. For a minimization problem, we say that a value $\hat{y}$ is an (α, β)-approximation to y if $y \leqslant \hat{y} \leqslant \alpha y + \beta$. For a maximization problem, we say that a value $\hat{y}$ is an (α, β)-approximation to y if $\frac{y}{\alpha} - \beta \leqslant \hat{y} \leqslant y$. An algorithm $\mathcal{A}$ is an (α, β)-approximation for a value $V(x)$ if it computes an (α, β)-approximation to $V(x)$ with probability at least $2/3$ for any proper input x.

For a graph $G = (V, E)$, we work in the *bounded degree* model, where one can query an i-th neighbor ($i \in [d]$) of a vertex in constant time; denote this query as $\mathrm{Nbr}(v, i)$. Here d is the maximum degree of the graph. If $i > \deg(v)$ for a particular vertex v, then $\mathrm{Nbr}(v, i) = \perp$. We also assume query access to the degree of a vertex, i.e., one can query $\deg(v)$ for any $v \in V$ in constant time.

▶ **Theorem 3.** *There is an ε-(node and edge) differentially-private algorithm for the maximum matching problem that reports a $(2, \rho n)$-approximation with probability $1-(2/n^4+1/n^{192\varepsilon/\rho})$, and runs in expected time $\tilde{O}\left((\bar{d}+1)/\rho^2\right)$, where $\bar{d}$ is the average degree of the input graph.*

▶ **Theorem 4.** *There is an ε-(node and edge) differentially-private algorithm for the minimum vertex cover problem that reports a $(2, \rho n)$-approximation with probability $1-(2/n^4+1/n^{96\varepsilon/\rho})$, and runs in expected time $\tilde{O}\left((\bar{d}+1)/\rho^2\right)$, where $\bar{d}$ is the average degree of the input graph.*

Typically, the privacy parameter ε is a constant, and so is the approximation parameter ρ, in which case the success probability in the theorems above is $1 - 1/(\mathrm{poly}(n))$.

The question of approximating the size of a vertex cover in sublinear-time was first posed by Parnas and Ron [27], who obtained a $(2, \rho n)$-approximation in time $d^{O(\log d/\rho^3)}$, where d is the maximum degree of the graph. Nguyen and Onak [24] improved upon this result by giving a $(2, \rho n)$-approximation for the maximum matching problem, and consequently a $(2, \rho n)$-approximation for the vertex cover problem, in time $O(2^{O(d)/\rho^2})$. The result of [24] was later improved by Yoshida, Yamamoto and Ito [35], who gave an ingenious analysis of the original algorithm to achieve a running time of $O(d^4/\rho^2)$. Onak, Ron, Rosen and Rubinfeld [26] proposed a near-optimal time complexity of $\tilde{O}(\bar{d} \cdot \mathrm{poly}(1/\rho))$, where $\bar{d}$ is the average degree of a graph, but Chen, Kannan, and Khanna [10] identified a subtlety in their analysis, which proved to be crucial to their improved time complexity claim. Very recently, building on ideas from the analysis of [35], Behnezhad [3] gave a new analysis for achieving a $(2, \rho n)$-approximation to the size of maximum matching and minimum vertex cover in time $\tilde{O}((\bar{d}+1)/\rho^2)$. Behnezhad's result nearly matches the lower bound given by Parnas and Ron [27], who showed that $\Omega(\bar{d}+1)$ queries are necessary for obtaining a $(O(1), \rho n)$-estimate in the case of the maximum matching or minimum vertex cover problem.

Our final DP algorithm simply runs the non-private approximation algorithm [3] and then adds Laplace noise proportional to the Coupled Global Sensitivity (of the non-private algorithm). Thus, our time complexity is identical to the non-private approximation algorithm. We show that the added Laplace noise is small enough that it preserves the approximation guarantees of the non-private approximation algorithm.

1.2 Organization

We define and motivate the notion of Coupled Global Sensitivity as a privacy tool in Section 1.3. Then we give a high-level overview of the techniques used for our results in Section 1.4. The formal privacy and accuracy analysis of Theorem 2 are in Sections 2 and 4. The formal analysis for Theorems 3 and 4 are in the full version [5]. We conclude with some open problems in Section 5.

1.3 Coupled Global Sensitivity as a Tool in Privacy analysis

Background and Motivation. Given a query $f : \mathcal{D} \to \mathbb{R}^d$ a general mechanism to answer the query privately is to compute $f(D)$ and then add noise. The global sensitivity of a function was introduced in the celebrated paper by Dwork, McSherry, Nissim and Smith [12], who showed that it suffices to perturb the output of the function with noise proportional to the global sensitivity of the function in order to preserve differential privacy.

▶ **Definition 5** (Global sensitivity). *For a query $f : \mathcal{D} \to \mathbb{R}^d$, the global sensitivity of f (wrt the ℓ_1-metric) is given by*

$$GS_f = \max_{A,B \in \mathcal{D}: A \sim B} \|f(A) - f(B)\|_1 .$$

One can preserve differential privacy by computing $f(D)$ and adding Laplacian noise[4] scaled to the global sensitivity of f, where D is a database. However, in many contexts we may not be able to compute the function f exactly. For example, if the dataset D is very large and our algorithm needs to run in sublinear-time or if the function f is intractable e.g., $f(G)$ is the size of the minimum vertex cover. In cases where we cannot compute f exactly, an attractive alternative is to use a randomized algorithm, say $\mathcal{A}_f$, to approximate the value of f. Given an approximation algorithm $\mathcal{A}_f$ it is natural to ask whether or not we can add noise to $\mathcal{A}_f(D)$ to obtain a differentially private approximation of $f(D)$ and (if possible) how to scale the noise. We first observe that computing $\mathcal{A}_f(D)$ and adding noise scaled to the global sensitivity of f does not necessarily work. Intuitively, this is because the sensitivity of $\mathcal{A}_f$ can be vastly different from that of f. For example, suppose that $GS_f = 1$, $f(D) = n = f(D') + 1$ for neighboring datasets $D \sim D'$ and that our approximation algorithm guarantees that $0.999 \cdot f(D) \leqslant \mathcal{A}_f(D) \leqslant 1.001 \cdot f(D)$. It is possible that $A_f(D) = 1.001n$ and $A_f(D') = 0.999(n-1)$ so that $|A_f(D) - A_f(D')| \geqslant 0.002n$ which can be arbitrarily larger than GS_f as n increases.

Coupled Global Sensitivity. We propose the notion of *coupled global sensitivity* of randomized algorithms as a framework for providing general-purpose privacy mechanisms for approximation algorithms running on a database D. In this framework, our differentially-private algorithms can follow a unified strategy, in which in the first step a non-private randomized approximation algorithm $\mathcal{A}_f(D)$ is run on the dataset, and privacy is obtained by adding Laplace noise proportional with the coupled global sensitivity of $\mathcal{A}_f$[5]. The concept of coupled global sensitivity has been used implicitly in prior work on differential privacy e.g., see [1, 8]. Our work formalizes this notion as a general tool that can be used to design and analyze differentially private approximation algorithms.

[4] Here, the probability density function of the Laplace distribution $\mathrm{Lap}(\lambda)$ is $h(z) = \frac{1}{2\lambda} \exp\left(-\frac{|z|}{\lambda}\right)$.

[5] We note that this is the simplest application of CGS, and as we will see in the analysis of estimating the average degree, we can use CGS to add noise to intermediate quantities used by the randomized algorithm as well.

Notation. When $\mathcal{A}$ is a randomized algorithm we use the notation $x := \mathcal{A}(D; r)$ to denote the output when running $\mathcal{A}$ on input D with fixed random coins r. Similarly, $\mathcal{A}(D)$ can be viewed as a random variable taken over the selection of the random coins r.

▶ **Definition 6** (Coupling). *Let Z and Z' be two random variables defined over the probability spaces $\mathcal{Z}$ and $\mathcal{Z}'$, respectively. A coupling of Z and Z', is a joint variable (Z_c, Z'_c) taking values in the product space $(\mathcal{Z} \times \mathcal{Z}')$ such that Z_c has the same marginal distribution as Z and Z'_c has the same marginal distribution as Z'. The set of all couplings is denoted by* $\mathsf{Couple}(Z, Z')$.

▶ **Definition 7** (Coupled global sensitivity of a randomized algorithm). *Let $\mathcal{A} : \mathcal{D} \times \mathcal{R} \to \mathbb{R}^k$ be a randomized algorithm that outputs a real-valued vector. Then the* coupled global sensitivity *of $\mathcal{A}$ is defined as*

$$\mathsf{CGS}_{\mathcal{A}} := \max_{D_1 \sim D_2} \min_{C \in \mathsf{Couple}(\mathcal{A}(D_1), \mathcal{A}(D_2))} \max_{(z,z') \in C} \|z - z'\|_1$$

▶ Remark 8. We can try to relax the definition of Coupled Global Sensitivity as follows: $\mathsf{CGS}_{\mathcal{A},\delta}$ is the minimum value, say x such that for all neighboring inputs $D_1 \sim D_2$, there exists a coupling C such that $\Pr_{(z,z') \sim C}[|z - z'| > x] \leqslant \delta$. We need to be careful here as we need to ensure that the minimum value x is always well-defined. If we can ensure this, then we can also show that adding noise proportional to $\mathsf{CGS}_{\mathcal{A},\delta}$ preserves (ε, δ)-differential privacy.

▶ **Fact 9.** *Let $\mathcal{A} : \mathcal{D} \times \mathcal{R} \to \mathbb{R}^k$ be a randomized algorithm viewed as a function that takes as input a dataset $\mathcal{D}$ and a random string in the finite set $\mathcal{R}$, and outputs a real-valued vector. For a finite set $\mathcal{R}$, denote by $\mathsf{Sym}(\mathcal{R})$ the symmetric group of all permutations on the elements in $\mathcal{R}$. Then,*

$$\mathsf{CGS}_{\mathcal{A}} \leqslant \max_{D_1 \sim D_2} \min_{\sigma \in \mathsf{Sym}(\mathcal{R})} \max_{R \in \mathcal{R}} \|\mathcal{A}(D_1; R) - \mathcal{A}(D_2; \sigma(R))\|_1$$

The following theorem formalizes the fact that adding noise proportional to the coupled global sensitivity of a randomized algorithm preserves differential privacy (see full version [5] for a formal proof).

▶ **Theorem 10.** *Let $\mathcal{A} : \mathcal{D} \to \mathbb{R}^k$ be a randomized algorithm and define the Laplace mechanism $\mathcal{M}_L(D) = \mathcal{A}(D) + (Y_1, \ldots, Y_k)$, where Y_i are i.i.d. random variables drawn from $Lap(\mathsf{CGS}_{\mathcal{A}}/\varepsilon)$. The mechanism $\mathcal{M}_L$ preserves ε-differential privacy.*

How we use Coupled Global Sensitivity. In our algorithm for estimating the average degree we divide the algorithm into randomized sub-routines and show that the CGS of these sub-routines is small, therefore enabling us to add Laplacian noise proportional to the CGS and ensure the privacy of each sub-routine, and by composition, the privacy of the entire algorithm (See Theorem 13). Similarly, we show that the existing non-private sublinear-time algorithms for maximum matching and minimum vertex cover have small CGS, therefore enabling us to add Laplace noise proportional to the CGS to their outputs thus making them differentially-private (See full version [5]).

1.4 Technical Overview

1.4.1 Privately Estimating the Average Degree

At a high-level, our private algorithm for estimating the average degree follows the non-private variant of Goldreich and Ron [17]. However, there are several challenges that prevent us from simply being able to add Laplacian noise to the output. We overcome these challenges

by first obtaining a new non-private algorithm with the same approximation ratio as that of [17], and then further add appropriate amounts of noise in several steps of the algorithm to obtain both privacy and accuracy guarantees. We begin by describing the algorithm of [17].

The Goldreich-Ron algorithm [17]. The strategy of the original non-private algorithm in [17] is to sample a set S of vertices and partition them into buckets S_i based on their degrees. In particular, for each i we set $S_i = B_i \cap S$ where the set B_i contains all vertices of degrees ranging between $((1+\beta)^{i-1}, (1+\beta)^i]$, where $\beta = \rho/c$ for some constant $c > 1$. Intuitively, as long as $|S_i|$ is sufficiently large the quantity $|S_i|/|S|$ is a good approximation for $|B_i|/n$ with high probability. Let I denote the indices i for which $|S_i|$ is sufficiently large. We can partition edges from the graph into three sets (1) edges with both endpoints in $\bigcup_{i\in I} B_i$, (2) edges with exactly one endpoint in $\bigcup_{i\in I} B_i$, and (3) edges with no endpoints in $\bigcup_{i\in I} B_i$. When the threshold for "large buckets" is tuned appropriately one can show that (whp) type 3 edges can be ignored as there are at most $o(n)$ such edges.

We could use $(1/|S|)\sum_{i\in I}|S_i|(1+\beta)^{i-1}$ as an approximation for $\frac{1}{n}\sum_{i\in I}\sum_{v\in B_i}\deg(v)$. The previous sum counts type (1) edges twice, type (2) edges once and type (3) edges zero times. While it is okay to ignore type (3) edges there could be a lot of type (2) edges which are under-counted. To correct for type (2) edges we can instead try to produce an approximation for the sum $\frac{1}{n}\sum_{i\in I}\sum_{v\in B_i}(1+\alpha_v)\deg(v)$ where α_v denotes the fraction of type (2) edges incident to v. Intuitively, α_v is included to ensure that type (2) edges are also counted twice. For each sampled node $v \in S_i$ we can pick a random neighbor $r(v)$ of v and define $X(v) = 1$ if $r(v) \notin \bigcup_{i\in I} B_i$; otherwise $X(v) = 0$. Observe that in the expected value of the random variable is $\mathbb{E}[X(v)] = \alpha_v$. Since $|S_i|$ is reasonably large for each $i \in I$ and $\deg(u) \approx \deg(v)$ for each pair $u, v \in S_i$ we can approximate the fraction of type (2) edges incident to B_i as $W_i/|S_i|$ where $W_i = \sum_{v\in S_i} X(v)$. Finally, we can use $(1/|S|)\sum_{i\in I}|S_i|(1+W_i/|S_i|)(1+\beta)^{i-1}$ as our final approximation for the average degree.

Challenges to making the original algorithm private by adding noise naively. The first naive attempt to transform the algorithm of [17] into a differentially private approximation would be to add noise to the final output. However, the coupled global sensitivity of this algorithm is large enough that the resulting algorithm is no longer a $(1+\rho)$-approximation.

A second natural strategy to make the above algorithm differentially private is to add Laplace noise to the degree of each vertex and partition vertices in S based on their noisy degrees $\tilde{d}(v) = \deg(v) + Y_v$ where $Y_v \sim \mathrm{Lap}(6/\varepsilon)$. (Note: To ensure that the algorithm still runs in sublinear time we could utilize lazy sampling and only sample $Y_v \sim \mathrm{Lap}(6/\varepsilon)$ when needed). In particular, we can let $\tilde{S}_i = S \cap \tilde{B}_i$ where $\tilde{B}_i$ denotes the set of all nodes v with noisy degree $\tilde{d}(v)$ ranging between $((1+\beta)^{i-1}, (1+\beta)^i]$. Now we can compute $W_i = Z_i + \sum_{v\in\tilde{S}_i} X(v)$ where $Z_i \sim \mathrm{Lap}(6/\varepsilon)$ and return $(1/|S|)\sum_{i\in I}|\tilde{S}_i|(1+\frac{W_i}{|\tilde{S}_i|})(1+\beta)^{i-1}$. While the above approach would preserve differential privacy, the final output may not be accurate. The problem is that the noise Y_v may cause a node v to shift buckets. It is not a problem if $v \in B_i$ shifts to an adjacent bucket i.e., $v \in \tilde{B}_{i-1}$ or $v \in \tilde{B}_{i+1}$ since $(1-\beta)^{i-2}$ and $(1-\beta)^{i+1}$ are still reasonable approximations for the original degree $\deg(v) \in ((1+\beta)^{i-1}, (1+\beta)^i]$. Indeed, when $\deg(v)$ is sufficiently large we can argue that $(1-\beta)\deg(v) < \tilde{d}(v) < (1+\beta)\deg(v)$ with high probability. However, this guarantee does not apply when $\deg(v)$ is small. In this case the Laplace noise Y_v might dominate $\deg(v)$ yielding an inaccurate approximation. Sivasubramaniam et al. [32], made similar observations, and because of these technical barriers, their paper analyzes the simpler strategy for estimating the average degree, which yields a less accurate result. The crucial observation here is that we need to deal with vertices having small degrees in our accuracy analysis separately.

Modified non-DP algorithm achieving the same approximation ratio. To address the challenges discussed above we first propose a modification to the strategy given by [17]. While the modified algorithm is still non-private it still achieves a $(1+\rho)$-approximation for any $\rho > 0$ and is amenable to differentially private adaptations. Our algorithm now samples vertices S *without replacement* and puts them into buckets $S_i = B_i \cap S$ according to their degrees. The key modification is that we merge all of the buckets with smaller degrees i.e., $i \leqslant K$[6] into one. We redefine B_1 to denote this merged bucket and $S_1 = S \cap B_1$ and we redefine I to be the set of all indices $i > K$ such that $|S_i|$ is sufficiently large. If B_1 is not too large then all of the edges incident to B_1 can simply be ignored as the total number of these edges will be small. Otherwise, we can account for edges that are incident to B_1 by adding $\frac{1}{|S|}\sum_{v\in S_1}(1+X(v))\deg(v)$ to our final output. Since we merged all of the buckets with smaller degrees we no longer have the guarantee that $\deg(u) \approx \deg(v)$ for all $u, v \in S_1$. However, since $\deg(v)$ is reasonably small for each $v \in S_1$ the variance is still manageable. Intuitively, the sum $\frac{1}{|S|}\sum_{v\in S_1}(1+X(v))\deg(v)$ approximates $\frac{1}{n}\sum_{v\in B_1}(1+\alpha_v)\deg(v)$ where α_v now denotes the fraction of edges incident to v whose second endpoint lies outside the set $B_1 \cup \bigcup_{i\in I} B_i$.

The differentially-private modified algorithm. We now introduce our sublinear-time differentially-private algorithm to approximate the average degree in Algorithm 4. Algorithm 4 relies on three subroutines given by Algorithms 1, 2, and 3. Splitting the algorithm into separate modules simplifies the privacy analysis as we can show that each subroutine is $\varepsilon/3$-differentially private – it follows that the entire algorithm is ε-differentially private. In Algorithm 1 we add Laplace noise to the degrees of *all* vertices in the graph and then return a sample of vertices, say S (sampled uniformly without replacement) along with their noisy degrees. For simplicity we describe Algorithm 1 in a way that the running time is linear in the size of the input. We do this to make our privacy analysis simpler. However, we can implement Algorithm 1 with lazy sampling of Laplace noise Y_u when required i.e., if node u is in our sample S or if $u = r(v)$ was the randomly selected neighbor of some node $v \in S$.

Algorithm 1 **NoisyDegree**.

NoisyDegree takes G as input and returns a set of sampled vertices along with the noisy degrees of every vertex in G.

1. Uniformly and independently select $\Theta(\sqrt{n}\cdot\mathrm{poly}(\log(n)/\rho)\cdot\mathrm{poly}(1/\varepsilon))$ vertices (without replacement) from V and let S denote the set of selected vertices.
2. For every $v \in V(G)$,

$$\tilde{d}(v) = \deg(v) + Y_v\ ,$$

   where $Y_v \sim \mathrm{Lap}(6/\varepsilon)$.
3. Return $\{\tilde{d}(v)\}_{v\in V(G)}, S$

Given the output of Algorithm 1 we can partition the sample S into buckets $\tilde{S}_i = S \cap \tilde{B}_i$ using their noisy degree. Here, we define $\tilde{B}_i = \left\{v : \tilde{d}(v) \in \left((1+\beta)^{i-1}, (1+\beta)^i\right)\right\}$ and we also define a merged bucket $S_1 = S \cap \left\{v : \tilde{d}(v) \leqslant (1+\beta)^{K-1}\right\}$ containing all sampled nodes

[6] where we fix $K := \left(2 + \log_{1+\beta}\left(\frac{2|S|\sqrt{\rho}}{\beta \log_{1+\beta}(n)\sqrt{n\sqrt{\log n}}}\right)\right)$ in the sequel

with noisy degree at most $(1+\beta)^{K-1}$. Here, K is a degree threshold parameter that we can tune. Now given a size threshold parameter T we can define $I = \left\{i \geqslant K \ : \ \left|\tilde{S}_i\right| \geqslant 1.2T \cdot |S|\right\}$ to be the set of big buckets. We remark that as a special case we define $|S_1|$ to be "small" if $|S_1| < 1.2T \cdot \sqrt{|S|} \cdot |S|$ instead of $|S_1| < 1.2T \cdot |S|$. As an intuitive justification we note that (whp) for each node v with noisy degree $\tilde{d}(v) \leqslant (1+\beta)^{K-1}$ the actual degree $\deg(v)$ will not be too much larger than $(1+\beta)^{K-1}$. In this case we have $\sum_{v:\tilde{d}(v)\leqslant(1+\beta)^{K-1}} \deg(v) \leqslant |S_1| \max_{v:\tilde{d}(v)\leqslant(1+\beta)^{K-1}} \deg(v) = o(n)$ so that we can safely ignore the edges incident to S_1.

Intuitively, for each large bucket $i \in I$, Algorithm 2 computes $\tilde{\alpha}_i = W_i/|\tilde{S}_i|$ our approximation of the fraction of type (2) edges incident to $\tilde{B}_i$. If S_1 is large then type (2) edges are (re)defined to be the edges with exactly one endpoint in $\left\{v : \tilde{d}(v) \leqslant (1+\beta)^{K-1}\right\} \cup \bigcup_{i\in I} \tilde{B}_i$. To preserve differential privacy we add laplace noise to W_i i.e., $W_i = Z_i + \sum_{v\in\tilde{S}_i} X(v)$ where $Z_i \sim \mathsf{Lap}(6/\varepsilon)$. We remark that (whp) we will have $Z_i = o(|\tilde{S}_i|)$ for each large bucket $i \in I$. Thus, the addition of laplace noise will have a minimal impact on the accuracy of the final result.

Algorithm 2 NoisyBigSmallEdgeCount. Here $M_{\rho,n}$ is a degree threshold parameter and T is a size threshold parameter. Note that the relationship between the parameters K (used informally as a degree threshold parameter in the overview) and $M_{\rho,n}$ is $K = 2 + \log_{(1+\beta)}\lceil 6M_{\rho,n}/\beta\rceil$.

NoisyBigSmallEdgeCount takes as input $G, I, \{\tilde{S}_i\}_{i=1}^t, S_1, \{\tilde{d}(v)\}_{v\in V(G)}, M_{\rho,n}, T$ and returns an approximation of the fraction of edges that are between big buckets and small buckets.

1. For every $i \in I$, ▷ count the edges between buckets in I and small buckets
   a. For all $v \in \tilde{S}_i$,
      i. Pick a random neighbor of v, say $r(v)$.
      ii. If $|S_1| < 1.2T\cdot\sqrt{|S|}\cdot|S|$, i.e., if S_1 is a small bucket. Then if $\tilde{d}(r(v)) \in ((1+\beta)^{i-1}, (1+\beta)^i]$ for some $i \notin I$, then $X(v) = 1$, otherwise $X(v) = 0$.
      iii. Otherwise, S_1 is not small. Therefore, if $\tilde{d}(r(v)) \in ((1+\beta)^{i-1}, (1+\beta)^i]$ for some $i \notin I$ and $i > \log_{1+\beta}\lceil\left(\frac{6M_{\rho,n}}{\beta}\right)\rceil + 2$, then $X(v) = 1$, otherwise $X(v) = 0$.
   b. Define $W_i := \sum_{v\in\tilde{S}_i} X(v) + Z_i$ where $Z_i \sim \mathrm{Lap}(6/\varepsilon)$ and $\tilde{\alpha}_i := \frac{W_i}{|\tilde{S}_i|}$.
2. return $\{W_i\}_{i\in I}, \{\tilde{\alpha}_i\}_{i\in I}$

If the merged bucket S_1 is small then we can ignore edges incident to S_1 and Algorithm 3 will simply output $\frac{1}{|S|}\sum_{i\in I} |\tilde{S}_i| \cdot (1+\tilde{\alpha}_i) \cdot (1+\beta)^i$. In this case the output can be computed entirely from the differentially private outputs that have already been computed by Algorithms 1 and 2 without even looking at the graph G. Intuitively, for any large bucket $i \in I$ and $v \in \tilde{S}_i$ we expect that (whp) $|Y_v| = |\tilde{d}(v) - \deg(v)|$ is small enough to ensure that $(1+\beta)^{i-2} \leqslant \deg(v) \leqslant (1+\beta)^{i+1}$. Thus, $(1+\beta)^i$ is still a reasonable approximation for $\deg(v)$.

If the merged bucket S_1 is sufficiently large, then we need to account for the edges within S_1 itself as well as the fraction of edges between S_1 and small buckets. We introduce a new estimator to approximate the fraction of edges between S_1 and small buckets given by $Z + \sum_{v\in S_1}(1+X(v))\cdot\deg'(v)$ where $Z \sim \mathrm{Lap}\left(36M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)\right)$ and $\deg'(v) = \min\{\deg(v), 6M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)\}$ (See Algorithm 3) – the relationship between the parameters K and $M_{\rho,n}$ is $K = 2 + \log_{(1+\beta)}\lceil 6M_{\rho,n}/\beta\rceil$. The Laplace Noise term is added to preserve differential privacy. We define the clamped degrees $\deg'(v)$ to ensure that the coupled global sensitivity of the randomized subroutine computing $\sum_{v\in S_1}(1+X(v))\cdot\deg'(v)$ is upper

bounded by $12M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)$. This way we can control the laplace noise parameters to ensure that $Z = o(|S_1|)$ with high probability so that the noise term Z does not adversely impact accuracy. Intuitively, we expect that $Y_v \leqslant M_{\rho,n}$ for all nodes v with high probability. In this case for any node $v \in S_1$ we will have $\deg'(v) = \deg(v) \leqslant 6M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)$.

Algorithm 3 **NoisyAvgDegree**.

NoisyAvgDegree takes $\{\tilde{S}_i\}_{i=1}^t, \{\tilde{d}(v)\}_{v\in V(G)}, \{\tilde{\alpha}_i\}_{i\in I}, I, M_{\rho,n}, T$ as input and returns the noisy estimator for average degree of the graph.

1. If $|S_1| < 1.2T\cdot\sqrt{|S|}\cdot|S|$ then output $\frac{1}{|S|}\sum_{i\in I}|\tilde{S}_i|\cdot(1+\tilde{\alpha}_i)\cdot(1+\beta)^i$.
2. Else, for every $v \in S_1$,
   a. Pick a random neighbor of v, say $r(v)$.
   b. If $\tilde{d}(r(v)) \in ((1+\beta)^{i-1},(1+\beta)^i]$ for some $i \notin I$ and $i > \log_{1+\beta}\lceil\left(\frac{6M_{\rho,n}}{\beta}\right)\rceil + 2$, then $X(v) = 1$, otherwise $X(v) = 0$.
   c. Output

$$\frac{1}{|S|}\left(\sum_{i\in I}|\tilde{S}_i|\cdot(1+\tilde{\alpha}_i)\cdot(1+\beta)^i + Z + \sum_{v\in S_1}(1+X(v))\cdot\deg'(v)\right),$$

where

$$Z \sim \mathrm{Lap}\left(36M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)\right)$$

and

$$\deg'(v) = \min\left\{\deg(v), 6M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)\right\}$$

Algorithm 4 **Main DP Algorithm**.

Main DP Algorithm that takes graph G as input and outputs an approximation of its average degree.

1. $\{\tilde{d}(v)\}_{v\in V(G)}, S :=$ **NoisyDegree**(G) ▷ see Algorithm 1
2. For $i = 1, 2\ldots, t$, let $\tilde{S}_i = \{v \in S : \tilde{d}(v) \in ((1+\beta)^{i-1},(1+\beta)^i]\}$ where $t := \lceil\log_{(1+\beta)}(n)\rceil$.
3. Define $M_{\rho,n} := \frac{1}{3}\cdot\sqrt{\frac{\rho}{n\sqrt{\log(n)}}}\cdot\frac{|S|}{t}$, $S_1 := \cup_{i\leqslant\log_{1+\beta}\left(\frac{6M_{\rho,n}}{\beta}\right)+2}\tilde{S}_i$, and, $I = \{i > \log_{1+\beta}\left(\frac{6M_{\rho,n}}{\beta}\right)+2 : |\tilde{S}_i| \geqslant 1.2T\cdot|S|\}$ where $T := \frac{1}{2}\sqrt{\frac{\rho}{n}}\cdot\frac{\varepsilon}{(1+\varepsilon)}\cdot\frac{1}{t}$.
4. $\{W_i\}_{i\in I}, \{\tilde{\alpha}_i\}_{i\in I} :=$ **NoisyBigSmallEdgeCount**$(G, I, \{\tilde{S}_i\}_{i=1}^t)$. ▷ see Algorithm 2
5. **NoisyAvgDegree**$(G, S, \{\tilde{S}_i\}_{i=1}^t, \{\tilde{\alpha}_i\}_{i\in I}, I, M_{\rho,n}, T)$. ▷ see Algorithm 3

The full analysis of Theorem 2 can be found in Sections 2 and 4.

▶ Remark 11. A simpler algorithm for estimating the average degree was given by Seshadri [31]. The main intuition behind this algorithm is that out of m edges of a graph, there are not "too many" edges that contribute a high degree. Thus the algorithm samples vertices and a random neighbor of each sampled vertex, but it only counts edges (scaled by a factor of 2 times the degree of the sampled vertex) for which the degree of the random neighbor is higher than that of the degree of the sampled vertex.

The Coupled Global Sensitivity of the final estimate returned by this algorithm is high (proportional to the degree of the sampled vertex and its random neighbor); thus adding Laplace noise directly to the estimate would result in a very inaccurate algorithm. It is unclear how to mitigate this issue and make this algorithm differentially-private with a reasonable accuracy guarantee.

1.4.2 Privately Estimating Maximum Matching and Vertex Cover Size

At a high-level our private algorithms for estimating the maximum matching and vertex cover add laplace noise (to the outputs) proportional to the coupled global sensitivity of the randomized non-private algorithms for the corresponding problems. The challenge lies in proving the coupled global sensitivity of these non-private algorithms is small.

We first describe and analyze the coupled global sensitivity of the classical polynomial-time greedy matching algorithm. This is helpful in our analysis of the non-private sublinear-time algorithm for maximum matching in the sequel.

We then describe and give a proof sketch of the coupled global sensitivity of the non-private sublinear-time matching algorithm [3]. The formal proofs for privately estimating the maximum matching and minimum vertex cover size are in the full version [5]. Recall that d is the maximum degree and $\bar{d}$ is the average degree of the graph .

The Polynomial-time Greedy Matching Algorithm $\mathcal{A}_{MM}$. This algorithm takes as input a graph $G = (V, E)$ and a random permutation π on the set of pairs $(x, y) \in V \times V$, with $x \neq y$, and processes each pair of vertices (x, y) in the increasing order of ranks given by π, and greedily adds edges to a maximal matching whose size is finally output[7]. Since the size of the maximal matching produced is known to be at least $\frac{1}{2}$ of the size of a maximum matching, this gives a non-private 2-approximation of the size of a maximum matching in G.

CGS of the Greedy Algorithm $\mathcal{A}_{MM}$. We show that the CGS of the greedy algorithm (with respect to node-neighboring graphs) is at most 1. Note that once the ranking on the edges is fixed the maximal matching obtained by $\mathcal{A}_{MM}$ is also fixed. Let σ_I be the identity permutation over the ranking of edges, i.e., we have $\sigma_I(\pi) = \pi$. We use Fact 9 to observe that,

$$\begin{aligned}
\mathsf{CGS}_{\mathcal{A}_{MM}} &\leqslant \max_{G_1 \sim G_2} \min_{\sigma} \max_{\pi} |\mathcal{A}_{MM}(G_1; \pi) - \mathcal{A}_{MM}(G_2; \sigma(\pi))| \\
&\leqslant \max_{\substack{G_1 \sim G_2 \\ \pi}} |\mathcal{A}_{MM}(G_1; \pi) - \mathcal{A}_{MM}(G_2; \sigma_I(\pi))| \\
&= \max_{\substack{G_1 \sim G_2 \\ \pi}} |\mathcal{A}_{MM}(G_1; \pi) - \mathcal{A}_{MM}(G_2; \pi).|
\end{aligned}$$

Therefore it is sufficient to analyze the relative size of the matching obtained on node-neighboring graphs G_1, G_2 that are processed by the greedy algorithm in the order given by the same π.

Let $G_1 \sim G_2$ where v^* is such that $E(V_1 \setminus \{v^*\}) = E(V_2 \setminus \{v^*\})$. Denote the greedy matchings obtained from $\mathcal{A}_{MM}(G_1, \pi)$ as M_1 and from $\mathcal{A}_{MM}(G_2, \pi)$ as M_2. Suppose edge e^* is incident to v^* such that $e^* \in E_2$, and $e^* \notin E_1$. We will show that $||M_1| - |M_2|| \leqslant 1$, which implies that $\max_{\substack{G_1 \sim G_2 \\ \pi}} |\mathcal{A}_{MM}(G_1; \pi) - \mathcal{A}_{MM}(G_2; \pi)| \leqslant 1$, thus proving that $\mathsf{CGS}_{\mathcal{A}_{MM}} \leqslant 1$.

[7] We note that the non-private algorithms [24, 35, 26] only consider the ranking π over m edges of the graph, whereas we consider the ranking over all $\binom{n}{2}$ pairs of vertices. This is because we want to define a "global" ranking so that we can define the same ranking consistently over neighboring graphs that may have different edges.

We first claim that if $e^* \notin M_1 \cup M_2$ then $|M_1| = |M_2|$. Since the greedy algorithm considers edges in the same order, the exact same edges must have been placed in M_1 as in M_2 before $e*$ is processed. Since $e^* = (v^*, u)$ is not chosen in M_2 it must have been the case that by this time u was matched in M_2, and thus the same matched edge must occur in M_1. From here on the algorithm again must make the same choices for the edges to be placed in M_1 and M_2.

Next, we claim that if $e^* \in M_1 \cup M_2$ then $M := M_1 \oplus M_2$[8] is one connected component containing e^*. Consequently, $||M_1| - |M_2|| \leqslant 1$. Since $e^* \in M_2$ and e^* cannot be in M_1, it is clear that $e^* \in M$. Suppose for the sake of contradiction, M consists of two connected components C_1, C_2 and WLOG $e^* \in C_1$. Consider edges in C_2. By Berge's Lemma [34], C_2 is either an alternating path or an alternating even cycle, with alternating edges from M_1 and M_2. Also, the edges in C_2 exist in both G_1 and G_2 with the same ranking. Observe that since C_2 is separate from C_1 containing e^*, if we replace edges in C_2 belonging to M_2 in the original graph G_2 by edges in C_2 belonging to M_1, this is still a valid maximal matching for the graph G_2. In fact, the greedy algorithm considers edges in C_2 in the same order for both graphs G_1, G_2, so the edges in M_1 and M_2 should be the same, in other words, C_2 cannot be a part of $M = M_1 \oplus M_2$, and hence M must have only one connected component, which contains e^*. Now, since M is either an alternating path or even cycle, $||M_1| - |M_2|| \leqslant 1$.

The Local Maximum Matching Algorithm $\mathcal{A}_{\mathbf{sub-MM}}$. We describe the local algorithm implemented by [3] in Algorithm 5. We modify the original algorithm to sample vertices *without replacement*. The algorithm then calls the vertex cover oracle (denoted as $\mathcal{O}^{\pi}_{VC}$) on each sampled vertex which subsequently calls the maximal matching oracle (denoted as $\mathcal{O}^{\pi}_{MO}$) on the incident edges to determine whether the sampled vertex is in the matching fixed by the ranking of edges π. Finally, the algorithm returns an estimate of the maximum matching size based on the number of sampled vertices in the matching. We note that in [3] the same sampling algorithm simultaneously outputs an approximation to maximum matching size and minimum vertex cover size. We choose to write the sampling procedure for estimating the maximum matching size and minimum vertex cover size separately so that it is easier to understand the Coupled Global Sensitivity for outputting the two different estimators.

Algorithm 5 Local Maximum Matching algorithm $\mathcal{A}_{sub-MM}$ using Oracle access.

Input. Input Graph $G = (V, E)$.

1. Uniformly sample $s = 16 \cdot 24(\ln n)/\rho^2$ vertices from V without replacement.
2. For $i = 1 \dots s$, if $\mathcal{O}^{\pi}_{VC}(v_i) =$ True then let $X_i = 1$, otherwise let $X_i = 0$.
3. return $\tilde{M} = \frac{n}{2s}(\sum_{i \in [s]} X_i) - \frac{\rho n}{2}$.

▶ Remark 12. [3] gives an efficient simulation of the matching and vertex cover oracles which exposes edges incident to a vertex in batches only when they are needed. We assume the efficient simulation of these oracles in our algorithms.

CGS of the Local Matching Algorithm $\mathcal{A}_{\mathbf{sub-MM}}$. Our main techniques involve identifying the sources of randomness in the local algorithm itself and then coupling the random coins of the runs of the algorithm on neighboring graphs. We follow the local algorithm given by [3] which samples vertices for both matching and vertex cover size estimation. We show that the identity coupling is sufficient in this case.

[8] $M_1 \oplus M_2$ is the *symmetric difference* of sets and this is defined as the set of edges in either M_1 or M_2 but not in their intersection.

In a previous version of our paper (before we were aware of the results of [3]), we analyzed the Coupled Global Sensitivity of the local matching algorithm given by [24, 35] which samples a set of edges uniformly at random, and calls a matching oracle on each sampled edge. The matching oracle indicates whether the edge is in the greedy matching fixed by the ranking π or not. Analyzing the Coupled Global Sensitivity of this algorithm is more challenging, i.e., considering the identity permutation σ_I over the ranking of edges π and sampled edges does not work. This is because for node-neighboring graphs G_1, G_2, it could be the case that all the edges sampled from G_1 belong to the matching M_1 fixed by the ranking π, but the same edges sampled from G_2 may not be in the matching M_2 fixed by the ranking π. Thus, we need to carefully define a bijection that maps edges in the matching M_1 to edges in the matching M_2.

2 Privacy Analysis of Theorem 2

▶ **Theorem 13.** *The Algorithm 4 is ε-DP.*

Proof. We will approach the privacy analysis in a modular fashion, i.e., we will analyze each sub-routine separately and show that by composition, the entire algorithm is ε-differentially private.

In the sequel, when analyzing the coupled global sensitivity of intermediate randomized quantities, we use Fact 9.

▷ Claim 14. Algorithm NoisyDegree (see Algorithm 1) is $\varepsilon/3$-DP.

Proof. First, fix any sample S. Define the function $f_{noisy-deg} := \{\tilde{d}(v)\}_{v\in V(G)}$. Observe that the degree of a node can change by at most 1 from adding or deleting an edge, and therefore $f_{noisy-deg}$ changes by at most 2 by adding or deleting an edge, in other words, the $GS_{f_{noisy-deg}} = 2$ and we can add noise proportional to $2/\varepsilon$. ◁

▷ Claim 15. Algorithm NoisyBigSmallEdgeCount (Algorithm 2) is $\varepsilon/3$-DP.

Proof. We fix noisy degrees $\{\tilde{d}(v)\}_{v\in V(G)}$, consequently fixing the buckets $\tilde{S}_1, \ldots, \tilde{S}_t$ and set I. Define the function $f_{t,\tilde{d}} := \{f_{\tilde{S}_i,\tilde{d}}(G; r)\}_{i\in I}$, and the function $f_{\tilde{S}_i,\tilde{d}}(G; r) = \sum_{v\in\tilde{S}_i} H(r(v))$ where $H(w) = 1$ if and only if we have $\tilde{d}(w) \in ((1+\beta)^{i-1}, (1+\beta)^i]$ for some $i \notin I$ and $|S_1| < 1.2T \cdot \sqrt{|S|} \cdot |S|$ or if $\tilde{d}(w) \in ((1+\beta)^{i-1}, (1+\beta)^i]$ for some $i \notin I$ and $i > \log_{1+\beta}\lceil\left(\frac{6M_{\rho,n}}{\beta}\right)\rceil + 2$; here $r(\cdot)$ defines the random coins used to sample a neighbor of v. We analyze $CGS_{f_{\tilde{S}_i,\tilde{d}}}$, and argue that $CGS_{f_{t,\tilde{d}}} \leqslant CGS_{f_{\tilde{S}_i,\tilde{d}}}$.

First, we show that for all fixed S, $\{\tilde{d}(v)\}_{v\in S}$ and $i \in I$, the $CGS_{f_{S_i,\tilde{d}}}$ is at most 2. Consider G and G' such that edge $(u^*, v^*) \in G$, but does not exist in G'. Fix any coupling such that $r(w) = r'(w)$ for all $w \neq u^*, v^*$, where r, r' defines the random coins for sampling neighbors of w in G and G' respectively. Now we have $X(w) = H(r(w)) = H(r'(w)) = X'(w)$ for all $w \neq u^*, v^*$. Thus, $CGS_{f_{\tilde{S}_i,\tilde{d}}} = |f_{\tilde{S}_i,\tilde{d}}(G; r) - f_{\tilde{S}_i,\tilde{d}}(G'; r')| = |\sum_{v\in\tilde{S}_i} H(r(v)) - \sum_{v\in\tilde{S}_i} H(r'(v))| = |H(r(v^*)) + H(r(u^*)) - H(r'(v^*)) - H(r'(u^*))| \leqslant 2$. Now, since the differing endpoints u^*, v^* can only appear in at most one of the i-th iterations simultaneously, it is clear to see that $CGS_{f_{t,\tilde{d}}}$ is also at most 2. ◁

▷ Claim 16. Algorithm NoisyAvgDegree (Algorithm 3) is $\varepsilon/3$-DP.

Proof. We fix noisy degrees $\{\tilde{d}(v)\}_{v\in V(G)}$, and sample S consequently fixing the buckets $\tilde{S}_1, \ldots, \tilde{S}_t$ and set I, and we fix $\{\tilde{\alpha}_i\}_{i=1}^t$. Note that the first output in Line 1 given by $\frac{1}{|S|}\sum_{i\in I} |\tilde{S}_i| \cdot (1+\tilde{\alpha}_i) \cdot (1+\beta)^i$ is already private since the terms in the summation consist of

parameters that are either noisy or public or both. We need to show that the second output in Line 2c is private. In particular, define the function $f_{S_1,\tilde{d}}(G;r) := \sum_{v\in S_1}(1+H_1(r(v)))\cdot\deg'(v)$ where $\deg'(v) = \min\{\deg(v), 6M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)\}$ and $H_1(w) = 1$ if and only if $\tilde{d}(w) \in ((1+\beta)^{i-1},(1+\beta)^i]$ for some $i \notin I$ and $i > \log_{1+\beta}\lceil\left(\frac{6M_{\rho,n}}{\beta}\right)\rceil+2$. We claim that for all fixed S and $\{\tilde{d}(v)\}_{v\in S}$, the $CGS_{f_{S_1,\tilde{d}}}$ is at most $12M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)$. Consider G and G' such that edge $(u^*,v^*)\in G$, but does not exist in G'. Fix any coupling such that $r(w) = r'(w)$ for all $w \neq u^*, v^*$, where r, r' defines the random coins for sampling neighbors of w in G and G' respectively. Now we have $X(w) = H_1(r(w)) = H_1(r'(w)) = X'(w)$ for all $w \neq u^*, v^*$. Thus, $|f_{S_1,\tilde{d}}(G;r) - f_{S_1,\tilde{d}}(G';r')| = |\sum_{v\in\tilde{S}_1}(1+H_1(r(v)))\cdot\deg'(v) - \sum_{v\in\tilde{S}_1}(1+H_1(r'(v)))\cdot\deg'(v)| = |(1+H(r(v^*)))\cdot\deg'(v^*) + (1+H(r(u^*)))\cdot\deg'(u^*) - (1+H(r'(v^*)))\cdot\deg'(v^*) - (1+H(r'(u^*)))\deg'(u^*)| \leqslant 2\cdot 6M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right) = 12M_{\rho,n}\left(3+\beta+\frac{1}{\beta}\right)$. Note that we introduce $\deg'(v)$, to ensure that the sensitivity of $f_{S_1,\tilde{d}}$ remains small. ◁

By composition, we have that the main algorithm is ε-DP. ◀

3 Preliminaries

We state the following tail bound for a random variable drawn from the Laplace Distribution.

▶ **Fact 17.** *If* $Y \sim \mathrm{Lap}(b)$, *then*

$$\Pr[|Y| \geqslant \ell\cdot b] = \exp(-\ell)\ .$$

Next, we state a well-known fact which implies that the concentration results for sampling with replacement obtained using Chernoff bounds type methods (bounding moment generating function + Markov inequality) can be transferred to the case of sampling without replacement.

▶ **Fact 18** ([2, 20]). *Let* $X = (x_1,\ldots,x_N)$ *be a finite population of* N *points and* $X_1,\ldots,X_n$ *be a random sample drawn without replacement from* X, *and* $Y_1,\ldots,Y_n$ *be a random sample drawn with replacement from* X. *If* $f:\mathbb{R}\to\mathbb{R}$ *is continuous and convex, then*

$$\mathbb{E}\left[f\left(\sum_{i=1}^{n}X_i\right)\right] \leqslant \mathbb{E}\left[f\left(\sum_{i=1}^{n}Y_i\right)\right]\ .$$

4 Accuracy Analysis of Theorem 2

4.1 Proof Sketch of Theorem 2

In this section, we give a sketch of the accuracy analysis. The more formal proofs can be found in the full version [5].

▶ **Theorem 19.** *For every* $\rho < 1/4$, $\beta \leqslant \rho/8$, *and* $\varepsilon^{-1} = o(\log^{1/4}(n))$, *for sufficiently large* n, *the main algorithm (see Algorithm 4) outputs a value* $\tilde{d}$ *such that with probability at least* $1-o(1)$, *it holds that*

$$(1-\rho)\cdot\bar{d} \leqslant \tilde{d} \leqslant (1+\rho)\cdot\bar{d}$$

Proof. The main proof strategy conditions on S_1 being sufficiently large or not. First, consider Case 1 when $|S_1| < 1.2T\cdot\sqrt{|S|}\cdot|S|$ where T is a size threshold parameter. We first show that for $i\in I$ the noisy buckets $|\tilde{B}_i|/n$ are approximated well by $|\tilde{S}_i|/|S|$. Next we show

that the number of vertices in buckets that are significantly smaller than the size threshold are of size $O(\sqrt{n})$ (for buckets $U' := \{v \in \tilde{B}_i : (i \notin I) \wedge (i > \log_{1+\beta}\left(\frac{6M_{\rho,n}}{\beta}\right) + 2)\}$, and of size $\tilde{O}(n^{3/4})$ (for bucket $B_1 := \cup_{i<\log_{1+\beta}\left(\frac{6M_{\rho,n}}{\beta}\right)+2}\tilde{B}_i$. This leads to the corollary (see full version [5] for the formal statement) which bounds the number of edges between small buckets as roughly $\tilde{O}(\rho n + n^{3/4})$.

One of our main contributions is showing that the actual fraction of edges between sufficiently large buckets and small buckets, denoted by α_i, is approximated well by our noisy estimator $\tilde{\alpha}_i$.

▶ **Corollary 20.** *Assuming that* $\varepsilon^{-1} = o(\log^{1/4}(n))$*, for every* $i \in I$*, for sufficiently large* n*, we have that with probability at least* $1 - o(1)$*,*

1. $|\tilde{\alpha}_i - \alpha_i| \leqslant \frac{\rho}{4}\alpha_i$ *if* $\alpha_i \geqslant \rho/8$.
2. $\tilde{\alpha}_i \leqslant \rho/4$*, if* $\alpha_i \leqslant \rho/8$.

Finally, we need to show that for sufficiently large noisy buckets, the actual degrees of the vertices (sans noise) only shifts to an adjacent noisy bucket. This helps us bound the number of edges whose one endpoint resides in a sufficiently large noisy bucket. We have shown that with high probability, all approximations of edges between the different types of buckets is good, which leads to the main Lemma for Case 1.

Now consider Case 2 when $|S_1| > 1.2T \cdot \sqrt{|S|} \cdot |S|$. We show that the bucket $|B_1|/n$ is now approximated well by $|S_1|/|S|$. We introduce a different estimator for counting edges between B_1 and small buckets given by $Z + \sum_{v \in S_1}(1 + X(v)) \cdot \deg'(v)$, where $Z \sim \text{Lap}\left(36M_{\rho,n}\left(3 + \beta + \frac{1}{\beta}\right)\right)$ and $\deg'(v) = \min\{\deg(v), 6M_{\rho,n}\left(3 + \beta + \frac{1}{\beta}\right)\}$. First, we show that for every $v \in S_1$, with high probability $\deg'(v) = \deg(v)$. Our main contribution in this case is showing that our estimator (sans noise) approximates the fraction of the sum of the edges between B_1 and all vertices in the graph (denoted by E_1), and the edges between B_1 and vertices in small buckets in the graph (denoted by E_1') well (see lemma below).

▶ **Lemma 21.** *Let* $\bar{d}_1$ *be the average degree of bucket* B_1*. If* $|B_1| > 1.5T \cdot \sqrt{|S|} \cdot n$*,*

1. *If* $\bar{d}_1 \geqslant 1$*, then with probability at least* $1 - o(1)$*,*

$$\left(1 - \frac{\rho}{4}\right) \cdot \frac{|E_1| + |E_1'|}{n} < \frac{1}{|S|}\sum_{v \in S_1}(1 + X(v)) \cdot \deg(v) < \left(1 + \frac{\rho}{4}\right) \cdot \frac{|E_1| + |E_1'|}{n}$$

2. *If* $\bar{d}_1 < 1$*, and* $\bar{d} \geqslant 1$*, then with probability at least* $1 - o(1)$*,*

$$\frac{|E_1| + |E_1'|}{n} - \rho/4 < \frac{1}{|S|}\sum_{v \in S_1}(1 + X(v)) \cdot \deg(v) < \frac{|E_1| + |E_1'|}{n} + \rho/4$$

To complete this part of the proof, we show that the noise added to the estimator (denoted by Z) is small and therefore, the noisy estimator also approximates the quantity $(|E_1| + |E_1'|)/n$ well.

The rest of the analysis is similar to Case 1 and we invoke the same lemmas to show that with high probability, the approximations of edges between the rest of the sufficiently large buckets, and between the small buckets, as well as between the sufficiently large buckets and small buckets is good, thus giving us the main Lemma for Case 2.

Combining these two main lemmas proves our main theorem statement. ◀

5 Conclusions and open questions

In this work we give a differentially-private sublinear-time $(1+\rho)$-approximation algorithm for estimating the average degree of the graph. We achieve a running time comparable to its non-private counterpart, which is also tight in terms of its asymptotic behaviour with respect to the number of vertices of the graph. We also give the first differentially-private approximation algorithms for the problems of estimating maximum matching size and vertex cover size of a graph.

To analyze the privacy of our algorithms, we proposed the notion of coupled global sensitivity, as a generalization of global sensitivity, which is applicable to randomized approximation algorithms. We show that coupled global sensitivity implies differential privacy, and use it to show that previous non-private algorithms from the literature, or variants, can be made private by finely tuning the amounts of noise added in various steps of the algorithms.

We propose several directions of investigation for developing the notion of coupled global sensivity further and open problems pertaining to differentially-private sublinear-time algorithms for graphs.

Other applications and limitations of CGS. In particular, what are the limitations of the CGS method? Can we characterize the set of algorithms with small CGS? Are there other natural problems for which we already have algorithms with small CGS, and hence that are easily amenable to privacy analogues? Are there algorithms for which we can prove large lower bounds on the CGS and yet they provide differential privacy?

Better approximations for maximum matching problems. In [24, 35], the authors also give a $(1, \rho n)$-approximation of maximum matching size with a query complexity that is exponential in d. Their analysis involves iterating over a sequence of oracles to augment paths of small length, in increasing order of lengths. The matching oracle considered in this work is used only in the first iteration. Analyzing the coupled global sensitivity of that algorithm appears to be much more involved, and we leave it as an open problem.

Better time complexity guarantees for $(2, \rho n)$-approximation matching and vertex cover algorithms. Note that our results in Theorems 3 and 4 achieve an expected running time. In contrast, the results in [3] achieve a high-probability bound on the time-complexity. This can be done by running multiple instances of the resulting approximation algorithm for enough time and returning the output of the instance that terminates first (the analysis involves a simple application of Markov inequality). Achieving this step in a way that preserves privacy would result in a degradation of the privacy parameter ε, due to composition. We leave it as an open question to provide a tighter privacy vs time-complexity analysis.

References

1 Daniel Alabi, Audra McMillan, Jayshree Sarathy, Adam D. Smith, and Salil P. Vadhan. Differentially private simple linear regression. *CoRR*, abs/2007.05157, 2020. `arXiv:2007.05157`.

2 Rémi Bardenet and Odalric-Ambrym Maillard. Concentration inequalities for sampling without replacement. *Bernoulli*, 21(3):1361–1385, 2015.

3 Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. *CoRR*, abs/2106.02942, 2021. `arXiv:2106.02942`.

4 Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 87–96. ACM, 2013. doi:10.1145/2422436.2422449.

5 Jeremiah Blocki, Elena Grigorescu, and Tamalika Mukherjee. Privately estimating graph parameters in sublinear time. *CoRR*, abs/2202.05776, 2022. arXiv:2202.05776.

6 Christian Borgs, Jennifer T. Chayes, and Adam D. Smith. Private graphon estimation for sparse graphs. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1369–1377, 2015. URL: https://proceedings.neurips.cc/paper/2015/hash/7250eb93b3c18cc9daa29cf58af7a004-Abstract.html.

7 Christian Borgs, Jennifer T. Chayes, Adam D. Smith, and Ilias Zadik. Revealing network structure, confidentially: Improved rates for node-private graphon estimation. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 533–543. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00057.

8 Kamalika Chaudhuri and Staal A. Vinterbo. A stability-based validation procedure for differentially private machine learning. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2652–2660, 2013. URL: https://proceedings.neurips.cc/paper/2013/hash/e6d8545daa42d5ced125a4bf747b3688-Abstract.html.

9 Shixi Chen and Shuigeng Zhou. Recursive mechanism: towards node differential privacy and unrestricted joins. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 653–664. ACM, 2013. doi:10.1145/2463676.2465304.

10 Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear algorithms and lower bounds for metric TSP cost estimation. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 30:1–30:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.30.

11 Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. On estimating the average degree. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 795–806. ACM, 2014. doi:10.1145/2566486.2568019.

12 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. *Journal of Privacy and Confidentiality*, 7(3):17–51, May 2017.

13 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. doi:10.1561/0400000042.

14 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM J. Comput.*, 35(4):964–984, 2006. doi:10.1137/S0097539704447304.

15 Hendrik Fichtenberger, Monika Henzinger, and Wolfgang Ost. Differentially private algorithms for graphs under continual observation. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 42:1–42:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.42.

16 Johannes Gehrke, Edward Lui, and Rafael Pass. Towards privacy for social networks: A zero-knowledge based definition of privacy. In Yuval Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 432–449. Springer, 2011. doi:10.1007/978-3-642-19571-6_26.

17 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008. doi:10.1002/rsa.20203.

18 Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially private combinatorial optimization. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1106–1125. SIAM, 2010. doi:10.1137/1.9781611973075.90.

19 Michael Hay, Chao Li, Gerome Miklau, and David D. Jensen. Accurate estimation of the degree distribution of private networks. In Wei Wang, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu, editors, *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, pages 169–178. IEEE Computer Society, 2009. doi:10.1109/ICDM.2009.11.

20 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

21 Vishesh Karwa, Sofya Raskhodnikova, Adam D. Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *ACM Trans. Database Syst.*, 39(3):22:1–22:33, 2014. doi:10.1145/2611523.

22 Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. Analyzing graphs with node differential privacy. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 457–476. Springer, 2013. doi:10.1007/978-3-642-36594-2_26.

23 Wentian Lu and Gerome Miklau. Exponential random graph estimation under differential privacy. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 921–930. ACM, 2014. doi:10.1145/2623330.2623683.

24 Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.81.

25 Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. Smooth sensitivity and sampling in private data analysis. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 75–84. ACM, 2007. doi:10.1145/1250790.1250803.

26 Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1123–1131. SIAM, 2012. doi:10.1137/1.9781611973099.88.

27 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007. doi:10.1016/j.tcs.2007.04.040.

28 Sofya Raskhodnikova and Adam D. Smith. Efficient lipschitz extensions for high-dimensional graph statistics and node private degree distributions. *CoRR*, abs/1504.07912, 2015. arXiv:1504.07912.

29 Dana Ron. Sublinear-time algorithms for approximating graph parameters. In *Computing and Software Science*, volume 10000 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2019.

30 Adam Sealfon and Jonathan R. Ullman. Efficiently estimating erdos-renyi graphs with node differential privacy. *J. Priv. Confidentiality*, 11(1), 2021. doi:10.29012/jpc.745.

31 C. Seshadhri. A simpler sublinear algorithm for approximating the triangle count. *CoRR*, abs/1505.01927, 2015. arXiv:1505.01927.

32 Harry Sivasubramaniam, Haonan Li, and Xi He. Differentially private sublinear average degree approximation.

33 Shuang Song, Susan Little, Sanjay Mehta, Staal A. Vinterbo, and Kamalika Chaudhuri. Differentially private continual release of graph statistics. *CoRR*, abs/1809.02575, 2018. arXiv:1809.02575.

34 Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

35 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM J. Comput.*, 41(4):1074–1093, 2012. doi:10.1137/110828691.

36 Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Private release of graph statistics using ladder functions. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 731–745. ACM, 2015. doi:10.1145/2723372.2737785.

The Complexity of Finding Fair Many-To-One Matchings

Niclas Boehmer
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Tomohiro Koana
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

We analyze the (parameterized) computational complexity of "fair" variants of bipartite many-to-one matching, where each vertex from the "left" side is matched to exactly one vertex and each vertex from the "right" side may be matched to multiple vertices. We want to find a "fair" matching, in which each vertex from the right side is matched to a "fair" set of vertices. Assuming that each vertex from the left side has one color modeling its attribute, we study two fairness criteria. In one of them, we deem a vertex set fair if for any two colors, the difference between the numbers of their occurrences does not exceed a given threshold. Fairness is relevant when finding many-to-one matchings between students and colleges, voters and constituencies, and applicants and firms. Here colors may model sociodemographic attributes, party memberships, and qualifications, respectively.

We show that finding a fair many-to-one matching is NP-hard even for three colors and maximum degree five. Our main contribution is the design of fixed-parameter tractable algorithms with respect to the number of vertices on the right side. Our algorithms make use of a variety of techniques including color coding. At the core lie integer linear programs encoding Hall like conditions. To establish the correctness of our integer programs, we prove a new separation result, inspired by Frank's separation theorem [Frank, Discrete Math. 1982], which may also be of independent interest. We further obtain complete complexity dichotomies regarding the number of colors and the maximum degree of each side.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph theory, polynomial-time algorithms, NP-hardness, FPT, ILP, color coding, submodular and supermodular functions, algorithmic fairness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.27

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2206.06988`

Funding *Niclas Boehmer*: Supported by the Deutsche Forschungsgemeinschaft (DFG) projects MaMu (NI 369/19) and ComSoc-MPMS (NI 369/22).
Tomohiro Koana: Supported by the Deutsche Forschungsgemeinschaft (DFG) project DiPa (NI 369/21).

Acknowledgements We are grateful to the anonymous *ICALP 2022* reviewers for their thoughtful, constructive, and helpful comments.

1 Introduction

A many-to-one matching in a bipartite graph $G = (U \cup V, E)$ is an edge subset $M \subseteq E$ such that each vertex in U is incident to at most one edge in M. We study the computational complexity of finding a "fair" many-to-one matching and call this problem Fair Matching: Given a bipartite graph $G = (U \cup V, E)$ in which every vertex in U is colored, it asks for a many-to-one matching M such that for each $v \in V$ the vertices matched to v meet a

fairness criterion. In this work, we require that M is "left-perfect", i.e., every vertex in U is incident to exactly one edge in M. Using a slightly different formulation, Stoica et al. [24] recently studied this problem in terms of a fairness requirement derived from *margin of victory* (MoV). Generally, the margin of victory of a multiset is defined as the number of occurrences of the most frequent value minus the number of occurrences of the second most frequent value. (Given a set of colored vertices, we obtain a multiset from the occurrences of colors in it.) By requiring that the margin of victory of a set of colored vertices shall not exceed a given threshold, we prevent one color from becoming a dominating majority (see Stoica et al. [24] for a more extensive motivation of this concept). As an alternative simple fairness measure, we consider Max-Min, which is defined as the difference between the number of occurrences of the most frequent value and the number of occurrences of the least frequent value in a multiset. In a set of colored vertices with a small value of Max-Min all colors appear more or less equally often.

Which of MoV or Max-Min is more appropriate depends not only on the specific application (as discussed in the next two paragraphs) but also on the underlying data. Suppose that we have $2n$ red, $2n$ blue, and one green vertex. Then, it would be natural to deem a subset consisting of n red, n blue, and one green vertex fair (as it is in some sense the best we can hope for). Now, MoV seems to be a better fit because the MoV of the described subset is zero whereas the Max-Min value is $n-1$. In contrast, if there are $2n$ red, $2n$ blue, and $2n$ green vertices, then the same subset with n red, n blue, and one green vertex should be considered as unfair, rendering Max-Min more suitable for this color distribution than MoV. In general, Max-Min seems to be a natural choice for homogeneous data. The first example illustrates, however, that in some scenarios, MoV may serve as a viable relaxation of Max-Min.[1]

A notable application of Fair Matching emerges in the context of district-based elections. In such elections, voters (modeled by vertices in U) are divided into constituencies (modeled by vertices in V), and then each constituency elects its own representative. Here, colors can represent various attributes. For instance, colors may represent political standings. A small margin of victory is particularly desirable in this case because it will lead to close elections, holding politicians accountable for their job. One could also strive for "fair" representation of different ethnic groups or age groups by modeling ethnicity or age with colors. Other applications include the assignment of school children to schools (where colors may model sociodemographic attributes) or the assignment of reviewers to academic papers (where colors may model the level of expertise or academic background of reviewers).

Similar fairness considerations also arise in modern online systems (see, e.g., [23] for a survey). For instance, fairness is a pressing issue to counter targeted advertising or to improve recommender systems. Here one task is to ensure that the content (each perhaps represented by multiple vertices in U) falling into different categories (colors) is assigned to users (vertices in V) in a way that each user is presented with a "diverse" selection of content. Lastly, we mention that (Max-Min) Fair Matching has also applications outside of the "fairness" context: Imagine a centralized job market for companies (vertices in V) and applicants (vertices in U), each having a specific skillset (color). Firms may wish to balance between applicants with different skillsets so that employees with various skillsets may be placed into teams. For instance, it may be desirable for a software company to hire roughly the same number of frontend and backend developers.

[1] Note that the Max-Min value is at least the MoV value for any multiset.

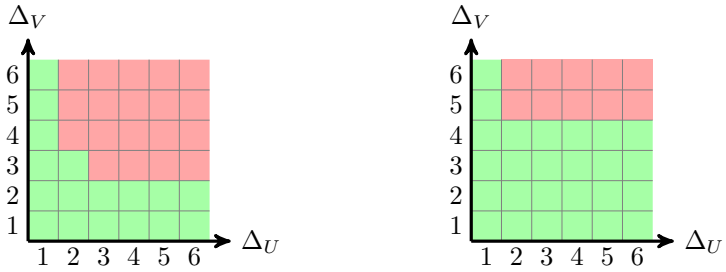


Figure 1 The complexity landscape of Max-Min Fair Matching (left) and MoV Fair Matching (right) for the maximum degree Δ_U (resp., Δ_V) in U (resp., V). Green denotes polynomial-time solvability and red denotes NP-hardness. All NP-hardness results hold for three colors.

Our Contributions. We perform a refined complexity analysis of the NP-hard Max-Min Fair Matching and MoV Fair Matching problems in terms of the size k of V, the number $|C|$ of colors, and the maximum degree Δ_U and Δ_V among vertices in U and V, respectively. Our main contribution are arguably involved FPT algorithms for the parameterization k (Section 4). At the heart of the design of our algorithms lies an integer linear program (ILP) of bounded dimension. We essentially determine whether Hall-like conditions that guarantee the existence of a fair matching are fulfilled by formulating these in a system of linear inequalities. In order to establish the correctness of our ILP formulations, we prove what we call *touching separation theorem*, getting inspiration from Frank's separation theorem on submodular and supermodular functions [14]. For MoV Fair Matching, we apply our approach in conjunction with the color coding technique [5]. To familiarize ourselves with the ideas underlying our ILPs, in Section 3, we start with a warm-up where we present ILP-based fixed-parameter tractable algorithms for the larger parameter $k + |C|$. To sum up, as it is straightforward to see that Max-Min/MoV Fair Matching are FPT with respect to the size n of U[2], we establish the fixed-parameter tractability of both problems for the two natural parameters n and k.[3] We then in Section 5 study the computational complexity of Max-Min/MoV Fair Matching with respect to Δ_U, Δ_V, and $|C|$. We show that Max-Min/MoV Fair Matching is polynomial-time solvable for $|C| = 2$ and that it becomes NP-hard for $|C| \geq 3$. Moreover, we settle all questions concerning the problems' classical complexity in terms of Δ_U and Δ_V, revealing a complete complexity landscape in this regard (see Figure 1). Finally, in Section 6, we show that Max-Min/MoV Fair Matching are linear-time solvable when every vertex in U can be matched to any vertex in V. Notably, all our algorithmic results hold even if we require that each vertex from V is matched to at least one vertex from U. This further constraint may appear when we need to divide the vertices into exactly k non-empty fair subsets. Although this constraint is seemingly simple, sometimes (e.g., in our FPT algorithm for Max-Min Fair Matching for k) non-trivial adaptions are needed.

Related Work. Stoica et al. [24] introduced three problems where the task is to partition a set of colored vertices into subsets with a small margin of victory satisfying some global size constraints. Among these three, the most general is Fair Connected Regrouping,

[2] We can enumerate all fair subsets of U in $2^n \cdot (n+k)^{O(1)}$ time. Then, Knapsack-like dynamic programming solves Fair Matching in $3^n \cdot (n+k)^{O(1)}$ time.

[3] In most described applications k is typically quite small and much smaller than n. For instance, Stoica et al. [24] performed some experiments for MoV Fair Matching to assign voters to districts with $n = 50,000$ and $k = 10$, and to assign students to schools with $n = 41,834$ and $k = 61$.

where one is given a vertex-colored graph G, an integer k, and a function that determines for each vertex which subsets it can be part of. The task is then to find a partitioning of G into k fair districts (i.e., connected components). In a follow-up work, Boehmer et al. [9] analyzed how the structure of G influences the (parameterized) computational complexity of Fair Connected Regrouping. The other two problems Stoica et al. [24] considered are special cases of Fair Connected Regrouping: One is Fair Regrouping where the connectivity constraints are dropped (corresponding to MoV Fair Matching). The other is Fair Regrouping_X, which is a special case of Fair Regrouping, where any vertex can belong to any district (corresponding to MoV Fair Matching on complete bipartite graphs; we study this special case in Section 6). They proved that Fair Regrouping is NP-hard for three colors (without any constraints on the degree of the graph) and is XP with respect to the number k of districts (i.e., polynomial-time solvable for constant k). They also showed that Fair Regrouping_X is XP with respect to the number of colors.

Coming back to Fair Matching as a matching problem, Ahmed et al. [4] proposed a global supermodular objective to model the fairness (which they call diversity) of a bipartite weighted many-to-many matching and developed a polynomial-time greedy heuristic for it. Ahmadi et al. [1] extended the work of Ahmed et al. [4] by generalizing the problem to the case where vertices can have multiple different colors and presented a pseudo-polynomial-time algorithm for it. Moreover, Dickerson et al. [13] applied this formulation of fairness to an online setting where vertices from the left side arrive over time and Ahmed et al. [3] applied it to the task of forming teams.

Fairness is also a popular topic when finding a stable many-to-one matching of vertices that have preferences over each other. Here, fairness constraints are typically modeled by imposing for each vertex from the right side certain lower and upper bounds on the number of vertices of each color that can be matched to it [7, 8, 10, 16, 18].

More broadly speaking, fairness has recently also been frequently applied to a variety of different problems from the area of combinatorial optimization. For instance, in the context of the Knapsack [22] or Maximum Coverage [6] problem, fairness means that all types are represented equally in the selected solution. For clustering, each cluster is considered fair when each type accounts for a certain fraction of vertices in it [2, 15].

The proof (or their completion) of all results marked by ($\star$) are omitted.

2 Preliminaries

For two integers $i < j \in \mathbb{N}$, let $[i, j] = \{i, i+1, \dots, j-1, j\}$ and let $[i] = [1, i]$. For a set S and an element $x \in S$, we sometimes write $S - x$ to denote $S \setminus \{x\}$.

Let $G = (U \cupdot V, E)$ be a bipartite graph, where U is the *left side* and V is the *right side* of G. Let $n := |U|$ and $k := |V|$ be the number of vertices in the left side and right side, respectively. For a vertex $w \in U \cupdot V$ and an edge set $M \subseteq E$, let $M(w)$ be the set of vertices *matched to* w in M, i.e., $M(w) = \{w' \in U \cupdot V \mid \{w, w'\} \in M\}$. We say that $M \subseteq E$ is a *many-to-one matching* in G if $|M(u)| \leq 1$ for every $u \in U$. A many-to-one matching M is *left-perfect* if $|M(u)| = 1$ for every $u \in U$. Note that we require M to be left-perfect as otherwise an empty set would constitute a trivial solution for our problem. When clear from context, we refer to a left-perfect many-to-one matching as a matching. For a vertex $w \in U \cupdot V$, let $N_G(w)$ be the set of its neighbors in G, i.e., $N_G(w) = \{w' \in U \cupdot V \mid \{w, w'\} \in E\}$. For $W \subseteq U \cupdot V$, let $N_G(W) = \bigcup_{w \in W} N_G(w)$ be the joint neighborhood of vertices from W and let $\nu_G(W) = \{w' \in U \cupdot V \mid N_G(w') \subseteq W\}$ be the set of vertices which are only adjacent to vertices in W. We drop the subscript $\cdot_G$ when it is clear from context.

Let C be the set of colors and let $\mathrm{col}\colon U \to C$ be a function that assigns a color to every vertex of U. For $U' \subseteq U$, let $U'_c \subseteq U'$ be the set of vertices $u \in U'$ of color c. For instance, given a matching M and a vertex $v \in V$, $M(v)_c$ denotes the set of vertices matched to v in M that have color c. We denote by $G_c = G[U_c \cup V]$ the graph G restricted to vertices from $U_c \cup V$. We use the shorthand $N_c(W)$ (resp., $\nu_c(W)$) for $N_{G_c}(W)$ (resp., $\nu_{G_c}(W)$).

Throughout the paper, we assume that the set C of colors is equipped with some linear order $\leq_C$, which serves as a tie breaker. So $\arg\max$ over C is well-defined. We write $\max^1$ for max and $\max^2$ for the second largest element.

We now define our two fairness measures. For a subset of vertices $U' \subseteq U$, let $\mathrm{MoV}(U') := \max^1_{c\in C} |U'_c| - \max^2_{c\in C} |U'_c|$ be the difference between the number of occurrences of the most and second most frequent color in U'. Similarly, for a subset of vertices $U' \subseteq U$, let $\mathrm{MmM}(U') := \max_{c\in C} |U'_c| - \min_{c\in C} |U'_c|$ be the difference between the number of occurrences of the most and least frequent color in U'. A subset of vertices $U' \subseteq U$ is ℓ-fair according to MoV (resp., Max-Min) if $\mathrm{MoV}(U') \leq \ell$ (resp., $\mathrm{MmM}(U') \leq \ell$).[4] A many-to-one matching M in G is ℓ-fair according to MoV (resp., Max-Min) if $M(v)$ is ℓ-fair according to MoV (resp., Max-Min) for all $v \in V$. The considered fairness notion will always be clear from context. We now define our central problem Π Fair Matching for some fairness measure Π:

Π Fair Matching	
Input:	A bipartite graph $G = (U \cup V, E)$, a set C of colors, a function $\mathrm{col}\colon U \to C$, and an integer $\ell \in \mathbb{N}$.
Question:	Is there a left-perfect many-to-one matching $M \subseteq E$ which is ℓ-fair according to the fairness measure Π?

We also sometimes consider Π Fair Matching with size constraints where additionally given two integers p and q, we require that the matching M to be found satisfies $p \leq |M(v)| \leq q$ for all $v \in V$. We refer to the case with $p = 1$ and $q = n$ as the non-emptiness constraint. This constraint is arguably crucial for some applications when we want to partition the vertices in the left side into exactly k non-empty subsets.

Let $\mathcal{I}$ be an instance of some problem and let $\mathcal{P}(\mathcal{I})$ be an integer linear program (ILP) constructed from $\mathcal{I}$. We say that $\mathcal{P}$ is *complete* if $\mathcal{P}(\mathcal{I})$ is feasible whenever $\mathcal{I}$ is a yes-instance. Conversely, we say that $\mathcal{P}$ is *sound* if $\mathcal{I}$ is a yes-instance whenever $\mathcal{P}(\mathcal{I})$ is feasible. In this work, we will make use of Lenstra's algorithm [19, 20] that decides whether an ILP of size L with p variables is feasible using $O(p^{2.5p+o(p)} \cdot |L|)$ arithmetic operations.

We assume that the reader is familiar with basic concepts in parameterized complexity (see for instance [12]). As a reminder, an FPT algorithm for a parameter k is an algorithm whose running time on input $\mathcal{I}$ is $f(k) \cdot |\mathcal{I}|^{O(1)}$ for some computable function f.

3 Warmup: FPT Algorithms for $k + |C|$

We prove that both Fair Matching problems are fixed-parameter tractable with respect to $k+|C|$: We present an integer linear programming (ILP) formulation of these problems whose number of variables is bounded in a function of the parameter $k+|C|$ and subsequently employ

[4] Notably, the definition of margin of victory of Stoica et al. [24] differs slightly from ours in that in their definition sets of vertices where the two most frequent colors have the same number of occurrences have a margin of victory of one (and not of zero). We chose our definition in accordance with Boehmer et al. [9] to be able to distinguish a tie between two colors from one color being one vertex ahead of another.

Lenstra's algorithm [19, 20]. Notably, one can upper-bound the number of "types" (according to their neighborhoods and colors) of vertices in U by $2^k \cdot |C|$. Using this observation, it is straightforward to give an ILP formulation of Max-Min/MoV Fair Matching using $O(2^k \cdot |C|)$ variables. Instead, we follow a theoretically more involved but more efficient approach. For this, we use a structural property of our problem related to Hall's theorem, which decreases the number of variables in our ILP to $O(|C| \cdot k)$. We reuse some of the results from this section in Section 4, where we prove that Max-Min/MoV Fair Matching are actually fixed-parameter tractable with respect to k.

To prove that the ILP we present in the following is complete, we use the following:

▶ **Lemma 1** (⋆). *Let $G = (U \cup V, E)$ be a bipartite graph and let M be a left-perfect many-to-one matching. Then, $|\nu(W)| \leq \sum_{v \in W} |M(v)| \leq |N(W)|$ for every $W \subseteq V$.*

For proving that our ILP is sound, we use the following:

▶ **Lemma 2.** *Let $G = (U \cup V, E)$ be a bipartite graph and let $\{z_v \in \mathbb{N} \mid v \in V\}$ be a set of integers. Suppose that $\sum_{v \in W} z_v \geq |\nu_G(W)|$ for every $W \subseteq V$ and that $\sum_{v \in V} z_v = |U|$. Then, there is a left-perfect many-to-one matching M such that $|M(v)| = z_v$ for every $v \in V$.*

Proof. Assume that the conditions stated in the lemma hold. We prove the existence of such a matching M by making use of Hall's theorem [17]. To do so, we introduce an auxiliary bipartite graph G' as follows: In G', the vertices on one of the two sides are the vertices from U. The vertices on the other side are $V' := \bigcup_{v \in V} Z_v$, where Z_v is a set of z_v vertices. There is an edge between $u \in U$ and $v' \in Z_v \subseteq V'$ if and only if $\{u, v\} \in E$. In order to apply Hall's theorem, we show that $|U'| \leq |N_{G'}(U')|$ for every $U' \subseteq U$.

Fix some $U' \subseteq U$ and let $W' = N_G(U')$. The construction of G' gives us $|N_{G'}(U')| = \sum_{v \in W'} z_v$. By the assumption of the lemma, we have $\sum_{v \in W'} z_v \geq |\nu_G(W')|$. Moreover, we have $\nu_G(W') = \nu_G(N_G(U')) \supseteq U'$. Consequently, we obtain $|N_{G'}(U')| = \sum_{v \in W'} z_v \geq |\nu_G(W')| \geq |U'|$. Hall's theorem then implies that G' admits a one-to-one matching M' which matches all vertices from U. In fact, M' is a perfect matching since $|V'| = \sum_{v \in V} z_v = |U|$ by our assumption. Now consider the matching M in G where a vertex $u \in U$ is matched to $v \in V$ if u is matched to a vertex from Z_v in M'. Then, M is a left-perfect many-to-one matching with $|M(v)| = z_v$ for every $v \in V$. ◀

Using Lemmas 1 and 2, we give an ILP formulation of Fair Matching with $O(|C| \cdot k)$ variables.

ILP formulation. Introduce a variable $z_v^c \in \mathbb{N}$ for every $v \in V$ and every $c \in C$. The variable z_v^c represents the number of vertices in U of color c that are matched to v. Suppose that the given instance admits a left-perfect ℓ-fair many-to-one matching M respecting the values of z_v^c. Then, for each $c \in C$, there is a matching M_c in G_c with $|M_c(v)| = z_v^c$ for $v \in V$ and $|M_c(u)| = 1$ for $u \in U_c$. As shown in Lemma 1, from this one can conclude that the following constraints must be fulfilled:

$$|\nu_c(W)| \leq \sum_{v \in W} z_v^c \leq |N_c(W)| \text{ for all } W \subseteq V, c \in C.$$

Next, we encode the fairness requirement. For Max-Min, we need to have that for every pair of colors the number of vertices of these two colors assigned to some vertex $v \in V$ differ by at most ℓ. Thus, we add the constraint

$$z_v^{c'} - z_v^c \leq \ell \text{ for all } v \in V \text{ and } c, c' \in C.$$

To model ℓ-fairness for MoV Fair Matching, we introduce two new binary variables $a_v^c, b_v^c \in \{0,1\}$ for all $v \in V$ and $c \in C$. Informally speaking, the intended meaning of these variables is that $a_v^c = 1$ (resp., $b_v^c = 1$) when c is the most (resp., second most) frequent color among vertices matched to v. We ensure that the values of these variables are set accordingly as follows:

$$z_v^c - z_v^{c'} \geq n(a_v^c + b_v^{c'} - 2), \quad z_v^c - z_v^{c'} \geq n(b_v^{c'} - a_v^c - 1), \text{ and } a_v^c + b_v^c \leq 1 \quad \forall v \in V, c, c' \in C;$$
$$\sum_{c \in C} a_v^c = \sum_{c \in C} b_v^c = 1 \quad \forall v \in V.$$

For the first constraint, note that it becomes $z_v^c \geq z_v^{c'}$ if $a_v^c = b_v^{c'} = 1$ and that it is always fulfilled otherwise. Similarly for the second constraint, observe that it becomes $z_v^c \geq z_v^{c'}$ if $a_v^{c'} = 0$ and $b_v^c = 1$ and that it is always fulfilled otherwise.

Finally, using a_v^c and b_v^c (and their meaning as proven above), we add the following constraint that encodes the ℓ-fairness in terms of margin of victory of $M(v)$ for all $v \in V$:

$$z_v^c - z_v^{c'} - n(2 - a_v^c - b_v^{c'}) \leq \ell \qquad \forall v \in V$$

Lastly, we can also add linear constraints ensuring that the number of vertices from U matched to each vertex $v \in V$ is between p and q: $p \leq \sum_{c \in C} z_v^c \leq q$ for all $v \in V$.

▶ **Theorem 3.** *Max-Min/MoV Fair Matching with arbitrary size constraints can be solved in* $O^\star((|C| \cdot k)^{O(|C| \cdot k)})$ *time.*

Proof. We show that an instance $(G = (U \cup V, E), C, \mathrm{col}, \ell, p, q)$ of Max-Min/MoV Fair Matching with size constraints admits an ℓ-fair left-perfect many-to-one matching if and only if the constructed ILP constructed is feasible. As described above, if the given instance is a yes-instance, there is an assignment to z_v^c satisfying all the constraints. Conversely, suppose that the ILP admits a solution $\{z_v^c \mid v \in V, c \in C\}$. Then, from our first set of constraints, we have for every $c \in C$ that $\sum_{v \in W} z_v^c \geq |\nu_c(W)|$ for every $W \subseteq V$ and $\sum_{v \in V} z_v^c = |U_c|$ (the later part follows from our first set of constraints for $W = V$). By Lemma 2, it follows that G_c has a matching M_c in which every vertex in U_c is matched and the values of z_v^c for $v \in V$ and $c \in C$ are respected. Aggregating M_c for all colors c yields a left-perfect ℓ-fair many-to-one matching for G respecting the given size constraints. Using Lenstra's algorithm [19, 20], the feasibility of the ILP can be determined in the claimed time. ◀

4 FPT Algorithms for k

In this section, we develop FPT algorithms for Max-Min/MoV Fair Matching for the parameterization $|V| = k$. We start with a discussion of the challenges for our algorithms. Afterwards, we obtain a new result on submodular and supermodular functions. Using this, in Section 4.1 we present the algorithm for Max-Min Fair Matching, and in Section 4.2 the slightly more involved algorithm for MoV Fair Matching.

The crux of our algorithms is an ILP as in Section 3. However, since we look into the parameterization without $|C|$, it would be too costly to introduce variables for each color. To illustrate our idea to work around this issue, take Max-Min Fair Matching as an example. One of the straightforward ideas how to formulate this problem as an ILP would be to introduce two variables $x_v \leq y_v$ for every vertex $v \in V$, where x_v (resp., y_v) encodes the minimum (resp., maximum) number of vertices of some color $c \in C$ matched to v. Informally speaking, to encode the ILP constraints from Section 3, we could now replace every occurrence of z_v^c with x_v (resp., y_v) if the constraint in which z_v^c occurs imposes an

upper (resp., lower) bound on z_v^c: the first set of constraints added in Section 3 translates to $\sum_{v\in W} y_v \geq \max_{c\in C} |\nu_c(W)|$ and $\sum_{v\in W} x_v \leq \min_{c\in C} |N_c(W)|$ for every $W \subseteq V$. Moreover, we add the constraint $y_v - x_v \leq \ell$ for all $v \in V$. Let $\mathcal{P}$ denote the ILP obtained this way. (See Section 4.1 for the formal construction of $\mathcal{P}$.) Although it is easy to see that $\mathcal{P}$ is complete, it turns out to be nontrivial to prove its soundness. To highlight this challenge, suppose that $\mathcal{P}$ is feasible for $\{x_v, y_v \mid v \in V\}$. To show that this implies that there is an ℓ-fair matching M, we have to show that for every color $c \in C$, the graph G_c has a matching M_c such that $x_v \leq |M_c(v)| \leq y_v$ for all $v \in V$. Unfortunately, we cannot directly apply Lemma 2 (as we were able to do in Section 3): To construct an ℓ-fair matching from $\{x_v, y_v \mid v \in V\}$ using Lemma 2, we have to show that for every color $c \in C$, there always exists a set of integers $\{z_v^c \mid v \in V\}$ with the following properties:

- **(i)** $\sum_{v\in W} z_v^c \geq |\nu_c(W)|$, $\forall W \subseteq V$
- **(ii)** $\sum_{v\in V} z_v^c = |U_c|$
- **(iii)** $x_v \leq z_v^c \leq y_v$, $\forall v \in V$.

Finding an assignment of variables z_v^c that fulfill (i) and (iii) is trivial; setting $z_v^c = y_v$ suffices because $\mathcal{P}$ dictates that $\sum_{v\in W} y_v \geq \max_{c'\in C} |\nu_{c'}(W)| \geq |\nu_c(W)|$ for every $W \subseteq V$. However, integers satisfying (i), (ii), and (iii) simultaneously are not trivially guaranteed to exist. To nevertheless prove their existence, we prove what we call the *touching separation theorem*, inspired by Frank's separation theorem [14] on submodular and supermodular functions. Our theorem implies that if there is a solution to $\mathcal{P}$, then there is an assignment of variables z_v^c satisfying (i), (ii), and (iii). Submodular and supermodular functions are defined as follows:

▶ **Definition 4.** *Let $f\colon 2^S \to \mathbb{N}$ be a set function over a set S. We say that f is submodular if $f(X)+f(Y) \geq f(X\cup Y)+f(X\cap Y)$ for every $X, Y \subseteq S$ and supermodular if $f(X)+f(Y) \leq f(X \cup Y) + f(X \cap Y)$ for every $X, Y \subseteq S$. Moreover, f is modular if f is both submodular and supermodular.*

Modular functions admit a simpler characterization, which will be useful in several proofs:

▶ **Lemma 5** (folklore, ⋆). *Let $f\colon 2^S \to \mathbb{N}$ be a set function. Then, f is modular if and only if $f(X) = f(\emptyset) + \sum_{x\in X}(f(x) - f(\emptyset))$ for every $X \subseteq S$.*

Frank's separation theorem [14] can be stated as follows: If $f\colon 2^S \to \mathbb{N}$ is submodular and $g\colon 2^S \to \mathbb{N}$ is supermodular, and $g(X) \leq f(X)$ for every $X \subseteq S$, then there exists a modular set function $h\colon 2^S \to \mathbb{N}$ such that $g(X) \leq h(X) \leq f(X)$ for every $X \subseteq S$. We prove an analogous separation theorem where the "lower-bound" function is the maximum of two functions – one being modular and the other being supermodular – and the "upper-bound" function is a modular function. (In general, the "lower-bound" function arising this way is neither submodular nor supermodular.) We additionally require that the function separating the two functions "touches" the lower-bound function. Our proof uses a rather involved induction on the size of S.

▶ **Theorem 6** (Touching separation theorem, ⋆). *Let $f, f'\colon 2^S \to \mathbb{N}$ be two modular set functions and let $g\colon 2^S \to \mathbb{N}$ be a supermodular set function. Suppose that $f(\emptyset) = f'(\emptyset) = g(\emptyset) = 0$ and $\max(f(X), g(X)) \leq f'(X)$ for every $X \subseteq S$. Then, there is a modular set function $h\colon 2^S \to \mathbb{N}$ such that $\max(f(X), g(X)) \leq h(X) \leq f'(X)$ for every $X \subseteq S$ and $h(S) = \max_{X\subseteq S} f(X) + g(S \setminus X)$.*

To apply Theorem 6, we use a supermodular function arising from a bipartite graph:

▶ **Lemma 7** (⋆). *Let $G = (U \uplus V, E)$ be a bipartite graph and let $g\colon 2^V \to \mathbb{N}$ be a set function such that $g(W) = |\nu(W)|$ for each $W \subseteq V$. Then, g is supermodular.*

4.1 Max-Min Fair Matching

We now present an FPT algorithm for Max-Min Fair Matching. As mentioned, our algorithm, which follows a similar approach as used in Section 3, builds an ILP. The difference is that the ILP constructed here involves $O(k)$ and not $\Theta(k \cdot |C|)$ variables. We prove the correctness of the ILP in Theorem 8, crucially relying on Theorem 6.

ILP formulation. We add two variables $x_v \leq y_v \in \mathbb{N}$ for every $v \in V$. The intended meaning for these variables is that for every color $c \in C$, the number of vertices from U_c matched to v is between x_v and y_v. We obtain the following constraints from Lemma 1:

$$\sum_{v \in W} y_v \geq \max_{c \in C} |\nu_c(W)| \text{ and } \sum_{v \in W} x_v \leq \min_{c \in C} |N_c(W)| \quad \forall W \subseteq V. \tag{1}$$

To encode ℓ-fairness, we add: $y_v - x_v \leq \ell$ for all $v \in V$.

▶ **Theorem 8.** *Max-Min Fair Matching can be solved in $O^\star(k^{O(k)})$ time.*

Proof. Our ILP uses $O(k)$ variables and $O(2^k)$ constraints; the construction takes $O^\star(2^k)$ time. Using Lenstra's algorithm [19, 20], it is possible to check whether the constructed ILP is feasible in $O^\star(k^{O(k)})$ time. It remains to prove the correctness of our ILP.

The completeness of our ILP follows from Lemma 1. For the soundness, suppose that $\{x_v, y_v \mid v \in V\}$ is a feasible solution for the ILP. For every color $c \in C$, we show the following: For the set $U_c \subseteq U$ of vertices of color c, there is a matching M_c in the bipartite graph $G_c = G[U_c \cup V]$ such that $x_v \leq |M_c(v)| \leq y_v$ for each $v \in V$, from which the soundness of the ILP directly follows. To show the existence of such a matching M_c, we will rely on Lemma 2. Note, however, that Lemma 2 asks for integers $\{z_v \mid v \in V\}$ meeting certain constraints, which we cannot choose arbitrarily as we have to respect the constraint $x_v \leq z_v \leq y_v$ for every $v \in V$. Let us fix some color $c \in C$. We find integers $\{z_v \mid v \in V\}$ via Theorem 6:

Let $f, f', g \colon 2^V \to \mathbb{N}$ be set functions such that $f(W) = \sum_{v \in W} x_v$ and $f'(W) = \sum_{v \in W} y_v$, and $g(W) = |\nu_c(W)|$ for each $W \subseteq V$. Note that f and f' are modular by Lemma 5 and that g is supermodular by Lemma 7. The constraints in the ILP ensure that $\max(f(W), g(W)) \leq f'(W)$ for every $W \subseteq V$. Consequently, Theorem 6 yields a set modular function $h \colon 2^V \to \mathbb{N}$ such that $\max(f(W), g(W)) \leq h(W) \leq f'(W)$ for each $W \subseteq V$ and $h(V) = \max_{W \subseteq V} f(W) + g(V \setminus W)$. Let $z_v = h(v)$ for every $v \in V$. Note that as h is modular and fulfills the constraints stated above, $\sum_{v \in W} z_v = h(W) \geq g(W) = |\nu_c(W)|$ for each $W \subseteq V$. Hence, in order to apply Lemma 2 on the integers $\{z_v \mid v \in V\}$, it remains to show that $\sum_{v \in V} z_v = h(V) = |U_c|$. Since our ILP requires that $f(W) = \sum_{v \in W} x_v \leq |N_c(W)|$ for $W \subseteq V$, we have that

$$\begin{aligned} f(W) + g(V \setminus W) &\leq |N_c(W)| + |\nu_c(V \setminus W)| \\ &= |\{u \in U_c \mid N_c(u) \cap W \neq \emptyset\}| + |\{u \in U_c \mid N_c(u) \subseteq V \setminus W\}| = |U_c|. \end{aligned}$$

Moreover, we have $f(\emptyset) + g(V) = |U_c|$, resulting in $h(V) = \max_{W \subseteq V} f(W) + g(V \setminus W) = |U_c|$ by Theorem 6. Therefore, the graph G_c admits a matching M_c with $x_v \leq |M_c(v)| \leq y_v$ for each $v \in V$ by Lemma 2. Combining these matchings yields an ℓ-fair matching in G. ◀

Non-emptiness constraint. For a matching M found by our algorithm, we may have $M(v) = \emptyset$ for some $v \in V$. So the non-emptiness constraint may be violated. Unfortunately, there is seemingly no simple linear constraint to ensure that $M(v) \neq \emptyset$ for each $v \in V$.[5]

[5] Adding $x_v > 0$ ensures that there is at least one vertex of each color matched to v. Adding $y_v > 0$ is not enough, as the ILP only ensures that for each color *at most* y_v vertices are matched to v.

Overcoming this challenge, we now develop an FPT algorithm for Max-Min Fair Matching with the non-emptiness constraint. We will build upon the ILP formulation for Theorem 8. First, observe that for $\ell = 0$, it suffices to add the constraint $y_v > 0$ for all $v \in V$ because this ensures that every vertex $v \in V$ is matched to at least $x_v = y_v > 0$ vertices of each color.

For $\ell > 0$, we develop a more involved algorithm. We will only describe its main ideas here. Suppose that there is an ℓ-fair many-to-one matching M such that $M(v) \neq \emptyset$ for each $v \in V$. By choosing an arbitrary element u_v of $M(v)$ for each $v \in V$, we obtain a matching $M^\star := \{\{u_v, v\} \mid v \in V\} \subseteq M$ with $|M^\star(v)| = 1$ for each $v \in V$. Since there are possibly $n^{\Omega(k)}$ choices for $M^\star$, we cannot assume that $M^\star$ is given. Instead, our algorithm only "guesses" some structural properties of $M^\star$, which we will incorporate into the above ILP for Max-Min Fair Matching by adding further constraints. (Our algorithm avoids guessing objects marked with a star.)

For each $v \in V$, let $\chi(v)$ be the color of $M^\star(v)$. By iterating over all possible partitions $\mathcal{V}$ of V, we first guess a partition of V according to $\chi(v)$, i.e., two vertices $v, v' \in V$ belong to the same subset of $\mathcal{V}$ if and only if $\chi(v) = \chi(v')$. For each $S \in \mathcal{V}$, let $\chi(S)$ be the color of $\chi(v)$ for an arbitrary vertex $v \in S$. Let us fix some $S \in \mathcal{V}$. We will formulate constraints that must be fulfilled when each vertex v in S is matched to $M^\star(v)$ (which has color $\chi(S)$). For this, let $U_S^\star \subseteq U_{\chi(S)}$ be the set of vertices from U of color $\chi(S)$ incident to an edge in $M^\star$ and let $G_S^\star$ be the graph obtained from $G_{\chi(S)}$ by deleting all edges incident to $U_S^\star$ and then adding the edges of $M^\star$ whose endpoint on the left side has color $\chi(S)$. Since every edge in M with an endpoint of color $\chi(S)$ is present in $G_S^\star$, these edges form a left-perfect many-to-one matching in $G_S^\star$. Thus, by Lemma 1, we should have $\sum_{v \in W} y_v \geq |\nu_{G_S^\star}(W)|$ and $\sum_{v \in W} x_v \leq |N_{G_S^\star}(W)|$ for each $W \subseteq V$ if there is a left-perfect matching containing $M^\star$ in $G_S^\star$. We need to evaluate $|\nu_{G_S^\star}(W)|$ and $|N_{G_S^\star}(W)|$ to include these constraints into our ILP. Note, however, that we cannot compute these values without $M^\star$ given. In the following, we explain how we can nevertheless incorporate these constraints by guessing further structural aspects of $M^\star$.

First, we rewrite $|\nu_{G_S^\star}(W)|$ and $|N_{G_S^\star}(W)|$ as follows:

$$|\nu_{G_S^\star}(W)| = |\nu_{\chi(S)}(W)| + |\{v \in S \cap W \mid N_{G_{\chi(S)}}(M^\star(v)) \not\subseteq W\}| \text{ and}$$
$$|N_{G_S^\star}(W)| = |N_{\chi(S)}(W)| - |\{v \in S \setminus W \mid N_{G_{\chi(S)}}(M^\star(v)) \cap W \neq \emptyset\}|.$$

To see why the first equation holds, observe that we have only deleted edges when constructing $G_S^\star$ from $G_{\chi(S)}$. Thus, we have $\nu_{G_{\chi(S)}}(W) \subseteq \nu_{G_S^\star}(W)$. Moreover, we have $u \in \nu_{G_S^\star}(W) \setminus \nu_{G_{\chi(S)}}(W)$ if and only if $u = M^\star(v)$ for some $v \in S \cap W$ (which implies that u is only adjacent to v in $G_S^\star$) and u has a neighbor outside of W in $G_{\chi(S)}$. The second equation follows similarly: As we only deleted edges, we have $N_{G_S^\star}(W) \subseteq N_{G_{\chi(S)}}(W)$. We also have $u \in N_{G_{\chi(S)}}(W) \setminus N_{G_S^\star}(W)$ if and only if $u = M^\star(v)$ for some $v \in S \setminus W$ and u has a neighbor in W in $G_{\chi(S)}$. To evaluate the second term in each of these equations, we guess a function $\mu \colon V \to 2^V$ such that $\mu(v) = N_{G_{\chi(S)}}(M^*(v))$ for each $v \in V$. For $\alpha(S, W) := |\{v \in S \cap W \mid \mu(v) \not\subseteq W\}|$ and $\beta(S, W) := |\{v \in S \setminus W \mid \mu(v) \cap W \neq \emptyset\}|$, we then have:

$$|\nu_{G_S^\star}(W)| = |\nu_{\chi(S)}(W)| + \alpha(S, W) \text{ and } |N_{G_S^\star}(W)| = |N_{\chi(S)}(W)| - \beta(S, W). \tag{2}$$

To incorporate the constraints $\sum_{v \in W} y_v \geq |\nu_{G_S^\star}(W)|$ and $\sum_{v \in W} x_v \leq |N_{G_S^\star}(W)|$, it remains to deal with $|\nu_{\chi(S)}(W)|$ and $|N_{\chi(S)}(W)|$. Note that we cannot compute $|\nu_{\chi(S)}(W)|$ or $|N_{\chi(S)}(W)|$ without $M^\star$ given. Moreover, it would be costly to guess these values (which can be $\Omega(n)$) for every $S \in \mathcal{V}$ and $W \subseteq V$ or the values of $\chi(S)$ (which can

be $\Omega(|C|)$) for every $S \in \mathcal{V}$. Instead, we relate these two values to $\max_{c\in C} |\nu_c(W)|$ and $\min_{c\in C} |N_c(W)|$ by guessing their respective differences. Although these differences may be $\Omega(n)$, we discover that we can cap them at k: When they are larger, then the arising constraints are already covered by Constraint (1) from the original ILP. Formally, we guess $X(S,W) = \min(|N_{\chi(S)}(W)| - \min_{c\in C} |N_c(W)|, k)$ and $Y(S,W) = \min(\max_{c\in C} |\nu_c(W)| - |\nu_{\chi(S)}(W)|, k)$ for each $S \in \mathcal{V}$ and $W \subseteq V$ by iterating over all possible $X, Y \colon \mathcal{V} \times 2^V \to \{0, \dots, k\}$ (for each of them we have at most $(k+1)^{k\cdot 2^k} \in O(2^{2^{O(k)}})$ choices).

To account for the constraints $\sum_{v\in W} y_v \geq |\nu_{G^\star_S}(W)|$ and $\sum_{v\in W} x_v \leq |N_{G^\star_S}(W)|$, we now add the following constraint to the ILP for each $S \in \mathcal{V}$ and $W \subseteq V$:

$$\sum_{v\in W} y_v \geq \max_{c\in C} |\nu_c(W)| - Y(S,W) + \alpha(S,W) \text{ and}$$
$$\sum_{v\in W} x_v \leq \min_{c\in C} |N_c(W)| + X(S,W) - \beta(S,W).$$

To see why these constraints must be fulfilled, we consider two cases. If $|N_{\chi(S)}(W)| - \min_{c\in C} |N_c(W)| \leq k$ (resp., $\max_{c\in C} |\nu_c(W)| - |\nu_{\chi(S)}(W)| \leq k$), then by the definition of $X(S,W)$ (resp. $Y(S,W)$) and Equation (2), we have $|N_{G^\star_S}(W)| = \min_{c\in C} |N_c(W)| + X(S,W) - \beta(S,W)$ (resp., $|\nu_{G^\star_S}(W)| = \max_{c\in C} |\nu_c(W)| - Y(S,W) + \alpha(S,W)$). Otherwise, we claim that the constraint $\sum_{v\in W} x_v \leq |N_{G^\star_S}(W)|$ (resp. $\sum_{v\in W} y_v \geq |\nu_{G^\star_S}(W)|$) is already captured by Constraint (1), since $X(S,W) = k \geq \beta(S,W)$ (resp., $Y(S,W) = k \geq \alpha(S,W)$), we have $\sum_{v\in W} x_v \leq \min_{c\in C} |N_c(W)| \leq \min_{c\in C} |N_c(W)| + X(S,W) - \beta(S,W)$ (resp., $\sum_{v\in W} y_v \geq \max_{c\in C} |\nu_c(W)| \geq \max_{c\in C} |\nu_c(W)| - Y(S,W) + \alpha(S,W)$).

Iterating over all described guesses and for each solving the constructed ILP, we get:

▶ **Theorem 9** (⋆). Max-Min Fair Matching *with the non-emptiness constraint can be solved in* $O^*(2^{2^{O(k)}})$ *time.*

4.2 MoV Fair Matching

We now develop an FPT algorithm for MoV Fair Matching for the parameter k, which also works with the non-emptiness constraint. Our algorithm has two parts. In the first part, we give an FPT algorithm (using an ILP) for an auxiliary problem called Targeted MoV Fair Matching. This is a variant of MoV Fair Matching, where for each $v \in V$, the two most frequent colors appearing in $M(v)$ are given as part of the input. We establish the soundness of the ILP for Targeted MoV Fair Matching using again Theorem 6 and Lemma 2. In the second part, we present a (randomized) parameterized reduction from MoV Fair Matching to Targeted Mov Fair Matching using the color coding technique [5]. To apply this technique, we show that the colors that appear (second) most frequently in $M(v)$ for some $v \in V$ "stand out" in a fair matching that fulfills certain conditions. Then, the color coding technique essentially allows us to determine these colors.

Part I. First, we define an auxiliary problem, which we call Targeted MoV Fair Matching. The input for MoV Fair Matching is also part of the input for Targeted MoV Fair Matching. Moreover, Targeted MoV Fair Matching takes as input two functions $\mu^1, \mu^2 \colon V \to C$. In Targeted MoV Fair Matching, we ask for an ℓ-fair matching M such that for every vertex $v \in V$, $\mu^1(v)$ (resp., $\mu^2(v)$) is the most (resp., second most) frequent color among the vertices $M(v)$ matched to v in M.

We now develop an FPT algorithm for Targeted MoV Fair Matching by means of an ILP. Let $C_{1,2} = \{\mu^1(v), \mu^2(v) \mid v \in V\}$ be the set of colors that appear (second) most frequent among the vertices matched to some vertex in V and let $C' = C \setminus C_{1,2}$ be the set

of other colors. Notably, the size of $C_{1,2}$ is linearly bounded in our parameter k. For every $v \in V$, we introduce a variable y_v which represents the number of vertices of color $\mu^2(v)$ matched to v. The values of y_v need to be chosen in a way such that, for every color $c' \in C'$, there is a matching $M_{c'}$ in $G_{c'}$ such that $|M_{c'}(v)| \leq y_v$ for all $v \in V$. By Lemma 1, we obtain the following constraint which must be fulfilled and add it to the ILP:

$$\sum_{v \in W} y_v \geq \max_{c' \in C'} |\nu_{c'}(W)| \quad \forall W \subseteq V.$$

For the vertices of colors in $C_{1,2}$, we impose constraints in the same way as in Section 3. For every $c \in C_{1,2}$ and $v \in V$, we introduce a variable z_v^c which represents the number of vertices of color c matched to v. Then, we add the following constraints according to Lemma 1.

$$|\nu_c(W)| \leq \sum_{v \in W} z_v^c \leq |N_c(W)| \quad \forall W \subseteq V, c \in C_{1,2}.$$

In case we have a lower bound $p = 1$, we additionally require that $z_v^{\mu^1(v)} \geq 1$ for all $v \in V$. Finally, we encode the ℓ-fairness: $y_v = z_v^{\mu^2(v)}$ and $z_v^{\mu^2(v)} \leq z_v^{\mu^1(v)} \leq z_v^{\mu^2(v)} + \ell$ for all $v \in V$.

In order to show the correctness of the ILP, with the help of Theorem 6, we prove the following adaptation of Lemma 2, in which we show that there exists a matching of the vertices of colors from C' to vertices from V respecting y_v for all $v \in V$.

▶ **Lemma 10** (⋆). *Let $G = (U \cup V, E)$ be a bipartite graph and let $\{z_v \in \mathbb{N} \mid v \in V\}$ be a set of integers. Suppose that $\sum_{v \in W} z_v \geq |\nu_G(W)|$ for every $W \subseteq V$. Then, there is a left-perfect many-to-one matching M such that $M(v) \leq z_v$ for every $v \in V$.*

Using Lemmas 2 and 10, we can now show that the above constructed ILP is feasible if and only if the given Targeted MoV Fair Matching is a yes-instance.

▶ **Proposition 11** (⋆). *Targeted MoV Fair Matching can be solved in $O^\star(k^{O(k^2)})$ time even with the non-emptiness constraint.*

Part II. We will employ the color coding technique to reduce MoV Fair Matching to Targeted Mov Fair Matching. To do so, we introduce another auxiliary problem called $\mathcal{Q}$-MoV Fair Matching. To define the problem, we first introduce additional notation. Suppose that M is a matching in the input graph $G = (U \cup V, E)$. Let $\mu_M^1, \mu_M^2 \colon V \to C$ be mappings such that for every vertex $v \in V$, $\mu_M^1(v)$ (resp., $\mu_M^2(v)$) is the most (resp., second most) frequent color among the vertices $M(v)$ matched to v in M. When the maximum or second maximum is achieved by more than one color, we break ties according to a fixed linear order $\leq_C$ on C. Let $\mathcal{V} = \{v^1, v^2 \mid v \in V\}$ be a set containing $2|V|$ elements and let $\mathcal{P}(M)$ be a partition of $\mathcal{V}$ into subsets $S \subseteq \mathcal{V}$ where for every $S \in \mathcal{P}(M)$, $v^i, v'^j \in S$ if and only if $\mu_M^i(v) = \mu_M^j(v')$ for $v, v' \in V$ and $i, j \in [2]$.

Using this, we define $\mathcal{Q}$-MoV Fair Matching. Here, $\mathcal{Q}$ is a partition of $\mathcal{V}$ and we assume that $\mathcal{Q}$ is fixed. The input of $\mathcal{Q}$-MoV Fair Matching is identical to the input of MoV Fair Matching. The difference is that $\mathcal{Q}$-MoV Fair Matching asks for a left-perfect ℓ-fair many-to-one matching M in G *consistent* with $\mathcal{Q}$, that is, $\mathcal{P}(M) = \mathcal{Q}$. Clearly, an instance of MoV Fair Matching is a yes-instance if and only if there exists a partition $\mathcal{Q}$ of $\mathcal{V}$ such that the corresponding $\mathcal{Q}$-MoV Fair Matching instance is a yes-instance. Since there are at most $k^{O(k)}$ ways to partition $\mathcal{V}$ (which is a set of $2k$ elements), we can afford to "guess" $\mathcal{Q}$ in our FPT algorithm. We thus focus on solving $\mathcal{Q}$-MoV Fair Matching. In particular, we assume without loss of generality that $\mathcal{Q}$ is cardinality-wise maximum – the input graph G has no ℓ-fair left-perfect many-to-one matching M such that $|\mathcal{P}(M)| > |\mathcal{Q}|$ (this assumption becomes essential in the proof of Lemma 12).

We solve $\mathcal{Q}$-MoV Fair Matching using color coding. We focus on $\ell > 0$ and omit the case $\ell = 0$. To apply the color coding method, we show a rather technical lemma stating that if a color c is among the two most frequent colors in $M(v)$ for some $v \in V$ in a certain fair matching M, then c needs to "stand out" – the number of its occurrence in the neighborhood of v in G is greater than for any other color c', unless c' also "stands out".

▶ **Lemma 12.** *Let $\mathcal{I}$ be a yes-instance of $\mathcal{Q}$-MoV Fair Matching with $\ell > 0$. Then, $\mathcal{I}$ admits an ℓ-fair left-perfect many-to-one matching M such that, for every $v \in V$ and $i \in [2]$, $|N_{\mu^i_M(v)}(v)| \geq |N_{c'}(v)|$ for every $c' \in \{c \in C \mid \forall v \in V, i \in \{1,2\}\colon c \neq \mu^i_M(v)\}$.*

Proof. For a matching M, let $\sigma(M) := \sum_{v \in V, i \in \{1,2\}} |M(v)_{\mu^i_M(v)}|$ be the total number of the occurrences of the two most frequent colors in $M(v)$ over all vertices $v \in V$. Consider an ℓ-fair matching M that maximizes $\sigma(M)$ among all ℓ-fair matchings consistent with $\mathcal{Q}$. Let $C_M := \{\mu^i_M(v) \in C \mid v \in V, i \in \{1,2\}\}$ be the set of colors which are one of the two most frequent colors in $M(v)$ for some $v \in V$ and let $C'_M := C \setminus C_M$ be the set of other colors.

Assume for the sake of contradiction that there there is some $v \in V$ and $i \in [2]$ with $\mu^i_M(v) = c \in C_M$ and color $c' \in C'_M$ such that $|N_{c'}(v)| > |N_c(v)|$. Recall that by the definition of C'_M it holds that $|M(v)_c| \geq |M(v)_{c'}|$. Consider a left-perfect matching M' obtained from M as follows. Initially, let $M' := M$. We then repeat the following procedure $|M(v)_c| - |M(v)_{c'}| + 1$ times: We delete from M' an arbitrary edge $\{u, v'\} \in M'$ such that $u \in N(v) \subseteq U$ is a vertex of color c' and v' is some vertex in $V - v$, and then add an edge $\{u, v\}$. Note that this is always possible, as $|M(v)_{c'}| \leq |M(v)_c| \leq |N_c(v)| < |N_{c'}(v)|$. We claim that M' is a left-perfect ℓ-fair matching. It is easy to verify the left-perfectness of M'. For the ℓ-fairness, since $c' \in C'_M$, for every $v' \in V - v$, the number of occurrences of the two most frequent colors in $M'(v')$ has not changed compared to $M(v')$, i.e., $\max^1_{c'' \in C} |M'(v')_{c''}| = \max^1_{c'' \in C} |M(v')_{c''}|$ and $\max^2_{c'' \in C} |M'(v')_{c''}| = \max^2_{c'' \in C} |M(v')_{c''}|$. Thus, $M'(v')$ is ℓ-fair for each $v' \in V - v$. It thus remains to show that $\max^1_{c'' \in C} |M'(v)_{c''}| - \max^2_{c'' \in C} |M'(v)_{c''}| \leq \ell$. We consider three cases:

- If $c = \mu^1_M(v)$, then as we added $|M(v)_c| - |M(v)_{c'}| + 1$ vertices of color c' to $M(v)$, we have $\max^1_{c'' \in C} |M'(v)_{c''}| = |M'(v)_{c'}| = |M(v)_c| + 1$ and $\max^2_{c'' \in C} |M'(v)_{c''}| = |M(v)_c|$, and thus $\max^1_{c'' \in C} |M'(v)_{c''}| - \max^2_{c'' \in C} |M'(v)_{c''}| = 1 \leq \ell$.
- Suppose that $c = \mu^2_M(v)$ and $\max^1_{c \in C} |M(v)_c| > \max^2_{c \in C} |M(v)_c|$. Then, we have $\max^1_{c \in C} |M'(v)_c| = \max^1_{c \in C} |M(v)_c|$ and $\max^2_{c \in C} |M(v)_c| = \max^2_{c \in C} |M(v)_c| + 1$, and thus $\max^1_{c'' \in C} |M'(v)_{c''}| - \max^2_{c'' \in C} |M'(v)_{c''}| \leq \ell - 1 \leq \ell$.
- Suppose that $c = \mu^2_M(v)$ and $\max^1_{c \in C} |M(v)_c| = \max^2_{c \in C} |M(v)_c|$. Then, we have $\max^1_{c \in C} |M'(v)_c| = \max^1_{c \in C} |M(v)_c| + 1$ and $\max^2_{c \in C} |M(v)_c| = \max^2_{c \in C} |M(v)_c|$, and thus $\max^1_{c'' \in C} |M'(v)_{c''}| - \max^2_{c'' \in C} |M'(v)_{c''}| \leq 1 \leq \ell$.

This proves the ℓ-fairness of M.

Recall that $\mu^i_M(v) = c$. We now show that the existence of M' contradicts one of our assumptions. To that end, we consider the following two cases:

- Suppose that $\mu^j_M(v') = c$ for some $v' \in V \setminus \{v\}$ and $j \in [2]$. This case contradicts the fact that $\mathcal{Q}$ is cardinality-wise maximum: To see why, note that for the set S of $\mathcal{Q}$ containing v^i (which is of size at least two, since either v'^1 or v'^2 is in S), we have $\mathcal{P}(M') = (\mathcal{P}(M) \setminus \{S\}) \cup \{S - v, \{v\}\}$, implying that $|\mathcal{P}(M')| > |\mathcal{P}(M)|$.
- Suppose that $\mu^j_M(v') \neq c$ for each $v' \in V \setminus \{v\}$ and $j \in [2]$. Observe that $\{v^i\} \in \mathcal{Q}$. The matching M' is thus consistent with $\mathcal{Q}$. Moreover, we have $\sigma(M') > \sigma(M)$, which is a contradiction. ◀

With Lemma 12 at hand, we are ready to give a randomized reduction from $\mathcal{Q}$-MoV Fair Matching to Targeted MoV Fair Matching.

▶ **Lemma 13** (⋆). *Let $\mathcal{I}$ be an instance of $\mathcal{Q}$-MoV Fair Matching. We can compute in polynomial time an instance $\mathcal{J}$ of Targeted MoV Fair Matching such that (i) $\mathcal{J}$ is a yes-instance with probability at least δ (where $1/\delta \in k^{O(k)}$) if $\mathcal{I}$ is a yes-instance and (ii) $\mathcal{J}$ is a no-instance if $\mathcal{I}$ is no-instance.*

Proof for $\ell > 0$. We describe a polynomial-time procedure to construct an instance $\mathcal{J}$ of Targeted MoV Fair Matching from a given instance $\mathcal{I} = (G = (U \cup V, E), C, \mathrm{col}, \ell)$ of $\mathcal{Q}$-MoV Fair Matching. We randomly assign each color $c \in C$ to one of the subsets in $\mathcal{Q}$ with the intended meaning that if we assign color $c \in C$ to $S \in \mathcal{Q}$, then in a matching M in $\mathcal{I}$ one of the following holds: (i) for $v^1 \in S$, c appears as the most frequent color in $M(v)$ and for $v^2 \in S$, c appears as the second most frequent color in $M(v)$ and c appears nowhere else as the most or second most frequent color or (ii) c does not appear as the most or second most frequent color in $M(v)$ for any $v \in V$. Formally let $\lambda\colon C \to \mathcal{Q}$ be a function assigning each color to a subset in $\mathcal{Q}$, where each assignment is chosen uniformly and independently at random.[6] For every $v \in V$ and $i \in [2]$ with $v^i \in S$ for some $S \in \mathcal{Q}$, we find $c_v^i = \arg\max |N_c(v)|$, where $\arg\max$ is taken over all colors c with $\lambda(c) = S$. Then, we construct an instance $\mathcal{J}$ of Targeted MoV Fair Matching, where $\mu^i(v) = c_v^i$ for every $v^i \in \mathcal{V}$. If a hypothetical matching M such that $\mu_M^i(u) = \mu^i(u) = c_v^i$ for $v \in V$ and $i \in [2]$ is not consistent with $\mathcal{Q}$, we let $\mathcal{J}$ be a trivial no-instance. Clearly, the construction of $\mathcal{J}$ takes polynomial time.

Suppose that $\mathcal{I}$ is a yes-instance. We show that in this case $\mathcal{J}$ is a yes-instance with probability at least δ with $\delta^{-1} \in k^{O(k)}$. Let M be an ℓ-fair matching for $\mathcal{I}$ that fulfills the properties described in Lemma 12 (which by Lemma 12 always exists). Assume that $\lambda(\mu_M^i(v)) = S_v^i$ holds for every $v \in V$ and $i \in [2]$, where $S_v^i \in \mathcal{Q}$ denotes the subset in $\mathcal{Q}$ to which v^i belongs. We claim that under this assumption on λ, the instance $\mathcal{J}$ constructed by our procedure is a yes-instance. In fact, we show that M is a solution of $\mathcal{J}$ (which also directly implies that there is a solution consistent with $\mathcal{Q}$ and thus that no trivial no-instance is returned). Since M is an left-perfect ℓ-fair matching in G, we only have to show that $\mu_M^i(v) = \mu^i(v) = c_v^i$ for every $v \in V$ and $i \in [2]$.

Let $C_M := \{\mu_M^i(v) \in C \mid v \in V, i \in [2]\}$ be the set of colors which are among the two most frequent colors in $M(v)$ for some $v \in V$ and let $C'_M := C \setminus C_M$ be the set of other colors. By Lemma 12, for every $v \in V$, $c' \in C'_M$, and $i \in [2]$, we have $|N_{\mu_M^i(v)}(v)| \geq |N_{c'}(v)|$. Since we always break ties according to a fixed linear order (including when we find $\mu_M^i(v)$), we have $\mu_M^i(v) = \arg\max_{c \in C'_M \cup \{\mu_M^i(v)\}} |N_c(v)|$. Recall that when constructing $\mathcal{J}$ we have defined $c_v^i = \arg\max |N_c(v)|$, where the maximum is taken over the set $C_v^i := \{c \in C \mid \lambda(c) = S_v^i\}$. By our assumption that $\lambda(\mu_M^i(v)) = S_v^i$ for every $v \in V$ and $i \in [2]$, we have $\mu_M^i(v) \in C_v^i$. We also have $c \notin C_v^i$ for every $c \in C_M \setminus \{\mu_M^i(v)\}$, which implies that $C_v^i \subseteq C'_M \cup \{\mu_M^i(v)\}$. Consequently, we obtain $c_v^i = \arg\max_{c \in C_v^i} |N_c(v)| = \mu_M^i(v)$ for every $v \in V$ and $i \in [2]$. It follows that M is a solution of $\mathcal{J}$.

Finally, observe that the probability that $\lambda(\mu_M^i(v)) = S_v^i$ for every $v \in V$ and $i \in [2]$ is at least $|\mathcal{Q}|^{-|\mathcal{Q}|} \geq (2k)^{-2k}$, since λ is chosen uniformly and independently at random. Thus, if $\mathcal{I}$ is a yes-instance, $\mathcal{J}$ is a yes-instance with probability at least $(2k)^{-2k}$.

If $\mathcal{J}$ is a yes-instance, then $\mathcal{I}$ is also a yes-instance, as, by construction of $\mathcal{J}$, a solution M for $\mathcal{J}$ needs to satisfy $\mathcal{P}(M) = \mathcal{Q}$ and is thus also a solution for $\mathcal{I}$. ◀

[6] In the language of color coding, the function λ is often described as assignments of "colors" (the "colors" in color coding are different from the colors used here).

We repeat the algorithm of Lemma 13 independently $\delta^{-1} \in k^{O(k)}$ times on the given instance $\mathcal{I}$ of $\mathcal{Q}$-Fair Matching. If $\mathcal{I}$ is a yes-instance, then at least one instance of Targeted Mov Fair Matching returned by the algorithm is a yes-instance with probability at least $1-(1-\delta)^{\delta^{-1}} \geq 1-1/e$. By Proposition 11, a yes-instance of Targeted Mov Fair Matching can be recognized in $O^\star(k^{O(k^2)})$ time. We thus have a randomized algorithm to solve $\mathcal{Q}$-MoV Fair Matching in $O^\star(k^{O(k^2)})$ time. Recall that $\mathcal{Q}$ is a partition of a $2k$-element set. So there are at most $k^{O(k)}$ choices for $\mathcal{Q}$, which gives us the following theorem (we remark that our algorithm can be derandomized using a standard method [12]):

▶ **Theorem 14.** *There is a randomized $O^\star(k^{O(k^2)})$-time algorithm to solve MoV Fair Matching even with the non-emptiness constraint.*

5 Complexity Dichotomies with respect to $|C|$ and Maximum Degree

In this section, we study the computational complexity of Max-Min/MoV Fair Matching for fixed values of Δ_U, Δ_V, and $|C|$. Recall that Δ_U (resp., Δ_V) is the maximum degree of all vertices in U (resp., V). We identify several computational dichotomies regarding these parameters (see Figure 1). We first show a dichotomy on $|C|$. In particular, Max-Min/MoV Fair Matching is polynomial-time solvable for $|C|=2$ (even if there are arbitrary lower size constraints), while it is NP-hard for $|C|=3$.

▶ **Theorem 15** (⋆). *Max-Min/MoV Fair Matching is polynomial-time solvable for $|C|=2$ and NP-hard for $|C|\geq 3$.*

The next two theorems concern complexity dichotomies with respect to Δ_U and Δ_V. All NP-hardness results here hold for three colors, while all polynomial-time results hold for an arbitrary number of colors.

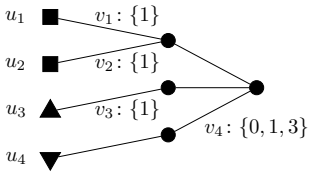
▶ **Theorem 16** (⋆). *MoV Fair Matching is polynomial-time solvable if $\Delta_U \leq 1$ or $\Delta_V \leq 4$ (even with the non-emptiness constraint) and NP-hard otherwise.*

As part of the proof of Theorem 16, we give an algorithm that solves MoV in polynomial-time for $\Delta_V \leq 4$. Our algorithm is a polynomial-time reduction to a polynomial-time solvable special case of General Factor, which we will define in the proof.

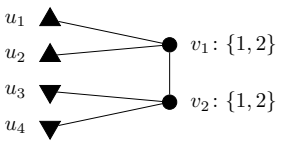
▶ **Proposition 17** (⋆). *MoV Fair Matching is polynomial-time solvable for $\Delta_V \leq 4$.*

Proof Sketch. We give a polynomial-time reduction to General Factor: In an instance of General Factor the input is an undirected graph $H=(W,F)$ and a degree list function $L\colon W \to 2^{\mathbb{N}}$ such that $L(w) \subseteq \{0,\dots,\deg_G(w)\}$ for every vertex $w \in W$. The problem asks for a spanning subgraph $H'=(W,F')$ for $F' \subseteq F$ such that $\deg_{H'}(w) \in L(w)$ for every $w \in W$. By a result of Cornuéjols [11], General Factor is polynomial-time solvable if for every $w \in W$, the degree list $L(w)$ has gaps of size at most one, i.e., $\{\min L(w), \min L(w)+1 \dots, \max L(w)\} \setminus L(w)$ does not contain any two consecutive integers. Given an instance $\mathcal{I}=(G=(U \cup V, E), C, \mathrm{col}, \ell)$ of MoV Fair Matching with $\Delta_V \leq 4$, we will construct an equivalent instance $\mathcal{J}=(H=(W,F),L)$ of this polynomial-time solvable special case of General Factor.

In the following, we give an overview of our construction. For every $v \in V$, we add a subgraph $H_v=(W_v,F_v)$ to H which contains all vertices from U adjacent to v in G ($N_G(v) \subseteq W_v$) but no other vertices from U. For the construction of H_v, we make extensive case distinctions depending on $N_G(v)$ and ℓ. See Figure 2 for two examples. The construction of H_v for all other cases are deferred to the appendix. The graph H is then the union of H_v



(a) H_v for $\ell = 0$ and $N(v)$ has two vertices of color α, one vertex of color β, and one vertex of color γ.



(b) H_v for $\ell = 1$ and $N(v)$ has two vertices of color α and two vertices of color β.

Figure 2 Exemplary constructions of H_v.

for all $v \in V$; notably, a vertex u from U may appear in multiple subgraphs H_v, in which case we identify all occurrences of u and merge them into one vertex. Moreover, for each $u \in U$, we set $L(u) = \{1\}$, which ensures that u is "matched" in every solution of $\mathcal{J}$.

To show that $\mathcal{I}$ and $\mathcal{J}$ are equivalent, it suffices to show the following for every $v \in V$:

A vertex set $S \subseteq N_G(v)$ is ℓ-fair if and only if there is a spanning subgraph $H'_v = (W_v, F'_v)$ of H_v such that $\deg_{H'_v}(u) = 1$ for every $u \in S$, $\deg_{H'_v}(u) = 0$ for every $u \in N(v) \setminus S$, and $\deg_{H'_v}(v') \in L(v')$ for every $v' \in W_v \setminus U$. $(\star)$

To see why $(\star)$ is sufficient, assume that $(\star)$ holds true. For the forward direction, suppose that $\mathcal{I}$ is a yes-instance, i.e., there is a ℓ-fair left-perfect many-to-one matching M in G. Then, $M(v) \subseteq N_G(v)$ is ℓ-fair for every $v \in V$. Hence, as we assume that $(\star)$ holds, we have a subgraph H'_v for every $v \in V$ that satisfies the degree constraints of $(\star)$. Consider a spanning subgraph H' whose edge set is the union of the edge set of H'_v over all v. It is easy to verify that H' constitutes a solution for $\mathcal{J}$. For the converse direction, suppose that $\mathcal{J}$ is a yes-instance, i.e., there is a spanning subgraph $H' = (W, F')$ with $\deg(w) \in L(w)$ for every $w \in W$. Then, the subgraph of H' induced by W_v satisfies all the degree constraints of $(\star)$. For $v \in V$, let $S_v \subseteq N_G(v)$ be the set of vertices that have a neighbor in $H'[W_v]$. By $(\star)$, S_v is ℓ-fair. Moreover, since $L(u) = \{1\}$ for every $u \in U$, every vertex appears in S_v for exactly one vertex $v \in V$. It follows that a matching M with $M(v) = S_v$ for every $v \in V$ is a ℓ-fair left-perfect many-to-one matching in $\mathcal{I}$. We remark that we can adapt our algorithm to handle the non-emptiness constraint. ◀

For Max-Min Fair Matching, we prove that fewer cases are polynomial-time solvable than for MoV Fair Matching:

▶ **Theorem 18** $(\star)$**.** *Max-Min Fair Matching is polynomial-time solvable if* $\Delta_U \leq 1$, $\Delta_V \leq 2$, *or* $(\Delta_U, \Delta_V) = (2, 3)$ *(even with the non-emptiness constraint) and NP-hard otherwise.*

6 Fair Matching on Complete Bipartite Graphs

A natural special case of Fair Matching is when the underlying graph is complete, i.e., each vertex in U can be assigned to any vertex in V. This special case is also among the three problems introduced by Stoica et al. [24] (they called it Fair Regrouping_X). Stoica et al. [24] presented a straightforward XP algorithm for MoV Fair Matching with size constraints parameterized by $|V|$ but left open the classical complexity. We partially settle this open question by proving that MoV Fair Matching on complete bipartite graphs is polynomial-time solvable even with the non-emptiness constraint. In fact, we find a precise characterization of yes-instances, which turns out to be surprisingly simple. However, it requires an intricate analysis to prove this, especially when the non-emptiness constraint is present. To simplify notation, we assume that $C = \{c_1, \ldots, c_{|C|}\}$ and that $|U_{c_i}| \geq |U_{c_{i+1}}|$ for each $i \in [1, |C| - 1]$ and set $|U_{c_i}| := 0$ for $i > |C|$.

▶ **Theorem 19** (⋆). *A MoV Fair Matching instance* $\mathcal{I} = (G = (U \cup V, E), C, \mathrm{col}, \ell)$ *with* G *being a complete bipartite graph is a yes-instance if and only if* $|U_{c_1}| \leq \ell k + \sum_{i \in [k]} |U_{c_{i+1}}|$. *With the non-emptiness constraint,* $\mathcal{I}$ *is a yes-instance if and only if it additionally satisfies:* $\ell > 0$ *and* $n \geq k$, *or* $\ell = 0$ *and* $n \geq 2k$.

▶ **Theorem 20** (⋆). *A Max-Min Fair Matching instance* $\mathcal{I} = (G = (U \cup V, E), C, \mathrm{col}, \ell)$ *with* G *being a complete bipartite graph is a yes-instance if and only if* $|U_{c_1}| \leq \ell k + |U_{c_{|C|}}|$. *With the non-emptiness constraint,* $\mathcal{I}$ *is a yes-instance if and only if it additionally satisfies:* $\ell > 0$ *and* $n \geq k$, *or* $\ell = 0$ *and* $|U_{c_1}| \geq k$.

Theorems 19 and 20 imply that Max-Min/MoV Fair Matching on a complete bipartite graph are solvable in linear time even with the non-emptiness constraint.

7 Conclusion

In this work, we have investigated the (parameterized) computational complexity of the Fair Matching problem. Two concrete directions of open questions are:

- We have provided algorithms that solve Fair Matching even if we require that every vertex in the right side is matched to at least one vertex. Can we extend our algorithms to handle arbitrary size constraints? In particular, does Fair Matching remain fixed-parameter tractable with respect to k? We have shown in Section 3 that Fair Matching is indeed FPT with respect to $k + |C|$ even for arbitrary size constraints. However, it does not seem straightforward to incorporate arbitrary size constraints in the ILPs given in Section 4. The complexity of Fair Matching on complete bipartite graphs (Section 6) is also open when arbitrary size constraints are present.
- Is Fair Matching solvable in $O^\star(2^k)$ time? Note that the ILP presented in Section 4.1 (which solves Max-Min Fair Matching without the non-emptiness constraint) is an ILP where the constraint matrix involves only zeros and ones when $\ell = 0$. Can we exploit such a structure in the constraint matrix to obtain a faster algorithm?

For future research, it would also be natural to study other variants of the Fair Matching problem. For instance, we may relax the left-perfect constraint studied in this work and consider a variant where the objective is to maximize the matching size under a fairness constraint. One may also look into other fairness notions such as proportionality constraints [21].

References

1 Saba Ahmadi, Faez Ahmed, John P. Dickerson, Mark Fuge, and Samir Khuller. An algorithm for multi-attribute diverse matching. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 3–9. ijcai.org, 2020.

2 Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Fair correlation clustering. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS '20)*, pages 4195–4205. PMLR, 2020.

3 Faez Ahmed, John Dickerson, and Mark Fuge. Forming diverse teams from sequentially arriving people. *J. Mech. Design*, 142(11):111401, 2020.

4 Faez Ahmed, John P. Dickerson, and Mark Fuge. Diverse weighted bipartite b-matching. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI '17)*, pages 35–41. ijcai.org, 2017.

5 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

6 Abolfazl Asudeh, Tanya Y. Berger-Wolf, Bhaskar DasGupta, and Anastasios Sidiropoulos. Maximizing coverage while ensuring fairness: a tale of conflicting objective. *CoRR*, abs/2007.08069, 2020. arXiv:2007.08069.

7 Haris Aziz, Serge Gaspers, Zhaohong Sun, and Toby Walsh. From matching with diversity constraints to matching with regional quotas. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19)*, pages 377–385. IFAAMAS, 2019.

8 Inacio Bo. Fair implementation of diversity in school choice. *Games Econ. Behav.*, 97:54–63, 2016.

9 Niclas Boehmer, Tomohiro Koana, and Rolf Niedermeier. A refined complexity analysis of fair districting over graphs. *CoRR*, abs/2102.11864, 2021. Accepted as an extended abstract at AAMAS '22. arXiv:2102.11864.

10 Jiehua Chen, Robert Ganian, and Thekla Hamm. Stable matchings with diversity constraints: Affirmative action is beyond NP. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 146–152. ijcai.org, 2020.

11 Gérard Cornuéjols. General factors of graphs. *J. Comb. Theory, Ser. B*, 45(2):185–198, 1988. doi:10.1016/0095-8956(88)90068-8.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 John P. Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. Balancing relevance and diversity in online bipartite matching via submodularity. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI '19)*, pages 1877–1884. AAAI Press, 2019.

14 András Frank. An algorithm for submodular functions on graphs. *Discrete Math.*, 16:97–120, 1982.

15 Zachary Friggstad and Ramin Mousavi. Fair correlation clustering with global and local guarantees. In *Proceedings of the 17th International Symposium on Algorithms and Data Structures (WADS '21)*, pages 414–427. Springer, 2021.

16 Isa E Hafalir, M Bumin Yenmez, and Muhammed A Yildirim. Effective affirmative action in school choice. *Theor. Econ.*, 8(2):325–363, 2013.

17 Philip Hall. On representatives of subsets. *Classic Papers in Combinatorics*, pages 58–62, 1987.

18 Chien-Chung Huang. Classified stable matching. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '10)*, pages 1235–1253. SIAM, 2010.

19 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

20 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983.

21 Thành Nguyen and Rakesh Vohra. Stable matching with proportionality constraints. *Oper. Res.*, 67(6):1503–1519, 2019.

22 Deval Patel, Arindam Khan, and Anand Louis. Group fairness for knapsack problems. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '21)*, pages 1001–1009. ACM, 2021.

23 Dana Pessach and Erez Shmueli. Algorithmic fairness. *CoRR*, abs/2001.09784, 2020. arXiv:2001.09784.

24 Ana-Andreea Stoica, Abhijnan Chakraborty, Palash Dey, and Krishna P. Gummadi. Minimizing margin of victory for fair political and educational districting. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '20)*, pages 1305–1313. IFAAMAS, 2020.

Factoring and Pairings Are Not Necessary for IO: Circular-Secure LWE Suffices

Zvika Brakerski
Weizmann Institute of Science, Rehovot, Israel

Nico Döttling
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Sanjam Garg
University of California, Berkeley, CA, USA
NTT Research, Sunnyvale, CA, USA

Giulio Malavolta
Max Planck Institute for Security and Privacy, Bochum, Germany

Abstract

We construct indistinguishability obfuscation (iO) solely under circular-security properties of encryption schemes based on the Learning with Errors (LWE) problem. Circular-security assumptions were used before to construct (non-leveled) fully-homomorphic encryption (FHE), but our assumption is stronger and requires circular randomness-leakage-resilience. In contrast with prior works, this assumption can be conjectured to be post-quantum secure; yielding the first provably secure iO construction that is (plausibly) post-quantum secure.

Our work follows the high-level outline of the recent work of Gay and Pass [STOC 2021], who showed a way to remove the heuristic step from the homomorphic-encryption based iO approach of Brakerski, Döttling, Garg, and Malavolta [EUROCRYPT 2020]. They thus obtain a construction proved secure under circular security assumption of natural homomorphic encryption schemes – specifically, they use homomorphic encryption schemes based on LWE and DCR, respectively. In this work we show how to remove the DCR assumption and remain with a scheme based on the circular security of LWE alone. Along the way we relax some of the requirements in the Gay-Pass blueprint and thus obtain a scheme that is secure under a different assumption. Specifically, we do not require security in the presence of a key-cycle, but rather only in the presence of a key-randomness cycle.

An additional contribution of our work is to point out a problem in one of the building blocks used by many iO candidates, including *all* existing provable post-quantum candidates. Namely, in the transformation from exponentially-efficient iO (XiO) from Lin, Pass, Seth and Telang [PKC 2016]. We show why their transformation inherently falls short of achieving the desired goal, and then rectify this situation by showing that *shallow* XiO (i.e. one where the obfuscator is depth-bounded) does translate to iO using LWE.

2012 ACM Subject Classification Theory of computation → Cryptographic primitives

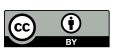
Keywords and phrases Cryptography, Obfuscation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.28

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://eprint.iacr.org/2020/1024

Funding *Zvika Brakerski*: Supported by the Israel Science Foundation (Grant No. 3426/21), and by the European Union Horizon 2020 Research and Innovation Program via ERC Project REACT (Grant 756482) and via Project PROMETHEUS (Grant 780701).
Nico Döttling: Supported by the Helmholtz Association within the project "Trustworthy Federated Data Analytics" (TFDA) (funding number ZT-I- OO1 4).
Sanjam Garg: Supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc. Any opinions, findings and conclusions or recommendations ex- pressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.



Giulio Malavolta: The work described in this paper has been partially supported by the German Federal Ministry of Education and Research BMBF (grant 16K15K042, project 6GEM).

1 Introduction

The goal of program obfuscation [4, 28] is to transform an arbitrary circuit Π into an unintelligible but functionally equivalent circuit $\tilde{\Pi}$. The aforementioned works showed that strong simulation-based notions of obfuscation were impossible for general purpose functionalities. However, the seemingly weaker *indistinguishability obfuscation* (iO) was not ruled out by prior work (and has in fact been shown to be the same as the best possible notion of obfuscation [27]). In broad terms, iO requires that if two circuits Π_0 and Π_1 are two implementations of the same function, then their obfuscations are computationally indistinguishable.

Garg et al. [19, 21] presented the first candidate for general purpose iO, paving the way for numerous other candidates based on a variety of mathematical structures. Although iO appears to be a weak notion of security, it has been shown to be sufficient for numerous cryptographic applications, including ones that were previously not known to exist under other assumptions (see [6, 20, 41] for examples). The first realizations of obfuscation relied an a new algebraic object called multilinear maps [15, 19, 24], which had only recently been constructed. Furthermore, the security of these objects relied on new (and poorly understood) computational intractability assumptions, or more commonly on plain heuristics. In fact, several attacks on multilinear map candidates [14, 30] and on obfuscation constructions based on multilinear maps [12, 39] were demonstrated. To defend against these attacks, several safeguards have been (e.g., [5, 13, 17, 22, 38]) proposed. Even with these heuristic safeguards, all but the schemes based on the Gentry et al. [24] multilinear maps are known to be broken against quantum adversaries.

Towards the goal of avoiding heuristics and obtaining provably secure constructions, substantial effort was made towards obtaining iO while minimizing (with the ultimate goal of removing) the use of multilinear maps [3, 33, 34, 36, 37]. These efforts culminated in replacing the use of multilinear maps with just bilinear maps [1, 2, 31], together with an additional pseudorandom generators of constant locality over the integers with polynomial stretch. Very recently this last limitation was removed by Jain, Lin and Sahai [32]. Specifically, they obtained iO based on the combined (sub-exponential) hardness of the Learning with Errors problem (LWE), a large-modulus variant of the Learning Parity with Noise problem (LPN), the existence of a pseudorandom generator in NC_0, and in addition the hardness of the external Diffie-Hellman problem in bilinear groups (SXDH). We note that the use of the pairings makes these construction insecure against quantum adversaries.

A different approach towards provably secure iO, which is more relevant to this work, was presented by Brakerski et al. [9]. They showed an iO candidate that is based on combining certain natural *homomorphic* encryption schemes. However, their construction was *heuristic* in the sense that the security argument could only be presented in the random oracle model. In a recent work, Gay and Pass [23] showed a way to remove the heuristic step and instead rely on a concrete assumption. Their construction is proved secure under the circular security of natural homomorphic encryption schemes – specifically, they use homomorphic encryption schemes based on LWE and Decisional Composite Residuosity (DCR, also known as Paillier's assumption). In terms of assumptions, their construction assumes sub-exponential security of (i) the Learning with Error (LWE) assumption, (ii) the Decisional Composite Residuosity (DCR) assumption, and (iii) a new notion of security that they call "shielded randomness

leakage" (SRL). The latter essentially requires that a fully homomorphic encryption scheme (specifically the GSW encryption scheme [26]) remains secure even in the presence of a key-cycle with the Damgård-Jurik encryption scheme [16]. Moreover, the notion of security is not the standard semantic security, but rather a new notion of security with respect to leakage of ciphertext randomness. We note that this construction is insecure against quantum attackers because of the use of the Damgård-Jurik encryption scheme [16].[1] In this work, we ask:

Can we realize provably secure constructions of iO with (plausible) post-quantum security?

1.1 Our results

We obtain a general purpose iO construction based solely on the circular security of LWE-based encryption schemes. On a technical level, we achieve this by introducing a "packed" variant of the dual-Regev LWE-based encryption scheme, and showing novel ways of manipulating ciphertexts of this variant in conjunction with ciphertexts of an FHE scheme. This allows us to remove the need for DCR-based encryption from the construction of [9, 23]. Furthermore, our technique allows us to relax the SRL security property that is required, so that we no longer need to require SRL security with respect to a key-cycle, but rather only with respect to a key-randomness cycle. We put forth this potentially weaker assumption as an object for further study.

More concretely, the circular security assumption made in [23], and thus also in this work, is that a scheme (in particular a leveled FHE scheme) maintains this property even in the presence of some leakage on the randomness of the ciphertext. In [23] it is shown that standard GSW encryption [26] satisfies SRL security (under the LWE assumption), and the additional assumption is therefore that SRL security is maintained in the presence of a key-randomness cycle, connecting GSW to another encryption scheme. While this assumption falls into the category of "circular security assumptions", similarly to the ones that underlie bootstrapping in FHE, the concrete assumption is quite different. While in the FHE setting it was only assumed that (standard) CPA security is preserved given a key cycle, here we assume that the stronger SRL property remains intact.

Let us now state our results somewhat more precisely.

▶ **Theorem 1** (Informal). *Assume the (sub-exponential) hardness of the LWE problem, and the SRL security of GSW in the presence of a randomness-key cycle with a packed variant of dual-Regev, then there exists indistinguishability obfuscation for all circuits.*

We note that if we further assume that circular security also maintains post-quantum security, then our assumption becomes post-quantum secure; yielding the first provably secure iO construction that is post-quantum secure.

Shallow XiO. As an additional contribution, we identify a gap in the transformation of "exponentially efficient iO" (XiO), a notion introduced by Lin, Pass, Seth and Telang [35] that was used almost universally in prior work. We show that this transformation has an inherent problem that does not allow to recover the result as stated. This gap affects most known iO constructions and, in particular, *all post-quantum provably secure candidates*. We rectify this situation by showing that a fairly simple technical modification (i.e. constraining

[1] Concurrently, [23] updated their manuscript to also include a solution based on LWE. See Section 1.3 for additional discussion.

the compiler to be *shallow*) allows us to recover the prior results. Along the way, we develop a framework for analyzing composition of compressing encodings, which can be a useful perspective for future research in this area.

1.2 Technical Overview

We now provide a technical outline of our construction and its properties.

Obfuscation via Homomorphic Encryption. The connection between (fully) homomorphic encryption and obfuscation is fairly straightforward. Given a program Π to be obfuscated, we can provide a ciphertext c_Π which encrypts Π under an FHE scheme. This will allow to use homomorphism to derive $c_x = \mathsf{Enc}(\Pi(x))$ for all x. Now all that is needed is a way to decrypt c_x in a way that does not reveal any information on Π. Early works (e.g. [21] and followups) attempted to use this approach and provide a "defective" version of the secret key of the FHE scheme, but a different approach was suggested in [9].

Specifically, [9] considered a homomorphic evaluation that takes c_Π to c_{TT}, an encryption of the *entire truth table* of Π, i.e. to an encryption of a multi-bit value. By relying on prior generic transformations [35], they showed that one can reduce the task of constructing general-purpose obfuscation to the task of computing a "decryption" hint for c_{TT} with the following properties:

- Succinctness: The size of the decryption hint must be sublinear in the size of the truth table $|\mathsf{TT}|$.
- Simulatability: The decryption hint should not reveal any additional information besides the truth table TT.

The reason why this is helpful is that some so-called "packed-encryption" schemes have the property that a short ciphertext-dependent decryption hint suffices in order to decrypt the ciphertext, in a way that does not seem to leak the secret key of the scheme itself. While standard FHE schemes do not natively support packed encryption, it was shown in [8] that it is possible to use the so-called key-switching technique to switch from an FHE scheme into a packed-encryption scheme.

Alas, when instantiating the components of the [9] approach in its simplistic form described above, the decryption hint leaks information that renders the scheme insecure. To counter this issue, [9] proposed to inject another source of randomness: By adding freshly sampled ciphertexts of the packed-encryption scheme (which in their case was instantiated with the Damgård-Jurik scheme [16]) one can smudge the leakage of the decryption hint. However the size of these fresh ciphertext would largely exceed the size of the truth table TT. Therefore, [9] proposed to heuristically sample them from a random oracle, leveraging the fact that the ciphertexts of [16] are *dense*, i.e. a uniformly sampled string lies in the support of the encryption algorithm with all but negligible probability. This led to a candidate, but without a proof of security.

A Provably Secure Scheme. In a recent work, Gay and Pass [23] observed that for the purpose of constructing obfuscation, it suffices to consider schemes in the common random string (CRS) model where, importantly, the size of the CRS can exceed the size of the truth table. This allowed them to place the Damgård-Jurik ciphertexts in the CRS and therefore avoid relying on random-oracle-like heuristics.

They propose a new method to prove the security of this approach: Leveraging the structural property of the GSW scheme [26]. They showed that adding a GSW encryption of 0 to the evaluated FHE ciphertext (before key-switching to Damgård-Jurik) allows one

to program the FHE ciphertext in the security proof. To sample these GSW encryptions of 0, they propose to draw the random coins $\mathbf{r}^*$ again from the CRS and let the evaluator recompute the correct ciphertext $\mathsf{GSW.Enc}(0;\mathbf{r}^*)$.

Taken together, these new ideas allow them to prove their construction secure against the shielded randomness leakage (SRL) security of the resulting FHE scheme. Loosely speaking, SRL security requires that semantic security of an encryption scheme is retained in the presence of an oracle that leaks the randomness $\mathbf{r}_f$ of the homomorphic evaluation of the function f over the challenge ciphertext. However the randomness $\mathbf{r}_f$ is not revealed in plain to the adversary, instead it is "shielded" by the random coins of a fresh GSW ciphertext $c = \mathsf{GSW.Enc}(0;\mathbf{r}^*)$. That is, the adversary is given $(\mathbf{r}_f - \mathbf{r}^*, c)$. In fact, the adversary can obtain polynomially-many samples from this distribution, for any function f, conditioned on the fact that the adversary knows the output of $f(m^*)$, where m^* is the hidden message.

To gain confidence in the veracity of the assumption, [23] show that the GSW encryption scheme satisfies SRL security if the (plain) LWE assumption holds. However, their obfuscation scheme requires one to publish a key cycle of GSW and Damgård-Jurik (i.e. an encryption of the GSW secrey key under Damgård-Jurik and vice versa). Thus their final assumption is that SRL security is retained in the presence of such a key cycle.

Obfuscation from Circular-Secure LWE. We wish to remove the need for the Damgård-Jurik encryption scheme from the above construction paradigm. The major obstacle to overcome consists in designing an LWE-based encryption scheme that simultaneously satisfies three properties.

- Linear Homomorphism: In order to key switch the GSW ciphertext into this form, the scheme must satisfy some weak notion of homomorphism. Specifically, it must support the homomorphic evaluation of linear functions.
- Succinct Randomness: The scheme must allow us to encrypt a long message string with a short randomness, that can then function as the decryption hint.
- Dense Ciphertexts: A uniformly sampled string must lie in the support of the encryption algorithm with all but negligible probability. This will allow us to parse the CRS as a collection of ciphertexts.[2]

Unfortunately all natural lattice-based candidates seem to fail to satisfy all of these properties. In particular, for all LWE-based schemes linear homomorphism seems to be at odds with dense ciphertexts: To ensure that the noise accumulated during the homomorphic evaluation does not impact the decryption correctness, one needs to ensure a gap between the noise bound and the modulus. More concretely, ciphertext are typically of the form $(\mathbf{a}, \mathbf{a}\cdot\mathbf{s}+e+q/2\cdot\mathsf{m}) \in \mathbb{Z}_q^{n+1}$ where $e \ll q$, which makes them inherently sparse.

Our Solution: A Packed Variant of Dual-Regev that is also Dense-Friendly. We show that the above requirements can be relaxed. Our starting point is devising a "packed" version of the dual-Regev encryption scheme [25]. This scheme will not have dense ciphertexts so it does not fit the requirements from previous works. However, we will show how we can define, for the same scheme, a family of ciphertexts which are both "almost dense" and can inter-operate with the non-dense scheme, so as to allow to construct the obfuscator.

[2] Note that for the purpose of constructing the obfuscator, one could make do with a common reference string which can have an arbitrary distribution. However, the string needs to be parsed as a ciphertext with respect to *all* public-keys. Requiring dense ciphertexts is a simple requirement that implies this property.

Let us start with our packed dual-Regev scheme. To pack a k-bit plaintext $\mathbf{m} \in \{0,1\}^k$ in a dual-Regev ciphertext we construct the public key as a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, which is statistically close to uniform but is sampled together with a trapdoor τ (whose role will be explained below), and another uniformly sampled matrix $\mathbf{B} \in \mathbb{Z}_q^{k \times n}$. The encryption algorithm computes a the ciphertext as

$$(\mathbf{A} \cdot \mathbf{r} + \mathbf{e}_0, \mathbf{B} \cdot \mathbf{r} + q/2 \cdot \mathbf{m} + \mathbf{e})$$

where $\mathbf{r} \leftarrow\!\!\$\ \mathbb{Z}_q^n$ is the encryption randomness and the vectors $\mathbf{e}_0$ and $\mathbf{e}$ are the encryption noises, where the norm of both vectors is bounded by some $B \ll q$. The property of the trapdoor τ is that it allows to recover $\mathbf{r}$ from $\mathbf{A} \cdot \mathbf{r} + \mathbf{e}_0$. The (semantic) security of the scheme follows directly by definition of LWE. To decrypt, therefore, one can first use the trapdoor τ to recover $\mathbf{r}$ from the first m elements of the ciphertext, and then recompute the mask $\mathbf{B} \cdot \mathbf{r}$ and recover each individual bit by rounding to the closest multiple of $q/2$. Setting the parameters appropriately, we can guarantee that the decryption is always successful. One important property of this scheme is that the random coins $\mathbf{r} \in \mathbb{Z}_q^n$ are sufficient to recover the entire message and furthermore the size of $\mathbf{r}$ is succinct (in particular independent of k).

In terms of homomorphism, the scheme is straightforwardly additively homomorphic. Furthermore, it supports key switching from any scheme with almost-linear decryption as per [8].[3] In particular it is possible to take a (long) message encrypted under an FHE scheme such as GSW and convert it to an encryption of the same message under packed dual-Regev, using precomputed key-switching parameters.[4]

As explained above, this scheme does not have dense ciphertexts. At this point we make two crucial observations that will allow us to bypass this hurdle.

(1) In order to construct the obfuscator using the [9] approach, dense ciphertexts only need to enjoy a very limited form of homomorphism, they only need to support a single addition with a non-dense ciphertext.

This is essentially because the obfuscator has the following outline. It starts by considering the dense ciphertext from the CRS (or oracle in the case of the original [9]), and homomorphically bootstraps it into a non-dense FHE ciphertext by evaluating the decryption circuit. Let $\mathbf{m}$ be the (random) message that is induced by the process. Then, the FHE encryption of $\mathbf{m}$ is processed in order to create a non-dense *packed* encryption of $\mathbf{m} \oplus \mathsf{TT}$, where TT is the truth table of the program to be obfuscated (or, more accurately, a chunk of this truth table, partitioning into chunks is required in order to allow reusability of the keys). Then a single homomorphic addition between the dense and non-dense ciphertext would imply a packed encryption of the truth table. All of this can be performed by the evaluator of the obfuscated program, so all that is needed is the decryption hint for this final ciphertext, that would allow to recover TT.

We note importantly, that in prior approaches (including the [23] blueprint) the aforementioned bootstrapping creates a key cycle, since a packed ciphertext is bootstrapped into an FHE ciphertext, which is afterwards key-switched into a packed ciphertext. However, we notice that it suffices to provide an encryption of the (succinct) *randomness* of the dense ciphertext in order to apply bootstrapping, thus leading to a relaxed key-randomness circular assumption. Interestingly, this observation is not very useful for actual dense ciphertexts (since finding the randomness would require using the key), however, our relaxed notion of density described below will allow to apply it and thus relax the circularity notion as well.

---

[3] This is done using the by-now-standard technique of encrypting powers-of-two of the elements of the secret key of the latter scheme, so that it is possible to evaluate any inner product homomorphically.

[4] We note that the key switching parameters are quite long so it is required for our method that they are reusable.

(2) A notion of *almost-everywhere* density suffices. A ciphertext distribution is almost-everywhere dense if it is dense except for a non-dense part whose length is independent of k (the message length).

The reason that this is sufficient is that the non-dense part of the ciphertext, which we refer to as the *header*, can be generated by the obfuscator and provided to the evaluator as a part of the obfuscated program. Since the header is short, and in particular the message length k can be selected to be much longer than the header, the effect on the length of the obfuscated program will be minimal. As hinted above, since the obfuscator generates the header, it in particular also samples the randomness for the final almost-everywhere dense ciphertext. This means that the obfuscator can generate the bootstrapping parameters using this randomness without requiring a key cycle.

Dense Encryption Mode. With these observations in mind we describe an alternative encryption mode ($\mathsf{DenseEnc}$) for the packed variant of dual-Regev where the bulk of the ciphertext is dense. On input a message $\mathbf{m} \in \{0,1\}^k$, the encryption algorithm in dense mode computes the following ciphertext

$$(\mathbf{A} \cdot \mathbf{r} + \mathbf{e}_0, \mathbf{B} \cdot \mathbf{r} + q/2 \cdot \mathbf{m} + \mathbf{u})$$

where $\mathbf{r}$ and $\mathbf{e}_0$ are sampled as before and $\mathbf{u} \leftarrow\!\!\$\ [-q/4, +q/4]^k$. For convenience, we are going to split the ciphertexts into two blocks: The header $\mathbf{h}_0 \in \mathbb{Z}_q^m$ and the message carrier $(h_1, \ldots, h_k) \in \mathbb{Z}_q^k$. Foremost, observe that the decryption algorithm as described before still returns the correct message with probability 1, since it recovers the same $\mathbf{r}$ from $\mathbf{h}_0$. Furthermore, note that (for a fixed header) all vectors $(h_1, \ldots, h_k) \in \mathbb{Z}_q^k$ are in the support of the encryption algorithm. Since $k \gg m$, most of the elements of the ciphertext in the alternative encryption mode are dense.

One can verify that the aforementioned limited form of homomorphism indeed holds, namely that

$$\mathsf{dR.Enc}(\mathbf{m}) + \mathsf{dR.DenseEnc}(\mathbf{m}') \in \mathsf{dR.DenseEnc}(\mathbf{m} \oplus \mathbf{m}').$$

This is the case since

$$\begin{aligned}
&(\mathbf{A} \cdot \mathbf{r} + \mathbf{e}_0, \mathbf{B} \cdot \mathbf{r} + q/2 \cdot \mathbf{m} + \mathbf{e}) + (\mathbf{A} \cdot \mathbf{r}' + \mathbf{e}_0', \mathbf{B} \cdot \mathbf{r}' + q/2 \cdot \mathbf{m}' + \mathbf{u}) \\
&= (\mathbf{A} \cdot (\mathbf{r} + \mathbf{r}') + \mathbf{e}_0 + \mathbf{e}_0', \mathbf{B} \cdot (\mathbf{r} + \mathbf{r}') + q/2 \cdot (\mathbf{m} \oplus \mathbf{m}') + \mathbf{e} + \mathbf{u}) \\
&= (\mathbf{A} \cdot \tilde{\mathbf{r}} + \tilde{\mathbf{e}}_0, \mathbf{B} \cdot \tilde{\mathbf{r}} + q/2 \cdot (\mathbf{m} \oplus \mathbf{m}') + \tilde{\mathbf{u}})
\end{aligned}$$

where $\tilde{\mathbf{u}} = \mathbf{e} + \mathbf{u} \in [-q/4, +q/4]^k$ with all but negligible probability over the random choice of $\mathbf{u}$, for an appropriate choice of the parameters.

Doing Away with the Header. We notice that given our two observations above, the goal of the header in the obfuscation scheme is quite minimal. The header is not needed for homomorphism, and is only needed for the purpose of extracting the randomness $\mathbf{r}$ at decryption time. We then observe that decrypting packed ciphertext is done in two contexts in the scheme. The first is when we bootstrap the almost-everywhere dense ciphertext into an FHE ciphertext, and the other is when the evaluator of the obfuscated program recovers TT from the final ciphertext. For the latter there is no need for a header since the decryption hint, i.e. the respective $\mathbf{r}$ value, is provided within the obfuscated program. For the former we do not need a header of a specific structure, but rather simply an encryption of $\mathbf{r}$ that allows bootstrapping the almost-dense ciphertext. It therefore suffices to provide $\mathsf{GSW.Enc}(\mathbf{r})$ directly, which makes the header completely redundant.

On the Assumption. Equipped with the newly developed packed version of dual-Regev we can follow the [9, 23] approach, with the aforementioned modifications, to construct the obfuscator. The resulting construction can be shown secure against the assumption that the SRL security of GSW is retained in the presence of a key cycle with the packed dual-Regev encryption scheme as presented above.

We then observe that it suffices to assume SRL security with respect to key-randomness cycles, rather than key cycles. We note that this assumption is no-stronger than key-cycle SRL since given a key-cycle it is possible to homomorphically generate a key-randomness cycle, but the converse is not known to be true.

Adding this to our observation about the redundancy of the header, the assumption we require is that SRL security is retained in the presence of a key-randomness cycle between GSW and packed dual-Regev, i.e.

$$(\mathsf{GSW.Enc}(\mathbf{r}), \mathsf{dR.Enc}(\mathsf{sk}_{\mathsf{GSW}}; \mathbf{r}))\,.$$

Since dual-Regev is randomness recoverable, this assumption is syntactically weaker than SRL security in the presence of a key-cycle: Given a GSW encryption of the dual-Regev secret key, one can homomorphically compute the randomness recovery circuit to obtain a GSW encryption of the randomness $\mathbf{r}$.

1.3 Related and Follow-up Work

Subsequently to the posting of this manuscript online (but concurrently and independently) [23] updated their manuscript to include a solution based on LWE in the place of DCR. They do not make the observations that a relaxed notion of density suffices (and is preferable) and thus they explicitly construct an encryption scheme with dense ciphertexts based on the (primal) Regev encryption scheme. The resulting scheme is more involved and in particular requires the two-key circular SRL security of GSW and (primal) Regev rather than the relaxed key-randomness circularity notion.

Wee and Wichs [42], again concurrently, presented another instantiation of the [9] approach which is arguably post-quantum secure. They rely on an indistinguishability assumption between two distributions and not directly on circular security. However, the underlying machinery developed shares many similarities with our approach. Specifically, while we essentially rely on randomness that is embedded in the CRS by interpreting it as an obliviously sampled ciphertext (which thus corresponds to one encrypted with fresh randomness), their approach is to use a pseudorandom function to transform the CRS into a randomizer for the output hint.

A follow-up work by Hopkins, Jain, and Lin [29] shows counterexamples to SRL security for general functions in the presence of a 2-key cycle, as stated in [23], and the conjecture from [42]. We stress that their findings do not imply an attack against the corresponding obfuscation scheme of [42] and [23] (as also pointed out by the authors in [29]). Rather, their results show that the veracity of SRL security depends on the concrete circuit representation of the functions under consideration. As a consequence of their findings, we updated the statement of our assumption (and adapted the analysis of our scheme) with a refined version, that further restricts the power of the adversary and more tightly characterize the security of our construction. However, the iO construction is unchanged from previous versions of this work.

A few remarks about the susceptibility of our scheme to the [29] attack are in order. In short, the attack exploits the randomness homomorphism of GSW to compute a biased leakage. The SRL function consists of a bootstrapping followed by a modular reduction (modulo 2). On the other hand, our admissible class of leakage functions consists of linear

functions (modulo q) followed by a rounding (i.e. outputting the most significant bit). We are not aware of a method to establish the same correlations exploited by [29] without violating the admissibility criteria for the leakage functions. Thus, we conjecture that SRL security with respect to such leakage function holds for all natural FHE candidates (see Section 3.1 for further details).

2 Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter. We say that a function negl is negligible if it vanishes faster than any inverse polynomial. Given a set S, we denote by $s \leftarrow\!\$\ S$ the uniform sampling from S. We say that an algorithm is PPT if it can be implemented by a probabilistic Turing machine M running in time $\mathsf{poly}(\lambda)$. The execution of a Turing machine M on input x and with random coins fixed to r is denoted by $M(x;r)$. We say that two distributions (D_0, D_1) are computationally (statistically, resp.) indistinguishable if for all PPT (unbounded, resp.) distinguishers, the probability to tell D_0 an D_1 apart is negligible. Matrices are denoted by $\mathbf{M}$ and vectors are denoted by $\mathbf{v}$. For convenience, we define $\mathbf{Bit}(\cdot)$ as the bit decomposition operation. We denote the infinity norm of a vector $\mathbf{v}$ by $\|\mathbf{v}\|_\infty$. We recall the smudging lemma.

▶ **Lemma 2** (Smudging). *Let $B_1 = B_1(\lambda)$ and $B_2 = B_2(\lambda)$ be positive integers and let $e_1 \in [-B_1, B_1]$ be a fixed integer. Let $e_2 \leftarrow\!\$\ [-B_2, B_2]$ chosen uniformly at random. Then the distribution of e_2 is statistically indistinguishable to that of $e_2 + e_1$ as long as $B_1/B_2 = \mathsf{negl}(\lambda)$.*

2.1 Indistinguishability Obfuscation

We recall the notion of indistinguishability obfuscation (iO) from [4].

▶ **Definition 3** (Indistinguishability Obfuscation). *A PPT machine iO is an indistinguishability obfuscator for a circuit class $\{\mathfrak{C}_\lambda\}_{\lambda\in\mathbb{N}}$ if the following conditions are satisfied:*

(Functionality) For all $\lambda \in \mathbb{N}$, all circuit $\Pi \in \mathfrak{C}_\lambda$, all inputs x it holds that: $\tilde{\Pi}(x) = \Pi(x)$, where $\tilde{\Pi} \leftarrow\!\$\ \mathsf{iO}(\Pi)$.

(Indistinguishability) For all $\lambda \in \mathbb{N}$, all pairs of circuit $(\Pi_0, \Pi_1) \in \mathfrak{C}_\lambda$ such that $|\Pi_0| = |\Pi_1|$ and $\Pi_0(x) = \Pi_1(x)$ on all inputs x, it holds that the following distributions are computationally indistinguishable: $\mathsf{iO}(\Pi_0) \approx \mathsf{iO}(\Pi_1)$.

Shallow XiO. In this work we construct a weaker version of iO called (shallow) XiO, which however is sufficient (along with the LWE assumption) to construct fully-fledged iO. Loosely speaking, a shallow XiO is a indistinguishability obfuscator (with pre-processing) for $\mathsf{P}^{\mathsf{log}}/\mathsf{poly}$ with non-trivial efficiency. Here $\mathsf{P}^{\mathsf{log}}/\mathsf{poly}$ denotes the class of polynomial-size circuits with inputs of length $\eta = O(\log(\lambda))$ and by non-trivial efficiency we mean that the size of the obfuscated circuit is bounded by $\mathsf{poly}(\lambda, |\Pi|) \cdot 2^{\eta\cdot(1-\varepsilon)}$, for some constant $\varepsilon > 0$. The runtime of the obfuscator can be any polynomial in λ, $|\Pi|$, and 2^η, except that its depth should not depend on 2^η. Furthermore, we allow the obfuscator to access a large uniform random string (the pre-processing) of size even larger than the truth table of the circuit. For a formal statement, we refer the reader to the full version [10].

2.2 The GSW Fully-Homomorphic Encryption

In the following we briefly recall the encryption scheme by Gentry, Sahai, and Waters [26] (henceforth, GSW). We denote by $n = n(\lambda)$ the lattice dimension and by $q = q(\lambda)$ the modulus (which we assume for simplicity to be even). Throughout the rest of this paper, we set $m = (n+1)(\log(q)+1)$ and $d = d(\lambda)$ as a bound on the depth of the arithmetic circuit to be evaluated.

KeyGen(1^λ): Sample a uniform matrix $\mathbf{A} \leftarrow\!\$\ \mathbb{Z}_q^{n\times m}$ and a vector $\mathbf{s} \leftarrow\!\$\ \chi^n$. Set the public key to $(\mathbf{A}, \mathbf{b} = \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$, where $\mathbf{e} \leftarrow\!\$\ \chi^m$. The secret key is set to $(-\mathbf{s}, 1)$.

Enc(pk, m): On input a message $\mathsf{m} \in \{0,1\}$, sample a uniform $\mathbf{R} \leftarrow\!\$\ \{0,1\}^{m\times m}$ and compute

$$\mathbf{C} = (\mathbf{A}, \mathbf{b}) \cdot \mathbf{R} + \mathsf{m} \cdot \mathbf{G}$$

where $\mathbf{G} = (1, 2, \ldots, 2^{\log(q)-1})^T \otimes I_{(n+1)}$ and $I_{(n+1)} \in \{0,1\}^{(n+1)\times(n+1)}$ denotes the identity matrix.

Eval(pk, Π, $(c_1, \ldots, c_\mu)$): There exists a (deterministic) polynomial-time algorithm that allows one to compute any d-bounded depth arithmetic circuit $\Pi : \{0,1\}^n \to \{0,1\}$ homomorphically over a vector of ciphertexts $(c_1, \ldots, c_\mu)$. For details about this algorithm, we refer the reader to [26]. For the purpose of this work, the only relevant information is that the evaluated ciphertext $\mathbf{c}_\Pi \in \mathbb{Z}_q^{(n+1)}$ is an $(n+1)$-dimensional vector. For multiple bits of output, the resulting ciphertext is defined to be the concatenation of the single-bit ciphertexts.

Dec(sk, c): We assume without loss of generality that the input ciphertext $\mathbf{c} \in \mathbb{Z}_q^{(n+1)}$ is the output of the evaluation algorithm. Such a ciphertext defines a linear function $\boldsymbol{\ell}_\mathbf{c}$ such that

$$\boldsymbol{\ell}_\mathbf{c}(\mathsf{sk}) = q/2 \cdot \mathsf{m} + e$$

where $|e| \le \hat{B} = (m+1)^d m B$. The message m is recovered by returning the most significant bit of the output.

Note that the decryption routine of GSW consists of the application of a linear function, followed by a rounding and we refer to this property as to *almost-linear* decryption. In a slight abuse of notation, we sometimes write $\mathsf{KeyGen}(1^\lambda; q)$ to denote the above key generation algorithm with a fixed modulus q.

Alternate Encryption. For convenience we also define a modified encryption algorithm, where the output ciphertexts consists of a single column vector. An additional difference is that we sample the randomness with norm $\tilde{B} = 2^\lambda \cdot \hat{B}$.

ColEnc(pk, m): On input a message m, sample a uniform $\mathbf{r} \leftarrow\!\$\ [-\tilde{B}, +\tilde{B}]^m$ and compute

$$\mathbf{c} = (\mathbf{A}, \mathbf{b}) \cdot \mathbf{r} + (0^n, q/2) \cdot \mathsf{m}.$$

This algorithm is going instrumental for our scheme, although ciphertexts in this form no longer support the homomorphic evaluation of arbitrary circuits. The multi-bit version of such an algorithm is defined accordingly to output the concatenation of independently sampled ciphertexts. We now recall a useful Lemma from [23].

▶ **Lemma 4** (GSW Smudging). *Let* $\tilde{B} = 2^\lambda \cdot \hat{B}$. *For all* $\lambda \in \mathbb{N}$, *for all* $(\mathsf{sk}, \mathsf{pk})$ *in the support of* $\mathsf{KeyGen}(1^\lambda)$, *for all messages* $\mathbf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_\mu)$, *for all depth-*$d$ *circuit* $(\Pi_1, \ldots, \Pi_\tau)$, *the following distributions are statistically indistinguishable*

$$\left(\begin{array}{l} c_1, \ldots, c_\mu, \mathbf{r}_1^*, \ldots, \mathbf{r}_\tau^*, \\ \mathsf{Eval}(\mathsf{pk}, \Pi_1, (c_1, \ldots, c_\mu)) + \mathsf{ColEnc}(\mathsf{pk}, 0; \mathbf{r}_1^*), \ldots, \\ \mathsf{Eval}(\mathsf{pk}, \Pi_\tau, (c_1, \ldots, c_\mu)) + \mathsf{ColEnc}(\mathsf{pk}, 0; \mathbf{r}_\tau^*) \end{array}\right)$$
$$\approx \left(\begin{array}{l} c_1, \ldots, c_\mu, \mathbf{r}_1^* - \mathsf{RandEval}(\mathsf{pk}, \Pi_1, \mathbf{m}, (\mathbf{R}_1, \ldots, \mathbf{R}_\mu)), \ldots, \\ \mathbf{r}_\tau^* - \mathsf{RandEval}(\mathsf{pk}, \Pi_\tau, \mathbf{m}, (\mathbf{R}_1, \ldots, \mathbf{R}_\mu)), \\ \mathsf{ColEnc}(\mathsf{pk}, \Pi_1(\mathsf{m}_1, \ldots, \mathsf{m}_\mu); \mathbf{r}_1^*), \ldots, \mathsf{ColEnc}(\mathsf{pk}, \Pi_\tau(\mathsf{m}_1, \ldots, \mathsf{m}_\mu); \mathbf{r}_\tau^*) \end{array}\right)$$

where $c_i \leftarrow\!\$\ \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_i; \mathbf{R}_i)$, $\mathbf{r}_i^* \leftarrow\!\$\ [-\tilde{B}, +\tilde{B}]^m$, *and* $\mathbf{R}_i \leftarrow\!\$\ \{0,1\}^{m \times m}$.

Randomness Homomorphism. We recall a useful property of the GSW scheme, namely that one can alternatively evaluate functions directly over the randomness of a ciphertext to obtain the same result. More formally, we say that a homomorphic encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ has randomness homomorphism for the circuit class $\{\mathfrak{C}_\lambda\}_{\lambda \in \mathbb{N}}$ if there exists an efficient algorithm $\mathsf{RandEval}$ such that for all $\Pi \in \mathfrak{C}_\lambda$, all $(\mathsf{sk}, \mathsf{pk})$ in the support of KeyGen, all vectors of messages $\mathbf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_\mu)$ and $\mathbf{R} = (\mathbf{R}_1, \ldots, \mathbf{R}_\mu)$, all ciphertexts $(c_1, \ldots, c_\mu)$ in the support of $(\mathsf{Enc}(\mathsf{pk}, \mathsf{m}_1; \mathbf{R}_1), \ldots, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_\mu; \mathbf{R}_\mu))$ it holds that

$$\mathsf{Eval}(\mathsf{pk}, \Pi, (c_1, \ldots, c_\mu)) = \mathsf{ColEnc}(\mathsf{pk}, \Pi(\mathbf{m}); \mathsf{RandEval}(\mathsf{pk}, \Pi, \mathbf{m}, \mathbf{R})).$$

Circuit Privacy. It is well known that the GSW encryption scheme satisfies the following notion of circuit privacy [7, 18, 40] (with a randomized evaluation algorithm).

▶ **Definition 5** (Circuit Privacy). *For all* $\lambda \in \mathbb{N}$, *all all* $\Pi \in \mathfrak{C}_\lambda$, *all* $(\mathsf{sk}, \mathsf{pk})$ *in the support of* KeyGen, *and all messages* m, *it holds that the following distributions are statistically indistinguishable*

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, \Pi(\mathsf{m}))) \approx (\mathsf{pk}, \mathsf{Eval}(\mathsf{pk}, \Pi, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}); r)).$$

where $r \leftarrow\!\$\ \{0,1\}^\lambda$.

3 Packed Encryption from LWE

In the following we describe a packed version of the dual-Regev encryption scheme [25]. We denote by $n = n(\lambda)$ the lattice dimension, by $q = q(\lambda)$ the modulus (which we assume for simplicity to be a power of 2), and by $k = k(\lambda)$ the expansion factor. We require the existence of a public-key encryption scheme $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$.

$\mathsf{KeyGen}(1^\lambda, 1^k)$: Sample a uniform $k \times n$ matrix $\mathbf{B} \leftarrow\!\$\ \mathbb{Z}_q^{k \times n}$ and a key pair of a public-key encryption scheme $(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{pk}_{\mathsf{PKE}}) \leftarrow\!\$\ \mathsf{PKE.KeyGen}(1^\lambda)$. The public key consists of $(\mathbf{B}, \mathsf{pk}_{\mathsf{PKE}})$ and the secret key is set to $\mathsf{sk}_{\mathsf{PKE}}$.

$\mathsf{Enc}(\mathsf{pk}, \mathbf{m})$: To encrypt a k-bit message $\mathbf{m} \in \{0,1\}^k$, sample a uniform randomness vector $\mathbf{r} \leftarrow\!\$\ \mathbb{Z}_q^n$ a noise vector $\mathbf{e} \leftarrow\!\$\ \chi^k$ and return the ciphertext

$$\mathbf{c} = (\mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, \mathbf{r}), \mathbf{B} \cdot \mathbf{r} + q/2 \cdot \mathbf{m} + \mathbf{e}).$$

$\mathsf{Dec}(\mathsf{sk}, \mathbf{c})$: Parse $\mathbf{c}$ as $(c_{\mathsf{PKE}}, c_1, \ldots, c_k)$ and recover the random coins by decrypting $\mathbf{r} = \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, c_{\mathsf{PKE}})$. Let $\mathbf{b}_i$ be the i-th row of $\mathbf{B}$. For $i = 1 \ldots k$, compute $\mathsf{m}_i = \mathsf{Round}(c_i - \mathbf{b}_i \cdot \mathbf{r})$, where Round rounds to the nearest multiple of $q/2$, i.e. it returns 1 if the input is closer to $q/2$ and 0 otherwise. Output $\mathbf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_k)$.

Clearly, the scheme is perfectly correct since

$$\begin{aligned}
&(\mathsf{Round}(c_1 - \mathbf{b}_1 \cdot \mathbf{r}), \ldots, \mathsf{Round}(c_k - \mathbf{b}_k \cdot \mathbf{r})) \\
&= (\mathsf{Round}(q/2 \cdot \mathsf{m}_1 + e_1), \ldots, \mathsf{Round}(q/2 \cdot \mathsf{m}_k + e_k)) \\
&= (\mathsf{Round}(q/2 \cdot \mathsf{m}_1), \ldots, \mathsf{Round}(q/2 \cdot \mathsf{m}_k)) \\
&= (\mathsf{m}_1, \ldots, \mathsf{m}_k) \\
&= \mathbf{m}.
\end{aligned}$$

Extended Encryption. It is not hard to see that the scheme presented above is (bounded) additively homomorphic over $\mathbb{Z}_2^k$. To lift the class of computable functions to all linear functions over $\mathbb{Z}_q^k$, we adopt the standard trick of encrypting the message multiplied by all powers of two $(1, 2, \ldots, 2^{\log(q)})$. For convenience, we define the following augmented encryption algorithm.

$\underline{\mathsf{ExtEnc}(\mathsf{pk}, \mathbf{m}):}$ On input an ℓ-dimensional message $\mathbf{m} \in \mathbb{Z}_q^\ell$, let $\mathbf{g} = (1, 2, \ldots, 2^{\log(q)-1})^T$ and define

$$\mathbf{M} = \begin{bmatrix} \mathsf{m}_1 \cdot \mathbf{g} & \mathsf{m}_2 \cdot \mathbf{g} & \ldots & 0^{\log(q)} \\ 0^{\log(q)} & 0^{\log(q)} & \ldots & 0^{\log(q)} \\ \vdots & \vdots & \ddots & \vdots \\ 0^{\log(q)} & 0^{\log(q)} & \ldots & \mathsf{m}_\ell \cdot \mathbf{g} \end{bmatrix} \in \mathbb{Z}_q^{k \times \ell \cdot k \cdot \log(q)}.$$

Sample a uniform randomness matrix $\mathbf{R} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times \ell \cdot k \cdot \log(q)}$ and a uniform noise matrix $\mathbf{E} \leftarrow\!\!\$\ \chi^{k \times \ell \cdot k \cdot \log(q)}$. Compute

$$\mathbf{C} = \mathbf{B} \cdot \mathbf{R} + \mathbf{M} + \mathbf{E}$$

and return the ciphertext $(\mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, \mathbf{R}), \mathbf{C})$.

Decryption works, as before, by recovering $\mathbf{R}$ from the public-key encryption scheme and then decrypting $\mathbf{m}$ component-wise.

Almost-Everywhere Dense Encryption. For convenience, we also define an alternative encryption algorithm in the following. Note that the encryption algorithm does not take as input any message, instead it encrypts a uniform k-bit binary vector. Syntactically, this is the equivalent of a key-encapsulation mechanism.

$\underline{\mathsf{DenseEnc}(\mathsf{pk}):}$ Sample a uniform randomness vector $\mathbf{r} \leftarrow\!\!\$\ \mathbb{Z}_q^n$ and return the ciphertext

$$\mathbf{c} = (c_{\mathsf{PKE}}, c_1, \ldots, c_k) = (\mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, \mathbf{r}), \mathbf{B} \cdot \mathbf{r} + \mathbf{u}).$$

where $\mathbf{u} \leftarrow\!\!\$\ \mathbb{Z}_q^k$.

We highlight two facts about this algorithm that are going to be important for our later construction: (i) The decryption algorithm works for both Enc and $\mathsf{DenseEnc}$ algorithms, where the plaintext of $\mathsf{DenseEnc}$ corresponds to $(\mathsf{Round}(u_1), \ldots, \mathsf{Round}(u_k))$. In fact, the scheme satisfies perfect correctness in both cases. (ii) The domain of the elements $(c_1, \ldots, c_k)$ is *dense*, i.e. the support of the scheme spans the entire vector space $\mathbb{Z}_q^k$. Since the element c_{PKE} is small (i.e. independent of k) for an appropriate choice of the public-key encryption scheme, we refer to such a property as *almost-everywhere* density.

Semantic Security. We argue that the scheme satisfies a strong form of semantic security, i.e. the honestly computed ciphertexts are computationally indistinguishable from uniform vectors in $\mathbb{Z}_q^k$. Semantic security for the extended encryption ExtEnc and the dense encryption $\mathsf{DenseEnc}$ follows along the same lines.

▶ **Theorem 6** (Semantic Security). *If* $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is semantically secure and the LWE assumption holds, then for all* $\lambda \in \mathbb{N}$ *and all messages* m *it holds that the following distributions are computationally indistinguishable*

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m})) \approx (\mathsf{pk}, \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, \mathbf{z}), \mathbf{u}).$$

where $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\$\ \mathsf{KeyGen}(1^\lambda, 1^k)$, $\mathbf{z} \leftarrow\!\$\ \mathbb{Z}_q^n$, *and* $\mathbf{u} \leftarrow\!\$\ \mathbb{Z}_q^k$.

Proof. The security of the scheme follows routinely by an invocation of semantic security of the public-key encryption scheme and an invocation of the LWE assumption. ◀

3.1 Key-Randomness SRL Security

We state a version of SRL security [23] tailored for our specific instance and adapted to the randomness-key circularity assumption (rather than the 2-key circularity, as stated in [23]).

▶ **Definition 7** (Key-Randomness SRL Security). *Let* $(\mathsf{GSW.KeyGen}, \mathsf{GSW.Enc}, \mathsf{GSW.Eval}, \mathsf{GSW.Dec})$ *be the GSW encryption scheme and* $(\mathsf{dR.KeyGen}, \mathsf{dR.Enc}, \mathsf{dR.Eval}, \mathsf{dR.Dec})$ *be the packed dual-Regev encryption scheme. Fix messages* $(\mathsf{m}_0, \mathsf{m}_1)$, *polynomials* $\tau = \tau(\lambda)$ *and* $k = k(\lambda)$ *and an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. *Consider the following experiment.*

$\mathsf{Exp}^{(b)}_{\mathsf{SRL}}(\mathcal{A})$*:*

- *Sample* $(\bar{\mathsf{sk}}, (\bar{\mathsf{pk}}, \mathbf{B})) \leftarrow\!\$\ \mathsf{dR.KeyGen}(1^\lambda, 1^k)$ *and* $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\$\ \mathsf{GSW.KeyGen}(1^\lambda)$
- *Compute* $\mathbf{c} \leftarrow\!\$\ \mathsf{GSW.Enc}(\mathsf{pk}, \mathsf{m}_b)$
- $\{\mathbf{P}_i, \mathbf{p}_i\}_{i=1\ldots\tau} = \mathcal{A}_1(\mathsf{pk}, \mathbf{B}, \mathbf{c})$
- *Compute* $\left(\bar{c}_{\mathsf{sk}}, \bar{\mathbf{C}}_{\mathsf{sk}}\right) = \mathsf{dR.ExtEnc}(\bar{\mathsf{pk}}, \mathsf{sk}; \mathbf{S})$ *and* $\mathbf{c}_{\mathbf{S}} \leftarrow\!\$\ \mathsf{GSW.Enc}(\mathsf{pk}, \mathbf{S}; \mathbf{R}_{\mathbf{S}})$
- *For all* $i = 1 \ldots \tau$*:*
  - *Sample* $\mathbf{C}_i^* = \mathsf{ColEnc}(0; \mathbf{r}_i^*)$ *where* $\mathbf{r}_i^* \leftarrow\!\$\ [-\tilde{B}, +\tilde{B}]^{m \cdot k}$
  - *Sample* $\mathbf{t}_i \leftarrow\!\$\ \mathbb{Z}_q^n$
  - *Sample* $\mathbf{c}_{i,\mathbf{r}} = \mathsf{GSW.Enc}(\mathsf{pk}, \mathbf{S} \cdot \mathsf{Bit}(\ell_i) + \mathbf{t}_i; \mathbf{R}_i) \leftarrow\!\$\ \mathsf{GSW.Eval}(\mathsf{pk}, \cdot\mathsf{Bit}(\ell_i) + \mathbf{t}_i, \mathbf{c}_{\mathbf{S}})$
- *Output* $\mathcal{A}_2(\bar{\mathbf{C}}_{\mathsf{sk}}, \{\mathbf{c}_{i,\mathbf{r}}, \mathbf{C}_i^*, \mathbf{u}_i, \mathbf{t}_i, \mathbf{r}_{\psi,i} - \mathbf{r}_i^*\}_{i=1\ldots\tau})$

Here, letting ℓ_i *be the linear function associated with* $\mathbf{C}_i^* + \mathbf{P}_i + q/2 \cdot \mathbf{v}_i$, *we set*

$$\mathbf{r}_{\psi,i} = \mathsf{RandEval}(\mathsf{pk}, \psi_i, \mathbf{S} \cdot \mathsf{Bit}(\ell_i) + \mathbf{t}_i, \mathbf{R}_i) \text{ and}$$
$$\psi_i(\mathbf{Z}) = \mathsf{Round}\left(\mathbf{B} \cdot \mathbf{t}_i + q/2 \cdot \mathbf{p}_i + \mathbf{w}_i - \bar{\mathbf{C}}_{\mathsf{sk}} \cdot \mathsf{Bit}(\ell_i) - \mathbf{B} \cdot \mathbf{Z}\right)$$

where $\mathbf{w}_i \leftarrow\!\$\ [-q/4, q/4]^k$, $\mathbf{v}_i \leftarrow\!\$\ \{0,1\}^k$, *and* $\mathbf{u}_i = \mathbf{w}_i + q/2 \cdot \mathbf{v}_i$, *for all* $i = 1 \ldots \tau$. *We say that an adversary* $\mathcal{A}$ *is admissible if for all* $i = 1 \ldots \tau$ *it holds that* $\mathbf{P}_i \in \mathsf{GSW.Enc}(\mathsf{pk}, \mathbf{p}_i)$. *The KR-SRL assumption conjectures that it holds for all admissible PPT adversaries* $\mathcal{A}$, *all messages* $(\mathsf{m}_0, \mathsf{m}_1)$ *and all polynomials* $\tau = \tau(\lambda)$ *and* $k = k(\lambda)$ *that*

$$|\Pr[\mathsf{Exp}^{(1)}_{\mathsf{SRL}}(\mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{(0)}_{\mathsf{SRL}}(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Since the SRL leakage depends on the specific circuit representation of the functions $(\psi_1 \ldots \psi_\tau)$, we propose a natural implementation for a class of functions that suffices to capture all possible leakage functions. Specifically, observe that ψ_i consist of a linear function (computed over $\mathbb{Z}_q$) followed by a rounding to the nearest multiple of $q/2$. Since all inputs are bit-wise encrypted the computation of modular additions (and multiplication by constants) is done via a canonical boolean circuit (see [11] for a concrete example) and the rounding is obtained by simply returning the ciphertext containing the most significant bit of the output.

4 Constructing (Shallow) XiO

In the following we present the construction of shallow XiO from the GSW scheme $(\mathsf{GSW.KeyGen}, \mathsf{GSW.Enc}, \mathsf{GSW.Eval}, \mathsf{GSW.Dec})$ and the packed version of the dual-Regev encryption $(\mathsf{dR.KeyGen}, \mathsf{dR.Enc}, \mathsf{dR.Eval}, \mathsf{dR.Dec})$ as described in Section 3.

4.1 Construction

The scheme assumes a long uniform string that is, for convenience, split in two chunks:

- A sequence of randomization vectors $(\mathbf{r}^*_1, \ldots, \mathbf{r}^*_{2^{\eta-\log(k)}})$ for the GSW scheme $\mathsf{GSW.PubCoin}$, where each $\mathbf{r}^*_i = (\mathbf{r}^*_{i,1}, \ldots, \mathbf{r}^*_{i,k}) \in [-\tilde{B}, +\tilde{B}]^{m\cdot k}$.
- A sequence of dense ciphertexts $(h_1, \ldots, h_{2^{\eta-\log(k)}})$ for packed dual-Regev scheme $\mathsf{dR.PubCoin}$, where each $h_i = (h_{i,1}, \ldots, h_{i,k}) \in \mathbb{Z}_q^k$.

On input the security parameter 1^λ and the circuit $\Pi : \{0,1\}^\eta \to \{0,1\}$, the obfuscator proceeds as follows.

Setting the Public Keys: Sample a dual-Regev key pair $(\bar{\mathsf{sk}}, \bar{\mathsf{pk}}) \leftarrow\!\!\$\ \mathsf{dR.KeyGen}(1^\lambda, 1^k)$ and GSW key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{GSW.KeyGen}(1^\lambda; q)$, where q is the modulus defined by the dual-Regev scheme. Compute a bit-by-bit GSW encryption $\mathbf{c}_\Pi \leftarrow\!\!\$\ \mathsf{GSWEnc}(\mathsf{pk}, \Pi)$ of the binary representation of the circuit Π.

Compute a Key Encryption: Compute a dual-Regev extended encryption of the GSW secret key $\left(\bar{c}_{\mathsf{sk}}, \bar{\mathbf{C}}_{\mathsf{sk}}\right) = \mathsf{dR.ExtEnc}(\bar{\mathsf{pk}}, \mathsf{sk}; \mathbf{S})$. where $\mathsf{sk} \in \mathbb{Z}_q^{n+1}$ and $\mathbf{S} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times\log(q)\cdot k\cdot(n+1)}$.

Decryption Hints: For all indices $i \in \{0,1\}^{\eta-\log(k)}$, do the following.

Evaluate the Circuit: Let $\Phi_{i,j} : \{0,1\}^{|\Pi|} \to \{0,1\}$ be the universal circuit that, on input a circuit description Π, returns the j-th bit of the i-th block (where each block consists of k bits) of the corresponding truth table. Compute

$$\mathbf{C}_i = \begin{bmatrix} \mathsf{GSW.Eval}(\mathsf{pk}, \Phi_{i,1}, \mathbf{c}_\Pi) \\ \cdots \\ \mathsf{GSW.Eval}(\mathsf{pk}, \Phi_{i,k}, \mathbf{c}_\Pi) \end{bmatrix} \in \mathbb{Z}_q^{k\times(n+1)}.$$

Compute the Low-Order Bits: Sample $\mathbf{r}_i \leftarrow\!\!\$\ \mathbb{Z}_q^n$ and compute $\mathbf{c}_{i,\mathbf{r}} \leftarrow\!\!\$\ \mathsf{GSW.Enc}(\mathsf{pk}, \mathbf{r}_i)$. Parse the i-th block of $\mathsf{dR.PubCoin}$ as

$$(h_{i,1}, \ldots, h_{i,k}) = \mathbf{B}\cdot\mathbf{r}_i + (u_{i,1}, \ldots, u_{i,k}) \in \mathbb{Z}_q^k$$

for some $(u_{i,1}, \ldots, u_{i,k}) \in \mathbb{Z}_q^k$. Let $\Psi_{i,j} : \{0,1\}^\lambda \to \{0,1\}$ be the circuit that, on input $\mathbf{r}_i$, computes the decryption of the j-th bit encrypted in $(h_{i,1}, \ldots, h_{i,k})$. I.e. it computes $\mathsf{Round}((h_{i,1}, \ldots, h_{i,k}) - \mathbf{B}\cdot\mathbf{r}_i)$. Compute homomorphically the matrix of ciphertexts

$$\mathbf{C}_{i,\mathsf{Round}} = \begin{bmatrix} \mathsf{GSW.Eval}(\mathsf{pk}, \Psi_{i,1}, \mathbf{c}_{i,\mathbf{r}}) \\ \cdots \\ \mathsf{GSW.Eval}(\mathsf{pk}, \Psi_{i,k}, \mathbf{c}_{i,\mathbf{r}}) \end{bmatrix} \in \mathbb{Z}_q^{k\times(n+1)}.$$

Rerandomize the Ciphertext: Parse the i-th block of $\mathsf{GSW.PubCoin}$ as $(\mathbf{r}^*_{i,1}, \ldots, \mathbf{r}^*_{i,k}) \in [-\tilde{B}, +\tilde{B}]^{m\cdot k}$ and compute

$$\mathbf{C}'_{i,\mathsf{Round}} = \mathbf{C}_{i,\mathsf{Round}} + \begin{bmatrix} \mathsf{GSW.ColEnc}(\mathsf{pk}, 0; \mathbf{r}^*_{i,1}) \\ \cdots \\ \mathsf{GSW.ColEnc}(\mathsf{pk}, 0; \mathbf{r}^*_{i,k}) \end{bmatrix} \in \mathbb{Z}_q^{k\times(n+1)}.$$

Proxy Re-Encrypt: Define $\mathbf{D}_i$ as the vector of GSW ciphertexts resulting from the homomorphic sum of $\mathbf{C}'_{i,\mathsf{Round}}$ and $\mathbf{C}_i$, i.e. $\mathbf{D}_i = \mathbf{C}'_{i,\mathsf{Round}} + \mathbf{C}_i$. Observe that $\mathbf{D}_i$ consists of k GSW ciphertexts and let $\boldsymbol{\ell}_{i,j} \in \mathbb{Z}_q^{(n+1)}$ be the linear function associated with the decryption of the j-th ciphertext. Define $\boldsymbol{\ell}_i = (\boldsymbol{\ell}_{i,1}, \ldots, \boldsymbol{\ell}_{i,k})$ and compute

$$\bar{\mathbf{c}}_i = \bar{\mathbf{C}}_{\mathsf{sk}} \cdot \mathsf{Bit}(\boldsymbol{\ell}_i) + (h_{i,1}, \ldots, h_{i,k}) \in \mathbb{Z}_q^k$$

where the function $\mathsf{Bit} : \mathbb{Z}_q^{k\cdot(n+1)} \to \{0,1\}^{\log(q)\cdot k\cdot(n+1)}$ is the bit decomposition operator.

Release Hint: Compute the i-th decryption hint as

$$\boldsymbol{\rho}_i = \mathbf{S} \cdot \mathsf{Bit}(\boldsymbol{\ell}_i) + \mathbf{r}_i \in \mathbb{Z}_q^n.$$

Output: The obfuscated circuit consists of the public keys $(\mathsf{pk}, \bar{\mathsf{pk}})$, the matrix $\bar{\mathbf{C}}_{\mathsf{sk}}$, the GSW encryption of the circuit $\mathbf{c}_\Pi$, the encryption headers $(\mathbf{c}_{1,\mathbf{r}}, \ldots, \mathbf{c}_{2^{\eta-\log(k)},\mathbf{r}})$, and the decryption hints $(\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_{2^{\eta-\log(k)}})$.

To evaluate the obfuscated circuit on input x, let i be the index of the block of the truth table of Π that contains $\Pi(x)$. The evaluator computes $\bar{\mathbf{c}}_i$ as specified above (note that all the operations are public, given the information included in the obfuscated circuit) and recovers $\Pi^{(i)}$ (the i-th block of the truth table of Π) by computing

$$\Pi^{(i)} = \mathsf{Round}(\bar{\mathbf{c}}_i - \mathbf{B} \cdot \boldsymbol{\rho}_i)$$

where $\mathsf{Round} : \mathbb{Z}_q^k \to \{0,1\}^k$ rounds the input to the nearest multiple of $q/2$.

Correctness. To see why the evaluation algorithm is correct, recall that

$$\bar{\mathbf{c}}_i = \bar{\mathbf{C}}_{\mathsf{sk}} \cdot \mathsf{Bit}(\boldsymbol{\ell}_i) + (h_{i,1}, \ldots, h_{i,k}).$$

First observe that $(\mathbf{r}_i, h_{i,1}, \ldots, h_{i,k})$ define a ciphertext in the support of the algorithm $\mathsf{dR.DenseEnc}(\bar{\mathsf{pk}})$, which we rewrite as

$$\begin{aligned}\mathsf{dR.DenseEnc}(\bar{\mathsf{pk}}) &= (\mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, \mathbf{r}_i), \mathbf{B} \cdot \mathbf{r}_i + (u_{i,1}, \ldots, u_{i,k})) \\ &= (\mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, \mathbf{r}_i), \mathbf{B} \cdot \mathbf{r}_i + \mathbf{u}_i)\,.\end{aligned}$$

Thus $\mathbf{C}'_{i,\mathsf{Round}}$ and $\mathbf{C}_i$ are in the support of

$$\begin{bmatrix}\mathsf{GSW.ColEnc}(\mathsf{pk}, \mathsf{Round}(u_1)) \\ \ldots \\ \mathsf{GSW.ColEnc}(\mathsf{pk}, \mathsf{Round}(u_k))\end{bmatrix} \text{ and } \begin{bmatrix}\mathsf{GSW.ColEnc}(\mathsf{pk}, \Pi_1^{(i)}) \\ \ldots \\ \mathsf{GSW.ColEnc}(\mathsf{pk}, \Pi_k^{(i)})\end{bmatrix}$$

respectively, by the evaluation correctness of the GSW scheme and by Lemma 2. Furthermore, recall that $\mathbf{D}_i = \mathbf{C}'_{i,\mathsf{Round}} + \mathbf{C}_i$. By an invocation of Lemma 2, we have that $\mathbf{D}_i$ is in the support of

$$\begin{bmatrix}\mathsf{GSW.ColEnc}(\mathsf{pk}, \mathsf{Round}(u_1)) \\ \ldots \\ \mathsf{GSW.ColEnc}(\mathsf{pk}, \mathsf{Round}(u_k))\end{bmatrix} + \begin{bmatrix}\mathsf{GSW.ColEnc}(\mathsf{pk}, \Pi_1^{(i)}) \\ \ldots \\ \mathsf{GSW.ColEnc}(\mathsf{pk}, \Pi_k^{(i)})\end{bmatrix}$$

$$\approx \begin{bmatrix}\mathsf{GSW.ColEnc}(\mathsf{pk}, \mathsf{Round}(u_1) \oplus \Pi_1^{(i)}) \\ \ldots \\ \mathsf{GSW.ColEnc}(\mathsf{pk}, \mathsf{Round}(u_k) \oplus \Pi_k^{(i)})\end{bmatrix}$$

with all but negligible probability. By the almost-linear decryption of GSW, it follows that

$$\bar{\mathbf{C}}_{\mathsf{sk}} \cdot \mathsf{Bit}(\boldsymbol{\ell}_i) = \mathbf{B} \cdot \tilde{\mathbf{s}}_i + \boldsymbol{\xi}_i + \boldsymbol{\zeta}_i + q/2 \cdot \left(\mathsf{Round}(u_1) \oplus \Pi_1^{(i)}, \ldots, \mathsf{Round}(u_k) \oplus \Pi_k^{(i)}\right)$$

where $\boldsymbol{\xi}_i$ is the decryption noise of the packed dual-Regev scheme (i.e. the subset sum of the noise terms of $\bar{\mathbf{C}}_{\mathsf{sk}}$) and $\boldsymbol{\zeta}_i$ is the decryption noise of the GSW ciphertext. It follows that $\|\boldsymbol{\xi}_i\|_\infty \leq B \cdot \log(q) \cdot k \cdot (n+1)$ and, by Lemma 2, $\|\boldsymbol{\zeta}_i\|_\infty \leq \tilde{B}$ with all but negligible probability. Note that, by linearity we have that $\tilde{\mathbf{s}}_i = \mathbf{S} \cdot \mathsf{Bit}(\boldsymbol{\ell}_i)$. Consequently, it holds that

$$\begin{aligned}
\bar{\mathbf{c}}_i &= \mathbf{B} \cdot \tilde{\mathbf{s}}_i + \boldsymbol{\xi}_i + \boldsymbol{\zeta}_i + q/2 \cdot \left(\mathsf{Round}(u_1) \oplus \Pi_1^{(i)}, \ldots, \mathsf{Round}(u_k) \oplus \Pi_k^{(i)}\right) + \mathbf{B} \cdot \mathbf{r}_i + \mathbf{u}_i \\
&= \mathbf{B} \cdot (\tilde{\mathbf{s}}_i + \mathbf{r}_i) + \boldsymbol{\xi}_i + \boldsymbol{\zeta}_i + q/2 \cdot \left(\mathsf{Round}(u_1) \oplus \Pi_1^{(i)}, \ldots, \mathsf{Round}(u_k) \oplus \Pi_k^{(i)}\right) + \mathbf{u}_i \\
&= \mathbf{B} \cdot (\tilde{\mathbf{s}}_i + \mathbf{r}_i) + q/2 \cdot \Pi^{(i)} + \mathbf{v}_i \\
&= \mathbf{B} \cdot \boldsymbol{\rho}_i + q/2 \cdot \Pi^{(i)} + \mathbf{v}_i
\end{aligned}$$

where $\mathbf{v}_i = \mathbf{u}_i + q/2 \cdot \mathsf{Round}(\mathbf{u}_i) + \boldsymbol{\xi}_i + \boldsymbol{\zeta}_i$ and $\|\mathbf{v}_i\|_\infty < q/4$ with all but negligible probability, over the random choice of $\mathbf{u}_i$. This is because $\mathbf{D}_i$ is statistically close to a fresh GSW encryption of $(\mathsf{Round}(u_1), \ldots, \mathsf{Round}(u_k)) \oplus \Pi^{(i)}$, by Lemma 4. Therefore we have that

$$\begin{aligned}
\mathsf{Round}\left(\mathbf{c}_i - \mathbf{B} \cdot \boldsymbol{\rho}_i\right) &= \mathsf{Round}\left(\mathbf{B} \cdot \boldsymbol{\rho}_i + q/2 \cdot \Pi^{(i)} + \mathbf{v}_i - \mathbf{B} \cdot \boldsymbol{\rho}_i\right) \\
&= \mathsf{Round}\left(q/2 \cdot \Pi^{(i)} + \mathbf{v}_i\right) \\
&= \mathsf{Round}\left(q/2 \cdot \Pi^{(i)}\right) \\
&= \Pi^{(i)}
\end{aligned}$$

with the same probability. Due to space constraints, we defer the analysis of our scheme to the full version [10].

References

1 Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 191–225, Darmstadt, Germany, May 19–23 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17653-2_7.

2 Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 284–332, Santa Barbara, CA, USA, August 18–22 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-26954-8_10.

3 Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 152–181, Paris, France, April 30 – May 4 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-56620-7_6.

4 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-44647-8_1.

5 James Bartusek, Jiaxin Guan, Fermi Ma, and Mark Zhandry. Return of GGH15: Provable security against zeroizing attacks. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 544–574, Panaji, India, November 11–14 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-03810-6_20.

6 Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499, Santa Barbara, CA, USA, August 17–21 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-44371-2_27.

7 Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 62–89, Santa Barbara, CA, USA, August 14–18 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_3.

8 Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437, Nuremberg, Germany, December 1–5 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-36033-7_16.

9 Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 79–109, Zagreb, Croatia, May 10–14 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-45721-1_4.

10 Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for io: Circular-secure lwe suffices. Cryptology ePrint Archive, Report 2020/1024, 2020. URL: https://ia.cr/2020/1024.

11 Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 22–25 2011. IEEE Computer Society Press. doi:10.1109/FOCS.2011.12.

12 Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 278–307, Paris, France, April 30 – May 4 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-56617-7_10.

13 Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 577–607, Santa Barbara, CA, USA, August 19–23 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96881-0_20.

14 Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 3–12, Sofia, Bulgaria, April 26–30 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46800-5_1.

15 Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493, Santa Barbara, CA, USA, August 18–22 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_26.

16 Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-44586-2_9.

17 Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Pratyay Mukherjee. Obfuscation from low noise multilinear maps. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018: 19th International Conference in Cryptology in India*, volume 11356 of *Lecture Notes in Computer Science*, pages 329–352, New Delhi, India, December 9–12 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-05378-9_18.

18 Léo Ducas and Damien Stehlé. Sanitization of FHE ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 294–310, Vienna, Austria, May 8–12 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49890-3_12.

19 Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17, Athens, Greece, May 26–30 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-38348-9_1.

20 Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94, San Diego, CA, USA, February 24–26 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-54242-8_4.

21 Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29 2013. IEEE Computer Society Press. doi:10.1109/FOCS.2013.13.

22 Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 241–268, Beijing, China, October 31 – November 3 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53644-5_10.

23 Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. Cryptology ePrint Archive, Report 2020/1010, 2020.

24 Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527, Warsaw, Poland, March 23–25 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46497-7_20.

25 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, BC, Canada, May 17–20 2008. ACM Press. doi:10.1145/1374376.1374407.

26 Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_5.

27 Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213, Amsterdam, The Netherlands, February 21–24 2007. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-70936-7_11.

28 Satoshi Hada. Zero-knowledge and code obfuscation. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 443–457, Kyoto, Japan, December 3–7 2000. Springer, Heidelberg, Germany. doi:10.1007/3-540-44448-3_34.

29 Sam Hopkins, Aayush Jain, and Huijia Lin. Counterexamples to new circular security assumptions underlying io, 2021.

30 Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 537–565, Vienna, Austria, May 8–12 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49890-3_21.

31 Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 251–281, Darmstadt, Germany, May 19–23 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17653-2_9.

32 Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. Cryptology ePrint Archive, Report 2020/1003, 2020.

33 Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 28–57, Vienna, Austria, May 8–12 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49890-3_2.

34 Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 599–629, Santa Barbara, CA, USA, August 20–24 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-63688-7_20.

35 Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 447–462, Taipei, Taiwan, March 6–9 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49387-8_17.

36 Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 630–660, Santa Barbara, CA, USA, August 20–24 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-63688-7_21.

37 Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, *57th Annual Symposium on Foundations of Computer Science*, pages 11–20, New Brunswick, NJ, USA, October 9–11 2016. IEEE Computer Society Press. doi:10.1109/FOCS.2016.11.

38 Fermi Ma and Mark Zhandry. The MMap strikes back: Obfuscation and new multilinear maps immune to CLT13 zeroizing attacks. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 513–543, Panaji, India, November 11–14 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-03810-6_19.

39 Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 629–658, Santa Barbara, CA, USA, August 14–18 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_22.

40 Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 536–553, Santa Barbara, CA, USA, August 17–21 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-44371-2_30.

41 Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3 2014. ACM Press. doi:10.1145/2591796.2591825.

42 Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious lwe sampling. Cryptology ePrint Archive, Report 2020/1042, 2020.

Characterization of Matrices with Bounded Graver Bases and Depth Parameters and Applications to Integer Programming

Marcin Briański
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Martin Koutecký
Computer Science Institute, Charles University, Prague, Czech Republic

Daniel Král'
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Kristýna Pekárková
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Felix Schröder
Institute of Mathematics, Technische Universität, Berlin, Germany

Abstract

An intensive line of research on fixed parameter tractability of integer programming is focused on exploiting the relation between the sparsity of a constraint matrix A and the norm of the elements of its Graver basis. In particular, integer programming is fixed parameter tractable when parameterized by the primal tree-depth and the entry complexity of A, and when parameterized by the dual tree-depth and the entry complexity of A; both these parameterization imply that A is sparse, in particular, the number of its non-zero entries is linear in the number of columns or rows, respectively.

We study preconditioners transforming a given matrix to an equivalent sparse matrix if it exists and provide structural results characterizing the existence of a sparse equivalent matrix in terms of the structural properties of the associated column matroid. In particular, our results imply that the ℓ_1-norm of the Graver basis is bounded by a function of the maximum ℓ_1-norm of a circuit of A. We use our results to design a parameterized algorithm that constructs a matrix equivalent to an input matrix A that has small primal/dual tree-depth and entry complexity if such an equivalent matrix exists.

Our results yield parameterized algorithms for integer programming when parameterized by the ℓ_1-norm of the Graver basis of the constraint matrix, when parameterized by the ℓ_1-norm of the circuits of the constraint matrix, when parameterized by the smallest primal tree-depth and entry complexity of a matrix equivalent to the constraint matrix, and when parameterized by the smallest dual tree-depth and entry complexity of a matrix equivalent to the constraint matrix.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Matroids and greedoids; Mathematics of computing → Combinatorial algorithms

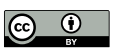
Keywords and phrases Integer programming, width parameters, matroids, Graver basis, tree-depth, fixed parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.29

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2202.05299`

Funding The third author was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 648509). This publication reflects only its authors' view; the ERC Executive Agency is not responsible for any



use that may be made of the information it contains. The third and fourth authors were supported by the MUNI Award in Science and Humanities (MUNI/I/1677/2018) of the Grant Agency of Masaryk University. The second author was partially supported by Charles University project UNCE/SCI/004 and by the project by the project 19-27871X of GA ČR. First author was partially supported by the Polish National Science Center grant (BEETHOVEN; UMO-2018/31/G/ST1/03718).

Acknowledgements All five authors would like to thank the Schloss Dagstuhl – Leibniz-Zentrum für Informatik for hospitality during the workshop "Sparsity in Algorithms, Combinatorics and Logic" in September 2021 where the work leading to the results contained in this paper was started.

1 Introduction

Integer programming is a problem of fundamental importance in combinatorial optimization with many theoretical and practical applications. It is known to be computationally very hard and is one of the 21 NP-complete problems in the original paper on NP-completeness by Karp [34]; the problem is known to be NP-complete even when the entries of the constraint matrix are zero and one only. On the positive side, Kannan and Lenstra [32, 41] showed that integer programming is polynomially solvable in fixed dimension, i.e., with a fixed number of variables. Another prominent tractable case is when the constraint matrix is unimodular, i.e., all determinants of its submatrices are equal to 0 or ± 1, in which case all vertices of the feasible region are integral and so linear programming algorithms can be applied.

Integer programming is known to be tractable for instances where the constraint matrix of an input integer program (IP) enjoys a certain block structure. The two most important cases are the cases of 2-stage IPs due to Hemmecke and Schultz [23], further investigated in particular in [1, 12, 27, 36, 37, 40], and n-fold IPs introduced by De Loera et al. [13] and further investigated in particular in [10, 11, 16, 22, 31, 40]. IPs of this kind appear in various contexts, see e.g. [29,38,39,44]. These (theoretical) tractability results complement well a vast number of empirical results demonstrating tractability of instances with a block structure, e.g. [2–4, 18, 19, 35, 45–47].

A more general approach to tractability of IPs with sparse constraint matrices involves depths/widths of graphs defined on columns or rows of the constraint matrices. Ganian and Ordyniak [20] initiated this line of study by showing that IPs with bounded primal tree-depth $\mathrm{td}_P(A)$ of a constraint matrix A and bounded coefficients and right hand sides $\|A, b\|_\infty$ can be solved efficiently. Levin, Onn and the second author [40] widely generalized this result by showing that IPs with bounded coefficients $\|A\|_\infty$ and bounded primal tree-depth $\mathrm{td}_P(A)$ or dual tree-depth $\mathrm{td}_D(A)$ of the constraint matrix A can be solved efficiently; such IPs include 2-stage IPs, n-fold IPs, and their generalizations. The existence of efficient algorithms in the case of constraint matrices A with bounded primal and dual tree-depth is closely linked to bounds on the norm of elements of the Graver basis of A, which is formed by minimal integer vectors of $\ker A$ in an orthant (see Section 2 for the rigorous definition). The maximum ℓ_1-norm and ℓ_∞-norm of an element of the Graver basis of A is denoted by $g_1(A)$ and $g_\infty(A)$, respectively. In particular, the following holds [40], where $\mathrm{ec}(A)$ denotes the entry complexity of a matrix A defined as the maximum number of bits needed to represent any of the entries of A.

▶ **Theorem 1.** *There exist functions $f_P, f_D : \mathbb{N}^2 \to \mathbb{N}$ such that the following holds for every matrix A: $g_\infty(A) \leq f_P(\mathrm{td}_P(A), \mathrm{ec}(A))$ and $g_1(A) \leq f_D(\mathrm{td}_D(A), \mathrm{ec}(A))$.*

Most of the existing algorithms for IPs assume that the input matrix is already given in its sparse form. This is a substantial drawback as existing algorithms cannot be applied to instances that are not sparse but can be transformed to a sparse instance, for example, the matrix in the left, whose dual tree-depth is 5, can be transformed by row operations to the matrix with dual tree-depth 2 given in the right.

$$\begin{pmatrix} 2 & 2 & 1 & 2 & 1 & 3 & 1 \\ 2 & 1 & 1 & 1 & 2 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 & 2 & 1 & 1 \\ 2 & 2 & 1 & 2 & 1 & 3 & 2 \end{pmatrix} \quad \to \quad \begin{pmatrix} 2 & 1 & 0 & 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 \end{pmatrix}$$

Such a transformation of an input matrix is a preconditioner as it changes an input matrix to an equivalent one that is computationally more tractable.

Preconditioning a matrix to make the problem computationally simpler, e.g., to make the feasible region not too flat in any direction, is a ubiquitous preprocessing step in mathematical programming solvers. In this paper, we are concerned with the existence and efficient computability of preconditioners to sparsity of matrices. Chan and Cooper together with the second, third and fourth authors [7,8] gave a structural characterization of matrices that are equivalent, i.e., can be transformed by row operations, to a matrix with small dual tree-depth, and used their structural results to design a fixed parameter algorithm to find such a matrix if it exists; we denote the smallest dual tree-depth of a matrix equivalent to A by $\mathrm{td}^*_D(A)$.

▶ **Theorem 2.** *There exists an algorithm parameterized by d and e that for an input matrix A with entry complexity e*

- *either outputs that $\mathrm{td}^*_D(A) > d$, or*
- *outputs a matrix A' equivalent to A such that the dual tree-depth of A' is $\mathrm{td}^*_D(A)$ and its entry complexity is $\mathcal{O}(d^2 2^{2d} e)$.*

The structural characterization given in [7,8] exhibits an interesting link to matroid theory: an equivalent matrix with small dual tree-depth exists if and only if the column matroid of the matrix, which is invariant under row operations, has small contraction*-depth (see Theorem 3 below). We remark that the term branch-depth was used in [7,8] following the terminology from [33] but as there is a competing notion of branch-depth [14], we decided to use a different name for this notion throughout the paper to avoid confusion.

In this paper, we provide a structural characterization of matrices that are equivalent to a matrix with small primal tree-depth or small incidence tree-depth (Theorems 4 and 5). We also study corresponding preconditioners and design fixed parameter algorithms for constructing an equivalent matrix with small primal tree-depth and small entry complexity and for constructing an equivalent matrix with small dual tree-depth and small entry complexity, if such a matrix exists (Theorems 7 and 8). Finally, we employ our structural results to resolve an open problem whether integer programming is fixed-parameter tractable parameterized by the largest ℓ_1-norm of the Graver basis element of a matrix A. Additionally, we show that the ℓ_1-norm of each element of the Graver basis of a matrix A is bounded by a function of the largest ℓ_1-norm of a circuit of the matrix A (Theorem 6). All these results are stated more precisely in Subsection 1.1.

The existence of appropriate preconditioners that we establish in this paper implies that integer programming is fixed parameter tractable when parameterized by

- $g_1(A)$, i.e., the ℓ_1-norm of the Graver basis of the constraint matrix,
- $c_1(A)$, i.e., the ℓ_1-norm of the circuits of the constraint matrix,

- $\mathrm{td}^*_P(A)$ and $\mathrm{ec}(A)$, i.e., the smallest primal tree-depth and entry complexity of a matrix equivalent to the constraint matrix, and
- $\mathrm{td}^*_D(A)$ and $\mathrm{ec}(A)$, i.e., the smallest dual tree-depth and entry complexity of a matrix equivalent to the constraint matrix.

We believe that our new tractability results significantly enhance the toolbox of tractable IPs as the nature of our tractability conditions substantially differ from prevalent block-structured sparsity-based tractability conditions. The importance of availability of various forms of tractable IPs can be witnessed by n-fold IPs, which were shown fixed-parameter tractable in [22], and, about a decade later, their applications has become ubiquitous, see e.g. [5, 6, 9, 10, 24, 28, 30, 39].

1.1 Our contribution

We now describe the results presented in this paper in detail; we refer the reader for the definitions of the notions not yet rigorously introduced to Section 2.

1.1.1 Characterization of depth parameters

The main structural result of [7, 8] is the following structural characterization of the existence of an equivalent matrix with small dual tree-depth in terms of the structural parameter of the column matroid, which is invariant under row operations.

▶ **Theorem 3.** *For every non-zero matrix A, it holds that the smallest dual tree-depth of a matrix equivalent to A is equal to the contraction*-depth of $M(A)$, i.e.,* $\mathrm{td}^*_D(A) = \mathrm{c}^*\mathrm{d}(A)$.

We discover structural characterizations of the existence of an equivalent matrix with small primal tree-depth and the existence of an equivalent matrix with small incidence tree-depth.

▶ **Theorem 4.** *For every matrix A, it holds that the smallest primal tree-depth of a matrix equivalent to A is equal to the deletion-depth of $M(A)$, i.e.,* $\mathrm{td}^*_P(A) = \mathrm{dd}(A)$.

▶ **Theorem 5.** *For every matrix A, it holds that the smallest incidence tree-depth of a matrix equivalent to A is equal to contraction*-deletion-depth of $M(A)$ increased by one, i.e.,* $\mathrm{td}^*_I(A) = \mathrm{c}^*\mathrm{dd}(A) + 1$.

1.1.2 Interplay of circuit and Graver basis complexity

As mentioned earlier, Graver bases play an essential role in designing efficient algorithms for integer programming. A *circuit* of a matrix A is a support-wise minimal integral vector contained in the kernel of A such that all its entries are coprime. Hence, every circuit of a matrix A is an element of the Graver basis of A and so the maximum ℓ_1-norm of an element of the Graver basis, which is denoted by $g_1(A)$, is an upper bound on the maximum ℓ_1-norm of a circuit of A, which is denoted by $c_1(A)$.

One of the open problems in the area, e.g. discussed during the Dagstuhl workshop 19041 "New Horizons in Parameterized Complexity", has been whether integer programming is fixed parameter tractable when parameterized by $g_1(A)$, the ℓ_1-norm of an element of the Graver basis of the constraint matrix A. An affirmative answer to this question follows from Theorems 6 and 8 below. We actually show that *the maximum ℓ_1-norm $g_1(A)$ of an element of the Graver basis of a matrix A is small if and only if A is equivalent to a matrix with a small dual tree-depth and small entry complexity*; the implication from left to right is given

in Theorem 1 and the other implication in Theorem 12 (recall that $c_1(A) \leq g_1(A)$ for every matrix). We summarize the relation between the the maximum ℓ_1-norm of a circuit of a matrix A, the maximum ℓ_1-norm of an element of the Graver basis of A, and the existence of an equivalent matrix with small dual tree-depth and small entry complexity in the next theorem (the second part of the theorem is given in Corollary 13).

▶ **Theorem 6.** *There exist a function $f_1 : \mathbb{N} \to \mathbb{N}$ such that the following holds for every matrix A with $\dim \ker A > 0$:*

- *the matrix A is equivalent to a matrix A' with $\mathrm{td}_D(A') \leq c_1(A)^2$ and $\mathrm{ec}(A') \leq 2\lceil c_1(A) \rceil$, and*
- *$c_1(A) \leq g_1(A) \leq f_1(c_1(A))$.*

Hence, informally speaking, the following statements are equivalent for every matrix A:

- The matrix A is equivalent to a matrix with bounded dual tree-depth and bounded entry complexity.
- The contraction*-depth of the matroid $M(A)$ is bounded.
- The ℓ_1-norm of every circuit of A is bounded.
- The ℓ_1-norm of every element of the Graver basis of A is bounded.

1.1.3 Algorithms to compute matrices with small depth parameters

We design a parameterized algorithm for computing an equivalent matrix with small primal tree-depth and small entry complexity if one exists.

▶ **Theorem 7.** *There exists a function $f : \mathbb{N}^2 \to \mathbb{N}$ and an FPT algorithm for the parameterization by d and e that, for a given rational matrix A with m rows and n columns, $m \leq n$:*

- *either outputs that A is not equivalent to a matrix with primal tree-depth at most d and entry complexity at most e, or*
- *outputs a matrix A' that is equivalent to A, its primal tree-depth is at most d and entry complexity is at most $f(d, e)$.*

We also design a parameterized algorithm for computing an equivalent matrix with small dual tree-depth and small entry complexity if one exists. Note that the algorithm described in Theorem 2 for computing an equivalent matrix with small dual tree-depth is parameterized by the dual tree-depth of the to be constructed matrix and the entry complexity of the *input* matrix while the algorithm given below is parameterized by the entry complexity of the to be constructed matrix and so the algorithm can be applied to a wider set of input matrices.

▶ **Theorem 8.** *There exists a function $f : \mathbb{N}^2 \to \mathbb{N}$ and an FPT algorithm for the parameterization by d and e that, for a given rational matrix A:*

- *either outputs that A is not equivalent to a matrix with dual tree-depth at most d and entry complexity at most e, or*
- *outputs a matrix A' that is equivalent to A, its dual tree-depth is at most d and entry complexity is at most $f(d, e)$.*

We remark that if a matrix A has entry complexity e and is equivalent to a matrix with dual tree-depth d, then there exists an equivalent matrix with dual tree-depth d and entry complexity bounded by a function of d and e (as implied by Theorem 2). However, the same is not true in the case of primal tree-depth. The entry complexity of every matrix with primal tree-depth equal to one that is equivalent to the following matrix A is linear in the number of rows of A, quite in a contrast to the case of dual tree-depth.

$$\begin{pmatrix} 1 & 2 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & & & \vdots & \vdots \\ \vdots & \vdots & & & \ddots & \ddots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 2 \end{pmatrix}$$

1.1.4 Hardness results

As our algorithmic results involve computing depth decompositions of matroids for various depth parameters in a parameterized way, we establish the computational hardness of these parameters in Theorem 18, primarily for the sake of completeness of our exposition. In particular, computing the following matroid parameters is NP-complete:

- deletion-depth,
- contraction-depth,
- contraction-deletion-depth,
- contraction*-depth, and
- contraction*-deletion-depth.

2 Preliminaries

In this section, we fix the notation used throughout the paper. We start with general notation and we then fix the notation related to graphs, matrices and matroids.

The set of all positive integers is denoted by $\mathbb{N}$ and the set of the first k positive integers by $[k]$. If A is a linear space, we write $\dim A$ for its dimension and if B is a set of vectors, we write $\mathcal{L}(B)$ for the linear hull of the vectors contained in B. If A is a linear space and K is a subspace of A, the *quotient space* A/K is the linear space of the dimension $\dim A - \dim K$ that consists of cosets of A given by K with the natural operations of addition and scalar multiplication; see e.g. [21] for further details. The quotient space A/K can be associated with a linear subspace of A of dimension $\dim A - \dim K$ formed by exactly a single vector from each coset of A given by K; we will often view the quotient space as such a subspace of A and write $w + K$ for the coset containing a vector w.

2.1 Graphs

All graphs considered in this paper are loopless simple graphs unless stated otherwise. If G is a graph, then we write $V(G)$ and $E(G)$ for the vertex set and the edge set of G, respectively; the number of vertices and edges of G is denoted by $|G|$ and $\|G\|$, respectively. If W is a subset of vertices of a graph G, then $G \setminus W$ is the graph obtained by removing the vertices of W (and all edges incident with them). If F is a subset of edges of a graph G, then $G \setminus F$ is the graph obtained by removing the edges contained in F and G/F is the graph obtained by contracting all edges contained in F and removing resulting loops and parallel edges.

We next define the graph parameter *tree-depth*, which is the central graph parameter in this paper. The *height* of a rooted tree is the maximum number of vertices on a path from the root to a leaf, and the *height* of a rooted forest, i.e., a graph whose each component

is a rooted tree, is the maximum height of its components. The *depth* of a rooted tree is the maximum number of edges on a path from the root to a leaf, and the *depth* of a rooted forest is the maximum depth of its components. The *closure* $\mathrm{cl}(F)$ of a rooted forest F is the graph obtained by adding edges from each vertex to all its descendants. Finally, the *tree-depth* $\mathrm{td}(G)$ of a graph G is the minimum height of a rooted forest F such that the closure $\mathrm{cl}(F)$ of the rooted forest F contains G as a subgraph.

2.2 Matroids

We next review basic definitions from matroid theory; for detailed information, we refer to the book of Oxley [42]. A *matroid* M is a pair $(X, \mathcal{I})$, where $\mathcal{I}$ is a non-empty hereditary collection of subsets of X that satisfies the *augmentation axiom*, i.e., if $X' \in \mathcal{I}$, $X'' \in \mathcal{I}$ and $|X'| < |X''|$, then there exists an element $x \in X'' \setminus X'$ such that $X' \cup \{x\} \in \mathcal{I}$. The set X is the *ground set* of M and the sets contained in $\mathcal{I}$ are referred to as *independent*. The *rank* of a subset X' of the ground set X, which is denoted by $r_M(X')$ or simply by $r(X')$ if M is clear from the context, is the maximum size of an independent subset of X' (it can be shown that all maximal independent subsets of X' have the same cardinality); the *rank* of the matroid M, which is denoted by $r(M)$, is the rank of its ground set. A *basis* of a matroid M is a maximal independent subset of the ground set of M and a *circuit* is a minimal subset of the ground set of M that is not independent. In particular, if X' is a circuit of M, then $r(X') = |X'| - 1$ and every proper subset of X' is independent. An element x of a matroid M is a *loop* if $r(\{x\}) = 0$, an element x is a *bridge* if it is contained in every basis of M, and two elements x and x' are *parallel* if $r(\{x\}) = r(\{x'\}) = r(\{x, x'\}) = 1$. If M is a matroid with ground set X, the *dual matroid*, which is denoted by M^* is the matroid with the same ground set X such that $X' \subseteq X$ is independent in M^* if and only if $r_M(X \setminus X') = r(M)$; in particular, $r_{M^*}(X') = r_M(X \setminus X') + |X'| - r(M)$ for every $X' \subseteq X$.

For a field $\mathbb{F}$, we say that a matroid M is $\mathbb{F}$-*representable* if every element of M can be assigned a vector from $\mathbb{F}^{r(M)}$ in such a way that a subset of the ground set of M is independent if and only if the set of assigned vectors is linearly independent. In particular, an element of M is a loop if and only if it is assigned the zero vector and two elements of M are parallel if and only if they are assigned non-zero multiples of the same non-zero vector. Such an assignment of vectors of $\mathbb{F}^{r(M)}$ to the elements of M is an $\mathbb{F}$-*representation* of M. Observe that the rank of a subset X' of the ground set is the dimension of the linear hull of the vectors assigned to the elements of X'. We say that a matroid M is $\mathbb{F}$-*represented* if the matroid M is given by its $\mathbb{F}$-representation. If a particular field $\mathbb{F}$ is not relevant in the context, we just say that a matroid M is *represented* to express that it is given by its representation.

Let M be a matroid with a ground set X. The matroid kM for $k \in \mathbb{N}$ is the matroid obtained from M by introducing $k-1$ parallel elements to each non-loop element and $k-1$ additional loops for each loop; informally speaking, every element of M is "cloned" to k copies. If $X' \subseteq X$, then the *restriction* of M to X', which is denoted by $M[X']$, is the matroid with the ground set X' such that a subset of X' is independent in $M[X']$ if and only if it is independent in M. In particular, the rank of $M[X']$ is $r_M(X')$. The matroid obtained from M by *deleting* X' is the restriction of M to $X \setminus X'$ and is denoted by $M \setminus X'$. The *contraction* of M by X', which is denoted by M/X', is the matroid with the ground set $X \setminus X'$ such that a subset X'' of $X \setminus X'$ is independent in M/X' if and only if $r_M(X'' \cup X') = |X''| + r_M(X')$. If X' is a single element set and e is its only element, we write $M \setminus e$ and M/e instead of $M \setminus \{e\}$ and $M/\{e\}$, respectively. If an $\mathbb{F}$-representation of M is given and X' is a subset of the ground set of M, then an $\mathbb{F}$-representation of M/X'

can be obtained from the $\mathbb{F}$-representation of M by considering it in the quotient space by the linear hull of the vectors representing the elements of X'. This leads us to the following definition: if M is an $\mathbb{F}$-represented matroid and A is a linear subspace of $\mathbb{F}^{r(M)}$, then the matroid M/A is the $\mathbb{F}$-represented matroid with the representation of M in the quotient space by A. Note that the ground sets of M and M/A are the same.

A matroid M is *connected* if every two distinct elements of M are contained in a common circuit. If M is an $\mathbb{F}$-represented matroid with at least two elements, then M is connected if and only if M has no loops and there do not exist two non-trivial vector spaces A and B of $\mathbb{F}^{r(M)}$ such that $A \cap B$ contains the zero vector only and every element of M is contained in A or B. A *component* of a matroid M is an inclusion-wise maximal connected restriction of M; a component is *trivial* if it consists of a single loop, and it is *non-trivial* otherwise. We often identify components of a matroid M with their element sets. Using this identification, it holds that a subset X' of a ground set of a matroid M is a component of M if and only if X' is a component of M^*. We remark that $(M^*)^* = M$ for every matroid M, and if e is an element of a matroid M, then $(M/e)^* = M^* \setminus e$ and $(M \setminus e)^* = M^*/e$.

2.3 Matrices

In this section, we define notation related to matrices. If $\mathbb{F}$ is a field, we write $\mathbb{F}^{m\times n}$ for the set of matrices with m rows and n columns over the field $\mathbb{F}$. If A is a rational matrix, the entry complexity $\mathrm{ec}(A)$ is the maximum length of a binary encoding of its entries, i.e., the maximum of $\lceil \log_2(|p|+1)\rceil + \lceil(\log_2|q|+1)\rceil$ taken over all entries p/q of A (where p and q are always assumed to be coprime). A rational matrix A is *z-integral* for $z \in \mathbb{Q}$ if every entry of A is an integral multiple of z. We say that two matrices A and A' are *equivalent* if one can be obtained from another by (equivalent) *row operations*, i.e., adding (a non-zero multiple of) one row to another and multiplying a row by a non-zero element. Observe that if A and A' are equivalent matrices, then their kernels are the same. For a matrix A, we define $M(A)$ to be the represented matroid whose elements are the columns of A. Again, if matrices A and A' are equivalent, then the matroids $M(A)$ and $M(A')$ are the same.

If A is a matrix, the *primal graph* of A is the graph whose vertices are columns of A and two vertices are adjacent if there exists a row having non-zero elements in the two columns associated with the vertices; the *dual graph* of A is the graph whose vertices are rows of A and two vertices are adjacent if there exists a column having non-zero elements in the two associated rows; the *incidence graph* of A is the bipartite graph with one part formed by rows of A and the other part by columns of A and two vertices are adjacent if the entry in the associated row and in the associated column is non-zero. The *primal tree-depth* of A, denoted by $\mathrm{td}_P(A)$, is the tree-depth of the primal graph of A, the *dual tree-depth* of A, denoted by $\mathrm{td}_D(A)$, is the tree-depth of the dual graph of A, and the *incidence tree-depth* of A, denoted by $\mathrm{td}_I(A)$, is the tree-depth of the incidence graph of A. Finally, $\mathrm{td}^*_P(A)$ is the smallest primal tree-depth of a matrix equivalent to A, $\mathrm{td}^*_D(A)$ is the smallest dual tree-depth of a matrix equivalent to A, and $\mathrm{td}^*_I(A)$ is the smallest incidence tree-depth of a matrix equivalent to A.

A *circuit* of a rational matrix A is a support-wise minimal integral vector contained in the kernel of A such that all its entries are coprime; the set of circuits of A is denoted by $\mathcal{C}(A)$. Note that a set X of columns is a circuit in the matroid $M(A)$ if and only if $\mathcal{C}(A)$ contains a vector with the support exactly equal to X. We write $c_1(A)$ for the maximum ℓ_1-norm of a circuit of A and $c_\infty(A)$ for the maximum ℓ_∞-norm of a circuit of A. Note if A and A' are equivalent rational matrices, then $\mathcal{C}(A) = \mathcal{C}(A')$ and so the parameters $c_1(\cdot)$ and $c_\infty(\cdot)$ are invariant under row operations. Following the notation from [17], we write $\dot\kappa_A$ for the least common multiple of the entries of the circuits of A. Observe that there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that $\dot\kappa_A \le f(c_\infty(A))$ for every matrix A.

If $\mathbf{x}$ and $\mathbf{y}$ are two d-dimensional vectors, we write $\mathbf{x} \sqsubseteq \mathbf{y}$ if $|\mathbf{x}_i| \leq |\mathbf{y}_i|$ for all $i \in [d]$ and $\mathbf{x}$ and $\mathbf{y}$ are in the same orthant, i.e., $\mathbf{x}_i$ and $\mathbf{y}_i$ have the same sign (or they both are zero) for all $i \in [d]$. The *Graver basis* of a matrix A, denoted by $\mathcal{G}(A)$, is the set of the $\sqsubseteq$-minimal non-zero elements of the integer kernel $\ker_{\mathbb{Z}}(A)$. We use $g_1(A)$ and $g_\infty(A)$ for the Graver basis of A analogously to the set of circuits, i.e., $g_1(A)$ is the maximum ℓ_1-norm of a vector in $\mathcal{G}(A)$ and $g_\infty(A)$ is the maximum ℓ_∞-norm of a vector in $\mathcal{G}(A)$. Again, the parameters $g_1(\cdot)$ and $g_\infty(\cdot)$ are invariant under row operations as the Graver bases of equivalent matrices are the same. Note that every circuit of a matrix A belongs to the Graver basis of A, i.e., $\mathcal{C}(A) \subseteq \mathcal{G}(A)$, and so it holds that $c_1(A) \leq g_1(A)$ and $c_\infty(A) \leq g_\infty(A)$ for every matrix A.

2.4 Matroid depth parameters

We now define matroid depth parameters that will be of importance further. We start with the notion of deletion-depth and contraction-depth, which were introduced in [14].

The *deletion-depth* of a matroid M, denoted by $\mathrm{dd}(M)$, is defined recursively as follows. If M has a single element, then $\mathrm{dd}(M) = 1$. If M is not connected, then $\mathrm{dd}(M)$ is the maximum deletion-depth of a component of M. Otherwise, $\mathrm{dd}(M)$ is 1 plus the minimum deletion-depth of $M \setminus e$ where the minimum is taken over all elements e of M.

The sequence of deletions of elements witnessing that the deletion-depth of a matroid M is $\mathrm{dd}(M)$ can be visualized by a rooted tree, which we call a *deletion-decomposition tree*, defined as follows. If M has a single element, then the deletion-decomposition tree of M consists of a single vertex labeled with the single element of M. If M is not connected, then the deletion-decomposition tree is obtained by identifying the roots of deletion-decomposition trees of the components of M. Otherwise, there exists an element e such that $\mathrm{dd}(M) = \mathrm{dd}(M \setminus e) + 1$ and the deletion-decomposition tree of M is obtained from the deletion-decomposition tree of $M \setminus e$ by adding a new vertex adjacent to the root of the deletion-decomposition tree of $M \setminus e$, changing the root of the tree to the newly added vertex and labeling the edge incident with it with the element e. Observe that the height of the deletion-decomposition tree is equal to the deletion-depth of M. In what follows, we consider deletion-decomposition trees that need not to be of optimal height, i.e., its edges can be labeled by a sequence of elements that decomposes a matroid M in a way described in the definition of the deletion-depth but its height is larger than $\mathrm{dd}(M)$. In this more general setting, the deletion-depth of a matroid M is the smallest height of a deletion-decomposition tree of M.

The *contraction-depth* of a matroid M, denoted by $\mathrm{cd}(M)$, is defined recursively as follows. If M has a single element, then $\mathrm{cd}(M) = 1$. If M is not connected, then $\mathrm{cd}(M)$ is the maximum contraction-depth of a component of M. Otherwise, $\mathrm{cd}(M)$ is 1 plus the minimum contraction-depth of M/e where the minimum is taken over all elements e of M. It is not hard to show that $\mathrm{dd}(M) = \mathrm{cd}(M^*)$ and $\mathrm{cd}(M) = \mathrm{dd}(M^*)$ for every matroid M. We define a *contraction-decomposition tree* analogously to a deletion-decomposition tree; the contraction-depth of a matroid M is the smallest height of a contraction-decomposition tree of M.

We next introduce the contraction-deletion-depth, which was studied under the name of type in [15], but we decided to use the name from [14], which we find to describe the parameter in the context considered here better. The *contraction-deletion-depth* of matroid M, denoted by $\mathrm{cdd}(M)$, is defined recursively as follows. If M has a single element, then $\mathrm{cdd}(M) = 1$. If M is not connected, then $\mathrm{cdd}(M)$ is the maximum contraction-deletion-depth of a component of M. Otherwise, $\mathrm{cdd}(M)$ is 1 plus the smaller among the minimum contraction-deletion-depth of the matroid $M \setminus e$ and the minimum contraction-deletion-depth of the matroid M/e where both minima are taken over all elements e of M. Clearly, it holds that $\mathrm{cdd}(M) = \mathrm{cdd}(M^*)$, $\mathrm{cdd}(M) \leq \mathrm{dd}(M)$ and $\mathrm{cdd}(M) \leq \mathrm{cd}(M)$ for every matroid M.

One of the key parameters in our setting is that of contraction*-depth; this parameter was introduced under the name branch-depth in [33] and further studied in [8] but we decided to use a different name to avoid a possible confusion with the notion of branch-depth introduced in [14]. A *contraction*-depth* of a matroid M, denoted by $\mathrm{c}^*\mathrm{d}(M)$, is the smallest depth of a rooted tree T with exactly $r(M)$ edges with the following property: there exists a function f from the ground set of M to the leaves of T such that for every subset X of the ground set of M the total number of edges contained in paths from the root to vertices of X is at least $r(X)$. There is an alternative equivalent definition of the parameter for represented matroids. The *contraction*-depth* of a represented matroid M can be defined recursively as follows. If M has rank zero, then $\mathrm{c}^*\mathrm{d}(M) = 0$. If M is not connected, then $\mathrm{c}^*\mathrm{d}(M)$ is the maximum contraction*-depth of a component of M. Otherwise, $\mathrm{c}^*\mathrm{d}(M)$ is 1 plus the minimum contraction*-depth of a matroid obtained from the matroid M by factoring along an arbitrary one-dimensional subspace. As the contraction in the definition is allowed to be by an arbitrary one-dimensional subspace, not only by a subspace generated by an element of M, it follows that $\mathrm{c}^*\mathrm{d}(M) \leq \mathrm{cd}(M)$.

Kardoš et al. [33] established the connection between the contraction*-depth and the existence of a long circuit, which is described in Theorem 9.

▶ **Theorem 9.** *Let M be a matroid and k the size of its largest circuit. It holds that $\log_2 k \leq \mathrm{c}^*\mathrm{d}(M) \leq k^2$. Moreover, there exists a polynomial-time algorithm that for an input oracle-given matroid M outputs a contraction*-decomposition tree of depth at most k^2.*

All contractions used in the proof of the inequality $\mathrm{c}^*\mathrm{d}(M) \leq k^2$ are contractions of elements of a matroid M, i.e., the one-dimensional subspaces as in the definition of the contraction*-depth are all generated by elements of M. We remark that this implies that $\mathrm{cd}(M) \leq k^2 + 1$. The sequence of such contractions can be visualized by a *contraction*-decomposition tree* that is defined in the same way as a contraction-decomposition tree except that one-vertex trees are associated with matroids of rank zero (rather than matroids consisting of a single element), however, the edges of the tree are still labeled by some of the elements of M. Note that the minimum depth of a contraction*-decomposition tree of a matroid M is an upper bound on its contraction*-depth, however, in general, the contraction*-depth of a matroid M can be smaller than the minimum depth of contraction*-decomposition tree of M as the definition of the contraction*-depth in the case of represented matroids permits contractions by arbitrary one-dimensional subspaces.

We next introduce the parameter of contraction*-deletion-depth, which we believe to have not been yet studied previously, but which is particularly relevant in our context. To avoid unnecessary technical issues, we introduce the parameter for represented matroids only. The *contraction*-deletion-depth* of a represented matroid M, denoted by $\mathrm{c}^*\mathrm{dd}(M)$, is defined recursively as follows. If M has rank zero, then $\mathrm{c}^*\mathrm{dd}(M) = 0$; if M has a single non-loop element, then $\mathrm{c}^*\mathrm{dd}(M) = 1$. If M is not connected, then $\mathrm{c}^*\mathrm{dd}(M)$ is the maximum contraction*-deletion-depth of a component of M. Otherwise, $\mathrm{c}^*\mathrm{dd}(M)$ is 1 plus the smaller among the minimum contraction*-deletion-depth of the matroid $M \setminus e$, where the minimum is taken over all elements of M, and the minimum contraction*-deletion-depth of a matroid obtained from M by factoring along an arbitrary one-dimensional subspace. Observe that $\mathrm{c}^*\mathrm{dd}(M) \leq \mathrm{cdd}(M)$ and $\mathrm{c}^*\mathrm{dd}(M) \leq \mathrm{c}^*\mathrm{d}(M)$ for every matroid M.

Finally, if A is a matrix, the *deletion-depth*, *contraction-depth*, etc. of A is the corresponding parameter of the vector matroid $M(A)$ formed by the columns of A, and we write $\mathrm{dd}(A)$, $\mathrm{cd}(A)$, etc. for the deletion-depth, contraction-depth, etc. of the matrix A. Observe that the deletion-depth, contraction-depth etc. of a matrix A is invariant under row operations as row operations preserve the matroid $M(A)$.

3 Structural results

In this section, we prove our structural results concerning optimal primal tree-depth and optimal incidence tree-depth of a matrix. We start with presenting an algorithm, which uses a deletion-decomposition tree of the matroid associated with a given matrix to construct a matrix with small primal tree-depth that is equivalent to the given matrix.

▶ **Lemma 10.** *There exists a polynomial-time algorithm that for an input matrix A and a deletion-decomposition tree of $M(A)$ with height d outputs a matrix A' equivalent to A such that $\mathrm{td}_P(A') \leq d$.*

Proof. We establish the existence of the algorithm by proving that $\mathrm{td}_P(A') \leq d$ in a constructive (algorithmic) way. Due to space constraints, we omit some details in the arguments that follow.

Fix a matrix A and a deletion-decomposition tree T of $M(A)$ with height d. Let X be the set of non-zero columns that are labels of the vertices of T. It can be shown that the columns contained in X form a basis of the column space of the matrix A. In particular, unless A is the zero matrix, the set X is non-empty. Let A' be the matrix obtained from A by row operations such that the submatrix of A' induced by the columns of X is the unit matrix with possibly some additional zero rows. We will prove by induction on the number of columns of an input matrix A that the primal tree-depth of A' is at most d.

The base of the induction is the case when A has a single column. In this case, the primal tree-depth of A' is one and the tree T is a single vertex labeled with the only column of A, and so its height is one.

We next present the induction step. First observe that every label of a vertex of T is either in X or a loop in $M(A)$, and every label of an edge e is a linear combination of labels of the vertices in the subtree delimited by e. It follows that if x is a label of the root of T, then x is either a loop or a bridge in the matroid $M(A)$. Let B be the matrix obtained from A by deleting the column x, and let T' be the deletion-decomposition tree of $M(B)$ obtained from T by removing the label x from the root. Since the matrix B' produced by the algorithm described above for B and T' is the submatrix of A' formed by the columns different from x (possibly after permuting rows) and the vertex associated with the column x is isolated in the primal graph of A', it follows that $\mathrm{td}_P(A') = \mathrm{td}_P(B') \leq d$ (the inequality holds by the induction hypothesis). Hence, we can assume that the root of T has no label.

We now analyze the case that the root of T has a single child and no label. Let x be the label of the single edge incident with the root of T, and let B' be the matrix obtained from A' by deleting the column x. By induction, the primal tree-depth of B' is at most $d-1$, which implies that the primal tree-depth of A' is at most $\mathrm{td}_P(B') + 1 = d$.

The final case to analyze is the case when the root of T has $k \geq 2$ children (in addition to having no label). Let $Y_1, \ldots, Y_k$ be the labels of the vertices and edges of the k subtrees of T delimited by the k edges incident with the root of T, and let $B_1, \ldots, B_k$ be the submatrices of A formed by the columns contained in $Y_1, \ldots, Y_k$. Since the support of the columns contained in Y_i contains only the unit entries of the columns of A' contained in $X \cap Y_i$, the primal graph of A' contains no edge joining a column of Y_i and a column of Y_j for $i \neq j$. It follows that the primal tree-depth of A' is at most the maximum primal tree-depth of B_i, which is at most d by the induction hypothesis. It follows that $\mathrm{td}_P(A') \leq d$ as desired. ◀

We are now ready to establish the link between the optimal primal tree-depth of a matrix and the deletion-depth of the matroid associated with the matrix. Due to space constraints, the proof of Theorem 4 is sketched only.

Sketch of the proof of Theorem 4. Fix a matrix A. By Lemma 10, it holds that $\mathrm{td}^*_P(A) \leq \mathrm{dd}(A)$. So, we focus on proving that $\mathrm{dd}(A) \leq \mathrm{td}^*_P(A)$. We will show that every matrix B satisfies that $\mathrm{dd}(B) \leq \mathrm{td}_P(B)$, which implies that $\mathrm{dd}(A) \leq \mathrm{td}^*_P(A)$.

The proof that $\mathrm{dd}(B) \leq \mathrm{td}_P(B)$ proceeds by induction on the number of columns. If B has a single column, then both $\mathrm{dd}(B)$ and $\mathrm{td}_P(B)$ are equal to one. We next present the induction step. If the matroid $M(B)$ is not connected, we apply induction to each of its components and derive that each component of $M(B)$ has deletion depth at most $\mathrm{td}_P(B)$. This implies that $\mathrm{dd}(B) \leq \mathrm{td}_P(B)$.

We next assume that the matroid $M(B)$ is connected. It can be shown that the primal graph of B is also connected, which yields that there exists a column such that the matrix B' obtained by deleting this column satisfies that $\mathrm{td}_P(B') = \mathrm{td}_P(B) - 1$. The induction assumption yields that $\mathrm{dd}(B') \leq \mathrm{td}_P(B) - 1$ and the definition of the deletion-depth yields that the deletion-depth of $M(B)$ is at most the deletion-depth of $M(B')$ increased by one. This implies that $\mathrm{dd}(B) = \mathrm{dd}(M(B)) \leq \mathrm{td}_P(B)$ as desired. ◀

Before proceeding with our structural result concerning incidence tree-depth, we use the structural results presented in Lemma 10 and Theorem 4 to get a parameterized algorithm for computing an optimal primal tree-depth of a matrix over a finite field. Due to space constraints, several steps of the proof of Corollary 11 are sketched only.

▶ **Corollary 11.** *There exists an FPT algorithm for the parameterization by a finite field $\mathbb{F}$ and an integer d that for an input matrix A over the field $\mathbb{F}$,*

- *either outputs that $\mathrm{td}^*_P(A) > d$, or*
- *computes a matrix A' equivalent to A with $\mathrm{td}_P(A') \leq d$ and also outputs the associated deletion-decomposition tree of $M(A)$ with height $\mathrm{td}_P(A')$.*

Proof. The property that a matroid M has deletion depth at most d can be expressed in monadic second order logic. Specifically, there exists a monadic second order formula $\psi_d(X)$ that describes whether the deletion-depth of a restriction of the matroid M to a subset X of the elements of M is at most d. In the formula that we present, small letters are used to denote elements of a matroid and capital letters subsets of the elements. Assuming that $\psi_c(x, y)$ is a monadic second order formula describing the existence of a circuit containing two elements x and y and $\psi_C(X)$ is a monadic second order formula describing whether a subset X is a component of a matroid, The sought formula $\psi_d(\cdot)$ is defined inductively (note that $\psi_d(\emptyset)$ is true for all d):

$$\psi_1(X) \equiv \forall x, y \in X : x \neq y \Rightarrow \neg\psi_c(x, y) \qquad \text{and}$$
$$\psi_d(X) \equiv \forall X' \subseteq X : \psi_C(X') \Rightarrow \exists x \in X' : \psi_{d-1}(X' \setminus \{x\}) \qquad \text{for } d \geq 2.$$

Hliněný [25, 26] proved that all monadic second order logic properties can be tested in a fixed parameter way for matroids represented over a finite field $\mathbb{F}$ with branch-width at most d when parameterized by the property, the field $\mathbb{F}$ and the branch-width d. Since the branch-width of a matroid M is at most its deletion-depth, this establishes the existence of a fixed parameter algorithm deciding whether $\mathrm{td}^*_P(A) = \mathrm{dd}(M(A)) \leq d$ (the equality follows from Theorem 4).

To obtain the algorithm claimed to exist in the statement of the corollary, we need to extend the algorithm for testing whether the deletion-depth of an input matroid M represented over $\mathbb{F}$ is at most d to an algorithm that also outputs a deletion-decomposition tree of M with height at most d. The deletion-depth of an input matroid M is one if and only if every element of M is a loop or a bridge. Hence, if the deletion-depth of M is $d = 1$, then the deletion-depth decomposition tree of height one consists of a single vertex labeled

with all elements of M. If the deletion-depth of M is at most $d \geq 2$ and M is not connected, we first identify its components, which can be done in polynomial time even in the oracle model, then proceed recursively with each component of M and eventually merge the roots of all deletion-depth decomposition trees obtained recursively. Finally, if the deletion-depth of M is at most $d \geq 2$ and M is connected, we loop over all elements e of M and test whether $\mathrm{dd}(M \setminus e) \leq d-1$. Such an element e must exist and when it is found, we recursively apply the algorithm to $M \setminus e$ to obtain a deletion-depth decomposition tree T of $M \setminus e$ with height $d-1$, which is then extended to a deletion-depth decomposition tree of M of height d. ◀

We conclude this section by establishing a link between the optimal incidence tree-depth and the contraction*-deletion-depth of the matroid associated with the matrix. Due to space constraints, the proof of Theorem 5 is sketched only.

Proof of Theorem 5. We prove the equality as two inequalities starting with the inequality $\mathrm{c}^*\mathrm{dd}(A) \leq \mathrm{td}^*_I(A) - 1$. To prove this inequality, we show that $\mathrm{c}^*\mathrm{dd}(A) \leq \mathrm{td}_I(A) - 1$ holds for every matrix A with m rows and n columns by induction on $m+n$. The base of the induction is formed by the cases when all entries of A are zero, $n = 1$ or $m = 1$. We focus on describing the induction step, which considers the case when A is non-zero, $m \geq 2$ and $n \geq 2$. First suppose that the matroid $M(A)$ is not connected. Let $X_1, \ldots, X_k$ be the component of $M(A)$ and let $A_1, \ldots, A_k$ be the submatrices of A formed by the columns $X_1, \ldots, X_k$, respectively. The definition of the contraction*-deletion-depth implies that $\mathrm{c}^*\mathrm{dd}(A)$ is the maximum of $\mathrm{c}^*\mathrm{dd}(A_i)$. The induction hypothesis yields that $\mathrm{c}^*\mathrm{dd}(A_i) \leq \mathrm{td}_I(A_i) - 1$. Since the incidence graph of A_i is a subgraph of the incidence graph of A, it follows that $\mathrm{td}_I(A_i) \leq \mathrm{td}_I(A)$ and so $\mathrm{c}^*\mathrm{dd}(A_i) \leq \mathrm{td}_I(A) - 1$. We conclude that $\mathrm{c}^*\mathrm{dd}(A) \leq \mathrm{td}_I(A) - 1$.

The core of the inductive argument showing that $\mathrm{c}^*\mathrm{dd}(A) \leq \mathrm{td}_I(A) - 1$ is formed by the case when the matroid $M(A)$ is connected and the incidence graph of A is also connected. The definition of the tree-depth implies that there exists a vertex w of the incidence graph whose deletion decreases the tree-depth of the incidence graph by one. Let A' be the matrix obtained from A by deleting the row or the column associated with the vertex w and note that $\mathrm{td}_I(A) = \mathrm{td}_I(A') + 1$. If the vertex w is associated with a column x, the matroid $M(A')$ is the matroid obtained from $M(A)$ by deleting the element x. If the vertex w is associated with a row x, the matroid $M(A')$ is the matroid obtained from $M(A)$ by contracting by the subspace generated by the unit vector with the entry in the row x. In either case, the definition of the contraction*-deletion-depth implies that $\mathrm{c}^*\mathrm{dd}(A) \leq \mathrm{c}^*\mathrm{dd}(A') + 1$. The induction hypothesis applied to A' yields that $\mathrm{c}^*\mathrm{dd}(A') \leq \mathrm{td}_I(A') - 1$, which yields that $\mathrm{c}^*\mathrm{dd}(A) \leq \mathrm{td}_I(A') = \mathrm{td}_I(A) - 1$.

To complete the proof of the theorem, it remains to show that the inequality $\mathrm{td}^*_I(A) \leq \mathrm{c}^*\mathrm{dd}(A) + 1$ holds for every matrix A. The proof proceeds by induction on the number n of columns of A. The core of the argument is the inductive step when the matroid $M(A)$ is connected, which we present next. The definition of the contraction*-deletion-depth implies that there exists an element x of $M(A)$ such that $\mathrm{c}^*\mathrm{dd}(M(A)\setminus x) = \mathrm{c}^*\mathrm{dd}(M(A)) - 1 = \mathrm{c}^*\mathrm{dd}(A) - 1$ or there exists a one-dimensional subspace such that the contraction by this subspace yields a matroid M' such that $\mathrm{c}^*\mathrm{dd}(M') = \mathrm{c}^*\mathrm{dd}(M(A)) - 1 = \mathrm{c}^*\mathrm{dd}(A) - 1$. In the former case, let A' be the matrix obtained from A by deleting the column x. By the induction hypothesis, there exists a matrix A'' equivalent to A' such that $\mathrm{td}_I(A'') \leq \mathrm{c}^*\mathrm{dd}(A') + 1 = \mathrm{c}^*\mathrm{dd}(A)$, and let A''' be the matrix obtained from A by the same row operations using that A'' is obtained from A'. Observe that the incidence graph of A'' can be obtained from the incidence graph of A''' by deleting the vertex associated with the column x. Hence, $\mathrm{td}_I(A''') \leq \mathrm{td}_I(A'') + 1$. Since A''' is equivalent to A, it follows that

$$\mathrm{td}^*_I(A) \leq \mathrm{td}_I(A''') \leq \mathrm{td}_I(A'') + 1 \leq \mathrm{c}^*\mathrm{dd}(A) + 1.$$

We now analyze the latter case, i.e., the case that there exists a one-dimensional subspace such that the contraction by this subspace yields a matroid M' such that $\mathrm{c}^*\mathrm{dd}(M') = \mathrm{c}^*\mathrm{dd}(A) - 1$. Let A' be the matrix obtained from A by row operations such that the contracted subspace used to obtain M' is generated by the unit vector with the non-zero entry being the first entry, and let B be the matrix obtained from A' by deleting the first row. By the induction hypothesis, there exists a matrix B' equivalent to B such that $\mathrm{td}_I(B') \leq \mathrm{c}^*\mathrm{dd}(A') + 1 = \mathrm{c}^*\mathrm{dd}(A)$, and let A'' be the matrix consisting of the first row of A and the matrix B'. Observe that A'' is equivalent to A. Since the incidence graph of B' can be obtained from the incidence graph of A'' by deleting the vertex associated with the first row, it holds that $\mathrm{td}_I(A'') \leq \mathrm{td}_I(B') + 1$. Hence, it follows that

$$\mathrm{td}^*_I(A) \leq \mathrm{td}_I(A'') \leq \mathrm{td}_I(B') + 1 \leq \mathrm{c}^*\mathrm{dd}(A) + 1.$$

The proof of the theorem is now completed. ◀

4 Primal tree-depth

In this section, we present a parameterized algorithm for computing an equivalent matrix with small primal tree-depth and bounded entry complexity if such a matrix exists. Due to space constraints, details of some of the arguments used in the proof are omitted.

Proof of Theorem 7. Let $f_P(\cdot,\cdot)$ be the function from the statement of Theorem 1 and set κ_0 to be the least common multiple of the integers $1, \ldots, f_P(d,e)$. Observe that the entry of every circuit of a matrix B with $\mathrm{td}_P(B) \leq d$ and $\mathrm{ec}(B) \leq e$ divides κ_0 as $c_\infty(B) \leq f_P(d,e)$.

We next describe the algorithm from the statement of the theorem. Without loss of generality, we can assume that the rank of the input matrix A is equal to the number of its rows, in particular, A is non-zero. The algorithm starts with diagonalizing the input matrix A by selecting an arbitrary basis of the column space and performing row-operations that the selected columns form the identity matrix. The resulting matrix is denoted by A_I. If the numerator or the denominator of any (non-zero) entry of A_I does not divide κ_0, the algorithm arrives at the first conclusion of the theorem. The algorithm next multiplies each row of A_I by κ_0, This yields an integral matrix A_0 with entries between $-\kappa_0^2$ and κ_0^2.

Let $M_{\mathbb{Q}}$ be the column matroid of A_0 when viewed as a matrix over rationals and let M_p be the column matroid of A_0 when viewed as a matrix over a field $\mathbb{F}_p$ for a prime $p > \kappa_0^2$; note that such a prime p can be found algorithmically as the algorithm is parameterized by d and e. The elements of both matroids $M_{\mathbb{Q}}$ and M_p are the columns of the matrix A_0, i.e., we can assume that their ground sets are the same.

We argue that *if A is equivalent to a matrix with primal tree-depth at most d and entry complexity at most e, then the matroids $M_{\mathbb{Q}}$ and M_p are the same.* As this is the key step in the proof, we present it in full detail. If a set X of columns forms a circuit in $M_{\mathbb{Q}}$, then there exists a linear combination of the columns of X that has all coefficients integral and coprime, i.e., not all are divisible by p, and that is equal to the zero vector (in fact, there exist such coefficients that they all divide κ_0 by the definition of κ_0); it follows that the set X is also dependent in M_p. If a set X of columns is independent in $M_{\mathbb{Q}}$, let B_I be a square submatrix of A_I formed by the columns X and some of the rows such that B_I is non-singular, and let B_0 be the square submatrix of A_0 formed by the same rows and columns. By [17, Lemma 3.3], the matrix B_I^{-1} is $1/\dot{\kappa}_A$-integral and the absolute values of its entries are between $1/c_\infty(A)$ and $c_\infty(A)$. Note that both $c_\infty(A)$ and $\dot{\kappa}_A$ divide κ_0 (here, we use the definition of κ_0 and the assumption that A is equivalent to a matrix with primal tree-depth at most d and entry complexity at most e) and so this the matrix B_I^{-1} is $1/\kappa_0$-integral and the absolute values of its entries are between $1/\kappa_0$ and κ_0. Let B' be the matrix obtained from B_I^{-1} by

multiplying each entry by κ_0; note that B' is an integer matrix with entries between $-\kappa_0^2$ and κ_0^2. The definitions of the matrices B_I, B_0 and B' yield that $B'B_0$ is a diagonal matrix with all diagonal entries equal to κ_0^2. It follows that the columns X form a set independent in M_p.

We now continue the description of the algorithm. As the next step, we apply the algorithm from Corollary 11 to the matrix A_0 viewed as over the field $\mathbb{F}_p$. If the algorithm determines that the deletion-depth of A_0 is larger than d, we arrive at the first conclusion of the theorem. If it determines that the deletion-depth of A_0 is at most d, it also outputs a deletion-decomposition tree of M_p with height at most d. If the output deletion-decomposition tree is not valid for the matroid $M_{\mathbb{Q}}$, we again arrive at the first conclusion of the theorem. If the output deletion-decomposition tree is also a deletion-decomposition tree of the matroid $M_{\mathbb{Q}}$, we use the algorithm from Lemma 10 to obtain a matrix A' equivalent to A such that the primal tree-depth of A' at most the height of the deletion-decomposition tree, i.e., $\mathrm{td}_P(A') \leq d$. As the matrix A' contains a unit submatrix formed by m rows and m columns, each (non-zero) entry of A' is a fraction that can obtained by dividing two entries of a circuit of A. If the absolute value of the numerator or the denominator of any these fractions exceeds κ_0, we again arrive at the first conclusion of the theorem. Otherwise, we output the matrix A'. Note that the primal tree-depth of A' is at most d and its entry complexity is at most $2\lceil\log_2(\kappa_0+1)\rceil$ and κ_0 depends on d and e only. ◀

5 Dual tree-depth, circuit complexity and Graver basis

In this section, we link minimum dual tree-depth of a matrix to its circuit complexity. Due to space constraints, details of some of the arguments used in the proof are omitted.

▶ **Theorem 12.** *There exists a polynomial-time algorithm that for a given matrix A whose rank is smaller than the number of its columns outputs an equivalent matrix A' such that $\mathrm{td}_D(A') \leq c_1(A)^2$ and $\mathrm{ec}(A') \leq 2\lceil\log_2 c_1(A)\rceil$.*

Proof. We start with the description of the algorithm from the statement of the theorem. Let A be the input matrix. We apply the algorithm from Theorem 9 to the matroid $M(A)$ given by the columns of the matrix A. Let T be the contraction*-decomposition tree output by the algorithm, let d be its depth and let X be the set of columns of A that are the labels of the edges of T, i.e., the elements of $M(A)$ used in the contractions. Since the columns contained in X form a base of the column space of A, we can perform row-operations on the matrix A in a way that the submatrix formed by the columns of X is an identity matrix; let A' be the resulting matrix. The algorithm outputs the matrix A'.

We now analyze the matrix A' that is output by the algorithm. Since the rank of A is smaller than the number of its columns, the matrix A has at least one circuit and so $c_1(A) \geq 2$. Observe that every circuit of $M(A)$ contains at most $c_1(A)$ elements and so it holds that $d \leq c_1(A)^2$, i.e., the depth of T is at most $c_1(A)^2$. Let F be a rooted forest obtained from T by removing the root and for each edge e, associating the vertex of e farther from the root of T with the (unique) row of A' that is non-zero in the column that is the label of e. Hence, the vertex set of F is formed by the rows of A'. Since it can be shown that the dual graph of A' is a subgraph of $\mathrm{cl}(F)$, it follows that that $\mathrm{td}_D(A') \leq c_1(A)^2$ (note that the height of F is at most $c_1(A)^2$).

It remains to analyze the entry complexity of A'. The entries of A' in the columns of X are zero or one. Every column z of A' that is not contained in X forms a circuit with some of the columns of X. The entries of A' in the column z are equal to $-c_x/c_z$ where c_x, $x \in X$, and c_z are the corresponding integer entries of this circuit of A' (and so of A). We conclude that the entry complexity of A' is at most $2\lceil\log_2 c_1(A)\rceil$. ◀

Theorem 12 implies that $g_1(A) \leq f_D\left(c_1(A)^2, 2\lceil \log_2 c_1(A)\rceil\right)$ for every matrix A with $\mathcal{C}(A) \neq \emptyset$, where f_D is the function from Theorem 1. Note that the condition $\mathcal{C}(A) \neq \emptyset$ is necessary as otherwise A has no circuit and so $c_1(A)$ is not defined. We state this bound as a corollary.

▶ **Corollary 13.** *There exists a function $f : \mathbb{N} \to \mathbb{N}$ such that $g_1(A) \leq f(c_1(A))$ for every matrix A with $\mathcal{C}(A) \neq \emptyset$.*

We next combine the algorithms from Theorems 2 and 12.

▶ **Corollary 14.** *There exists a function $f : \mathbb{N} \to \mathbb{N}$ and an FPT algorithm for the parameterization by k that for a given rational matrix A:*

- *either outputs that $c_1(A) > k$, or*
- *outputs a matrix A' that is equivalent to A, its dual tree-depth is $\mathrm{td}^*_D(A)$ and its entry complexity is at most $f(k)$.*

Proof of Corollary14. For an input matrix A, we apply the algorithm from Theorem 12 to get a matrix A' equivalent to A. If the dual tree-depth of A' is larger than k^2 or the entry complexity of A' is larger than $2\lceil \log_2 k\rceil$, then $c_1(A) > k$ and we arrive at the first conclusion. Otherwise, we apply the algorithm from Theorem 2 with parameters $d = k^2$ and $e = 2\lceil \log_2 k\rceil$ to compute a matrix A'' equivalent to A' and so to A such that the dual tree-depth of A'' is $\mathrm{td}^*_D(A)$ and the entry complexity of A'' is $O(k^4 2^{2k^2} \log k)$. ◀

Finally, the previous corollary together with Theorem 1 yields the parameterized algorithm for testing whether an input matrix is equivalent to a matrix with small dual tree-depth and small entry complexity as given in Theorem 8.

Proof of Theorem 8. Let f_D be the function from the statement of Theorem 1 and set $k = f_D(d, e)$. Apply the algorithm from Corollary 14 with the parameter k to an input matrix A. If the algorithm reports that $c_1(A) > k$, then A is not equivalent to a matrix with dual tree-depth at most d and entry complexity at most e. If the algorithm outputs a matrix A' and $\mathrm{td}_D(A') > d$, then $\mathrm{td}^*_D(A) > d$ and so the matrix A is not equivalent to a matrix with dual tree-depth at most d. Otherwise, the dual tree-depth of A' is at most d and its entry complexity is $2^{O(k^2)} = 2^{O\left(f_D(d,e)^2\right)}$, i.e., it is bounded by a function of d and e only as required. ◀

6 Computational hardness of depth parameters

In this section, we complement our algorithmic results by establishing computational hardness of matroid depth parameters that we discussed in this paper. The hardness results apply even when the input matroid is given by its representation over a fixed (finite or infinite) field.

We start with defining a matroid $M_{\mathbb{F}}(G)$ derived from a graph G. Fix a field $\mathbb{F}$. For a graph G, we define an $\mathbb{F}$-represented matroid $M_{\mathbb{F}}(G)$ as follows. The matroid $M_{\mathbb{F}}(G)$ contains $|G| + \|G\|$ elements, which correspond to the vertices and the edges of G. We next associate each element of $M_{\mathbb{F}}(G)$ with a vector of $\mathbb{F}^{V(G)}$. An element of $M_{\mathbb{F}}(G)$ corresponding to a vertex w of G is represented by e_w and an element of $M_{\mathbb{F}}(G)$ corresponding to an edge ww' of G is represented by $e_w - e_{w'}$.

We next define a graph G/A for a graph G and a linear subspace A of $\mathbb{F}^{V(G)}$. Let W be the subset of vertices of $V(G)$ such that $e_w \in A$ for $w \in W$, and let F be a maximal subset of edges $ww' \in E(G)$ such that

- A contains a vector $e_w + \alpha e_{w'}$ for a non-zero element $\alpha \in \mathbb{F}$, and
- the set containing all vectors e_w, $w \in W$, and all vectors $e_w - e_{w'}$, $ww' \in F$, is linearly independent.

The graph G/A is obtained by deleting all vertices of W and then contracting all edges contained in F. The next lemma asserts that G/A is well-defined. We remark that the proof of the next lemma as well as the proofs of Lemmas 16 and 17 are omitted due to space constraints.

▶ **Lemma 15.** *Let G be a graph, $\mathbb{F}$ a field and A a linear subspace of $\mathbb{F}^{V(G)}$. The graph G/A is well-defined, i.e., the graph G/A does not depend on the choice of the set F in its definition.*

The next lemma relates the number of components of the matroid $M_{\mathbb{F}}(G)/A$ and the number of components of the graph G/A for a graph G and a linear subspace A of $\mathbb{F}^{V(G)}$.

▶ **Lemma 16.** *Let G be a graph, $\mathbb{F}$ a field and A a linear subspace of $\mathbb{F}^{V(G)}$. The number of components of $M_{\mathbb{F}}(G)/A$ is at most the number of components of the graph G/A.*

We next link the existence of a balanced independent set in a bipartite graph to the contraction*-depth of a suitably defined matroid. We remark that the idea of using a bipartite graph with cliques added between the vertices of its parts was used in [43] to establish that computing tree-depth of a graph is NP-complete.

▶ **Lemma 17.** *Let G be a bipartite graph with parts X and Y, let $\mathbb{F}$ be a field, and let k be an integer. Let G' be the graph obtained from G by adding all edges between the vertices of X and between the vertices of Y. The following three statements are equivalent.*

- *There exists an independent set containing k elements of X and k elements of Y.*
- *The contraction*-depth of $M_{\mathbb{F}}(G')$ is at most $|X| + |Y| - k$.*
- *The contraction-depth of the matroid $2M_{\mathbb{F}}(G')$ is at most $|X| + |Y| - k + 1$.*

We are now ready to state and prove our hardness result.

▶ **Theorem 18.** *For every field $\mathbb{F}$, each of the following five decision problems, whose input is an $\mathbb{F}$-represented matroid M and an integer d, is NP-complete:*

- *Is the contraction-depth of M at most d?*
- *Is the contraction*-depth of M at most d?*
- *Is the contraction-deletion-depth of M at most d?*
- *Is the contraction*-deletion-depth of M at most d?*
- *Is the deletion-depth of M at most d?*

Proof. It is NP-complete to decide for a bipartite graph G with parts X and Y and an integer k whether there exist k-element subsets $X' \subseteq X$ and $Y' \subseteq Y$ such that $X' \cup Y'$ is independent [43]. For an input bipartite graph G, let G' be the graph obtained from G by adding all edges between the vertices of X and between the vertices of Y. We claim that the existence of such subsets X' and Y' is equivalent to each of the following four statements:

- The matroid $2M_{\mathbb{F}}(G')$ has contraction-depth at most $|X| + |Y| - k + 1$.
- The matroid $M_{\mathbb{F}}(G')$ has contraction*-depth at most $|X| + |Y| - k$.
- The matroid $(|G'| + 1)M_{\mathbb{F}}(G')$ has contraction-deletion-depth at most $|X| + |Y| - k + 1$.
- The matroid $(|G'| + 1)M_{\mathbb{F}}(G')$ has contraction*-deletion-depth at most $|X| + |Y| - k$.

The equivalence of the first and second statements follow directly from Lemma 17. Since the rank of the matroid $(|G'|+1)M_{\mathbb{F}}(G')$ is $|G'|$, its contraction-deletion-depth is at most $|G'|+1$ and its contraction*-deletion-depth is at most $|G'|$. As each element of the matroid $(|G'|+1)M_{\mathbb{F}}(G')$ is parallel to (at least) $|G'|$ elements of the matroid, it follows that the contraction-deletion-depth of $M_{\mathbb{F}}(G')$ is the same as its contraction-depth and its contraction*-deletion-depth is the same as its contraction*-depth. Lemma 17 now implies the equivalence of the third and fourth statements. As the matroids $2M_{\mathbb{F}}(G')$, $M_{\mathbb{F}}(G')$ and $(|G'|+1)M_{\mathbb{F}}(G')$ can be easily constructed from the input graph G in time polynomial in $|G|$, the NP-completeness of the first four problems listed in the statement of the theorem follows.

For an $\mathbb{F}$-represented matroid M, it is easy to construct an $\mathbb{F}$-represented matroid M^* that is dual to M in time polynomial in the number of the elements of M [42, Chapter 2]. Since the contraction-depth of M is equal to the deletion-depth of M^*, it follows that the fifth problem listed in the statement of the theorem is also NP-complete. ◀

References

1 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007.

2 Cevdet Aykanat, Ali Pinar, and Ümit V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 25(6):1860–1879, 2004.

3 Martin Bergner, Alberto Caprara, Alberto Ceselli, Fabio Furini, Marco E Lübbecke, Enrico Malaguti, and Emiliano Traversi. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1-2):391–424, 2015.

4 Ralf Borndörfer, Carlos E Ferreira, and Alexander Martin. Decomposing matrices into blocks. *SIAM Journal on Optimization*, 9(1):236–269, 1998.

5 Robert Bredereck, Aleksander Figiel, Andrzej Kaczmarczyk, Dušan Knop, and Rolf Niedermeier. High-multiplicity fair allocation made more practical. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 260–268. International Foundation for Autonomous Agents and Multiagent Systems, 2021.

6 Robert Bredereck, Andrzej Kaczmarczyk, Dušan Knop, and Rolf Niedermeier. High-multiplicity fair allocation: Lenstra empowered by n-fold integer programming. In *Proceedings of the 20th ACM Conference on Economics and Computation*, pages 505–523. Association for Computing Machinery, 2019.

7 Timothy F. N. Chan, Jacob W. Cooper, Martin Koutecký, Daniel Král', and Kristýna Pekárková. Matrices of optimal tree-depth and row-invariant parameterized algorithm for integer programming. To appear in SIAM Journal on Computing.

8 Timothy F. N. Chan, Jacob W. Cooper, Martin Koutecký, Daniel Král', and Kristýna Pekárková. Matrices of optimal tree-depth and row-invariant parameterized algorithm for integer programming. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:19, 2020.

9 Hua Chen, Lin Chen, and Guochuan Zhang. Fpt algorithms for a special block-structured integer program with applications in scheduling. *preprint arXiv:2107.01373*, 2021.

10 Lin Chen and Dániel Marx. Covering a tree with rooted subtrees–parameterized and approximation algorithms. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2801–2820. SIAM, 2018.

11 Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time. In *32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1666–1681. SIAM, 2021.

12 Jana Cslovjecsek, Friedrich Eisenbrand, Michal Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:14, 2021.

13 Jesús A. De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. N-fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008.

14 Matt DeVos, O-joung Kwon, and Sang-il Oum. Branch-depth: Generalizing tree-depth of graphs. *European Journal of Combinatorics*, 90:103186, 2020.

15 G. Ding, B. Oporowski, and J. Oxley. On infinite antichains of matroids. *Journal of Combinatorial Theory Series B*, 63(1):21–40, 1995.

16 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster Algorithms for Integer Programs with Block Structure. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:13, 2018.

17 Farbod Ekbatani, Bento Natura, and László A. Végh. Circuit imbalance measures and linear programming. *preprint arXiv:2108.03616*, 2021.

18 Michael C. Ferris and Jeffrey D. Horn. Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80(1):35–61, 1998.

19 Gerald Gamrath and Marco E. Lübbecke. Experiments with a generic Dantzig–Wolfe decomposition for integer programs. *Experimental Algorithms*, 6049:239–252, 2010.

20 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artificial Intelligence*, 257:61–71, 2018.

21 P.R. Halmos. *Finite-Dimensional Vector Spaces*. Undergraduate Texts in Mathematics. Springer, 1993.

22 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, 137:325–341, 2013.

23 Raymond Hemmecke and Rüdiger Schultz. Decomposition of test sets in stochastic integer programming. *Mathematical Programming*, 94:323–341, 2003.

24 Danny Hermelin, Hendrik Molter, Rolf Niedermeier, and Dvir Shabtay. Equitable scheduling for the total completion time objective. *preprint arXiv:2112.13824*, 2021.

25 Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. In *20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *LNCS*, pages 319–330, 2003.

26 Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *Journal of Combinatorial Theory Series B*, 96(3):325–351, 2006.

27 Klaus Jansen, Kim-Manuel Klein, and Alexandra Lassota. The double exponential runtime is tight for 2-stage stochastic ILPs. In *Integer Programming and Combinatorial Optimization - 22nd International Conference (IPCO)*, volume 12707 of *Lecture Notes in Computer Science*, pages 297–310. Springer, 2021.

28 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-ip: new ptas results for scheduling with setup times. To appear in Mathematical Programming.

29 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-IP – new PTAS results for scheduling with setups times. *preprint arXiv:1801.06460*, 2018.

30 Klaus Jansen, Alexandra Lassota, and Marten Maack. Approximation algorithms for scheduling with class constraints. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 349–357. Association for Computing Machinery, 2020.

31 Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. Near-linear time algorithm for n-fold ILPs via color coding. *SIAM Journal on Discrete Mathematics*, 34(4):2282–2299, 2020.

32 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, August 1987.

33 František Kardoš, Daniel Král', Anita Liebenau, and Lukáš Mach. First order convergence of matroids. *European Journal of Combinatorics*, 59:150–168, 2017.

34 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations, The IBM Research Symposia Series*, pages 85–103. Springer, 1972.

35 Taghi Khaniyev, Samir Elhedhli, and Fatih Safa Erenay. Structure detection in mixed-integer programs. *INFORMS Journal on Computing*, 30(3):570–587, 2018.

36 Kim-Manuel Klein. About the complexity of two-stage stochastic IPs. To appear in Mathematical Programming.

37 Kim-Manuel Klein and Janina Reuter. Collapsing the tower - on the complexity of multistage stochastic ips. In *33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 348–358. SIAM, 2022.

38 Dušan Knop and Martin Koutecký. Scheduling kernels via configuration LP. *preprint arXiv:2003.02187*, 2018.

39 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018.

40 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107, pages 85:1–85:14, 2018.

41 Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

42 James Oxley. *Matroid Theory*. Oxford Graduate Texts in Mathematics. Oxford University Press, 2011.

43 Alex Pothen. The complexity of optimal elimination trees. *Technical Report CS-88-13, Pennsylvania State University*, 1988.

44 Rüdiger Schultz, Leen Stougie, and Maarten H. van der Vlerk. Solving stochastic programs with integer recourse by enumeration: A framework using gröbner basis reductions. *Mathematical Programming*, 83:229–252, 1998.

45 François Vanderbeck and Laurence A Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

46 Jiadong Wang and Ted Ralphs. Computational experience with hypergraph-based methods for automatic decomposition in discrete optimization. In *Proceedings of the 10th International Conference on Integration of Constraint Programming*, pages 394–402. Springer, 2013.

47 Roman L. Weil and Paul C. Kettler. Rearranging matrices to block-angular form for decomposition (and other) algorithms. *Management Science*, 18(1):98–108, 1971.

A Structural Investigation of the Approximability of Polynomial-Time Problems

Karl Bringmann
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Alejandro Cassis
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Nick Fischer
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Marvin Künnemann
TU Kaiserslautern, Germany

Abstract

An extensive research effort targets optimal (in)approximability results for various NP-hard optimization problems. Notably, the works of (Creignou'95) as well as (Khanna, Sudan, Trevisan, Williamson'00) establish a tight characterization of a large subclass of MaxSNP, namely Boolean MaxCSPs and further variants, in terms of their polynomial-time approximability. Can we obtain similarly encompassing characterizations for classes of *polynomial-time optimization problems*?

To this end, we initiate the systematic study of a recently introduced polynomial-time analogue of MaxSNP, which includes a large number of well-studied problems (including Nearest and Furthest Neighbor in the Hamming metric, Maximum Inner Product, optimization variants of k-XOR and Maximum k-Cover). Specifically, for each k, MaxSP_k denotes the class of $O(m^k)$-time problems of the form $\max_{x_1,\dots,x_k} \#\{y : \phi(x_1,\dots,x_k,y)\}$ where ϕ is a quantifier-free first-order property and m denotes the size of the relational structure. Assuming central hypotheses about clique detection in hypergraphs and exact Max-3-SAT, we show that for any MaxSP_k problem definable by a quantifier-free m-edge *graph* formula ϕ, the *best possible* approximation guarantee in faster-than-exhaustive-search time $O(m^{k-\delta})$ falls into one of four categories:

- optimizable to exactness in time $O(m^{k-\delta})$,
- an (inefficient) approximation scheme, i.e., a $(1+\varepsilon)$-approximation in time $O(m^{k-f(\varepsilon)})$,
- a (fixed) constant-factor approximation in time $O(m^{k-\delta})$, or
- an m^{ε}-approximation in time $O(m^{k-f(\varepsilon)})$.

We obtain an almost complete characterization of these regimes, for MaxSP_k as well as for an analogously defined minimization class MinSP_k. As our main technical contribution, we show how to rule out the existence of *approximation schemes* for a large class of problems admitting constant-factor approximations, under a hypothesis for *exact* Sparse Max-3-SAT algorithms posed by (Alman, Vassilevska Williams'20). As general trends for the problems we consider, we observe: (1) Exact optimizability has a simple algebraic characterization, (2) only few maximization problems do not admit a constant-factor approximation; these do not even have a subpolynomial-factor approximation, and (3) constant-factor approximation of minimization problems is *equivalent* to deciding whether the optimum is equal to 0.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Approximation algorithms analysis

Keywords and phrases Classification Theorems, Hardness of Approximation in P, Fine-grained Complexity Theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.30

Category Track A: Algorithms, Complexity and Games

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 30; pp. 30:1–30:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version *Full Version*: https://arxiv.org/abs/2204.11681

Funding *Karl Bringmann, Alejandro Cassis, Nick Fischer:* This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979). *Marvin Künnemann:* Part of this research was performed at ETH Zürich, Institute for Theoretical Studies, supported by Dr. Max Rössler, by the Walter Haefner Foundation, and by the ETH Zürich Foundation.

1 Introduction

For many optimization problems, the best known exact algorithms essentially explore the complete search space, up to low-order improvements. While this holds true in particular for NP-hard optimization problems such as maximum satisfiability, also common polynomial-time optimization problems like Nearest Neighbor search are no exception. When facing such a problem, the perhaps most common approach is to relax the optimization goal and ask for *approximations* rather than optimal solutions. Can we obtain an approximate value significantly faster than exhaustive search, and if so, what is the best approximation guarantee we can achieve?

Even considering polynomial-time problems only, the study of such questions has led to significant algorithmic breakthroughs, such as locality-sensitive hashing (LSH) [49, 41, 11], subquadratic-time approximation algorithms for Edit Distance [56, 18, 17, 19, 14, 12, 26, 55, 21, 43, 13], scaling algorithms for graph problems [37, 70, 34], and fast approximation algorithms via the polynomial method [5, 6] (which lead to the currently fastest known exponential-time approximation schemes for MaxSAT).

These algorithmic breakthroughs have recently been complemented by exciting tools for proving hardness of approximation in P: Most notably, the distributed PCP framework [4] has lead to strong conditional lower bounds, including fundamental limits for Nearest Neighbor search [62], as well as tight approximability results for the Maximum Inner Product problem [28]. Other technical advances include evidence against deterministic approximation schemes for Longest Common Subsequence [1, 3, 29], strong (at times even tight) problem-specific hardness results such as [61, 22, 16, 25, 52], the first fine-grained equivalences of approximation in P results [30, 29], and related works on parameterized inapproximability [27, 51, 59], see [36] for a survey.

These strong advances from both sides shift the algorithmic frontier and the frontier of conditional hardness towards each other. Consequently, it becomes increasingly important to generalize isolated results – both algorithms and reductions – towards making these frontiers explicit: A more comprehensive description of current techniques might enable us to understand general trends underlying these works and to highlight the most pressing limitations of current methods. In fact, we view this as one of the fundamental tasks and uses of fine-grained algorithm design & complexity.

Optimization Classes in P. To study (hardness of) approximation in P in a general way, we study a class MaxSP_k recently introduced in [23] (see the full version of this paper for a comparison to the classic class MaxSNP, which motivated the definition of MaxSP_k). This class consists of polynomial-time optimization problems of the form:

$$\max_{x_1,\dots,x_k} \#\{y : \phi(x_1,\dots,x_k,y)\},$$

where ϕ is a quantifier-free first-order property.[1] One obtains an analogous minimization class MinSP$_k$ by replacing maximization by minimization. A large number of natural and well-studied problems can be expressed this way: In particular, we may think of each x_i ranging over a set X_i of n vectors in $\{0,1\}^d$, and the task is to maximize (or minimize) the number of coordinates y satisfying an arbitrary Boolean function over the coordinate values $x_1[y], \dots, x_k[y]$ (here, ϕ is defined using a binary relation $R \subseteq (X_1 \cup \dots \cup X_k) \times Y$ that expresses whether the y-th coordinate of x_i is 0 or 1; see Section 2 for details). In particular, this class of problems includes:

- (Offline) Furthest/Nearest Neighbor search in the Hamming metric ($\max_{x_1,x_2} / \min_{x_1,x_2} d_H(x_1,x_2)$),
- Maximum/Minimum Inner Product; the latter is the optimization formulation of the Most-Orthogonal Vectors problem [2] ($\max_{x_1,x_2} / \min_{x_1,x_2} \langle x_1, x_2 \rangle$),
- A natural similarity search problem that we call Maximum k-Agreement ($\max_{x_1,\dots,x_k} \#\{y : x_1[y] = \dots = x_k[y]\}$),
- Maximum k-Cover [35, 31, 59] and its variation Maximum Exact-k-Cover [54] ($\max_{x_1,\dots,x_k} \#\{y : x_i[y] = 1 \text{ for some } i\}$, $\max_{x_1,\dots,x_k} \#\{y : x_i[y] = 1 \text{ for exactly one } i\}$)[2],
- The canonical optimization variants of the k-XOR problem [50, 33] ($\max_{x_1,\dots,x_k} / \min_{x_1,\dots,x_k} \#\{y : x_1[y] \oplus \dots \oplus x_k[y] = 0\}$).

  By the standard split-and-list technique [66], it is easy to see that any c-approximation for Maximum k-XOR or Minimum k-XOR in time $O(n^{k-\delta}\operatorname{poly}(d))$ gives a c-approximation for Max-LIN (maximize the number of satisfied constraints of a linear system over $\mathbb{F}_2$, see [45, 66, 6]) or the Minimum Distance Problem[3] (finding the minimum weight of a non-zero code word of a linear code over $\mathbb{F}_2$, see [63]), respectively, in time $O(2^{n(1-\delta')})$.

By a simple baseline algorithm, all of these problems can be solved in time $O(m^k)$, where m denotes the input size (for the above setting of k sets of n vectors in $\{0,1\}^d$ we have $m = O(nd)$). A large body of work addresses problems with $k = 2$, typically inventing or adapting strong techniques to each specific problem as needed:

- Abboud, Rubinstein, and Williams introduced the distributed PCP in P framework and ruled out almost-polynomial approximations for the Maximum Inner Product problem assuming the Strong Exponential Time Hypothesis (SETH). Subsequently, Chen [28] strengthened the lower bound and gave an approximation algorithm resulting in tight bounds on the approximability in strongly subquadratic time, assuming SETH. Corresponding inapproximability results for its natural generalization to k-Maximum Inner Product have been obtained in [51].
- In contrast, strong approximation algorithms have a rich history for the Nearest Neighbor search problem: Using LSH, we can obtain an $(1+\varepsilon)$-approximation in time $O(n^{2-\Theta(\varepsilon)})$ [44, 9, 10, 15]. Using further techniques, the dependence on ε has been improved to $O(n^{2-\Omega(\sqrt{\varepsilon})})$ [64] and $O(n^{2-\tilde{\Omega}(\sqrt[3]{\varepsilon})})$ [5, 6]. On the hardness side, Rubinstein [62] shows that the dependence on ε cannot be improved indefinitely, by proving that for every δ, there exists an ε such that $(1+\varepsilon)$-approximate Nearest Neighbor search requires time $\Omega(n^{2-\delta})$ assuming SETH. In particular, this rules out $\operatorname{poly}(1/\varepsilon)n^{2-\delta}$-time algorithms with $\delta > 0$.

[1] Note that [23] more generally defines classes MaxSP$_{k,\ell}$ for $\ell \geq 1$. We focus on MaxSP$_k$ = MaxSP$_{k,1}$, as it was determined as computationally harder than MaxSP$_{k,\ell}$ with $\ell \geq 2$, and contains many natural problems (see below).

[2] These problems are typically studied in the setting where k is part of the input, while we consider them for a fixed constant k.

[3] In fact, even for the Nearest Codeword Problem over $\mathbb{F}_2$.

- For the dual problem of Furthest Neighbor search (i.e., diameter in the Hamming metric), [20] gives a $O(n^{2-\Theta(\varepsilon^2)})$-time algorithm, which was improved to $O(n^{2-\Theta(\varepsilon)})$ via reduction to Nearest Neighbor search in [47]. Following further improvements [42, 48], also here [5, 6] give an $O(n^{2-\tilde{\Omega}(\sqrt[3]{\varepsilon})})$-time algorithm. Analogous inapproximability to Rubinstein's result are given in [30].
- For Minimum Inner Product, the well-known Orthogonal Vectors hypothesis [65] (which is implied by SETH [66]) is precisely the assumption that already distinguishing whether the optimal value is 0 or at least 1 cannot be done in strongly subquadratic time. Interestingly, Chen and Williams [30] show a converse: a refutation of the Orthogonal Vectors hypothesis would give a subquadratic-time constant-factor approximation for Minimum Inner Product.

In the above list, we focus on the difficult case of moderate dimension $d = n^{o(1)}$ (when measuring the time complexity with respect to the input size). Lower-dimensional settings such as $d = \Theta(\log n)$, $d = \Theta(\log \log n)$ or even lower are addressed in other works [69, 30].

While the above collection of results gives a detailed understanding of isolated problems, we know little about general phenomena of (in)approximability in MaxSP_k and MinSP_k using faster-than-exhaustive-search algorithms: Are there problems for which constant-factor approximations are best possible? Is maximizing (or minimizing) Inner Product the only problem without a constant-factor approximation? Which problems can we optimize exactly?

There are precursors to our work that show fine-grained *equivalence classes* of approximation problems in P [29, 30]. However, establishing membership of a problem in these classes requires a problem-specific proof, while we are interested in *syntactically* defined classes, where class membership can be immediately read off from the definition of a problem. Our aim is to understand the approximability landscape of such classes fully. Finally, the previous works either focus on lower-dimensional settings [30][4], or target more powerful problems than we consider, such as Closest-LCS-Pair [29].

MaxSP_k as Polynomial-Time Analogue of MaxSNP. Investigating NP-hard optimization problems, Papadimitriou and Yannakakis [60] introduced the class MaxSNP, which motivates the definition of MaxSP_k as a natural polynomial-time analogue (see [23] and the full version of this paper for details). As a general class containing prominent, constant-factor approximable optimization problems, MaxSNP was introduced to give the first evidence that Max-3-SAT does not admit a PTAS, by proving that Max-3-SAT belongs to the hardest-to-approximate problems in MaxSNP.

Ideally, one would like to understand the approximability landscape in MaxSNP fully and give tight approximability results for each such problem. Major advances towards this goal have been achieved by Creignou [32] and Khanna, Sudan, Trevisan, and Williamson [53] who gave a complete classification of a large subclass of MaxSNP, namely, maximum Boolean Constraint Satisfaction Problems (MaxCSP): Each Boolean MaxCSP either is polynomial-time optimizable or it does not admit a PTAS unless P = NP, rendering a polynomial-time constant-factor approximation best possible. For minimization analogues, including Boolean MinCSPs, the situation is more diverse with several equivalence classes needed to describe the result [53].

We initiate the study of the same type of questions in the polynomial-time regime. Our aim is to achieve a detailed understanding of MaxSP_k and MinSP_k akin to the classification theorems achieved for MaxCSPs and MinCSPs [32, 53].

[4] Chen and Williams also give some results for the moderate-dimensional case; we discuss these in Section 3.

1.1 Our Results

We approach the classification of MaxSP$_k$ and MinSP$_k$ by considering the simplest, yet expressive case of a *single, binary relation* involved in the first-order formula; this route was also taken in earlier classification work for first-order properties [24]. We may thus view the relational structure as a graph, and call such a formula a *graph formula*. Note that despite its naming, this includes problems not usually viewed as graph problems, such as all examples given in the introduction, which are natural problems on sets of *vectors* in $\{0,1\}^d$.

We obtain a classification of each graph formula into one of four regimes, assuming central fine-grained hardness hypotheses whose plausibility we detail in the full version of this paper. All of our hardness results are implied by the Sparse Max-3-SAT hypothesis [7], which states that for all $\delta > 0$ there is some $c > 0$ such that Max-3-SAT on n variables and cn clauses has no $O(2^{n(1-\delta)})$-time algorithm.[5] (Actually, most of our hardness results already follow from weaker assumptions.)

▶ **Theorem 1.** *Let ψ be a MaxSP$_k$ or MinSP$_k$ graph formula. Assuming the Sparse Max-3-SAT hypothesis, ψ belongs to precisely one of the following regimes:*

R1 Efficiently optimizable:
There is some $\delta > 0$ such that ψ can be solved exactly in time $O(m^{k-\delta})$.

R2 Admits an approximation scheme, but not an efficient one:
For all $\varepsilon > 0$, there is some $\delta > 0$ such that ψ can be $(1+\varepsilon)$-approximated in time $O(m^{k-\delta})$. However, for all $\delta > 0$, there is some $\varepsilon > 0$ such that ψ cannot be $(1+\varepsilon)$-approximated in time $O(m^{k-\delta})$.

R3 Admits a constant-factor approximation, but no approximation scheme:
There are $\varepsilon, \delta > 0$ such that ψ can be $(1+\varepsilon)$-approximated in time $O(m^{k-\delta})$. However, there also exists an $\varepsilon > 0$ such that for all $\delta > 0$, ψ cannot be $(1+\varepsilon)$-approximated in time $O(m^{k-\delta})$.

R4 Arbitrary polynomial-factor approximation is best possible (maximization):
For every $\varepsilon > 0$, there is some $\delta > 0$ such that ψ can be $O(m^{\varepsilon})$-approximated in time $O(m^{k-\delta})$. However, for every $\delta > 0$, there exists some $\varepsilon > 0$ such that ψ cannot be $O(m^{\varepsilon})$-approximated in time $O(m^{k-\delta})$.

No approximation at all (minimization):
For all $\delta > 0$, we cannot decide whether the optimum value of ψ is 0 or at least 1 in time $O(m^{k-\delta})$.

Note that the characteristics of the fourth (i.e., hardest) regime differ between the maximization and the minimization case.

This theorem has immediate consequences for the approximability landscape in MaxSP$_k$ and MinSP$_k$, based on fine-grained assumptions: In particular, while any MaxSP$_k$ graph formula can be approximated within a subpolynomial factor $O(m^{\varepsilon})$, we can rule out optimal approximation ratios that *grow with m but are strictly subpolynomial* (i.e., there are *no* graph formulas whose optimal approximability within $O(m^{k-\Omega(1)})$ time is $\Theta(\log\log m)$, $\Theta(\log^2 m)$ or $2^{\Theta(\sqrt{\log m})}$). Furthermore, there are no graph formulas with an $f(1/\varepsilon)m^{k-\Omega(1)}$ approximation scheme that cannot already be optimized to exactness in time $m^{k-\Omega(1)}$.

In fact, beyond Theorem 1 we even give an almost complete characterization of each regime: Specifically, we introduce integer-valued hardness parameters $0 \le H_{\text{and}}(\psi) \le H_{\text{deg}}(\psi) \le k$ (defined in Section 3). As illustrated in Figure 1, we show how to place any graph formula ψ

[5] This hypothesis is a stronger version of the Max-3-SAT hypothesis [58].

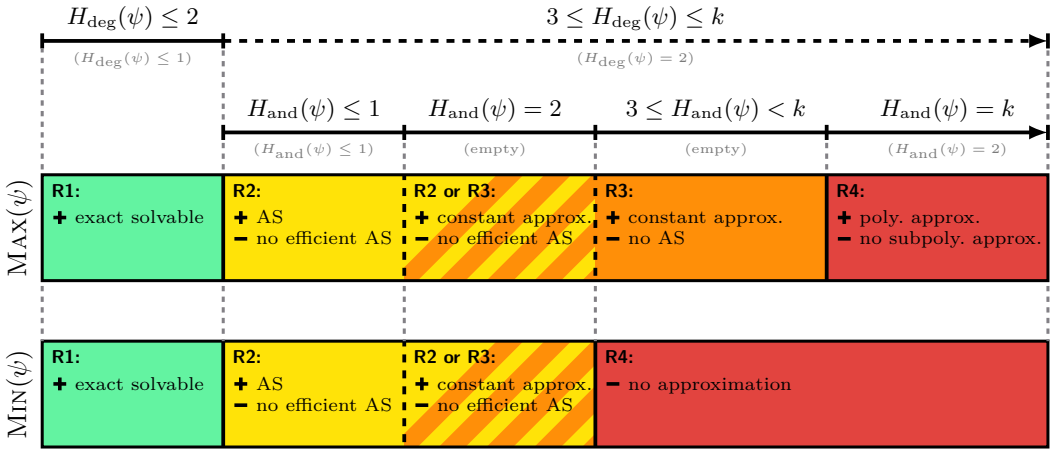


Figure 1 Visualizes our classification of first-order optimization problems $\textsc{Max}(\psi)$ and $\textsc{Min}(\psi)$ for all $k \geq 3$, in terms of the hardness parameters H_{and} and $H_{\deg}$ (as defined in Definitions 4 and 6). See Definition 10 for the precise definition of an (efficient) approximation scheme (AS). The pale labels indicate how to change the conditions to obtain the picture for $k = 2$.

Table 1 Some interesting examples classified according to Figure 1. For each problem ψ, an instance consists of k sets of n vectors $X_1, \ldots, X_k \subseteq \{0,1\}^d$. We write $\psi = \max_{x_1,\ldots,x_k} / \min_{x_1,\ldots,x_k} \phi$ and only list the inner formulas ϕ in the table.

Problem ψ	ϕ	$H_{\deg}(\psi)$	$H_{\text{and}}(\psi)$	Hardness regime
Maximum/Minimum 2-Agreement	$\#\{y : x_1[y] = x_2[y]\}$	2	1	R2
Maximum/Minimum 3-Agreement	$\#\{y : x_1[y] = x_2[y] = x_3[y]\}$	2	2	R1
Maximum k-Agreement, $k \geq 4$	$\#\{y : x_1[y] = \cdots = x_k[y]\}$	$2\lfloor \frac{k}{2} \rfloor$	$k-1$	R3
Minimum k-Agreement, $k \geq 4$	$\#\{y : x_1[y] = \cdots = x_k[y]\}$	$2\lfloor \frac{k}{2} \rfloor$	$k-1$	R4
Maximum/Minimum k-XOR	$\#\{y : x_1[y] \oplus \cdots \oplus x_k[y]\}$	k	1	R2
Maximum k-Inner Product	$\#\{y : x_1[y] \wedge \cdots \wedge x_k[y]\}$	k	k	R4
Minimum k-Inner Product	$\#\{y : x_1[y] \wedge \cdots \wedge x_k[y]\}$	k	k	R4

into its corresponding regime depending solely on $H_{\text{and}}(\psi)$ and $H_{\deg}(\psi)$ – with the *single exception* of formulas ψ with $H_{\text{and}}(\psi) = 2$ and $H_{\deg}(\psi) \geq 3$. For these formulas (e.g., Maximum Exact-3-Cover), it remains open whether they belong to Regime 2 or 3, i.e., whether or not they admit an approximation scheme (see Open Problem 1).

In the full version we give natural problems for each regime (showing in particular that each regime is indeed non-empty). A particular highlight is that we can prove existence of *constant-factor approximable formulas that do not admit an approximation scheme*, such as Maximum-4-Agreement (see Section 3, Theorem 11 for a technical discussion of the lower bound). In the full version of this paper, we give the details on how to calculate the hardness parameters $H_{\text{and}}(\psi)$ and $H_{\deg}(\psi)$ for each example in Table 1.

While the approximability of problems in the third and fourth regime seem rather unsatisfactory, we also consider the setting of additive approximation, and show that for every MaxSP_k and MinSP_k graph formula, there is an additive approximation scheme; however, assuming the Sparse MAX-3-SAT hypothesis it cannot be an efficient one unless we can optimize the problem exactly.

▶ **Theorem 2** (Additive Approximation). *For every ψ, we give an additive approximation scheme, i.e., for every $\varepsilon > 0$, there is a $\delta > 0$ such that we compute the optimum value up to an additive $\varepsilon|Y|$ error in time $O(m^{k-\delta})$.*

If ψ does not belong to Regime 1, we show that unless the Sparse Max-3-SAT hypothesis fails, for every $\delta > 0$, there is some $\varepsilon > 0$ such that we cannot compute the optimum value up to an additive $\varepsilon|Y|$ error in time $O(m^{k-\delta})$.

Finally, we remark that our classification identifies general trends in MaxSP_k and MinSP_k, including that exact optimizability is described by a simple algebraic criterion, and that constant-factor approximation for minimization problems is equivalent to testing whether the optimum is 0. We address these trends in Section 3.

On Plausibility of the Hypotheses. As tight unconditional lower bounds for polynomial-time problems are barely existent, we give *conditional* lower bounds, based on established assumptions in fine-grained complexity theory, such as SETH (see [65] for an excellent survey). The essentially only exception is the only recently introduced *Sparse* Max-3-SAT hypothesis [7]; we use this hypothesis only for giving evidence against approximation schemes and for ruling out certain additive approximation schemes. While it is possible that this hypothesis could ultimately be refuted, our classification describes the frontier of the current state of the art. At the very least, our conditional lower bound for approximation schemes reveals a rather surprising connection: To obtain an *approximation scheme* for *polynomial-time problems* such as Maximum 4-Agreement, we need to give an *exponential*-time improvement for *exact* solutions for Sparse Max-3-SAT!

1.2 Further Related Work

Parameterized Inapproximability. The problem settings we consider are related to a recent and strong line of research on parameterized inapproximability, including [27, 51, 31, 59, 57] (see [36] for a recent survey). In these contexts, one seeks to determine optimal approximation guarantees within some running time of the form $f(k)n^{g(k)}$ (such as FPT running time $f(k)\,\text{poly}(n)$ or running time $n^{o(k)}$) for some parameter k (such as the solution size). Unfortunately, many of these results do not immediately establish hardness for specific values of k. An important exception is work by Karthik, Laekhanukit, and Manurangsi [51], which among other results establishes inapproximability of k-Dominating Set and the Maximum k-Inner Product within running time $O(n^{k-\varepsilon})$, assuming SETH. We give a detailed comparison of our setting to notions used in Karthik et al.'s work in the full version of this work.

Fine-Grained Complexity of First-Order Properties. Studying the fine-grained complexity of polynomial-time problem classes defined by first-order properties has been initiated in [68, 40], with different settings considered in [39, 38]. In particular, for model-checking first-order properties, [40] provides a completeness result (a fine-grained analogue of the Cook-Levin Theorem) and [24] provides a classification theorem (a fine-grained analogue of Schaefer's Theorem, see also Section 3.1).

For *optimization* in P, [23] provides completeness theorems for MaxSP$_k$ (a fine-grained analogue of MaxSNP-completeness of Max-3-SAT [60]). In contrast, the present paper gives a classification theorem (building a fine-grained analogue of the approximability classifications of optimization variants of CSPs [32, 53]).

2 Preliminaries

For an integer $k \geq 0$, we set $[k] = \{1, \ldots, k\}$. For a set A and integer $k \geq 0$ we denote by $\binom{A}{k}$ the set of all size-k subsets of A. Let $\phi(z_1, \ldots, z_k)$ a Boolean function, and let $S \subseteq [k]$. Any function obtained by instantiating all variables z_i ($i \notin S$) in ϕ by constant values is called an *S-restriction* of ϕ. Finally, we write $\tilde{O}(\cdot)$ to hide poly-logarithmic factors and denote by $\omega < 2.373$ the exponent of square matrix multiplication [8].

2.1 First-Order Model-Checking

A *relational structure* consists of n objects and relations $R_1, \ldots, R_\ell$ (of arbitrary arities) between these objects. A *first-order formula* is a quantified formula of the form

$$\psi = (Q_1 x_1) \ldots (Q_{k+1} x_{k+1})\, \phi(x_1, \ldots, x_{k+1}),$$

where ϕ is a Boolean formula over the predicates $R(x_{i_1}, \ldots, x_{i_\ell})$ and each Q_i is either a universal or existential quantifier. Given a $(k+1)$-partite structure on objects $X_1 \cup \cdots \cup X_{k+1}$, the *model-checking problem* (or *query evaluation problem*) is to check whether ψ holds on the given structure, that is, for x_i ranging over X_i and by instantiating the predicates $R(x_{i_1}, \ldots, x_{i_\ell})$ in ϕ according to the structure, ψ is valid.

Following previous work in this line of research [40, 24], we usually assume that the input is represented sparsely – that is, we assume that the relational structure is written down as an exhaustive enumeration of all records in all relations; let m denote the total number of such entries[6]. The convention is reasonable as this data format is common in the context of database theory and also for the representation of graphs (where it is called the *adjacency list* representation), see Section 4 for a further discussion.

A first-order formula ψ is called a *graph formula* if it is defined over a single binary predicate $E(x_i, x_j)$. Many natural problems fall into this category; see [24] for a detailed discussion on the subject.

2.2 MaxSP$_k$ and MinSP$_k$

In analogy to first-order properties with quantifier structure $\exists^k \forall$ (with maximization instead of $\exists$ and counting instead of $\forall$) and following the definition in [23], we now introduce the class of optimization problems. Let MaxSP$_k$ be the class containing all formulas of the form

[6] By ignoring objects not occurring in any relation, we may always assume that $n \leq O(m)$.

$$\psi = \max_{x_1,\dots,x_k} \#_{y}\, \phi(x_1,\dots,x_k,y), \tag{1}$$

where, as before, ϕ is a Boolean formula over some predicates $R(x_{i_1},\dots,x_{i_\ell})$. We similarly define MinSP$_k$ with "min" in place of "max". Occasionally, we write OptSP$_k$ to refer to both of these classes simultaneously, and we write "opt" as a placeholder for either "max" or "min". In analogy to the model-checking problem for first-order properties, we associate to each formula $\psi \in$ OptSP$_k$ an algorithmic problem:

▶ **Definition 3** (Max(ψ) and Min(ψ)). *Let $\psi \in$ MaxSP$_k$ be as in (1). Given a $(k+1)$-partite structure on objects $X_1 \cup \dots \cup X_k \cup Y$, the Max($\psi$) problem is to compute*

$$\mathrm{OPT} = \max_{x_1\in X_1,\dots,x_k\in X_k} \#_{y\in Y}\, \phi(x_1,\dots,x_k,y).$$

We similarly define Min(ψ) for $\psi \in$ MinSP$_k$. Occasionally, for $\psi \in$ OptSP$_k$, we write Opt(ψ) to refer to both problems simultaneously.

For convenience, we introduce some further notation: For objects $x_1 \in X_1,\dots,x_k \in X_k$, we denote by $\mathrm{Val}(x_1,\dots,x_k) = \#_{y\in Y}\,\phi(x_1,\dots,x_k,y)$ the *value* of $(x_1,\dots,x_k)$.

The problem Opt(ψ) can be solved in time $O(m^k)$ for all OptSP$_k$ formulas ψ, by a straightforward extension of the model-checking baseline algorithm; see [23] for details. As this is clearly optimal for $k = 1$, we will often implicitly assume that $k \geq 2$ in the following.

For a clean analogy between model-checking and the optimization classes MaxSP$_k$ and MinSP$_k$, we will from now view model-checking as "testing for zero". More precisely, the model-checking problem of $\exists x_1 \dots \exists x_k \forall y \neg\phi(x_1,\dots,x_k,y)$ is equivalent to testing whether Min(ψ) has optimal solution OPT $= 0$, where $\psi = \min_{x_1,\dots,x_k} \#_y\, \phi(x_1,\dots,x_k,y)$. We refer to the latter problem as Zero(ψ).

Definition 3 introduces Max(ψ) and Min(ψ) as *exact* optimization problems (i.e., OPT is required to be computed exactly). We say that an algorithm computes a *(multiplicative) c-approximation* for Max(ψ) if it computes any value in the interval $[c^{-1}\cdot \mathrm{OPT}, \mathrm{OPT}]$. Similarly, a (multiplicative) c-approximation for Min(ψ) computes a value in $[\mathrm{OPT}, c\cdot\mathrm{OPT}]$.

3 Technical Overview

This section serves the purpose of stating our results formally, to provide the main proof ideas and techniques, and to convey some intuition whenever possible. Due to space constraints, we omit precise definitions of the fine-grained hypotheses here and instead refer to the full version of this paper.

We first introduce our hardness parameters: the and-hardness $H_{\mathrm{and}}(\psi)$ borrowed from previous work [24] (reviewed below), as well as a novel algebraic parameter that we call degree-hardness $H_{\deg}(\psi)$. In the subsequent sections, we give an overview over our various algorithmic and hardness results based on the values $H_{\deg}(\psi), H_{\mathrm{and}}(\psi)$ (see Figure 1), with the formal proof of Theorem 1 given at the end of this section.

3.1 Bringmann et al.'s Model-Checking Dichotomy

Bringmann, Fischer and Künnemann [24] established a fine-grained classification of all $\exists^k\forall$-quantified graph properties into computationally easy and hard model-checking problems. As our work extends that classification (and also since our results are of a similar flavor), we briefly summarize their results. The hardness parameter presented in Definition 4 forms the basis for the dichotomy.

Here, and for the remainder of this section, we write ψ_0 to denote the Boolean function obtained from ψ in the following way: Let $\psi = \mathrm{opt}_{x_1,\dots,x_k} \#_y \phi(x_1,\dots,x_k,y)$ be any graph formula, where ϕ is a propositional formula over the atoms $E(x_i,x_j)$ $(i,j \in [k])$ and $E(x_i,y)$ $(i \in [k])$. Consider the Boolean function obtained from ϕ by replacing every atom $E(x_i,x_j)$ $(i,j \in [k])$ by *false*. What remains is a Boolean function over the k atoms $E(x_i,y)$ $(i \in [k])$ and we denote this formula by ψ_0. For example, if $\psi = \max_{x_1,x_2} \#_y(\neg E(x_1,x_2) \wedge E(x_1,y) \wedge E(x_2,y))$ then we define $\psi_0 : \{0,1\}^2 \to \{0,1\}$ by $\psi_0(a_1,a_2) = a_1 \wedge a_2$.

As apparent in Definition 4, the core difficulty of a formula ψ is captured by ψ_0 and thus not affected by the predicates $E(x_i,x_j)$. In the full version we elaborate on this phenomenon.

▶ **Definition 4** (And-Hardness). *Let ϕ be a Boolean function on k inputs. The* and-hardness $H_{\mathrm{and}}(\phi)$ *of ϕ is the largest integer $0 \le h \le k$ such that, for any index set $S \in \binom{[k]}{h}$, there exists some S-restriction of ϕ with exactly one satisfying assignment. (Set $H_{\mathrm{and}}(\phi) = 0$ for constant-valued ϕ.) For an OptSP_k graph formula ψ, we define $H_{\mathrm{and}}(\psi) = H_{\mathrm{and}}(\psi_0)$.*

This hardness parameter essentially specifies the computational hardness of the model-checking problems $\textsc{Zero}(\phi)$; here, $H_{\mathrm{and}}(\psi) \le 2$ is the critical threshold:

▶ **Theorem 5** (Model-Checking, [24]). *Let ψ be a MinSP_k graph formula.*

- *If $H_{\mathrm{and}}(\psi) \le 2$ and $H_{\mathrm{and}}(\psi) < k$, then $\textsc{Zero}(\psi)$ can be solved in time $O(m^{k-\delta})$ for some $\delta > 0$.*
- *If $3 \le H_{\mathrm{and}}(\psi)$ or $H_{\mathrm{and}}(\psi) = k$, then $\textsc{Zero}(\psi)$ cannot be solved in time $O(m^{k-\delta})$ for any $\delta > 0$ unless the Max-3-SAT hypothesis fails.*

3.2 Exact Optimization

We are now ready to detail our results. Our main contribution is a dichotomy for the exact solvability and approximability of $\textsc{Max}(\psi)$ and $\textsc{Min}(\psi)$ for all MaxSP_k and MinSP_k graph formulas ψ. For the exact case, the decisive criterion for the hardness of some formula ψ can be read off the polynomial *extension* of the function ψ_0. Specifically, for any Boolean function ϕ there exists a (unique) multilinear polynomial with real coefficients that computes ϕ on binary inputs. By abuse of notation, we refer to that polynomial by writing ϕ as well. The *degree* $\deg(\phi)$ of ϕ is the degree of its polynomial extension.

As an example, consider the Exact-3-Cover property: $\phi(z_1,z_2,z_3)$ is true if and only if exactly one of its inputs z_1, z_2 or z_3 is true. Then its unique multilinear polynomial extension is

$$\phi(z_1,z_2,z_3) = 3z_1z_2z_3 - 2(z_1z_2 + z_2z_3 + z_3z_1) + (z_1 + z_2 + z_3),$$

and therefore $\deg(\phi) = 3$.

▶ **Definition 6** (Degree Hardness). *Let ϕ be a Boolean function on k inputs. The* degree hardness $H_{\deg}(\phi)$ *of ϕ is the largest integer $0 \le h \le k$ such that, for any index set $S \in \binom{[k]}{h}$, there exists some S-restriction of ϕ of degree h. For an OptSP_k graph formula ψ, we define $H_{\deg}(\psi) = H_{\deg}(\psi_0)$.*

It always holds that $H_{\mathrm{and}}(\psi) \le H_{\deg}(\psi)$, but in general these parameters behave very differently. With $H_{\deg}$ in place of H_{and}, we are able to recover the same classification as Theorem 5 for both exact maximization and minimization:

▶ **Theorem 7** (Exact Optimization). *Let ψ be an OptSP_k graph formula.*

- *If $H_{\deg}(\psi) \leq 2$ and $H_{\deg}(\psi) < k$, then $\text{Opt}(\psi)$ can be solved in time $O(m^{k-\delta})$ for some $\delta > 0$.*
- *If $3 \leq H_{\deg}(\psi)$ or $H_{\deg}(\psi) = k$, then $\text{Opt}(\psi)$ cannot be solved in time $O(m^{k-\delta})$ for any $\delta > 0$ unless the Max-3-SAT hypothesis fails.*

The algorithmic part is proven along the same lines as [24]: We first brute-force over all but $k = 3$ quantifiers, and solve the remaining problem by a reduction to maximum-weight triangle detection with small edge weights. Intuitively, since the degree of the resulting 3-variable problem is at most 2, we can express the objective as a sum of three parts depending on two variables each (this approach is also used in a similar context in [67, Chapter 6.5]). This allows us to label the edges of a triangle instance with corresponding parts, which can be shown to yield average edge weight $O(1)$.

The conditional lower bound is more interesting. Our reduction is inspired by a standard argument proving quadratic-time hardness of the Maximum Inner Product problem (MaxIP). That lower bound is based on the OV hypothesis, so it consists of a reduction from an OV instance $X_1, X_2 \subseteq \{0,1\}^d$ to an instance of MaxIP. The idea is to use a *gadget* that maps every entry in $x_i \in X_i$ to a constant number of new entries;[7] let $x_i' \in \{0,1\}^{O(d)}$ denote the vector after applying the gadget coordinate-wise. The crucial property is that $\langle x_1', x_2' \rangle = d - \langle x_1, x_2 \rangle$, and thus a pair of orthogonal vectors x_1, x_2 corresponds to a pair of vectors x_1', x_2' of maximum inner product.

To mimic the reduction for all problems which are hard in the sense of Theorem 7, we settle for the weaker but sufficient property that the value of (x_1', x_2') equals $\beta_1 d - \beta_2 \langle x_1, x_2 \rangle$, for some positive integers β_1, β_2. It follows from our algebraic characterization of hard functions that a gadget with such guarantees always exists. Ultimately, our hardness proof makes use of that gadget in a similar way as for MaxIP.

The hardness part of Theorem 7 can in fact be stated in a more fine-grained way: If some problem $\text{Opt}(\psi)$ has degree hardness $h = H_{\deg}(\psi) \geq 3$, then the hardness proof can be conditioned on the weaker h-Uniform HyperClique assumption. That connection can be complemented by a partial converse, thus revealing a certain *equivalence* between exact optimization and hyperclique detection. An analogous equivalence for model-checking graph formulas could not be proved and was left as an open problem in [24].

▶ **Theorem 8** (Equivalence of $\text{Opt}(\psi)$ and HyperClique). *Let ψ be an OptSP_k graph formula of degree hardness $h = H_{\deg}(\psi) \geq 2$.*

- *If $\text{Opt}(\psi)$ can be solved in time $O(m^{k-\delta})$ for some $\delta > 0$, then, for some (large) $k' = k'(k, h, \delta)$, h-Uniform k'-HyperClique can be solved in time $O(n^{k'-\delta'})$ for some $\delta' > 0$.*
- *If h-Uniform $(h+1)$-HyperClique can be solved in time $O(n^{h+1-\delta})$ for some $\delta > 0$, then $\text{Opt}(\psi)$ can be solved in time $O(m^{k-\delta'})$ for some $\delta' > 0$.*

3.3 Approximation

Since Theorem 7 gives a complete classification for the exact solvability of $\text{Opt}(\psi)$, the next natural question is to study the approximability of properties which are hard to compute exactly. Unlike the exact case, we use different techniques and tools to give our classification for maximization and minimization problems.

[7] Every entry a in x_1 is replaced by three new coordinates $(a, 1-a, 1-a)$ and every entry b in x_2 is replaced by $(1-b, b, 1-b)$. The contribution to the inner product of the new vectors is equal to $a(1-b) + (1-a)b + (1-a)(1-b) = 1 - ab$.

3.3.1 Maximization

We obtain a simple classification of all constant-factor approximable maximization problems, conditioned on the Strong Exponential Time Hypothesis (SETH).

▶ **Theorem 9** (Constant Approximation – Maximization). *Let ψ be a MaxSP$_k$ graph formula.*
- *If $H_{\text{and}}(\psi) < k$ (or equivalently, if ψ_0 does not have exactly one satisfying assignment), then there exists a constant-factor approximation for Max(ψ) in time $O(m^{k-\delta})$ for some $\delta > 0$.*
- *Otherwise, if $H_{\text{and}}(\psi) = k$ (or equivalently, ψ_0 has exactly one satisfying assignment), then there exists no constant-factor approximation for Max(ψ) in time $O(m^{k-\delta})$ for any $\delta > 0$, unless SETH fails.*

We give a high-level explanation of our proof by focusing on two representative problems: On the one hand, strong conditional hardness results have been shown for the MaxIP problem [4, 28]. When ψ_0 has a single satisfying assignment, Max(ψ) is equivalent (up to complementation) to MaxIP, so we adapt the hardness results for our setting.

On the other hand, consider the Furthest Neighbor problem: Given two sets of bit-vectors $X_1, X_2 \subseteq \{0,1\}^d$, compute the maximum Hamming distance between vectors $x_1 \in X_1$ and $x_2 \in X_2$. There exists a simple linear-time 3-approximation for this problem: Fix some $x_1 \in X_1$ and compute its furthest neighbor $x_2 \in X_2$. Then compute the furthest neighbor $x_1' \in X_1$ of x_2 and return the distance between x_1' and x_2 as the answer. By applying the triangle inequality twice, it is easy to see that this indeed yields a 3-approximation. That argument generalizes for approximating Max(ϕ) whenever ψ_0 satisfies the following property: If α is a satisfying assignment of ψ_0, then the component-wise negation of α is also satisfying. Finally, if ψ_0 has at least two satisfying assignments, then Max(ψ) can be reduced to this special case via a reduction which worsens the approximation ratio by at most a constant factor. The essential insight for that last step is that we can always "cover" all satisfying assignments by only two satisfying assignments, as there always exists one satisfying assignment which contributes a constant fraction (depending on k) to the optimal value.

We give a finer-grained view of the classification in Theorem 9 in two ways. First, we want to isolate properties which admit *arbitrarily good* constant-factor approximations. We make that notion precise in the following definition:

▶ **Definition 10** (Approximation Scheme). *Let ψ be an OptSP$_k$ formula. We say that Opt(ψ) admits an* approximation scheme *if for any $\varepsilon > 0$ there exists some $\delta > 0$ and an algorithm computing a $(1+\varepsilon)$-approximation of Opt(ψ) in time $O(m^{k-\delta})$.*[8]

In the following theorem, we identify some formulas which admit such an approximation scheme, and some formulas for which this is unlikely:

▶ **Theorem 11** (Approximation Scheme – Maximization). *Let ψ be a MaxSP$_k$ graph formula.*
- *If $H_{\text{and}}(\psi) \leq 1$, then there exists a randomized approximation scheme for Max(ψ).*
- *If $3 \leq H_{\text{and}}(\psi)$ or $H_{\text{and}}(\psi) = k$, then there exists no approximation scheme for Max(ψ) unless the Sparse Max-3-SAT hypothesis fails.*

Unfortunately, we were not able to close the gap in Theorem 11, and it remains open whether problems with $H_{\text{and}}(\phi) = 2$ admit approximation schemes. In Section 4, we give a specific example falling into this category.

[8] We stress that in our work, ε is always a constant and cannot depend on m.

For the first item, we give approximation schemes for any $\textsc{Max}(\psi)$ with $H_{\text{and}}(\psi) = 1$ via a reduction to Furthest Neighbor, which, as mentioned in the introduction, admits an approximation scheme.

The lower bound is more interesting: By Theorem 9 we know that if ψ has and-hardness $H_{\text{and}}(\psi) < k$, then it admits a constant-factor approximation. Nevertheless, the lower bound in Theorem 9 only addresses formulas of and-hardness $H_{\text{and}}(\psi) = k$. In particular, Theorem 11 identifies a class of problems for which some (fixed) constant-factor approximation is *best-possible in terms of approximation.*

At a technical level, we make use of interesting machinery to obtain the lower bound. The starting point is the *distributed PCP* framework, which was introduced in [4] and further strengthened in [62, 28] to give hardness of approximation for Maximum k-Inner Product (k-MaxIP). In this case, we cannot use this tool directly, since it crucially relies on the fact that the *target* problem $\textsc{Max}(\psi)$ has full hardness $H_{\text{and}}(\psi) = k$.[9] Instead, we first show Max-3-SAT-hardness of the following intermediate problem:

▶ **Problem 12** ($(k,3)$-OV). *Given sets of n vectors $X_1, \ldots, X_k \subseteq \{0,1\}^d$, where each coordinate $y \in [d]$ is associated to three* active *indices $a, b, c \in [k]$, detect if there are vectors $x_1 \in X_1, \ldots, x_k \in X_k$ such that for all $y \in [d]$, it holds that $x_a[y] \cdot x_b[y] \cdot x_c[y] = 0$ where a, b, c are the active indices at y.*

We then provide a gap introducing reduction from $(k,3)$-OV to $\textsc{Max}(\psi)$, in the same spirit as the PCP reduction gives such a reduction from k-OV to k-MaxIP. This involves several technical steps as outlined in Figure 2. At a high level, we decompose a $(k,3)$-OV instance into a combination of multiple 3-OV instances and use the PCP reduction as a black box on each of these. After combining the outputs of the reduction, we obtain instances of $\textsc{Max}(\psi)$ with the desired gap. The issue with this approach is that the PCP reduction blows up the dimension of the input vectors exponentially,[10] which makes the reduction inapplicable to our case if we start from a moderate-dimensional $(k,3)$-OV instance. To show that $(k,3)$-OV does not even have a $O(m^{k-\delta})$-time algorithm when the dimension is $d = O(\log n)$, we use the stronger *Sparse* Max-3-SAT hypothesis.[11] See the full version of this paper for further discussion of this hypothesis.

The second way in which we get a closer look at Theorem 9 is by inspecting the hardest regime, i.e., when $H_{\text{and}}(\psi) = k$:

▶ **Theorem 13** (Polynomial-Factor Approximation). *Let ψ be a $\textsc{MaxSP}_k$ graph formula of full and-hardness $H_{\text{and}}(\psi) = k$.*

- *For every $\varepsilon > 0$, there exists some $\delta > 0$ such that an m^{ε}-approximation for $\textsc{Max}(\psi)$ can be computed in time $O(m^{k-\delta})$.*
- *For every $\delta > 0$, there exists an $\varepsilon > 0$ such that there exists no m^{ε}-approximation for $\textsc{Max}(\psi)$ in time $O(m^{k-\delta})$ unless SETH fails.*

The lower bound is obtained by applying the subsequent improvements on the distributed PCP framework by [62, 28, 51], which improve the parameters of the reduction via algebraic geometry codes and expander graphs. For the upper bound, we give a simple algorithm which exploits the sparsity of the instances.

[9] More precisely, the reduction exploits the fact that ψ_0 has a unique satisfying assignment to encode the communication protocol used in the reduction.

[10] An instance of k-OV on dimension $d = c \log n$ is reduced to multiple instances of k-MaxIP on dimension $\exp(c) \log n$.

[11] Morally, just as SETH implies the hardness of low-dimensional OV, the *Sparse* Max-3-SAT Hypothesis implies the hardness of *low-dimensional* $(k,3)$*-OV*.

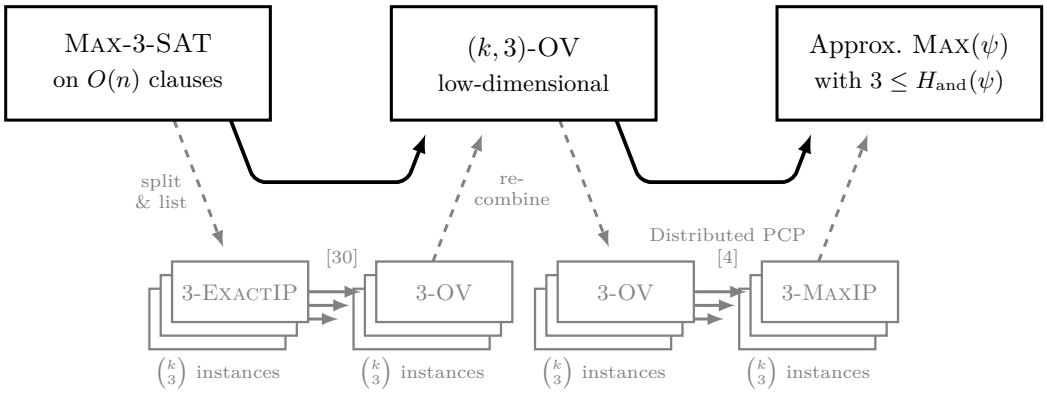


Figure 2 The chain of reductions from Sparse Max-3-SAT to VMax(ψ) involving two main steps. Both proofs involve several intermediate steps as illustrated in the gray parts.

3.3.2 Minimization

There is an easier criterion for the hardness of approximating minimization problems. Namely, observe that giving a multiplicative approximation to a minimization problem Min(ψ) is at least as hard as testing if the optimal value is zero (recall that we refer to this problem as Zero(ψ)). More precisely, suppose that we are given an instance with OPT $= 0$. Then any multiplicative approximation must return an optimal solution.

It turns out that this is the only source of hardness for Min(ψ) problems. We show a fine-grained equivalence of deciding Zero(ψ) and approximating Min(ψ) within a constant factor:

▶ **Theorem 14** (Constant Approximation is Equivalent to Testing Zero). *Let ψ be a MinSP$_k$ graph formula. Via a randomized reduction, there exists a constant-factor approximation algorithm for Min(ψ) in time $O(m^{k-\delta})$ for some $\delta > 0$ if and only if Zero(ψ) can be solved in time $O(m^{k-\delta'})$ for some $\delta' > 0$.*

To reduce approximating Min(ψ) to Zero(ψ), we make use of *locality-sensitive hashing* (LSH). This technique was for instance used to solve the Approximate Nearest Neighbors problem in Hamming spaces [44], and was recently adapted by Chen and Williams to show that Minimum Inner Product can be reduced to OV [30]. The latter result constitutes a singular known case for the general trend in MinSP$_k$ revealed by Theorem 14. In comparison, our reduction is simpler and more general, but gives weaker guarantees on the constant factor.

We specifically use LSH for a reduction from approximating Min(ψ) to the intermediate problem of *listing* solutions to Zero(ψ) – the better the listing algorithm performs, the better the approximation guarantee. For instance, an algorithm listing L solutions in time $O(m^{k-\delta} \cdot L^{\delta/k})$ for some $\delta > 0$ results in a constant-factor approximation, while a listing algorithm in time $\tilde{O}(m^{k-\delta} + L)$ leads to an approximation scheme. To finish the proof of Theorem 14, we show that any Zero(ψ) problem with an $O(m^{k-\delta})$-time decider, for some $\delta > 0$, also admits a listing algorithm in time $O(m^{k-\delta} \cdot L^{\delta/k})$.

Since the hardness of Zero(ψ) is completely classified [24], we obtain the following dichotomy as a consequence of Theorem 14:

▶ **Corollary 15** (Constant Approximation – Minimization). *Let ψ be a $\textsc{MinSP}_k$ graph formula.*

- *If $H_{\text{and}}(\psi) \leq 2$ and $H_{\text{and}}(\psi) < k$, then there exists a randomized constant-factor approximation algorithm for $\textsc{Min}(\psi)$ in time $O(m^{k-\delta})$ for some $\delta > 0$.*
- *If $3 \leq H_{\text{and}}(\psi)$ or $H_{\text{and}}(\psi) = k$, then computing any approximation for $\textsc{Min}(\psi)$ in time $O(m^{k-\delta})$ for any $\delta > 0$ is not possible unless the Max-3-SAT hypothesis fails.*

Similar to the maximization case, let us next consider approximation schemes for minimization problems. We can reuse the general framework outlined in the previous paragraphs to obtain an approximation scheme by giving an output-linear listing algorithm in time $\tilde{O}(m^{k-\delta} + L)$, for all formulas with and-hardness at most $H_{\text{and}}(\psi) \leq 1$.

▶ **Theorem 16** (Approximation Scheme – Minimization). *Let ψ be a $\textsc{MinSP}_k$ graph formula. If $H_{\text{and}}(\psi) \leq 1$, then there exists a randomized approximation scheme for $\textsc{Min}(\psi)$.*

3.4 Efficient (Multiplicative) Approximation Schemes

Theorems 11 and 16 show that if $H_{\text{and}}(\psi) \leq 1$, then we can give approximation schemes for $\textsc{Opt}(\psi)$. In the full version, we complement this result by ruling out the existence of *efficient* approximation schemes for most regimes. We say that $\textsc{Opt}(\psi)$ admits an efficient (multiplicative) approximation scheme if there is some fixed constant $\delta > 0$ such that for any $\varepsilon > 0$, a multiplicative $(1+\varepsilon)$-approximation for $\textsc{Opt}(\psi)$ can be computed in time $O(m^{k-\delta})$.

▶ **Theorem 17.** *Let ψ be an $\textsc{OptSP}_k$ graph formula. If $3 \leq H_{\text{deg}}(\psi)$ or $H_{\text{deg}}(\psi) = k$, then there exists no efficient approximation scheme for $\textsc{Opt}(\psi)$ assuming the Sparse Max-3-SAT Hypothesis.*

3.5 Proving the Main Theorem

We can now put things together to prove Theorem 1.

Proof of Theorem 1. We only sketch the proof for a maximization problem ψ with $k \geq 3$; the other cases are similar. We show how to classify ψ into one of the four stated regimes with a case distinction based on the hardness parameters $H_{\text{and}}(\psi)$ and $H_{\text{deg}}(\psi)$. This case distinction can also be read off Figure 1.

- If $H_{\text{deg}}(\psi) \leq 2$: By Theorem 7, ψ is efficiently optimizable, so it lies in R1.
- Otherwise, it holds that $3 \leq H_{\text{deg}}(\psi) \leq k$. In this case, we make a further distinction:
  - $H_{\text{and}}(\psi) \leq 1$: By Theorem 11, ψ admits an approximation scheme but by Theorem 17 not an efficient one, so it lies in R2.
  - $H_{\text{and}}(\psi) = 2$: By Theorem 9, ψ admits an efficient constant-factor approximation but by Theorem 17, it does not admit an efficient approximation scheme. Thus, depending on whether ψ admits an approximation scheme or not, it lies in R2 or R3. (As mentioned below Theorem 1, this is the single case where we cannot place the formula in its precise regime, see also Open Problem 1.)
  - $3 \leq H_{\text{and}}(\psi) < k$: By Theorem 9, ψ admits an efficient constant factor approximation but by Theorem 11 it has no approximation scheme, so it lies in R3.
  - $H_{\text{and}}(\psi) = k$: By Theorem 13, ψ admits an efficient polynomial-factor approximation, and this is best possible, so it lies in R4. ◀

4 Discussion and Open Problems

Our investigation reveals all possible approximability types (in better-than-exhaustive-search time) for general classes of polynomial-time optimization problems, namely graph formulas in MaxSP_k and MinSP_k. Our results, which give an almost complete characterization, open up the following questions:

- Can we extend our classification beyond graph formulas, i.e., when we allow more binary relations, or even higher-arity relations? Such settings include, e.g., generalizations of Max k-XOR from $\mathbb{F}_2$ to $\mathbb{F}_q$, or variants of the densest subgraph problem on hypergraphs.
- In our setting, each variable x_i ranges over a separate set X_i, also known as a *multichromatic* setting. We leave it open to transfer our results to the *monochromatic* setting (see e.g. [52]).
- While we consider running times expressed in the input size (as usual in database contexts), it would also be natural to consider parameterization in the number n of objects in the relational structure, see [68].

Besides these extensions, we ask whether one can close the remaining gap in our classification: Do formulas ψ with $H_{\text{and}}(\psi) = 2$ and $H_{\text{deg}}(\psi) \geq 3$ admit an approximation scheme? As a specific challenge, we give the following open problem:

▶ **Open Problem 1.** *Is there an approximation scheme for Maximum Exact-3-Cover (or its minimization variant)? Specifically, can we prove or rule out that for every $\varepsilon > 0$, there is some $\delta > 0$ such that we can $(1+\varepsilon)$-approximate Maximum Exact-3-Cover in time $O(m^{3-\delta})$?*

It appears likely that showing existence of an approximation scheme for Maximum Exact-3-Cover would lead to a full characterization of MaxSP_k.

Finally, while we focused on the qualitative question whether or not exhaustive search can be beaten, a follow-up question is to determine precise *approximability-time tradeoffs*. In this vein, consider the well-studied Maximum k-Cover problem: A simple linear-time greedy approach is known to establish a $(1-1/e)^{-1}$-approximation [46]. Subsequent lower bounds show that this is conditionally best possible in polynomial-time [35] and even $f(k)m^{o(k)}$-time (under Gap-ETH) [31, 59]. On the other hand, for every fixed k, we show (1) existence of an approximation scheme, but (2) rule out an efficient one assuming SETH, i.e., for every $\delta > 0$, there is some $\varepsilon > 0$ such that an $(1+\varepsilon)$-approximation requires time $\Omega(m^{k-\delta})$.

▶ **Open Problem 2.** *Let $k \geq 2$. Can we determine, for every $1 \leq \gamma \leq (1-1/e)^{-1}$, the optimal exponent α of the fastest γ-approximation for Maximum k-Cover running in time $O(m^{\alpha \pm o(1)})$, assuming plausible fine-grained hardness assumptions?*

Note that the extreme cases for $\gamma = 1$ and $\gamma = (1-1/e)^{-1}$ are already settled and that [59] shows that for all immediate cases, α must have a linear dependence on k, assuming Gap-ETH.

References

1 Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *Proceedings of the 8th Conference on Innovations in Theoretical Computer Science*, volume 67 of *ITCS '17*, pages 11:1–11:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '15, pages 59–78. IEEE Computer Society, 2015.

3 Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *Proceedings of the 9th Conference on Innovations in Theoretical Computer Science*, volume 94 of *ITCS '18*, pages 35:1–35:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

4 Amir Abboud, Aviad Rubinstein, and Ryan Williams. Distributed PCP theorems for hardness of approximation in P. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '17, pages 25–36. IEEE Computer Society, 2017.

5 Josh Alman, Timothy Chan, and Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '16, pages 467–476. IEEE Computer Society, 2016.

6 Josh Alman, Timothy Chan, and Ryan Williams. Faster deterministic and Las Vegas algorithms for offline approximate nearest neighbors in high dimensions. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 637–649. SIAM, 2020.

7 Josh Alman and Virginia Vassilevska Williams. OV graphs are (probably) hard instances. In *Proceedings of the 11th Conference on Innovations in Theoretical Computer Science*, volume 151 of *ITCS '20*, pages 83:1–83:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

8 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 522–539. SIAM, 2021.

9 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the 47th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '06, pages 459–468. IEEE Computer Society, 2006.

10 Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya P. Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1018–1028. SIAM, 2014.

11 Alexandr Andoni, Piotr Indyk, and Ilya P. Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians*, ICM '18, pages 3287–3318, 2018.

12 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 377–386. IEEE Computer Society, 2010.

13 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '20, pages 990–1001. IEEE, 2020.

14 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pages 199–204. ACM, 2009.

15 Alexandr Andoni and Ilya P. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, STOC '15, pages 793–801. ACM, 2015.

16 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing*, STOC '18, pages 267–280. ACM, 2018.

17 Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '04, pages 550–559. IEEE Computer Society, 2004.

18 Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, STOC '03, pages 316–324. ACM, 2003.

19 Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '06, pages 792–801. SIAM, 2006.

20 Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, STOC '99, pages 435–444. ACM, 1999.

21 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing*, STOC '20, pages 685–698. ACM, 2020.

22 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '15, pages 661–670. IEEE Computer Society, 2014.

23 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. Fine-grained completeness for optimization in P. In *APPROX-RANDOM*, volume 207 of *LIPIcs*, pages 9:1–9:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

24 Karl Bringmann, Nick Fischer, and Marvin Künnemann. A fine-grained analogue of Schaefer's theorem in P: Dichotomy of $\exists^k\forall$-quantified first-order graph properties. In *Proceedings of the 34th Computational Complexity Conference*, volume 137 of *CCC '19*, pages 31:1–31:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

25 Karl Bringmann, Marvin Künnemann, and Karol Wegrzycki. Approximating APSP without scaling: Equivalence of approximate min-plus and exact min-max. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing*, STOC '19, pages 943–954. ACM, 2019.

26 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '18, pages 979–990. IEEE Computer Society, 2018.

27 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '17, pages 743–754. IEEE Computer Society, 2017.

28 Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. In *Proceedings of the 33th Computational Complexity Conference*, volume 102 of *CCC '18*, pages 14:1–14:45. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

29 Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1–20. SIAM, 2019.

30 Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 21–40. SIAM, 2019.

31 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *ICALP '19*, pages 42:1–42:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

32 Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3):511–522, 1995.

33 Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A subquadratic algorithm for 3XOR. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117 of *MFCS '18*, pages 59:1–59:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

34 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014.

35 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

36 Andreas Emil Feldmann, C. S. Karthik, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Electronic Colloquium on Computational Complexity (ECCC)*, 27:86, 2020.

37 Harold N. Gabow. Scaling algorithms for network problems. *J. Comput. Syst. Sci.*, 31(2):148–168, 1985.

38 Jiawei Gao. On the fine-grained complexity of least weight subsequence in multitrees and bounded treewidth DAGs. In *Proceedings of the 14th International Symposium on Parameterized and Exact Computation*, volume 148 of *IPEC '19*, pages 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

39 Jiawei Gao and Russell Impagliazzo. The fine-grained complexity of strengthenings of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:9, 2019.

40 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Trans. Algorithms*, 15(2):23:1–23:35, 2019.

41 Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529. Morgan Kaufmann, 1999.

42 Ashish Goel, Piotr Indyk, and Kasturi R. Varadarajan. Reductions among high dimensional proximity problems. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 769–778. SIAM, 2001.

43 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing*, STOC '20, pages 657–670. ACM, 2020.

44 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

45 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

46 Dorit S. Hochbaum. *Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems*, pages 94–143. PWS Publishing Co., 1996.

47 Piotr Indyk. Dimensionality reduction techniques for proximity problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 371–378. SIAM, 2000.

48 Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 539–545. SIAM, 2003.

49 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613. ACM, 1998.

50 Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, triangles. *Algorithmica*, 74(1):326–343, 2016.

51 C. S. Karthik, Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019.

52 Karthik C. S. and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science*, volume 124 of *ITCS '19*, pages 17:1–17:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

53 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000.

54 Donald E. Knuth. Dancing links. *CoRR*, abs/cs/0011047, 2000. `arXiv:0011047`.

55 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing*, STOC '20, pages 699–712. ACM, 2020.

56 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, 1998.

57 Bingkai Lin. Constant approximating k-clique is w[1]-hard. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing*, STOC '21, pages 1749–1756. ACM, 2021.

58 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 1236–1252. SIAM, 2018.

59 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 62–81. SIAM, 2020.

60 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

61 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, STOC '13, pages 515–524. ACM, 2013.

62 Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing*, STOC '18, pages 1260–1268. ACM, 2018.

63 Noah Stephens-Davidowitz and Vinod Vaikuntanathan. SETH-hardness of coding problems. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '19, pages 287–301. IEEE Computer Society, 2019.

64 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *J. ACM*, 62(2), 2015.

65 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians*, ICM '18, pages 3447–3487, 2018.

66 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

67 Ryan Williams. *Algorithms and resource requirements for fundamental problems*. ProQuest LLC, Ann Arbor, MI, 2007. Thesis (Ph.D.)–Carnegie Mellon University.

68 Ryan Williams. Faster decision of first-order graph properties. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14. ACM, 2014.

69 Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 1207–1215. SIAM, 2018.

70 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

Faster Knapsack Algorithms via Bounded Monotone Min-Plus-Convolution

Karl Bringmann
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Alejandro Cassis
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We present new exact and approximation algorithms for 0-1-Knapsack and Unbounded Knapsack:

- *Exact Algorithm for 0-1-Knapsack:* 0-1-Knapsack has known algorithms running in time $\widetilde{O}(n + \min\{n \cdot \mathrm{OPT}, n \cdot W, \mathrm{OPT}^2, W^2\})$ [Bellman '57], where n is the number of items, W is the weight budget, and OPT is the optimal profit. We present an algorithm running in time $\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$. This improves the running time in case n, W, OPT are roughly equal.
- *Exact Algorithm for Unbounded Knapsack:* Unbounded Knapsack has known algorithms running in time $\widetilde{O}(n + \min\{n \cdot p_{\max}, n \cdot w_{\max}, p_{\max}^2, w_{\max}^2\})$ [Axiotis, Tzamos '19, Jansen, Rohwedder '19, Chan, He '22], where n is the number of items, $w_{\max}$ is the largest weight of any item, and $p_{\max}$ is the largest profit of any item. We present an algorithm running in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$, giving a similar improvement as for 0-1-Knapsack.
- *Approximating Unbounded Knapsack with Resource Augmentation:* Unbounded Knapsack has a known FPTAS with running time $\widetilde{O}(\min\{n/\varepsilon, n + 1/\varepsilon^2\})$ [Jansen, Kraft '18]. We study *weak* approximation algorithms, which approximate the optimal profit but are allowed to overshoot the weight constraint (i.e. resource augmentation). We present the first approximation scheme for Unbounded Knapsack in this setting, achieving running time $\widetilde{O}(n + 1/\varepsilon^{1.5})$. Along the way, we also give a simpler FPTAS with lower order improvement in the standard setting.

For all of these problem settings the previously known results had matching conditional lower bounds. We avoid these lower bounds in the approximation setting by allowing resource augmentation, and in the exact setting by analyzing the time complexity in terms of weight and profit parameters (instead of only weight or only profit parameters).

Our algorithms can be seen as reductions to Min-Plus-Convolution on monotone sequences with bounded entries. These structured instances of Min-Plus-Convolution can be solved in time $\widetilde{O}(n^{1.5})$ [Chi, Duan, Xie, Zhang '22] (in contrast to the conjectured $n^{2-o(1)}$ lower bound for the general case). We complement our results by showing reductions in the opposite direction, that is, we show that achieving our results with the constant 1.5 replaced by any constant < 2 implies subquadratic algorithms for Min-Plus-Convolution on monotone sequences with bounded entries.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

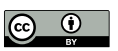
Keywords and phrases Knapsack, Approximation Schemes, Fine-Grained Complexity, Min-Plus Convolution

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.31

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2205.08493

Funding This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).



49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 31; pp. 31:1–31:21

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this paper we present new exact and approximation algorithms for Knapsack problems.

Exact Pseudopolynomial Algorithms for Unbounded Knapsack. In the Unbounded Knapsack problem we are given a set of n items, where each item has a weight w_i and a profit p_i, along with a knapsack capacity W. The goal is to find a *multiset* of items which maximizes the total profit and has total weight at most W. The textbook dynamic programming algorithm for Unbounded Knapsack due to Bellman [6] runs in time $O(nW)$ or in time $O(n \cdot \mathrm{OPT})$, where OPT is the value of the optimal solution. Recent literature on Unbounded Knapsack studies alternative parameters: Axiotis and Tzamos [3] and Jansen and Rohwedder [25] independently presented algorithms running in time $\widetilde{O}(w_{\max}^2)$ and $\widetilde{O}(p_{\max}^2)$,[1] where $w_{\max}$ is the largest weight of any item and $p_{\max}$ the largest profit of any item – in general $w_{\max}$ could be much smaller than W and $p_{\max}$ much smaller than OPT, so these algorithms improve upon Bellman's algorithm for some parameter settings. Chan and He [15] presented further improvements[2] achieving time $\widetilde{O}(n\, w_{\max})$ and $\widetilde{O}(n\, p_{\max})$ Note that when $w_{\max} \approx p_{\max} \approx n$ all mentioned algorithms require at least quadratic time $\Omega(n^2)$. Can we overcome this quadratic barrier? In this paper we answer this positively by considering the combined parameter $w_{\max} + p_{\max}$.

▶ **Theorem 1.** *Unbounded Knapsack can be solved in expected time* $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$.

This result is particularly interesting in light of recent fine-grained lower bounds for Unbounded Knapsack. Indeed, for each previous result that we have mentioned above, a matching conditional lower bound is known [19, 28]. For example, Unbounded Knapsack cannot be solved in time $O((n+W)^{2-\delta})$ for any constant $\delta > 0$ under a plausible hypothesis. Inspecting these conditional lower bounds, we observe that they construct hard instances where only the profit parameters or only the weight parameters are under control; one of the two must be very large to obtain a hardness reduction. We thus avoid these conditional lower bounds by considering the combined profit and weight parameter $w_{\max} + p_{\max}$.

Exact Pseudopolynomial Algorithms for 0-1 Knapsack. The 0-1 Knapsack problem is the variant of Unbounded Knapsack where every input item can appear at most once in any solution. Bellman's algorithm also solves 0-1 Knapsack in time $O(nW)$ or $O(n \cdot \mathrm{OPT})$. However, the landscape is more diverse when considering other parameters. In particular, it is open whether 0-1 Knapsack can be solved in time $\widetilde{O}(n + w_{\max}^2)$ or $\widetilde{O}(n + p_{\max}^2)$.[3] Table 1 shows a non-exhaustive list of pseudopolynomial-time algorithms for 0-1 Knapsack using different combinations of the parameters $n, w_{\max}, p_{\max}, W, \mathrm{OPT}$. Note that when all

[1] We use the notation $\widetilde{O}(T) = \cup_{c>0} O(T \log^c T)$ to supress polylogarithmic factors

[2] Note that we can assume that $n \leq w_{\max}$ without loss of generality since if there are multiple items with the same weight, we can keep only the one with the largest profit. Similarly, $n \leq p_{\max}$.

[3] This gap is analogous to the case of Subset Sum where we are given a set of numbers X and a target number t. For the unbounded case, where the goal is to find whether a multiset of items in X sums to t, Jansen and Rohwedder [25] gave an algorithm in time $\widetilde{O}(n+u)$ where u is the largest number in the input. For the more standard "0-1" case where we ask for a subset of X summing to t, the best known running times are $\widetilde{O}(n+t)$ [8, 26], $O(nu)$ [33], $\widetilde{O}(n+u^2/n)$ by combining [21] and [8, 26], and $\widetilde{O}(n+u^{3/2})$ by combining [21] and [8, 26] and [33]; see also [11, 34] for generalizations to X being a multiset and related results.

Table 1 Non-exhaustive list of pseudopolynomial-time algorithms for 0-1 Knapsack.

Running Time	Reference
$O(n \cdot \min\{W, \mathrm{OPT}\})$	[6]
$O(n \cdot p_{\max} \cdot w_{\max})$	[33]
$\widetilde{O}(n + w_{\max} \cdot W)$	[27, 5, 3]
$\widetilde{O}(n + p_{\max} \cdot W)$	[5]
$\widetilde{O}(n \cdot \min\{w_{\max}^2, p_{\max}^2\})$	[3]
$\widetilde{O}((n+W) \cdot \min\{w_{\max}, p_{\max}\})$	[5]
$O(n + \min\{w_{\max}^3, p_{\max}^3\})$	[34]
$\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$	**Theorem 2**

these parameters are bounded by $O(n)$, all existing algorithms require at least quadratic time $\Omega(n^2)$. In this paper we show that by considering the combined weight and profit parameter $W + \mathrm{OPT}$ we can overcome this quadratic barrier.

▶ **Theorem 2.** *There is a randomized algorithm for 0-1 Knapsack that runs in time $\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$ and succeeds with high probability.*

Similar to the unbounded case, matching conditional lower bounds ruling out time $O((n+W)^{2-\delta})$ and $O((n+\mathrm{OPT})^{2-\delta})$ for any $\delta > 0$ are known [19, 28]. These lower bounds construct hard instances where only one of W, OPT is under control, the other needs to be very large. We thus avoid these lower bounds by considering the combined weight and profit parameter $W + \mathrm{OPT}$.

Approximation Schemes for Unbounded Knapsack. Since Unbounded Knapsack is well known to be NP-hard, it is natural to study approximation algorithms. In particular, a *fully polynomial-time approximation scheme* (FPTAS) given $0 < \varepsilon < 1$ computes a solution x with total weight $w(x) \leq W$ and total profit $p(x) \geq (1-\varepsilon)\mathrm{OPT}$ in time $\mathrm{poly}(n, 1/\varepsilon)$. The first FPTAS for Unbounded Knapsack was designed by Ibarra and Kim in 1975 [23] and runs in time $\widetilde{O}(n + (1/\varepsilon)^4)$. In 1979 Lawler [30] improved the running time to $\widetilde{O}(n + (1/\varepsilon)^3)$. This was the best known until Jansen and Kraft in 2018 [24] presented an FPTAS running in time $\widetilde{O}(n + (1/\varepsilon)^2)$. This algorithm has a matching conditional lower ruling out time $O((n + 1/\varepsilon)^{2-\delta})$ for any $\delta > 0$ [19, 28, 32].

We present a new FPTAS for Unbounded Knapsack which is (as we believe) simpler than Jansen and Kraft's, and has a lower order improvement in the running time:

▶ **Theorem 3.** *Unbounded Knapsack has an FPTAS with running time $\widetilde{O}\left(n + \frac{(1/\varepsilon)^2}{2^{\Omega(\sqrt{\log(1/\varepsilon)})}}\right)$.*

Bringmann and Nakos [10] recently gave an FPTAS for the related Subset Sum problem which achieves the same running time.

Weak Approximation for Unbounded Knapsack. Motivated by the matching upper and conditional lower bounds for FPTASs for Unbounded Knapsack, we study the relaxed notion of *weak approximation* as coined in [32]: we relax the weight constraint and seek a solution x with total weight $w(x) \leq (1+\varepsilon) \cdot W$ and total profit $p(x) \geq (1-\varepsilon)\mathrm{OPT}$. Note that OPT is still the optimal value of any solution with weight at most W. This can be

interpreted as bicriteria approximation (approximating both the weight and profit constraint) or as resource augmentation (the optimal algorithm is allowed weight W while our algorithm is allowed a slightly larger weight of $(1+\varepsilon)\cdot W$). All of these are well-established relaxations of the standard (=strong[4]) notion of approximation. Such weaker notions of approximation are typically studied when a PTAS for the strong notion of approximation is not known. More generally, studying these weaker notions is justified whenever there are certain limits for strong approximations, to see whether these limits can be overcome by relaxing the notion of approximation. In particular, we want to understand whether this relaxation can overcome the conditional lower bound ruling out time $O((n+1/\varepsilon)^{2-\delta})$. For the related Subset Sum problem this question has been resolved positively: Bringmann and Nakos [10] conditionally ruled out strong approximation algorithms in time $O((n+1/\varepsilon)^{2-\delta})$ for any $\delta > 0$, but Mucha, Węgrzycki and Włodarczyk [32] designed a weak FPTAS in time $\widetilde{O}(n+(1/\varepsilon)^{5/3})$.

In this paper, we give a positive answer for Unbounded Knapsack:

▶ **Theorem 4.** *Unbounded Knapsack has a weak approximation scheme running in expected time* $\widetilde{O}(n+(\frac{1}{\varepsilon})^{1.5})$.

Our theorem gives reason to believe that resource augmentation indeed makes the problem easier. Specifically, obtaining a strong approximation scheme with the same running time as our weak one would refute the existing conditional lower bound for strong approximation.

Min-Plus-Convolution. All conditional lower bounds mentioned above are based on a hypothesis about the MinConv problem: Given sequences $A, B \in \mathbb{Z}^n$ compute their $(\min,+)$-convolution, which is the sequence $C \in \mathbb{Z}^{2n}$ with $C[k] = \min_{i+j=k} A[i]+B[j]$.[5] The MinConv problem can be trivially solved in time $O(n^2)$. This can be improved to time $n^2/2^{\Omega(\sqrt{\log n})}$ via a reduction to $(\min,+)$-matrix product due to Bremner et al. [7], and using Williams' algorithm for the latter [37] (which was derandomized later by Chan and Williams [17]). The lack of truly subquadratic algorithms despite considerable effort has led researchers to postulate the MinConv hypothesis, namely that MinConv cannot be solved in time $O(n^{2-\delta})$ for any constant $\delta > 0$ [19, 28]. Many problems are known to have conditional lower bounds from the MinConv hypothesis, see, e.g., [4, 14, 19, 25, 28, 29].

Central to our work is a reduction from MinConv to Unbounded Knapsack shown independently by Cygan et al. [19] and Künnemann et al. [28]. In particular, they showed that if Unbounded Knapsack on n items and $W = O(n)$ can be solved in subquadratic time, then MinConv can be solved in subquadratic time. This reduction immediately implies matching conditional lower bounds for the previously known exact algorithms with running times $O(n\cdot W)$, $\widetilde{O}(w_{\max}^2)$ and $\widetilde{O}(n\cdot w_{\max})$, as mentioned earlier.

A small modification of this reduction extends to the dual case, i.e., an exact subquadratic-time algorithm for Unbounded Knapsack with $\mathrm{OPT} = O(n)$ would result in a subquadratic-time algorithm for MinConv. This establishes matching conditional lower bounds for the algorithms in time $O(n\cdot\mathrm{OPT})$, $\widetilde{O}(p_{\max}^2)$ and $\widetilde{O}(n\cdot p_{\max})$. Moreover, by setting $\varepsilon = \Theta(1/\mathrm{OPT})$, an FPTAS for Unbounded Knapsack would yield an exact algorithm for MinConv, establishing that the $\widetilde{O}(n+(1/\varepsilon)^2)$-time FPTAS is conditionally optimal. This last observation was pointed out in [32].

[4] From now on, by "strong" approximation we mean the standard (non-weak) notion of approximation.
[5] If we replace the min by a max, we obtain the MaxConv problem, which is equivalent to MinConv after negating the sequences. Therefore, we will sometimes use these two names interchangeably.

In fact, the reduction due to Cygan et al. [19] is even an equivalence, showing that Unbounded Knapsack is intimately connected to MinConv. The same reduction is also known for 0-1 Knapsack [19, 28], so a similar discussion applies to 0-1 Knapsack.

Bounded Monotone Min-Plus-Convolution. Despite its postulated hardness, there are restricted families of instances of MinConv for which subquadratic algorithms are known, see e.g. [9, 12, 16, 18]. Central to this paper is the case when the input sequences $A, B \in \mathbb{Z}^n$ are monotone non-decreasing and have entries bounded by $O(n)$; we call this task BMMinConv.

In a celebrated result, Chan and Lewenstein gave an algorithm for BMMinConv that runs in expected time $O(n^{1.859})$ [16]. As a big hammer, their algorithm uses the famous Balog-Szemerédi-Gowers theorem from additive combinatorics. Very recently, Chi, Duan, Xie and Zhang showed how to avoid this big hammer and improved the running time to expected $\widetilde{O}(n^{1.5})$ [18].

All of our results mentioned so far use this BMMinConv algorithm as a subroutine. That is, our algorithms are reductions from various Knapsack problems to BMMinConv.

Equivalence with Bounded Monotone Min-Plus-Convolution. We complement our results by showing reductions in the opposite direction: Following the same chain of reductions as in [19, 28] but starting from bounded monotone instances of $(\min,+)$-convolution, we reduce BMMinConv to $O(n)$ instances of Unbounded Knapsack with $O(\sqrt{n})$ items each, where it holds that $w_{\max}, p_{\max}, W$ and OPT are all bounded by $O(\sqrt{n})$. Instantiating this reduction for the exact and approximate setting, we show the following theorem.

▶ **Theorem 5** (Equivalence). *For any problems A and B from the following list, if A can be solved in time $\widetilde{O}(n^{2-\delta})$ for some $\delta > 0$, then B can be solved in randomized time $\widetilde{O}(n^{2-\delta/2})$:*

1. *BMMaxConv on sequences of length n*
2. *Unbounded Knapsack on n items and $W, \mathrm{OPT} = O(n)$*
3. *0-1 Knapsack on n items and $W, \mathrm{OPT} = O(n)$*
4. *Weak $(1+\varepsilon)$-approximation for Unbounded Knapsack on n items and $\varepsilon = \Theta(1/n)$*

On the one hand, Theorem 1 solves Unbounded Knapsack in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$ by using Chi, Duan, Xie and Zhang's subquadratic BMMinConv algorithm [18]. On the other hand, Theorem 5 shows that any algorithm solving Unbounded Knapsack in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{2-\delta})$ can be transformed into a subquadratic BMMinConv algorithm. This shows that both our exact and approximation algorithms take essentially *the only possible route* to obtain subquadratic algorithms, by invoking a BMMinConv algorithm.

Is randomness necessary? The algorithms given by Theorems 1, 2 and 4 are all randomized. If we insist on deterministic algorithms, we note that by applying Chan and Lewenstein's deterministic $\widetilde{O}(n^{1.864})$-time algorithm for BMMaxConv [16], we can obtain deterministic versions of Theorem 1 and Theorem 4 with exponent 1.864 instead of 1.5 (i.e. the only part where we use randomness is in applying Chi, Duan, Xie and Zhang's algorithm [18]). On the other hand, we do not know[6] how to derandomize Theorem 2.

[6] Our algorithm closely follows Bringmann's algorithm for Subset Sum [8] whose derandomization is an open problem.

Organization of the extended abstract. Due to space constraints, we only include the details of our exact algorithm for Unbounded Knapsack and 0-1 Knapsack in the main part of this manuscript. We defer our approximation schemes and the equivalence between BMMinConv and knapsack problems to the full version.

1.1 Technical Overview

Exact algorithm for Unbounded Knapsack. In Section 3 we present our exact algorithm for Unbounded Knapsack. Let $(\mathcal{I}, W)$ be an instance of Unbounded Knapsack. We denote by $\mathcal{P}_i[0, \dots, W]$ the array where $\mathcal{P}_i[j]$ is the maximum profit of a solution of weight at most j using at most 2^i items. Since each item has weight at least 1, any feasible solution consists of at most W items. Thus, our goal is to compute the value $\mathcal{P}_{\lceil \log W \rceil}[W] = \mathrm{OPT}$.

The natural approach is to use dynamic programming: since $\mathcal{P}_0$ consists of solution of at most one item, it can be computed in time $O(n)$. For $i > 0$ we can compute $\mathcal{P}_i[0, \dots, W]$ by taking the $(\max, +)$-convolution of $\mathcal{P}_{i-1}[0, \dots, W]$ with itself. This gives an algorithm in time $O(W^2 \log W)$.

Jansen and Rohwedder [25] and Axiotis and Tzamos [3] showed that instead of convolving sequences of length W, it suffices to convolve only $O(w_{\max})$ entries of $\mathcal{P}_{i-1}$ in each iteration. This improves the running time to $O(w_{\max}^2 \log W)$ by using the naive algorithm for $(\max, +)$-convolution. The approach of Jansen and Rohwedder [25] is as follows. Suppose x is the optimal solution for a target value $\mathcal{P}_i[j]$. They showed that x can be split into two solutions x_1, x_2 such that (i) the number of items in each part is at most 2^{i-1} and (ii) the difference between the weights of both parts is at most $O(w_{\max})$. Thus, (i) guarantees that both x_1 and x_2 are optimal solutions for two entries of $\mathcal{P}_{i-1}$, and (ii) implies that these entries lie in an interval in $\mathcal{P}_{i-1}$ of length $O(w_{\max})$. In this way, they can afford to perform the $(\max, +)$-convolution of only $O(w_{\max})$ entries in $\mathcal{P}_{i-1}$.

To show the existence of such a partitioning of x they make use of Steinitz' Lemma [36], which shows that any collection of m vectors in $\mathbb{R}^d$ with infinity norm at most 1, whose sum is 0, can be permuted such that every prefix sum has norm at most $O(d)$ (see Lemma 11 for the precise statement). The partitioning of x follows from Steinitz' Lemma by taking the weights of the items picked by x as 1-dimensional vectors. The usage of Steinitz' Lemma to reduce the number of states in dynamic programs was pioneered by Eisenbrand and Weismantel [20], and further refined by Jansen and Rohwedder [25].

In our algorithm, we use Steinitz' Lemma in a similar way to split the number of items and the weight of x, but additionally we use it to ensure that the profits of the solutions x_1, x_2 differ by at most $O(p_{\max})$ (see Lemma 12). In this way, by carefully handling the subproblems $\mathcal{P}_{i-1}$ we can enforce that the values of the $O(w_{\max})$ entries that need to be convolved have values in a range of size $O(p_{\max})$. Since the arrays $\mathcal{P}_i$ are monotone non-decreasing, we can then apply the algorithm for BMMaxConv, and thus handle each subproblem in time $\widetilde{O}((p_{\max} + w_{\max})^{1.5})$.

Exact algorithm for 0-1 Knapsack. Cygan et al. [19] showed that there is a reduction from 0-1 Knapsack to MaxConv. More precisely, they showed that if MaxConv can be solved in time $T(n)$, then 0-1 Knapsack can be solved in randomized time $\widetilde{O}(T(W))$. Their reduction is a generalization of Bringmann's Subset Sum algorithm [8], which can be seen as a reduction from Subset Sum to Boolean convolution. Cygan et al. showed that the reduction for 0-1 Knapsack can be obtained by essentially replacing the Boolean convolutions by $(\max, +)$-convolutions in Bringmann's algorithm.

In Section 4 we observe that this reduction produces instances of MaxConv which are monotone non-decreasing and have entries bounded by OPT. That is, we obtain BMMaxConv instances of size $O(W + \mathrm{OPT})$, and following the analysis of [19] we obtain an algorithm for 0-1 Knapsack in time $\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$, which yields Theorem 2.

Approximating Unbounded Knapsack. Let $(\mathcal{I}, W)$ be an instance of Unbounded Knapsack. Consider the array $\mathcal{P}[0, \ldots, W]$ where $\mathcal{P}[j]$ is the maximum profit of a solution with weight at most j. In particular, $\mathcal{P}[W] = \mathrm{OPT}$ is the optimal value of the instance. We will use that the sequence $\mathcal{P}[0, \ldots, W]$ is monotone.

We present the following simple algorithm to compute $\mathcal{P}[0, \ldots, W]$, which is based on an algorithm for (unbounded) Subset Sum from Bringmann [8]. Fix an optimal solution x for an entry $\mathcal{P}[j]$. We can split x into three smaller solutions x_1, x_2, x_3 such that the total weight of both x_1 and x_2 is at most $j/2$, and x_3 is consists of at most one item. It follows that if we know $\mathcal{P}[0, \ldots, j/2]$, then we can compute $\mathcal{P}[0, \ldots, j]$ by taking the $(\max, +)$-convolution of $\mathcal{P}[0, \ldots, j/2]$ with itself and possibly adding one extra item from $\mathcal{I}$. In particular, if we compute all entries $\mathcal{P}[0, \ldots, W/n]$ as a base case using dynamic programming in time $O(W)$, then we can compute the entire sequence $\mathcal{P}[0, \ldots, W]$ by applying $O(\log n)$ $(\max, +)$-convolutions on sequences of length at most W. The overall running time is $O(W^2 \log n)$.

Although this exact algorithm is not particularly exciting or new, it can be nicely extended to the approximate setting. We show that by replacing the exact $(\max, +)$-convolutions with approximate ones, we obtain an FPTAS for Unbounded Knapsack in time $\widetilde{O}(n+1/\varepsilon^2)$. To this end, we preprocess the item set to get rid of *light* items with weight smaller than $\varepsilon \cdot W$, and *cheap* items with profit smaller than $\varepsilon \cdot \mathrm{OPT}$, while decreasing the optimal value by only $O(\varepsilon \cdot \mathrm{OPT})$. We now proceed as in the exact case, starting with the base case $\mathcal{P}[0, \ldots, \varepsilon \cdot W]$, which is all-zeroes since there are no more light items. Then we can build up $\mathcal{P}[0, \ldots, 2^j \varepsilon W]$ for increasing values of j by performing approximate $(\max, +)$-convolutions, until we have computed $\mathcal{P}[0, \ldots, W]$. For approximating $(\max, +)$-convolutions we use an algorithm due to Chan [13], which in our setting without cheap items runs in time $\widetilde{O}(1/\varepsilon^2)$. Thus, after applying the preprocessing in time $O(n)$, we compute a $(1+\varepsilon)$-approximation of $\mathcal{P}[0, \ldots, W]$ by applying $O(\log 1/\varepsilon)$ approximate $(\max, +)$-convolutions in overall time $\widetilde{O}(n + (1/\varepsilon)^2)$.

Then, we treat the case of weak approximation. The main steps of the algorithm are virtually the same as before. The crucial difference is that now we can afford to round weights. In this way, we can adapt Chan's algorithm and construct MaxConv instances which are monotone non-decreasing and have bounded entries. This yields BMMaxConv instances, and by using Chi, Duan, Xie and Zhang's algorithm for this special case [18], we can compute a weak approximation of $(\max, +)$-convolution in time $\widetilde{O}((1/\varepsilon)^{1.5})$. By similar arguments as for the strong approximation, this yields a weak approximation scheme running in time $\widetilde{O}(n + (1/\varepsilon)^{1.5})$.

Equivalence between BMMinConv and Knapsack problems. As mentioned earlier in the introduction, Cygan et al. [19] and Künnemann et al. [28] independently showed a reduction from MinConv to Unbounded Knapsack. In the full version of the paper, we show that following the same chain of reductions from MaxConv to Unbounded Knapsack but instead starting from BMMaxConv, with minor adaptations we can produce instances of Unbounded Knapsack with $W, \mathrm{OPT} = O(n)$. Together with our exact algorithm for Unbounded Knapsack, which we can phrase as a reduction to BMMaxConv, we obtain an equivalence of BMMaxConv and Unbounded Knapsack with $W, \mathrm{OPT} = O(n)$ – if one of these problems can be solved in subquadratic time, then both can.

Note that for Unbounded Knapsack with $W, \mathrm{OPT} = O(n)$ a weak $(1+\varepsilon)$-approximation for $\varepsilon = \Theta(1/n)$ already computes an exact optimal solution. This yields the reduction from BMMaxConv to the approximate version of Unbounded Knapsack. We similarly obtain a reduction to 0-1 Knapsack with $W, \mathrm{OPT} = O(n)$. This yields our equivalences from Theorem 5.

2 Preliminaries

We write $\mathbb{N} = \{0, 1, 2, \dots\}$. For $t \in \mathbb{N}$ we let $[t] = \{0, 1, \dots, t\}$. For reals $a \le b$ we write $[a, b]$ for the interval from a to b, and for reals a, b with $b \ge 0$ we write $[a \pm b]$ for the interval $[a - b, a + b]$.

We formally define the Unbounded Knapsack problem. We are given a set of items $\mathcal{I} = \{(p_1, w_1), \dots, (p_n, w_n)\}$, where each item i has a profit $p_i \in \mathbb{N}$ and a weight $w_i \in \mathbb{N}$, and a knapsack capacity $W \in \mathbb{N}$. The task is to maximize $\sum_{i=1}^{n} p_i \cdot x_i$ subject to the constraints $\sum_{i=1}^{n} w_i \cdot x_i \le W$ and $x \in \mathbb{N}^n$. The more standard 0-1 Knapsack problem is defined in the same way, but the solution x is constrained to $x \in \{0, 1\}^n$.

Given an instance $(\mathcal{I}, W)$, we denote by $x \in \mathbb{N}^n$ a multiset of items, where x_i is the number of copies of the i-th item. We sometimes refer to x as a *solution*. We write $p_{\mathcal{I}}(x)$ for the total profit of x, i.e., $p_{\mathcal{I}}(x) := \sum_i x_i \cdot p_i$. Similarly, we write $w_{\mathcal{I}}(x) := \sum_i x_i \cdot w_i$ for the weight of x. When the item set $\mathcal{I}$ is clear from context, we drop the subscript and simply write $p(x)$ and $w(x)$. We denote the number of items contained in a solution x by $\|x\|_1 := \sum_i x_i$. A solution x is *feasible* if it satisfies the constraint $w(x) \le W$. We denote by OPT the maximum profit $p(x)$ of any feasible solution x. We denote by $p_{\max} := \max_{(p,w)\in\mathcal{I}} p$ the maximum profit of any input item and by $w_{\max} := \max_{(p,w)\in\mathcal{I}} w$ the maximum weight of any input item.

Notions of Approximation. We say that an algorithm gives a *strong* $(1+\varepsilon)$-approximation for Unbounded Knapsack if it returns a solution $x \in \mathbb{N}^n$ with weight $w(x) \le W$ and profit $p(x) \ge (1-\varepsilon) \cdot \mathrm{OPT}$. We say that an algorithm gives a *weak* $(1+\varepsilon)$-approximation for Unbounded Knapsack if it returns a solution $x \in \mathbb{N}^n$ with profit $p(x) \ge (1-\varepsilon) \cdot \mathrm{OPT}$ and weight $w(x) \le (1+\varepsilon) \cdot W$. We stress that here OPT still denotes the optimum value with weight at most W, i.e., $\mathrm{OPT} = \max\{p(x) \mid x \in \mathbb{N}^n, w(x) \le W\}$.

Profit Sequences. Given an item set $\mathcal{I}$ and capacity W, we define the array $\mathcal{P}_{\mathcal{I}}[0, \dots, W]$, where $\mathcal{P}_{\mathcal{I}}[j]$ is the maximum profit achievable with capacity j, i.e.,

$$\mathcal{P}_{\mathcal{I}}[j] := \max\{p_{\mathcal{I}}(x) \colon x \in \mathbb{N}^n, w_{\mathcal{I}}(x) \le j\}.$$

Note that $\mathcal{P}_{\mathcal{I}}[0] = 0$. A textbook way to compute $\mathcal{P}_{\mathcal{I}}[0, \dots, W]$ is by dynamic programming:

▶ **Fact 6.** *$\mathcal{P}_{\mathcal{I}}[0, \dots, W]$ can be computed using dynamic programming in time $O(n \cdot W)$.*

We will also consider the array $\mathcal{P}_{\mathcal{I},k}[0, \dots, W]$, where we restrict to solutions with at most 2^k items, for any non-negative integer k, i.e., for any $j \in [W]$ we set

$$\mathcal{P}_{\mathcal{I},k}[j] := \max\{p_{\mathcal{I}}(x) \colon x \in \mathbb{N}^n, w(x) \le j, \|x\|_1 \le 2^k\}.$$

When $\mathcal{I}$ is clear from context, we will drop the subscript and write $\mathcal{P}[0, \dots, W]$ and $\mathcal{P}_k[0, \dots, W]$. When we work with 0-1 Knapsack instead of Unbounded Knapsack, we will use the same notation $\mathcal{P}_{\mathcal{I}}$ and $\mathcal{P}_{\mathcal{I},k}$, where we restrict to $x \in \{0, 1\}^n$ instead of $x \in \mathbb{N}^n$.

MaxConv. The $(\max,+)$-convolution $A \oplus B$ of two sequences $A[0,\dots,n], B[0,\dots,n] \in \mathbb{Z}^{n+1}$ is a sequence of length $2n+1$ where $(A \oplus B)[k] := \max_{i+j=k} A[i] + B[j]$. We call MaxConv the task of computing the $(\max,+)$-convolution of two given sequences.

We will use the following handy notation: Given sequences $A[0,\dots,n], B[0,\dots,n]$ and intervals $I, J \subseteq [n]$ and $K \subseteq [2n]$, we denote by $C[K] := A[I] \oplus B[J]$ the computation of $C[k] := \max\{A[i] + B[j] \colon i \in I, j \in J, i+j=k\}$ for each $k \in K$. The following proposition shows that this can be computed efficiently. We defer the proof to Appendix A.

▶ **Proposition 7.** *If MaxConv can be solved in time $T(n)$, then $C[K] = A[I] \oplus B[J]$ can be computed in time $O(T(|I|+|J|)+|K|)$.*

Sometimes we will refer to the $(\min,+)$-convolution, where we replace max by a min. The two problems MinConv and MaxConv are equivalent after negating the sequences.

BMMaxConv. In the BMMaxConv problem, we compute the $(\max,+)$-convolution of sequences of length n which are monotone non-decreasing and have bounded values. For this setting Chan and Lewenstein [16] gave the first subquadratic algorithm, and recently Chi, Duan, Xie and Zhang gave the following remarkable result:

▶ **Theorem 8** (BMMaxConv [18]). *Given monotone non-decreasing sequences $A[0,\dots,n]$ and $B[0,\dots,n]$ with entries $A[i], B[i] \in [O(n)] \cup \{-\infty\}$ for all $i \in [n]$, their $(\max,+)$-convolution $A \oplus B$ can be computed in expected time $\widetilde{O}(n^{1.5})$.*

Note that Chi, Duan, Xie and Zhang phrase their result for $(\min,+)$-convolution of monotone increasing sequences with entries in $[O(n)]$. We prove in Appendix A that both statements are equivalent, so their result also works for $(\max,+)$-convolution with entries in $[O(n)] \cup \{-\infty\}$.

Witnesses. Let $A[0,\dots,n], B[0,\dots,n]$ be an instance of MaxConv. Let $C := A \oplus B$. Given $k \in [2n]$, we say that $i \in [n]$ is a *witness* for $C[k]$ if $C[k] = A[i] + B[k-i]$. We say that an array $M[0\dots,2n]$ is a *witness array*, if each entry $M[k]$ contains some witness for $C[k]$.

For the general case of MaxConv it is well known (e.g. [35, 1]) that computing the witness array has the same time complexity as $(\max,+)$-convolution, up to a polylog(n) overhead. This reduction does not immediately apply to BMMaxConv because the sequences might not remain monotone. However, we make it work with some extra care, see Appendix B for the proof.

▶ **Lemma 9** (Witness Finding). *If BMMaxConv can be computed in time $T(n)$, then a witness array $M[0,\dots,2n]$ can be computed in time $\widetilde{O}(T(n))$.*

Niceness asumptions on time bounds. For all time bounds $T(n)$ in this paper, we make the following niceness assumptions: (1) $T(\widetilde{O}(n)) \le \widetilde{O}(T(n))$, and (2) $k \cdot T(n) \le O(T(kn))$ for any $k, n \ge 1$. This is satisfied for all natural time bounds of polynomial-time or pseudopolynomial-time algorithms, in particular it holds for all functions of the form $T(n) = \Theta(n^\alpha \log^\beta n)$ for any constants $\alpha \ge 1, \beta \ge 0$.

3 Exact algorithm for Unbounded Knapsack

In this section we prove the following Theorem:

▶ **Theorem 10.** *If BMMaxConv on length-n sequences can be solved in time $T(n)$, then Unbounded Knapsack can be solved in time $\widetilde{O}(n + T(p_{\max} + w_{\max}))$, where $p_{\max}$ is the largest profit of any item and $w_{\max}$ is the largest weight of any item.*

Note that Theorem 1 follows as an immediate corollary of Theorem 10 by plugging in Chi, Duan, Xie and Zhang's algorithm (Theorem 8).

For the entire section, fix an instance $(\mathcal{I}, W)$ of the Unbounded Knapsack problem. Recall that $\mathcal{P}_i[0, \dots, W]$ is defined as $\mathcal{P}_i[j] := \max\{p(x) \colon w(x) \leq j, \|x\|_1 \leq 2^i\}$, and set $\Delta := p_{\max} + w_{\max}$. Suppose we know that the optimal solution consists of at most 2^k items. Then, our goal is to compute the value $\mathcal{P}_k[W]$. The natural approach is to use dynamic programming: if we have computed $\mathcal{P}_{i-1}$, then $\mathcal{P}_i = \mathcal{P}_{i-1} \oplus \mathcal{P}_{i-1}$. To get our desired running time, we will show that we only need to convolve $O(\Delta)$ entries of $\mathcal{P}_{i-1}$ and that we can enforce that all of these fall in a range of $O(\Delta)$ values. By monotonicity of $\mathcal{P}_{i-1}$, we end up with a BMMaxConv instance, which can be solved in time $O(T(\Delta))$. The resulting total time to compute $\mathcal{P}_i[W]$ is $O(n + k \cdot T(\Delta))$. With additional preprocessing we ensure that $k = O(\log \Delta)$, turning the running time into $\widetilde{O}(n + T(\Delta))$.

3.1 Preparations

We need to show that when computing the optimal answer for some entry $\mathcal{P}_i[j]$, we can split it in such a way that both its total profit and its total weight are roughly halved. Our main tool to show this is the Steinitz Lemma [22, 36]. A beautiful proof for it can be found in [31].

▶ **Lemma 11** ([22, 36, Steinitz Lemma])**.** *Let $\|.\|$ be a norm in $\mathbb{R}^m$ and let M be an arbitrary collection of t vectors in $\mathbb{R}^m$ such that $\|v\| \leq 1$ for every $v \in M$ and $\sum_{v \in M} v = 0$. Then, it is possible to permute the vectors in M into a sequence $(v_1, \dots, v_t)$ such that $\|v_1 + \cdots + v_k\| \leq m$ holds for every $k \in [t]$.*

We use the Steinitz Lemma to argue that the items in a solution can be split in two parts in such a way that both the total profit and the total weight are roughly halved:

▶ **Lemma 12** (Splitting Lemma)**.** *Let $i \geq 1$ and consider a solution $x \in \mathbb{N}^n$ with $\|x\|_1 \leq 2^i$. Then there is a partition of x into two solutions $x_1, x_2 \in \mathbb{N}^n$ with the following properties:*

1. *(Splitting of Items) $\|x_1\|_1, \|x_2\|_1 \leq 2^{i-1}$ and $x = x_1 + x_2$,*
2. *(Approximate Splitting of Weight) $|w(x_1) - \frac{1}{2}w(x)| \leq 2\Delta$ and $|w(x_2) - \frac{1}{2}w(x)| \leq 2\Delta$,*
3. *(Approximate Splitting of Value) $|p(x_1) - \frac{1}{2}p(x)]| \leq 2\Delta$ and $|p(x_2) - \frac{1}{2}p(x)]| \leq 2\Delta$.*

Proof. Let $t := \|x\|_1 \leq 2^i$. First assume that t is even; we will remove this assumption later. Write $x = \sum_{j=1}^{t} x^{(j)}$ where each $x^{(j)}$ corresponds to one copy of some item, i.e. $\|x^{(j)}\|_1 = 1$, and set $v^{(j)} = \begin{pmatrix} w(x^{(j)}) \\ p(x^{(j)}) \end{pmatrix}$. Note that $\|v^{(j)}\|_\infty \leq \Delta$. By applying the Steinitz Lemma on the vectors $v^{(j)} - \frac{1}{t}\begin{pmatrix} w(x) \\ p(x) \end{pmatrix}$ (after normalizing by Δ), we can assume that the $v^{(j)}$'s are ordered such that

$$\left\| \sum_{j=1}^{t/2} v^{(j)} - \frac{1}{2}\begin{pmatrix} w(x) \\ p(x) \end{pmatrix} \right\|_\infty \leq 2\Delta. \tag{1}$$

Fix this ordering, and let $x_1 = x^{(1)} + \ldots + x^{(t/2)}$, corresponding to $v^{(1)}, \ldots, v^{(t/2)}$, and let $x_2 = x^{(t/2+1)} + \ldots + x^{(t)}$, corresponding to the remaining vectors $v^{(t/2+1)}, \ldots, v^{(t)}$. We now check that x_1, x_2 satisfy the properties of the lemma:

- Property 1 is clearly satisfied by construction.
- For property 2, note that (1) implies $|w(x_1) - \frac{1}{2}w(x)| \leq 2\Delta$. Since $w(x_2) = w(x) - w(x_1)$, we have that $|w(x_2) - \frac{1}{2}w(x)| = |\frac{1}{2}w(x) - w(x_1)| \leq 2\Delta$.
- Property 3 follows in the same way as property 2.

If t is odd, then $t + 1 \leq 2^i$, so we can add a dummy vector $x^{(t+1)} = 0$ with corresponding $v^{(t+1)} := \binom{0}{0}$ and repeat the same argument with $t := t + 1$. ◀

When we apply Lemma 12 to an optimal solution corresponding to an entry of the array $\mathcal{P}_i$, we obtain the following lemma.

▶ **Lemma 13.** *Let $\beta > 0$. For any index $j \in [\beta \pm 8\Delta] \cap [W]$ there are indices $j_1, j_2 \in [\frac{\beta}{2} \pm 8\Delta] \cap [W]$ with the following properties:*

(i) $j_1 + j_2 = j$,
(ii) $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2]$,
(iii) $|\mathcal{P}_{i-1}[j_1] - \frac{1}{2}\mathcal{P}_i[j]| \leq 2\Delta$ *and* $|\mathcal{P}_{i-1}[j_2] - \frac{1}{2}\mathcal{P}_i[j]| \leq 2\Delta$.

Proof. Let $x \in \mathbb{N}^n$ be an optimal solution for $\mathcal{P}_i[j]$, that is, we have $p(x) = \mathcal{P}_i[j]$, $w(x) \leq j$, and $\|x\|_1 \leq 2^i$. We apply Lemma 12 to x and obtain $x_1, x_2 \in \mathbb{N}^n$ such that $x_1 + x_2 = x$. We do a case distinction based on $w(x_1), w(x_2)$:

- $w(x_1), w(x_2) \in [\frac{j}{2} \pm 4\Delta]$: Let $j_1 := w(x_1)$ and $j_2 := j - j_1$; note that $j_1, j_2 \in [\frac{j}{2} \pm 4\Delta] \subseteq [\frac{\beta}{2} \pm 8\Delta]$. We argue that $p(x_1) = \mathcal{P}_{i-1}[j_1]$ and $p(x_2) = \mathcal{P}_{i-1}[j_2]$. Indeed, since $w(x_1) = j_1$ the solution x_1 is feasible for weight j_1, so $p(x_1) \leq \mathcal{P}_{i-1}[j_1]$. Similarly, since $w(x_2) = w(x) - w(x_1) \leq j - w(x_1) = j_2$ the solution x_2 is feasible for weight j_2, so $p(x_2) \leq \mathcal{P}_{i-1}[j_2]$. Moreover, by optimality of x we have $p(x_1) + p(x_2) = p(x) = \mathcal{P}_i[j] \geq \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2]$, so we obtain $p(x_1) = \mathcal{P}_{i-1}[j_1]$ and $p(x_2) = \mathcal{P}_{i-1}[j_2]$. Using these equations together with $p(x) = \mathcal{P}_i[j]$, property (ii) follows from $p(x) = p(x_1) + p(x_2)$, property (iii) follows from Property 3 of Lemma 12, and property (i) holds by definition of j_2.
- $w(x_1) < \frac{j}{2} - 4\Delta$: Property 2 of Lemma 12 implies that $|w(x_1) - w(x_2)| \leq 4\Delta$, and thus $w(x_2) \leq \frac{j}{2}$. Therefore, x_1 and x_2 are feasible for weights $j_1 := \lfloor \frac{j}{2} \rfloor$ and $j_2 := \lceil \frac{j}{2} \rceil$, respectively. Note that $j_1, j_2 \in [\frac{\beta}{2} \pm (4\Delta + 1)] \subseteq [\frac{\beta}{2} \pm 8\Delta]$. Property (i) is obvious, and properties (ii) and (iii) now follow as in the first case.
- $w(x_1) > \frac{j}{2} + 4\Delta$: Similarly as the previous case, property 2 of Lemma 12 implies that $w(x_2) \geq \frac{j}{2}$. Therefore, we have $w(x) = w(x_1) + w(x_2) > j + 4\Delta$, which contradicts the assumption $w(x) \leq j$.
- $w(x_2) < \frac{j}{2} - 4\Delta$ or $w(x_2) > \frac{j}{2} + 4\Delta$: Symmetric to the previous two cases. ◀

3.2 The algorithm

We are now ready to present our algorithm. The idea is to use the Splitting Lemma 12 to convolve smaller sequences which are bounded and monotone.

Let $\mathcal{I} = \{(w_i, p_i)\}_{i=1}^n$ with capacity constraint W be an instance of Unbounded Knapsack. Since any item has $w_i \geq 1$, we know that any solution $x \in \mathbb{N}^n$ consists of at most W items. Thus, to compute the value of the optimal solution it suffices to compute $\mathcal{P}_k[W]$ where $k := \lceil \log W \rceil$.

Our approach is as follows. We do binary search for OPT in the range $[p_{\max} \cdot W]$. Suppose we have the current guess α. Instead of computing the arrays $\mathcal{P}_i$, we compute *clipped versions*, i.e., C_i which has the property that $C_i[j] \geq \alpha$ if and only if $\mathcal{P}_i[j] \geq \alpha$.

We compute C_i as follows: At every step, we only compute the values for $O(\Delta)$ weights $C_i[W \cdot 2^{i-k} \pm 8\Delta]$. For the base case $i = 0$, we simply set $C_0[0, \dots, 8\Delta] := \mathcal{P}_0[0, \dots, 8\Delta]$. Note that this can be done in time $O(n + \Delta)$ by doing one pass over the item set, since $\mathcal{P}_0$ only considers solutions with at most one item. Moreover, observe that $C_0[0, \dots, 8\Delta]$ is monotone non-decreasing by definition of $\mathcal{P}_0$.

For the general case $i > 0$ we first compute an array $A_i[W \cdot 2^{i-k} \pm 8\Delta]$ by taking the $(\max, +)$-convolution of $C_{i-1}[W \cdot 2^{i-1-k} \pm 8\Delta]$ with itself. To obtain $C_i[W \cdot 2^{i-k} \pm 8\Delta]$, we clip the values in A_i which are too large, and set to $-\infty$ the values which are to small. This ensures that all values in C_i lie within a range of $O(\Delta)$, except for values that are $-\infty$. Algorithm 1 contains the pseudocode.

Algorithm 1 Given an instance $(\mathcal{I}, W)$ of Unbounded Knapsack and a guess $\alpha \in [p_{\max} \cdot W]$, the algorithm computes a value $C_k[W]$ satisfying the guarantee in Lemma 15.

1: $k := \lceil \log W \rceil$
2: Initialize $C_0[0, \dots, 8\Delta] := \mathcal{P}_0[0, \dots, 8\Delta]$ by iterating over the item set $\mathcal{I}$ once
3: **for** $i = 1, \dots, k$ **do**
4: $\quad A_i[W \cdot 2^{i-k} \pm 8\Delta] := C_{i-1}[W \cdot 2^{i-1-k} \pm 8\Delta] \oplus C_{i-1}[W \cdot 2^{i-1-k} \pm 8\Delta]$
5: $\quad C_i[j] := \begin{cases} \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta & \text{if } A_i[j] > \alpha \cdot 2^{i-k} + 24\Delta \\ -\infty & \text{if } A_i[j] < \alpha \cdot 2^{i-k} - 40\Delta \\ A_i[j] & \text{otherwise} \end{cases}$
return $C_k[W]$

Due to the clipping, at every step we compute a $(\max, +)$-convolution of sequences of length $O(\Delta)$ and values in $[O(\Delta)] \cup \{-\infty\}$ (after shifting the indices and values appropriately). Furthermore, note that all convolutions involve monotone non-decreasing sequences. Indeed, as noted above the starting sequence C_0 is monotone non-decreasing. Convolving it with itself produces a monotone non-decreasing sequence again, and the clipping in line 5 of Algorithm 1 preserves monotonicity. The same argument applies for further iterations. Thus, the running time of Algorithm 1 is $O(n + T(\Delta) \log W)$, where $T(\Delta)$ is the running time to compute BMMaxConv on sequences of length Δ.

Regarding correctness, we claim the following:

▷ Claim 14. For every $i \in [k]$ and every index $j \in [W \cdot 2^{i-k} \pm 8\Delta] \cap [W]$ the following holds:
- If $\mathcal{P}_i[j] \in [\alpha \cdot 2^{i-k} - 40\Delta, \alpha \cdot 2^{i-k} + 24\Delta]$, then $C_i[j] = \mathcal{P}_i[j]$.
- If $\mathcal{P}_i[j] > \alpha \cdot 2^{i-k} + 24\Delta$, then $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$.
- If $\mathcal{P}_i[j] < \alpha \cdot 2^{i-k} - 40\Delta$, then $C_i[j] = -\infty$.

Intuitively, the claim says that entries "close" to the (scaled) guess $\alpha \cdot 2^{i-k}$ get computed exactly, while entries below and above get clipped appropriately.

Proof. We prove the claim by induction on i. In the base case $i = 0$, note that since $\alpha \in [p_{\max} \cdot W]$ and $k = \lceil \log W \rceil$ we have $\alpha \cdot 2^{-k} \leq p_{\max} \leq \Delta$. Thus, $[\alpha \cdot 2^{-k} - 40\Delta, \alpha \cdot 2^{-k} + 24\Delta]$ contains the whole interval $[0, \Delta]$ of possible values of $\mathcal{P}_0[j] = C_0[j]$ (for any $0 \leq j \leq 8\Delta$).

Now we show that the claim holds for any $1 \leq i \leq k$ assuming it holds for $i - 1$. Fix any $j \in [W \cdot 2^{i-k} \pm 8\Delta]$. Note that the thresholding in line 4 of Algorithm 1 does not increasy any of the entries in A_i, so $C_i[j] \leq A_i[j]$. Moreover, since inductively $C_{i-1}[j'] \leq \mathcal{P}_{i-1}[j']$ for all j', by definition of A_i we have $A_i[j] \leq \mathcal{P}_i[j]$. Hence, we obtain $C_i[j] \leq \mathcal{P}_i[j]$. We use this observation to obtain the claim, by showing an appropriate lower bound for $C_i[j]$ in the following.

Pick indices j_1, j_2 as guaranteed by Lemma 13. Property (i) of Lemma 13 guarantees that the computation of $A_i[j]$ in line 3 of Algorithm 1 looks at the entries j_1, j_2 in C_{i-1}. Hence, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2]$.

We proceed by a case distinction on the values of the entries $\mathcal{P}_{i-1}[j_1]$ and $\mathcal{P}_{i-1}[j_2]$:

Case 1: $\mathcal{P}_{i-1}[j_1], \mathcal{P}_{i-1}[j_2] \in [\alpha \cdot 2^{i-1-k} - 40\Delta, \alpha \cdot 2^{i-1-k} + 24\Delta]$. By the induction hypothesis, both values are computed exactly, that is, $C_{i-1}[j_1] = \mathcal{P}_{i-1}[j_1]$ and $C_{i-1}[j_2] = \mathcal{P}_{i-1}[j_2]$. Thus, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] = \mathcal{P}_i[j]$, using property (ii) of Lemma 13. Since we observed above that $A_i[j] \leq \mathcal{P}_i[j]$, we obtain $A_i[j] = \mathcal{P}_i[j]$. The thresholding in line 4 of Algorithm 1 now yields the claim for this case.

Case 2: $\mathcal{P}_{i-1}[j_1] > \alpha \cdot 2^{i-1-k} + 24\Delta$. Property (iii) of Lemma 13 implies that $|\mathcal{P}_{i-1}[j_1] - \mathcal{P}_{i-1}[j_2]| \leq 4\Delta$, and hence $\mathcal{P}_{i-1}[j_2] \geq \alpha \cdot 2^{i-1-k} + 20\Delta$. Thus, property (ii) of Lemma 13 implies $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] > \alpha \cdot 2^{i-k} + 24\Delta$. Therefore, we want to show that $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$.

By the induction hypothesis, we have $C_{i-1}[j_1] = \lceil \alpha \cdot 2^{i-1-k} \rceil + 24\Delta$ and $C_{i-1}[j_2] \geq \alpha \cdot 2^{i-1-k} + 20\Delta$. Hence, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2] > \alpha \cdot 2^{i-k} + 40\Delta$. Due to the thresholding, we conclude that $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$, as desired.

Case 3: $\mathcal{P}_{i-1}[j_1] < \alpha \cdot 2^{i-k} - 40\Delta$. Similarly as in case 2, property (iii) of Lemma 13 implies that $\mathcal{P}_{i-1}[j_2] \leq \alpha \cdot 2^{i-1-k} - 36\Delta$. Thus, $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] < \alpha \cdot 2^{i-k} - 76\Delta$. Since $C_i[j] \leq \mathcal{P}_i[j]$, but $C_i[j]$ takes values in $\{-\infty\} \cup [\alpha \cdot 2^{i-k} \pm 40\Delta]$ it follows that $C_i[j] = -\infty$. ◁

Given the claim, it is easy to see that $C_k[W] \geq \alpha$ if and only if $\mathcal{P}_k[W] \geq \alpha$. Along with the running time analysis argued earlier, we obtain the following lemma.

▶ **Lemma 15.** *Algorithm 1 runs in time $O(n + T(\Delta) \log W)$, where $T(\Delta)$ is the time complexity of BMMaxConv on sequences of length Δ, and computes a value $C_k[W]$ which satisfies $C_k[W] \geq \alpha$ if and only if $\mathrm{OPT} = \mathcal{P}_k[W] \geq \alpha$.*

Given Lemma 15, we can do binary search to find the optimal value. This gives an algorithm for Unbounded Knapsack in time $O((n + T(\Delta)) \log W \log \mathrm{OPT})$. To shave the polylog$(W, \mathrm{OPT})$ factors and obtain the running time $\widetilde{O}(n + T(\Delta))$ claimed in Theorem 10, we make use of the following lemma. It allows us to reduce the capacity of the instance by repeatedly adding copies of the item with maximum profit-to-weight ratio. Similar results have been shown for general ILPs [20], for Unbounded Knapsack [5] and for the Coin Change problem [15]. For completeness, we include the proof by Chan and He [15, Lemma 4.1].[7]

▶ **Lemma 16.** *Let $(p_{i^*}, w_{i^*}) := \mathrm{argmax}_{(p,w) \in \mathcal{I}} \frac{p}{w}$. If $W \geq 2w_{\max}^3$, then there exists an optimal solution containing (p_{i^*}, w_{i^*}).*

Proof. Consider an optimal solution x that does not contain item (p_{i^*}, w_{i^*}). If there is an item (p_j, w_j) that appears at least w_{i^*} times in x, then we can replace w_{i^*} of the copies of item (p_j, w_j) by w_j copies of item (p_{i^*}, w_{i^*}). By definition of (p_{i^*}, w_{i^*}), this does not decrease the total profit of the solution, so by optimality of x the new solution x' is also optimal. Therefore, some optimal solution contains (p_{i^*}, w_{i^*}).

[7] Both Chan and He [15] and Bateni et al. [5] show that the same conclusion of the lemma holds if $W > w_{i^*}^2$, with a slightly more involved argument. For our purposes, this simple variant is enough.

It remains to consider the case that x contains less than w_{i^*} copies of every item, so its total weight is at most $n \cdot w_{i^*} \cdot w_{\max}$. Note that $n \leq w_{\max}$, because without loss of generality there is at most one item per distinct weight (otherwise we can keep only the item with the largest profit for each weight). Thus, the total weight of x is at most $w_{\max}^3$. It follows that $W < w_{\max}^3 + w_{i^*} \leq 2w_{\max}^3$, since otherwise we could add at least one copy of (p_{i^*}, w_{i^*}) to x, contradicting its optimality. ◀

We now put all pieces together: As a preprocessing step we repeatedly add the item (p_{i^*}, w_{i^*}) and decrease W by w_{i^*}, as long as $W > 2w_{\max}^3$. After this preprocessing, we have $W = O(w_{\max}^3) = O(\Delta^3)$, and thus $\mathrm{OPT} \leq W \cdot p_{\max} = O(\Delta^4)$. The we do binary search for OPT, using Algorithm 1 as a decision procedure. By Lemma 15, the overall running time is $O((n + T(\Delta)) \log^2 \Delta) = \widetilde{O}(n + T(\Delta))$. This completes the proof of Theorem 10.

3.3 Solution Reconstruction

The algorithm we described gives us the value OPT of the optimal solution. In this section we will describe how to use witness arrays (Lemma 9) to reconstruct a feasible solution $x \in \mathbb{N}^n$ such that $p(x) = \mathrm{OPT}$ with only a polylogarithmic overhead in the overall running time.

▶ **Lemma 17.** *A optimal solution x can be reconstructed in time $\widetilde{O}(n + T(p_{\max} + w_{\max}))$.*

Proof Sketch. Let $k = \lceil \log W \rceil$ be as in Algorithm 1. After determining the value of OPT, run Algorithm 1 again with the guess $\alpha = \mathrm{OPT}$. For every BMMaxConv in line 4 compute the witness array M_i corresponding to A_i via Lemma 9. This takes time $\widetilde{O}(n+T(p_{\max}+w_{\max}))$. Now, the idea is to start from $C_k[W]$ and traverse the *computation tree* of Algorithm 1 backwards. That is, we look at the pair of entries $C_{k-1}[M_k[W]], C_{k-1}[W - M_k[W]]$ which define the value of $C_k[W]$ and recursively obtain the pair of entries in C_{k-2} determining the value of $C_{k-1}[M_k[W]]$, etc. By proceeding in this way, we eventually hit the leaves, i.e., the entries of $C_0[0, \ldots, 8\Delta] = \mathcal{P}_0[0, \ldots, 8\Delta]$, which correspond to the items in an optimal solution. A naive implementation of this idea takes time $O(\sum_{i \leq k} 2^i) = O(2^k) = O(W)$, which is too slow.

Now we describe an efficient implementation of the same idea. For each $i \in [k]$ construct an array $Z_i[W \cdot 2^{i-k} \pm 8\Delta]$ initialized to zeros. Set $Z_k[W] := 1$. We will maintain the invariant that $Z_i[j]$ stores the number of times we arrive at $C_i[j]$ by traversing the computation tree starting at $C_k[W]$. This clearly holds for $Z_k[W] = 1$ by definition. Now we describe how to fill the entries for the levels below. Iterate over $i = k, k-1, \ldots, 1$. For each entry $j \in [W \cdot 2^{i-k} \pm 8\Delta] \cap [W]$ add $Z_i[j]$ to its witness entries in the level below, i.e., increase $Z_{i-1}[M_i[j]]$ by $Z_i[j]$ and $Z_{i-1}[j - M_i[j]]$ by $Z_i[j]$. The invariant is maintained by definition of the witnesses, and because Algorithm 1 guarantees that $M_i[j], j - M_i[j] \in [2 \cdot 2^{i-1-k} \pm 8\Delta] \cap [W]$. Note that this procedure takes time $O(k\Delta) = \widetilde{O}(\Delta)$.

Finally, note that for the base case we have that each $Z_0[j]$ for $j \in [8\Delta]$ counts the number of times that we hit the entry $C_0[j] = \mathcal{P}_0[j]$ in the computation tree starting from $C_k[W]$. Recall that by definition, $\mathcal{P}_0[j]$ is the maximum profit of an item in $\mathcal{I}$ with weight at most j. Hence, every entry $\mathcal{P}_0[j]$ corresponds to a unique item in $\mathcal{I}$. Therefore, we can read off from $Z_0[0, \ldots, 8\Delta]$ the multiplicity of each item included in an optimal solution. The overall time of the procedure is $\widetilde{O}(n + T(p_{\max} + w_{\max}))$, as claimed. ◀

4 Exact Algorithm for 0-1 Knapsack

Cygan et al. [19] showed the following reduction from 0-1 Knapsack to BMMaxConv:

▶ **Theorem 18** ([19, Theorem 13]). *If MaxConv on length-n sequences can be solved in time $T(n)$, then 0-1 Knapsack can be solved in time $O(T(W\log W)\log^3(n/\delta)\log n)$ with probability at least $1-\delta$.*

Their reduction is a generalization of Bringmann's algorithm for Subset Sum [8], replacing Boolean convolutions by $(\max,+)$-convolutions. We observe that essentially the same reduction yields sequences of length $O(W)$ which are monotone and have entries bounded by OPT. In particular, these are BMMaxConv instances. This yields the following:

▶ **Theorem 19** (0-1 Knapsack $\to$ BMMaxConv). *If BMMaxConv on length-n sequences can be solved in time $T(n)$, then 0-1 Knapsack can be solved in time $\widetilde{O}(n+T(W+\mathrm{OPT}))$ with high probability.*

Proof. The proof is virtually the same as [19, Theorem 13], so we omit some details. In particular, we emphasize how the constructed instances can be seen to be monotone and bounded, but we omit some details of the correctness argument. The idea of the algorithm is the following: split the item set $\mathcal{I}$ into groups $G_{(a,b)}\subseteq\mathcal{I}$ such that all items $(p,w)\in\mathcal{I}$ with $p\in[2^{a-1},2^a)$ and $w\in[2^{b-1},2^b)$ are in group $G_{(a,b)}$. That is, all items within each group have weights and profits within a factor of 2 of each other, and thus there are $O(\log W\log\mathrm{OPT})$ many groups. We will describe how to compute $\mathcal{P}_{G_{(a,b)}}[0,\dots,W]$ for each $G_{(a,b)}$. Having that, we simply combine all the profit arrays into $\mathcal{P}_{\mathcal{I}}[0,\dots,W]$ using $(\max,+)$-convolutions. Since we have $O(\log W\log\mathrm{OPT})$ groups, and each profit array is a monotone non-decreasing sequence of length W with entries bounded by OPT, the combination step takes time $\widetilde{O}(T(W+\mathrm{OPT}))$.

Fix some group $G_{(a,b)}$. Since every $(p,w)\in G_{(a,b)}$ has $w\in[2^{a-1},2^a)$ and $p\in[2^{b-1},2^b)$, any feasible solution from $G_{(a,b)}$ consists of at most $z:=\lceil\min\{W/2^{a-1},\mathrm{OPT}/2^{b-1}\}\rceil$ items. Thus, by splitting the items in $G_{(a,b)}$ randomly into z subgroups $G_{(a,b),1},\dots,G_{(a,b),z}$, any fixed feasible solution has at most $O(\log z)$ items in each subgroup $G_{(a,b),k}$ with high probability. To see this, fix a solution x and note that,

$$\mathbf{Pr}[\text{at least } r \text{ items from } x \text{ fall in } G_{(a,b),k}]\le\binom{z}{r}\left(\frac{1}{z}\right)^r\le\left(\frac{e\cdot z}{r}\right)^r\left(\frac{1}{z}\right)^r=\left(\frac{e}{r}\right)^r,$$

where the first inequality follows due to a union bound over all subsets of items of size r. By setting $r=O(\log z)$, we can bound this probability by z^{-c} for any constant c. So by a union bound, none of the z groups $G_{(a,b),1},\dots,G_{(a,b),z}$ has more than $\kappa:=O(\log z)$ elements from the fixed solution x with probability at least $1-1/\operatorname{poly}(z)$.

Therefore, to obtain the value of any fixed solution it suffices to compute the optimal solution consisting of at most κ items from $G_{(a,b),k}$ for every target weight $\le O(2^a\kappa)$, and then merge the results. More precisely, for every $1\le i\le z$ we compute the array $\mathcal{P}_{G_{(a,b),i},\log(\kappa)}[0,\dots,O(2^a\kappa)]$. Recall that this is defined as

$$\mathcal{P}_{G_{(a,b),i},\log(\kappa)}[j]:=\max\{p(x)\colon x \text{ is a solution from } G_{(a,b),i} \text{ with } w(x)\le j,\|x\|_1\le\kappa\}$$

for each $j\in[O(2^a\kappa)]$. For ease of notation, we denote the array by $\mathcal{P}_{G_i,\kappa}:=\mathcal{P}_{G_{(a,b),i},\log\kappa}$.

Then, we merge the $\mathcal{P}_{G_i,\kappa}$'s using $(\max,+)$-convolutions. Now we describe these two steps in more detail:

Computing $\mathcal{P}_{G_i,\kappa}[0,\dots,O(2^a\kappa)]$. Since we only care about solutions with at most κ items, we use randomization again[8]: split the items in $G_{(a,b),i}$ into κ^2 buckets $A_1,\dots,A_{\kappa^2}$. By the birthday paradox, with constant probability it holds that any fixed solution is shattered among the buckets, i.e., each bucket contains at most 1 item of the solution. Thus, for each bucket A_k we construct the array $\mathcal{P}_{A_k,0}[0,\dots,2^a]$. Recall that this is defined as

$$\mathcal{P}_{A_k,0}[j] := \max\{p(x)\colon x \text{ is a solution from } A_k \text{ with } w(x)\le j, \|x\|_1\le 1\}$$

for each entry $j\in[2^a]$. To combine the results, we compute $\mathcal{P}_{A_1,0}\oplus\mathcal{P}_{A_2,0}\oplus\cdots\oplus\mathcal{P}_{A_{\kappa^2},0}$. By definition, every $\mathcal{P}_{A_i,0}$ is a monotone non-decreasing sequence of length 2^a with entries bounded by 2^b. Thus, the merging step takes time $O(T((2^a+2^b)\cdot\kappa^2)\cdot\kappa^2)$.

Each entry of the resulting array has the correct value $\mathcal{P}_{i,\kappa}[j]$ with constant probability, since a corresponding optimal solution is shattered with constant probability. By repeating this process $O(\log z)$ times and keeping the entrywise maximum among all repetitions, we boost the success probability to $1-1/\operatorname{poly}(z)$. Thus, by a union bound over the z subgroups $G_{(a,b),1},\dots,G_{(a,b),z}$, we get that any z fixed entries $\mathcal{P}_{G_1,\kappa}[j_1],\dots,\mathcal{P}_{G_z,\kappa}[j_z]$ corresponding to a solution which is partitioned among the z subgroups get computed correctly with probability at least $1-1/\operatorname{poly}(z)$. This adds an extra $O(\log z)=O(\kappa)$ factor to the running time.

Merging $\mathcal{P}_{G_1,\kappa}\oplus\cdots\oplus\mathcal{P}_{G_z,\kappa}$. This computation is done in a binary tree-like fashion. That is, in the first level we compute $(\mathcal{P}_{G_1,\kappa}\oplus\mathcal{P}_{G_2,\kappa}),(\mathcal{P}_{G_3,\kappa}\oplus\mathcal{P}_{G_4,\kappa}),\dots,(\mathcal{P}_{G_{z-1},\kappa}\oplus\mathcal{P}_{G_z,\kappa})$. In the second level we merge the results from the first level in a similar way. We proceed in the same way with further levels. Since we merge z sequences, we have $\lceil\log z\rceil$ levels of computation. In the j-th level, we compute the $(\max,+)$-convolution of $z/2^j$ many monotone non-decreasing sequences of length $O(2^j\cdot 2^a\cdot\kappa)$ with entries bounded by $O(2^j\cdot 2^b\cdot\kappa)$. Therefore, overall the merging takes time

$$O\Big(\sum_{j=1}^{\lceil\log z\rceil}\frac{z}{2^j}\cdot T((2^a+2^b)\cdot 2^j\cdot\kappa)\Big)\le\widetilde{O}(T((2^a+2^b)\cdot z)),$$

where we used both of our niceness assumptions $k\cdot T(n)\le O(T(k\cdot n))$ for any $k>1$ and $T(\widetilde{O}(n))\le\widetilde{O}(T(n))$. Since $z=\lceil\min\{W/2^{a-1},\mathrm{OPT}/2^{b-1}\}\rceil$, we have $\widetilde{O}(T((2^a+2^b)\cdot z)=\widetilde{O}(T(W+\mathrm{OPT}))$.

Wrapping up. To recap, the algorithm does the following steps:

1. Split the items into $O(\log W\log\mathrm{OPT})$ groups $G_{(a,b)}$. This takes time $O(n)$.
2. Randomly split each group $G_{(a,b)}$ into $z:=\lceil\min\{W/2^{a-1},\mathrm{OPT}/2^{b-1}\}\rceil$ subgroups $G_{(a,b),i}$ for $i\in[z]$.
3. For each $G_{(a,b),i}$ compute the array $\mathcal{P}_{G_i,\kappa}[0,\dots,O(2^a\kappa)]$ in time $O(T((2^a+2^b)\kappa^2)\cdot\kappa^3)$. Since $\kappa=O(\log z)$, the total time over all $i\in[z]$ is

   $$O(z\cdot T((2^a+2^b)\kappa^2)\cdot\kappa^3)\le O(T((2^a+2^b)\cdot z\cdot\kappa^2)\kappa^3)\le\widetilde{O}(T((2^a+2^b)\cdot z))\le\widetilde{O}(T(W+\mathrm{OPT})).$$

   Note that here we use the niceness assumptions on $T(n)$. In particular, first we used that $k\cdot T(n)\le O(T(k\cdot n))$ for any $k>1$, and then that $T(\widetilde{O}(n))\le\widetilde{O}(T(n))$.
4. Merge the arrays $\mathcal{P}_{G_1,\kappa}\oplus\cdots\oplus\mathcal{P}_{G_z,\kappa}$ in time $\widetilde{O}(T(W+\mathrm{OPT}))$ to obtain $\mathcal{P}_{G_{(a,b)}}[0,\dots,W]$.
5. Merge the arrays $\mathcal{P}_{G_{(a,b)}}$ using $O(\log W\log\mathrm{OPT})$ convolutions in total time $\widetilde{O}(T(W+\mathrm{OPT}))$.

[8] This step is called "Color Coding" in [8, 19].

Thus, the overall time of the algorithm is $\widetilde{O}(n+T(W+\mathrm{OPT}))$. Note that as mentioned earlier in the proof, the algorithm succeeds in computing any *fixed* entry $\mathcal{P}_{\mathcal{I}}[j]$ with probability at least $1-1/\operatorname{poly}(z)$. In particular, this is sufficient to compute the optimal solution $\mathcal{P}_{\mathcal{I}}[W]$ with good probability. As described, the algorithm only returns the value of the optimal solution. In the full version of the paper, we show how to reconstruct an optimal solution. ◀

References

1. Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. Witnesses for Boolean matrix multiplication and for shortest paths. In *FOCS*, pages 417–426. IEEE Computer Society, 1992.
2. Noga Alon and Moni Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4/5):434–449, 1996.
3. Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster Knapsack and graph algorithms. In *ICALP*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
4. Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. Better approximations for tree sparsity in nearly-linear time. In *SODA*, pages 2215–2229. SIAM, 2017.
5. MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for Knapsack via convolution and prediction. In *STOC*, pages 1269–1282. ACM, 2018.
6. Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.
7. David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014.
8. Karl Bringmann. A near-linear pseudopolynomial time algorithm for Subset Sum. In *SODA*, pages 1073–1084. SIAM, 2017.
9. Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM J. Comput.*, 48(2):481–512, 2019.
10. Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating Subset sum and Partition. In *SODA*, pages 1797–1815. SIAM, 2021.
11. Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *SODA*, pages 1777–1796. SIAM, 2021.
12. Michael R. Bussieck, Hannes Hassler, Gerhard J. Woeginger, and Uwe T. Zimmermann. Fast algorithms for the maximum convolution problem. *Oper. Res. Lett.*, 15(3):133–141, 1994.
13. Timothy M. Chan. Approximation schemes for 0-1 Knapsack. In *SOSA*, volume 61 of *OASICS*, pages 5:1–5:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
14. Timothy M. Chan and Sariel Har-Peled. Smallest k-enclosing rectangle revisited. In *SoCG*, volume 129 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
15. Timothy M. Chan and Qizheng He. More on change-making and related problems. *J. Comput. Syst. Sci.*, 124:159–169, 2022.
16. Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via Additive Combinatorics. In *STOC*, pages 31–40. ACM, 2015.
17. Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021.
18. Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC*. ACM, 2022.
19. Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019.

20 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020.

21 Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991.

22 V. S. Grinberg and S. V. Sevastyanov. Value of the Steinitz constant. *Funktsional. Anal. i Prilozhen.*, 14(2):56–57, 1980.

23 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the Knapsack and Sum of Subset problems. *J. ACM*, 22(4):463–468, 1975.

24 Klaus Jansen and Stefan Erich Julius Kraft. A faster FPTAS for the Unbounded Knapsack problem. *Eur. J. Comb.*, 68:148–174, 2018.

25 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *ITCS*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

26 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for Subset Sum. In *SOSA*, volume 69 of *OASICS*, pages 17:1–17:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

27 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004.

28 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *ICALP*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

29 Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. On lower bounds for the maximum consecutive subsums problem and the (min, +)-convolution. In *ISIT*, pages 1807–1811. IEEE, 2014.

30 Eugene L. Lawler. Fast approximation algorithms for Knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979.

31 Jiří Matoušek. *Thirty-three miniatures: Mathematical and Algorithmic applications of Linear Algebra*. American Mathematical Society Providence, RI, 2010.

32 Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. A subquadratic approximation scheme for Partition. In *SODA*, pages 70–88. SIAM, 2019.

33 David Pisinger. Linear time algorithms for Knapsack problems with bounded weights. *J. Algorithms*, 33(1):1–14, 1999.

34 Adam Polak, Lars Rohwedder, and Karol Wegrzycki. Knapsack and Subset Sum with small items. In *ICALP*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

35 Raimund Seidel. On the All-Pairs-Shortest-Path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

36 Ernst Steinitz. Bedingt konvergente Reihen und konvexe Systeme. *Journal für die reine und angewandte Mathematik*, 143:128–176, 1913. URL: `http://eudml.org/doc/149403`.

37 R. Ryan Williams. Faster All-Pairs Shortest Paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018.

A Missing Proofs from Section 2

Proof of Proposition 7. By shifting the indices, we can assume that $A[I]$ and $B[J]$ are sequences $A'[0,\dots,|I|-1]$ and $B'[0,\dots,|J|-1]$. Compute $C' := A' \oplus B'$ in time $T(|A|+|B|)$. By shifting the indices back, we can infer the values of the entries $C[I+J] = A[I] \oplus B[J]$. Thus, we can simply read off the entries in $C[K]$ from the array C'. ◀

Equivalence between variants of BMMaxConv

As stated in Section 2, the algorithm of Chi, Duan, Xie and Zhang [18] computes the $(\min,+)$-convolution of sequences which are monotone increasing and have values in $[O(n)]$. The following proposition shows that this is equivalent to solving MaxConv on monotone non-decreasing sequences with values in $[O(n)] \cup \{-\infty\}$, which justifies Theorem 8.

▶ **Proposition 20.** *MaxConv on monotone non-decreasing sequences of length n and values in $[O(n)] \cup \{-\infty\}$ is equivalent to MinConv on monotone increasing sequences of length n and values in $[O(n)]$, in the sense that if one can be solved in time $T(n)$ then the other can be solved in time $O(T(n))$.*

Proof. We first describe how to reduce MaxConv on monotone non-decreasing sequences and values in $[O(n)] \cup \{-\infty\}$ to MinConv on monotone increasing sequences and values in $[O(n)]$ via a simple chain of reductions:

- *Removing $-\infty$:* Let $A[0,\dots,n], B[0,\dots,n]$ be an instance of MaxConv where A, B are monotone non-decreasing and $A[i], B[i] \in [O(n)] \cup \{-\infty\}$. We start by reducing it to an equivalent instance of MaxConv on monotone non-decreasing sequences and values in $[O(n)]$ (i.e. we remove the $-\infty$ entries). Let Δ be the maximum entry of A and B. Construct a new sequence $A'[0,\dots,n]$ where $A'[i] := 0$ if $A[i] = -\infty$, and $A'[i] := A[i] + 2\Delta$ otherwise. Construct $B'[0,\dots,n]$ from B in the same way. Note that A' and B' are monotone non-decreasing and have values in $[O(n)]$. Moreover, we can infer the values of any entry $(A \oplus B)[k]$ from C': if $C'[k] \le 3\Delta$ then $(A \oplus B)[k] = -\infty$ and otherwise $(A \oplus B)[k] = C'[k] - 4\Delta$.
- *Reducing to MaxConv on non-increasing sequences:* Now we reduce an instance $A[0,\dots,n]$, $B[0,\dots,n]$ of MaxConv on monotone non-decreasing sequences and values in $[O(n)]$ to an instance of MinConv on monotone non-increasing sequences and values in $[O(n)]$. Let Δ be the maximum entry of A and B. Construct two new sequences A' and B' by setting $A'[i] := \Delta - A[i]$ and $B'[i] := \Delta - B[i]$. Then A' and B' are monotone non-increasing and given their $(\min,+)$-convolution we can easily infer the $(\max,+)$-convolution of A and B.
- *Reducing to MinConv on increasing sequences:* Next, we reduce an instance $A[0,\dots,n]$, $B[0,\dots,n]$ of MinConv on monotone non-increasing sequences and values in $[O(n)]$ to an instance of MinConv on increasing sequences and values in $[O(n)]$. Construct two new sequences A' and B' by reversing and adding a linear function to A and B, i.e., set $A'[i] := A[n-i] + i$ and $B'[i] := B[n-i] + i$ for every $i \in [n]$. Note that A' and B' are monotone increasing sequences, and given their $(\min,+)$-convolution we can infer the $(\min,+)$-convolution of A and B.

Combining the reductions above, we conclude that MaxConv on monotone non-decreasing sequences with values in $[O(n)] \cup \{-\infty\}$ can be reduced in linear time to MinConv on monotone increasing sequences with values in $[O(n)]$. To show the reduction in the other direction, we can apply the same ideas: we first negate the entries and shift them to make them non-negative, then reverse the resulting sequences. We omit the details. ◀

B Witnesses for BMMaxConv

In this section we give the proof of Lemma 9. For the remainder of this section, fix an instance $A[0,\dots,n], B[0,\dots,n]$ of BMMaxConv and let $C := A \oplus B$. To compute the witness array $M[0,\dots,2n]$, we first show that if an entry $C[k]$ has a unique witness then

we can easily find it. Then we reduce to the unique witness case with randomization. This approach is well known [35, 1], but some extra care is needed to ensure that the instances remain monotone and bounded. The standard derandomization for this approach [2] also works in our setting.

▶ **Lemma 21.** *If BMMaxConv on length-n sequences can be computed in time $T(n)$, then in time $\widetilde{O}(T(n))$ we can compute an array $U[0,\dots,2n]$ such that for every $k \in [2n]$ if $C[k]$ has a unique witness $M[k]$ then $U[k] = M[k]$. The output $U[k]$ is undefined otherwise.*

Proof. We will describe how to compute the unique witnesses bit by bit. For each bit position $b \in [\lceil \log n \rceil]$ let $i_b \in \{0,1\}$ be the b-th bit of i. Construct sequences A_b, B_b defined as $A_b[i] := 2A[i] + i + i_b$ and $B_b[i] := 2B[i] + i$ for $i \in [n]$. Note that A_b, B_b are still monotone non-decreasing and have entries bounded by $O(n)$. Compute $C_b := A_b \oplus B_b$. For each $k \in [2n]$, we set the b-th bit of $U[k]$ to $C_b[k] \bmod 2$. It is not hard to see that for those entries $C[k]$ which have unique witnesses $U[k]$, this procedure indeed gives the b-th bit of $U[k]$. Indeed, note that because we double every entry in A_b, B_b and add a linear function, adding i_b does not change the maximizer. Therefore, if $C[k]$ has a unique witness then $C'[k]$ has a unique witness whose b-th bit can be read from the least significant bit of $C'[k]$. Thus, by repeating this over all bit positions $b \in [\lceil \log n \rceil]$, we compute the entire array of unique witnesses U using $O(\log n)$ invocations to BMMaxConv, as desired. ◀

Proof of Lemma 9. Fix a set $S \subseteq [n]$. We say that an entry $C[k]$ gets *isolated* by S if the number of witnesses of $C[k]$ in S is exactly one. We will now describe how to find the witnesses of all entries isolated by S (the idea and argument is similar as in the proof of Lemma 21). Construct sequences A', B' where for each $i \in [n]$ we set

$$A'[i] := \begin{cases} 2A[i] + i + 1 & \text{if } i \in S \\ 2A[i] + i & \text{otherwise} \end{cases}$$

and $B'[i] := 2B[i] + i$. Note that these sequences are monotone non-decreasing and have entries bounded by $O(n)$. Let $C' := A' \oplus B'$. We claim that if an entry $C[k]$ is isolated by S, then $C'[k]$ has a unique witness. To see this, note that if no witness of $C[k]$ gets included in S, then we have that $C'[k] = 2C[k] + k$. If at least one witness gets included in S, then $C'[k] = 2C[k] + k + 1$. In particular, if a witness gets isolated then $C'[k]$ will have a unique witness, as claimed. Thus, by Lemma 21 we can compute in time $\widetilde{O}(T(n))$ an array $U[0,\dots,2n]$ which contains the witnesses of all entries that are isolated by S. Note that some entries of U might be undefined, but we can simply check in time $O(n)$ which entries of U are true witnesses, by iterating over $k \in [2n]$ and checking whether the equality $C[k] = A[U[k]] + B[k - U[k]]$ holds.

Now we show how to select appropriate sets S. Fix an entry $C[k]$ and denote by $R \in [n]$ its number of witnesses. We sample $S \subseteq [n]$ by including each element $i \in [n]$ independently with probability $p := 2^{-\alpha}$ where $\alpha \in \mathbb{N}$ is chosen such that $2^{\alpha-2} \le R \le 2^{\alpha-1}$. Let X be the random variable counting the number of witnesses of $C[k]$ that get sampled in S. By keeping the first two terms in the inclusion-exclusion formula we have that $\mathbf{Pr}[X \ge 1] \ge p \cdot R - \binom{R}{2} p^2$, and by a union bound $\mathbf{Pr}[X \ge 2] \le \binom{R}{2} p^2$. Thus,

$$\mathbf{Pr}[X = 1] = \mathbf{Pr}[X \ge 1] - \mathbf{Pr}[X \ge 2] \ge p \cdot R(1 - p \cdot R) \ge 1/8$$

where the last inequality holds because $1/8 \le p \cdot R \le 1/4$ due to the choice of α. In particular, S isolates $C[k]$ with probability at least $1/8$.

We now put the pieces together. Iterate over the $O(\log n)$ possible values for α. Sample a set S and find all witnesses of entries isolated by S as described earlier in time $\widetilde{O}(T(n))$. As we argued above, if $C[k]$ has R witnesses and $2^{\alpha-2} \leq R \leq 2^{\alpha-1}$, then $C[k]$ gets isolated with constant probability. Thus, by repeating this step with the same α for $O(\log n)$ freshly sampled sets S we find a witness for all such entries $C[k]$ with probability at least $1-1/\operatorname{poly}(n)$. Combining the results across iterations we obtain the array of witnesses $M[0,\ldots,2n]$ in time $\widetilde{O}(T(n))$, as desired.

Finally, we note that this procedure can be derandomized with standard techniques [2]. ◀

Improved Sublinear-Time Edit Distance for Preprocessed Strings

Karl Bringmann
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Alejandro Cassis
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Nick Fischer
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Vasileios Nakos
RelationalAI, Berkeley, CA, USA

Abstract

We study the problem of approximating the edit distance of two strings in sublinear time, in a setting where one or both string(s) are preprocessed, as initiated by Goldenberg, Rubinstein, Saha (STOC '20). Specifically, in the (k, K)-*gap edit distance* problem, the goal is to distinguish whether the edit distance of two strings is at most k or at least K. We obtain the following results:

- After preprocessing *one* string in time $n^{1+o(1)}$, we can solve $(k, k \cdot n^{o(1)})$-gap edit distance in time $(n/k + k) \cdot n^{o(1)}$.
- After preprocessing *both* strings separately in time $n^{1+o(1)}$, we can solve $(k, k \cdot n^{o(1)})$-gap edit distance in time $kn^{o(1)}$.

Both results improve upon some previously best known result, with respect to either the gap or the query time or the preprocessing time.

Our algorithms build on the framework by Andoni, Krauthgamer and Onak (FOCS '10) and the recent sublinear-time algorithm by Bringmann, Cassis, Fischer and Nakos (STOC '22). We replace many complicated parts in their algorithm by faster and simpler solutions which exploit the preprocessing.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Edit Distance, Property Testing, Preprocessing, Precision Sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.32

Category Track A: Algorithms, Complexity and Games

Funding This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

The *edit distance* (also known as *Levenshtein distance*) is a fundamental measure of similarity between strings. It has numerous applications in several fields such as information retrieval, computational biology and text processing. Given strings X and Y, their edit distance denoted by $\mathsf{ED}(X, Y)$ is defined as the minimum number of character insertions, deletions and substitutions needed to transform X into Y.

Table 1 A comparison of sublinear-time algorithms for the $(k, k \cdot g)$-gap edit distance problem for different gap parameters g. All algorithms in this table are randomized and succeed with high probability. Note that some of these results are subsumed by others.

Source	Gap g	Preprocessing time	Query time
Goldenberg, Krauthgamer, Saha [20]	$O(k)$	no preprocessing	$\widetilde{O}(n/k + k^3)$
Kociumaka, Saha [24]	$O(k)$	no preprocessing	$\widetilde{O}(n/k + k^2)$
Brakensiek, Charikar, Rubinstein [13]	$O(k)$	no preprocessing	$\widetilde{O}(n/\sqrt{k})$
Bringmann, Cassis, Fischer, Nakos [15]	$O(k)$	no preprocessing	$O^*(n/k^2+k^8)$
Goldenberg, Kociumaka, Krauthgamer, Saha [19]	$O(k)$	no preprocessing	$\widetilde{O}(n/k^{3/2})$
Bringmann, Cassis, Fischer, Nakos [15]	$O^*(1)$	no preprocessing	$O^*(n/k + k^4)$
Goldenberg, Rubinstein, Saha [21]	$O(k)$	one-sided, $\widetilde{O}(n)$	$\widetilde{O}(n/k + k^2)$
Brakensiek, Charikar, Rubinstein [13]	g	one-sided, $\widetilde{O}(nk/g)$	$\widetilde{O}(n/g+k^2/g)$
This work, Theorem 1	$O^*(1)$	one-sided, $O^*(n)$	$O^*(n/k + k)$
Chakraborty, Goldenberg, Koucký [18]	$O(k)$	two-sided, $\widetilde{O}(n)$	$O(\log n)$
Brakensiek, Charikar, Rubinstein [13]	g	two-sided, $\widetilde{O}(nk/g)$	$\widetilde{O}(k^2/g)$
Ostrovsky, Rabani [26]	$O^*(1)$	two-sided, $\widetilde{O}(n^2)$	$O(\log n)$
This work, Theorem 2	$O^*(1)$	two-sided, $O^*(n)$	$O^*(k)$
Goldenberg, Rubinstein, Saha [21]	$O(1)$	two-sided, $\widetilde{O}(n^2)$	$O^*(n^{3/2})$
Goldenberg, Rubinstein, Saha [21]	1	two-sided, $\widetilde{O}(n)$	$\widetilde{O}(k^2)$

A textbook dynamic programming algorithm computes the edit distance of two strings of length n in time $O(n^2)$. Popular conjectures such as the *Strong Exponential Time Hypothesis* imply that that this algorithm is essentially optimal, as there is no strongly subquadratic-time algorithm [8, 1, 16, 2]. As for some applications involving enormous strings (such as DNA sequences) quadratic-time algorithms are impractical, a long line of research developed progressively better and faster *approximation* algorithms [9, 11, 26, 7, 4, 17, 25, 14]. The current best approximation guarantee in near-linear time is an algorithm by Andoni and Nosatzki [6] computing an $f(1/\varepsilon)$-approximation in time $O(n^{1+\varepsilon})$.

Another more recent line of research studies edit distance in the *sublinear-time* setting. Here the goal is to approximate the edit distance without reading the entire input strings. More formally, in the (k, K)-gap edit distance problem the goal is to distinguish whether the edit distance between X and Y is at most k or greater than K. The performance of gap algorithms is typically measured in terms of the string length n and the gap parameters k and K. This problem has been studied in several works [10, 7, 20, 13, 24] most of which focus on the (k, k^2)-gap problem. Currently, there are two incomparable best known results: A recent result by Goldenberg, Kociumaka, Krauthgamer and Saha [19] established a non-adaptive algorithm for the (k, k^2)-gap problem in time $\widetilde{O}(n/k^{3/2})$.[1] Another recent result by Bringmann, Cassis, Fischer and Nakos [15] reduces the gap to $O^*(1)$ and solves the $(k, O^*(k))$-gap problem in time $O^*(n/k+k^4)$.[2] See Table 1 for a more detailed comparison.

Our starting point is the work by Goldenberg, Rubinstein and Saha [21] which studies sublinear algorithms for edit distance in the *preprocessing model*. Here, we are allowed to preprocess one or both input strings X and Y *separately*, and then use the precomputed

[1] We write $\widetilde{O}(\cdot)$ to hide polylogarithmic factors $(\log n)^{O(1)}$.
[2] We write $O^*(\cdot)$ to hide subpolynomial factors $n^{o(1)}$ in n.

information to solve the (k, K)-gap edit distance problem. This model is motivated by applications where many long strings are compared against each other. For example, the *string similarity join* problem is to find all pairs of strings in a database (containing e.g. DNA sequences) which are close in edit distance; see [27] for a survey on practically relevant algorithms. Note that in these applications, if we have an algorithm with almost-linear preprocessing time (which is the case for all the algorithms we present in this paper), then the overhead incurred by preprocessing is comparable to the time necessary to read and store the strings in the first place. In [21], the authors pose and investigate the following open question:

> *"What is the complexity of approximate edit distance with preprocessing when $k \ll n$?" [21]*

This question has spawned significant interest in the community [18, 21, 13], and with this paper we also make progress towards this question. We give an overview of results in Table 1. Note that most results are hard to compare to each other (one-sided versus two-sided preprocessing, exact versus $O(1)$-approximate versus $O(k)$-approximate).

In the two-sided model, all known algorithms (with almost-linear preprocessing time[3] and, say, subpolynomial gap $g = n^{o(1)}$) share the common barrier that the query time is $\Omega(k^2)$. Due to this barrier, Goldenberg et al. [21] specifically ask whether there exists an approximation algorithm with sub-k^2 query time. One of our contributions is that we answer this question in the affirmative.

Our Results. We develop sublinear-time algorithms for the $(k, O^*(k))$-gap edit distance problem, in the one-sided and two-sided preprocessing model, respectively.

▶ **Theorem 1** (One-Sided Preprocessing). *Let X, Y be length-n strings. After preprocessing Y in time $O^*(n)$, we can solve the $(k, k \cdot n^{o(1)})$-gap edit distance problem for X and Y in time $O^*(n/k + k)$ with high probability.*

In comparison to the (k, k^2)-gap algorithms from [21, 13] with best query time[4] $\widetilde{O}(n/k+k)$, we contribute the following improvement: Ignoring lower-order factors, we reduce the gap from k to $O^*(1)$ while achieving the same query time $O^*(n/k+k)$ and the same preprocessing time $O^*(n)$. In comparison to the $(k, O^*(k))$-gap algorithm in time $O^*(n/k + k^4)$ from [15], we achieve the same gap but an improved query time for large k, at the cost of preprocessing one of the strings.

In the two-sided model, we obtain an analogous result, where the query time no longer depends on n/k.

▶ **Theorem 2** (Two-Sided Preprocessing). *Let X, Y be length-n strings. After preprocessing both X and Y (separately) in time $O^*(n)$, we can solve the $(k, k \cdot n^{o(1)})$-gap edit distance problem for X and Y in time $O^*(k)$ with high probability.*

We remark that all hidden factors in both theorems are $2^{\widetilde{O}(\sqrt{\log n})}$. For a detailed comparison of this algorithm to the previously known results, see Table 1. We point out that Theorem 2 settles the open question from [21] whether there exists an edit distance approximation algorithm with small gap and sub-k^2 query time.

[3] Here we insist on almost-linear preprocessing time since the celebrated embedding of edit distance into the ℓ_1-metric with distortion $n^{o(1)}$ due to Ostrovsky and Rabani [26] achieves query time $O(\log n)$ but requires preprocessing time $\Omega(n^2)$.

[4] For the (k, k^2)-gap problem, the running time bounds $\widetilde{O}(n/k + k)$ and $\widetilde{O}(n/k)$ can be considered equal, as for $k \geq \sqrt{n}$ the algorithm may return a trivial answer.

Our Techniques. To achieve our results we build on the recent sublinear-time algorithm by Bringmann, Cassis, Fischer and Nakos [15], which itself builds on an almost-linear-time algorithm by Andoni, Krauthgamer and Onak [4]. The basic idea of the original algorithm is to split the strings into several smaller parts and recur on these parts with non-uniform precisions. The idea of [15] is to prune branches in the recursion tree, by detecting and analyzing periodic substructures. Towards that, they designed appropriate property testers to efficiently detect these structures. In our setting, we observe that having preprocessed the string(s), we can prune the computation tree much more easily. Thus, our algorithm proceeds in the same recursive fashion as [15] and uses a similar set of techniques, but due to the preprocessing it turns out to be simpler and faster.

Further Related Work. In the previous comparison about sublinear-time algorithms, we left out streaming and sketching algorithms [12, 18, 9] and document exchange protocols [23, 12, 22].

Future Directions. There are several interesting directions for future work. We specifically mention two open problems.

1. *Constant gap?* As Table 1 shows, so far no constant-gap sublinear-time algorithm is known. Maybe the one-sided preprocessing setting is more approachable for this challenge. We believe that our approach is hopeless to achieve a constant gap, since we borrow from the recursive decomposition introduced in [4] which inherently incurs a polylogarithmic overhead in the approximation factor.
2. *Improving the query time?* The well-known $\Omega(n/K)$ lower bound against the (k, K)-gap *Hamming distance* problem (and therefore against edit distance) continues to hold in the one-sided preprocessing setting. In particular, the most optimistic hope is an algorithm with query time $O^*(n/k)$ for the $(k, O^*(k))$-gap edit distance problem. Can this be achieved or is the extra $+k$ in the query time of Theorem 1 necessary? For two-sided preprocessing, to the best of our knowledge no lower bound is known.

2 Preliminaries

We set $[\,i\,..\,j\,] = \{\,i, i+1, \ldots, j-1\,\}$ (in particular, $[\,i\,..\,i\,] = \emptyset$) and $[\,j\,] = [\,0\,..\,j\,]$. We say that an event happens *with high probability* if it happens with probability at least $1 - 1/\operatorname{poly}(n)$, where the degree of the polynomial can be an arbitrary constant. We write $\operatorname{poly}(n) = n^{O(1)}$ and $\widetilde{O}(n) = n(\log n)^{O(1)}$.

Let X, Y be strings over an alphabet Σ with polynomial size. We denote by $|X|$ the length of X. We denote by $X \circ Y$ the *concatenation* of X and Y. We denote by $X[\,i\,]$ the i-th character in X starting with index zero. We denote by $X[\,i\,..\,j\,]$ the substring of X with indices in $[\,i\,..\,j\,]$, that is, including i and excluding j. For out-of-bounds indices we set $X[\,i\,..\,j\,] = X[\,\max(i,0)\,..\,\min(j,|X|)\,]$. If X and Y have the same length, we define their *Hamming distance* $\mathsf{HD}(X,Y)$ as the number of non-matching characters $\mathsf{HD}(X,Y) = |\{\,i : X[\,i\,] \neq Y[\,i\,]\,\}|$. For two strings X, Y with possibly different lengths, we define their *edit distance* $\mathsf{ED}(X,Y)$ as the smallest number of character *insertions*, *deletions* and *substitutions* necessary to transform X into Y. An *optimal alignment* between X and Y is a monotonically non-decreasing function $A : \{\,0, \ldots, |X|\,\} \to \{\,0, \ldots, |Y|\,\}$ such that $A(0) = 0$, $A(|X|) = |Y|$ and

$$\mathsf{ED}(X,Y) = \sum_{i=0}^{|X|-1} \mathsf{ED}(X[\,i\,], Y[\,A(i)\,..\,A(i+1)\,]).$$

It is easy to see that there is an optimal alignment between any two strings X, Y: Trace an optimal path through the standard dynamic program for edit distance and assign $A(i)$ to the smallest j for which the path crosses (i, j).

Let T be a rooted tree, and let v be a node in T. We denote by $\mathsf{root}(T)$ the root node in T. We denote by $\mathsf{parent}(v)$ the parent node of v. We denote by $\mathsf{depth}(v)$ the length of the root-to-v path, and by $\mathsf{height}(v)$ the length of the longest v-to-leaf path.

3 Overview

3.1 A Linear-Time Algorithm à la Andoni-Krauthgamer-Onak

We start to outline an almost-linear-time algorithm to approximate the edit distance of two strings X, Y following the framework of the Andoni-Krauthgamer-Onak algorithm [4], with some changes as in [15] and some additional modifications (see the novel trick outlined at the end of this subsection).

First Ingredient: A Divide-and-Conquer Scheme. The basic idea of the algorithm is to apply a divide-and-conquer scheme to reduce the approximation of the global edit distance to approximating the edit distance of several smaller strings. The straightforward idea of partitioning both strings X, Y into parts $X_1, \dots, X_m, Y_1, \dots, Y_m$ and computing the edit distances $\mathsf{ED}(X_i, Y_i)$ does not immediately work; instead we need to consider *several shifts* of the string Y_i. We remark that this concept of recurring on smaller strings for several shifts is quite standard in previous work. The following lemma uses the same ideas as the "$\mathcal{E}$-distance" defined in [4]. We give a proof in Appendix A.

▶ **Lemma 3** (Divide and Conquer). *Let X, Y be length-n strings, and let $0 = j_0 < \cdots < j_B = n$. We write $X_i = X[\, j_{i-1} \,..\, j_i \,]$ and $Y_{i,s} = Y[\, j_{i-1} + s \,..\, j_i + s \,]$.*

- *For all shifts $s_1, \dots, s_B$ we have that $\mathsf{ED}(X, Y) \le \sum_i \mathsf{ED}(X_i, Y_{i,s_i}) + 2|s_i|$.*
- *There are shifts $s_1, \dots, s_B$ with $\sum_i \mathsf{ED}(X_i, Y_{i,s}) \le 2\mathsf{ED}(X, Y)$ and $2|s_i| \le \mathsf{ED}(X, Y)$ for all i.*

To explain how to apply Lemma 3, we first specify on which substrings our algorithm is supposed to recur. To this end, let T be a balanced B-ary tree with n leaves. T will act as the "recursion tree" of the algorithm. For a string X of length n, we define a substring X_v for every node v in T as follows: If the subtree below X_v spans from the i-th to the j-th leaf (ordered from left to right), then we set $X_v = X[\, i \,..\, j+1 \,]$. In particular, X_v is a single character for each leaf v, and $X_{\mathsf{root}(T)} = X$. We further define $X_{v,s} = X[\, i+s \,..\, j+s+1 \,]$. For concreteness, we set $B = 2^{\sqrt{\log n \log\log n}} = n^{o(1)}$ throughout the paper.

A Simple Algorithm. Based on Lemma 3, we next present a simple (yet slow) algorithm. Our goal is to compute, for each node v in the tree T, an approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ of $\mathsf{ED}(X_v, Y_{v,s})$ for all shifts s. The result at the root node is returned as the desired approximation of $\mathsf{ED}(X, Y)$. The algorithm works as follows: For each leaf we can cheaply compute $\mathsf{ED}(X_v, Y_{v,s})$ exactly by comparing the single characters X_v and $Y_{v,s}$. For each internal node with children $v_1, \dots, v_B$ we compute

$$\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) = \sum_{i=1}^{B} \min_{s_i \in \mathbf{Z}} \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i|. \tag{1}$$

A careful application of Lemma 3 shows that if the recursive approximations $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i})$ have multiplicative error at most α, then by approximating $\mathsf{ED}(X_v, Y_{v,s})$ as in (1) the multiplicative error becomes $2\alpha + B$. Since we repeat this argument recursively up to depth $\mathsf{depth}(T) \leq \log_B(n)$, the multiplicative error accumulates to $B \cdot \exp(O(\log_B(n))) = n^{o(1)}$.

This simple algorithm achieves the desired approximation quality, however, it is not fast enough: For every node v we have to compute $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ for too many shifts s (naively speaking, for up to n shifts). As a first step towards dealing with this issue, we first show that at every node v we can in fact tolerate a certain *additive* error (in addition to the multiplicative error discussed before) using a technique called *Precision Sampling*. Then we exploit the freedom of additive errors to run this algorithm for a *restricted* set of shifts s.

Second Ingredient: Precision Sampling. It ultimately suffices to compute an approximation of $\mathsf{ED}(X, Y)$ with *additive* error k in order to solve the constant-gap edit distance problem. We leverage this freedom to also solve the recursive subproblems up to some additive error. Specifically, we will work with the following data structure:

▶ **Definition 4** (Precision Tree). *Let T be a balanced B-ary tree with n leaves. For $t \in \mathbf{N}$, we randomly associate a* tolerance t_v *to every node v in T as follows:*

- *If v is the root, then set $t_v = t$;*
- *otherwise set $t_v = t_{\mathsf{parent}(v)} \cdot u_v/3$, where we sample $u_v \sim \mathrm{Exp}(O(\log n))$ (the exponential distribution with parameter $O(\log n)$).*

We refer to T as a precision tree with initial tolerance t.

The tolerance t_v at a node v determines the additive error which we can tolerate at v. That is, our goal is to approximate $\mathsf{ED}(X_v, Y_{v,s})$ with additive error t_v (and the same multiplicative error as before). The initial tolerance is set to $t = k$. The critical step is how to combine the recursive approximations with additive error $t_{v_1}, \ldots, t_{v_B}$ to an approximation with additive error t_v. The naive solution would incur error $\sum_i t_{v_i} \gg t_v$. Instead, we employ the *Precision Sampling Lemma* [4, 5, 3, 15] (see Lemma 15 in Section 4) to recombine the recursive approximation and avoid this blow-up in the additive error.

An Improved Algorithm. We can now improve the simple algorithm to a near-linear-time algorithm. In the original Andoni-Krauthgamer-Onak algorithm this was achieved by pruning most recursive subproblems (depending on their tolerances t_v). We will follow a different avenue: Our algorithm recurs on every node in the precision tree, and we obtain a linear-time algorithm by bounding the expected running time per node by $n^{o(1)}$.

We achieve this by the following novel trick: We restrict the set of feasible shifts s at each node v with respect to the tolerance t_v. In fact, we require two constraints: First, we restrict s to values smaller than $\approx k$ in absolute value. This first restriction is correct since we only want to maintain edit distances bounded by k; this idea was also used in previous works. Second, we restrict the feasible shifts s at any node v to multiples of $\lfloor t_v/2 \rfloor$. Then, in order to approximate $\mathsf{ED}(X_v, Y_{v,s})$ for any shift s we let $\tilde{s}$ denote the closest multiple of $\lfloor t_v/2 \rfloor$ to s, and approximate $\mathsf{ED}(X_v, Y_{v,s})$ by $\mathsf{ED}(X_v, Y_{v,\tilde{s}})$. Since $|s - \tilde{s}| \leq t_v/2$, both edit distances differ by at most t_v. Recall that we can tolerate this error using the Precision Sampling technique. Let $S_v = \{ -k \cdot n^{o(1)}, \ldots, k \cdot n^{o(1)} \} \cap \lfloor t_v/2 \rfloor \mathbf{Z}$ denote the set of shifts respecting these restrictions (the precise lower-order term $n^{o(1)}$ will be fixed later).

In terms of efficiency, we have improved as follows: At every node the running time is essentially dominated by the number of feasible shifts s. Using our discretization trick, there are only $|S_v| = k \cdot n^{o(1)}/t_v$ such shifts. By the following Lemma 5, we can bound this number in expectation by $k \cdot n^{o(1)}/t_{\mathsf{root}(T)} = n^{o(1)}$.

▶ **Lemma 5** (Expected Precision). *Let T be a B-ary precision tree with initial tolerance t and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$, and let v be a node in T. Then, conditioned on a high-probability event E, it holds that*

$$\mathbf{E}\left(\frac{1}{t_v}\,\middle|\,E\right) \leq \frac{(\log n)^{O(\mathsf{depth}(v))}}{t} \leq \frac{n^{o(1)}}{t}.$$

We include a proof in Appendix A. For technical reasons, the lemma is only true conditioned on some high-probability event E. For the remainder of this paper, we implicitly condition on this event E.

3.2 How to Go Sublinear?

Following the idea of [15], our strategy is to turn this algorithm into a sublinear-time algorithm by not exploring the whole precision tree recursively, and instead only exploring a smaller fragment. To achieve this, the goal is to approximate $\mathsf{ED}(X_v, Y_{v,s})$ for many nodes v directly, without the need to explore their children – we say that we *prune* v. The algorithm by [15] uses several structural insights on periodic versus non-periodic strings to implement pruning rules. We can avoid the complicated treatment and follow a much simpler avenue, exploiting that we can preprocess the strings.

In the following, we will assume that we have access to an oracle answering the following two queries. In the next section we will argue how to efficiently implement data structures to answer these queries.

- $\textsc{Matching}(X, Y, v)$: Returns either $\textsc{Close}(s^*)$ where s^* satisfies $|s^*| \leq k \cdot n^{o(1)}$ and $\mathsf{HD}(X_v, Y_{v,s^*}) \leq t_v/2$, or $\textsc{Far}$ in case that there is no shift s^* with $X_v = Y_{v,s^*}$, see Definition 8. (Note that if $1 \leq \min_{s^*} \mathsf{HD}(X_v, Y_{v,s^*}) \leq t_v/2$, the query can return either $\textsc{Close}(s^*)$ or $\textsc{Far}$.)
- $\textsc{ShiftedDistance}(Y, v, s, s')$: For shifts $|s|, |s'| \leq k \cdot n^{o(1)}$, returns an approximation of $\mathsf{ED}(Y_{v,s}, Y_{v,s'})$ with additive error $t_v/2$ (and $n^{o(1)}$-multiplicative error, see Definition 9).

Suppose for the moment that both queries can be answered in constant time. Then we can reduce the running time of the previous linear-time algorithm to time $k \cdot n^{o(1)}$ as follows: We try to prune each node v by querying $\textsc{Matching}(X, Y, v)$. If the $\textsc{Matching}$ query reports $\textsc{Far}$, we simply continue recursively as before (i.e., no pruning takes place). However, if the matching query reports $\textsc{Close}(s^*)$, then we can prune v as follows: Query $\textsc{ShiftedDistance}(Y, v, s^*, s)$ for all shifts $s \in S_v$ and return the outcome as an approximation of $\mathsf{ED}(X_v, Y_{v,s})$ for all shifts s. For the correctness we apply the triangle inequality and argue that the additive error is bounded by $t_v/2 + t_v/2 = t_v$.

It remains to argue that the number of recursive computations is bounded by $k \cdot n^{o(1)}$. The intuitive argument is as follows: Assume that $\mathsf{ED}(X, Y) \leq k$ (i.e., we are in the "close" case) and consider an optimal alignment between X and Y, which contains at most k mismatches. Recall that each level of the precision tree induces a partition of X into consecutive substrings X_v. Thus, there are at most k substrings X_v which contain a mismatch in the optimal alignment. For all *other* substrings, there are no mismatches and hence $X_v = Y_{v,s^*}$ for some shift s^*. It follows that on every level there are at most k nodes for which the $\textsc{Matching}$ test fails, and in total there are only $k \cdot \mathsf{height}(T) = O(k \log n)$ such nodes.

3.3 How to Answer Matching and Shifted-Distance Queries?

It remains to find data structures which answer these queries efficiently. We assume that the precision tree T has been generated in advance and is shared across all precomputations. In particular, this requires "public randomness" for the otherwise independent precomputations.

Matching Queries. The idea is to precompute and store fingerprints (i.e. hashes) of the substrings $Y_{v,s}$ for every node v in the partition tree and all shifts $s \in \{-k \cdot n^{o(1)}, \dots, k \cdot n^{o(1)}\}$. Then, to answer a query we simply compute the fingerprint of X_v and lookup whether it equals one of the precomputed fingerprints. Alas, a naive implementation of this idea is too slow since upon query we might need to read the whole string X. To obtain the desired sublinear query time, we instead subsample the strings X_v and $Y_{v,s}$ with rate $\approx 1/t_v$. In this way we incur additive Hamming error at most $t_v/2$, as desired. Formally, we show the following lemma in Section 4.1:

▶ **Lemma 6** (Matching Queries). *We can preprocess Y in expected time $n^{1+o(1)}$ to answer $\textsc{Matching}(X, Y, v)$ queries in time $\widetilde{O}(|X_v|/t_v)$ with high probability. Moreover, we can separately preprocess both X and Y in expected time $n^{1+o(1)}$ to answer $\textsc{Matching}(X, Y, v)$ queries in time $O(1)$ with high probability.*

The key difference between the one-sided and the two-sided preprocessing is that in the former we need to compute the fingerprint for X_v, which takes time $O(|X_v|/t_v)$ (we shave the factor t_v due to the subsampling), while in the latter we can afford to precompute these fingerprints and answer the queries faster.

Shifted-Distance Queries. We give two ways to answer ShiftedDistance queries. The first one relies in a black-box manner on any almost-linear-time algorithm to compute an edit distance approximation with multiplicative error $n^{o(1)}$ [4, 7, 6]. Using the same trick as before to restrict the set of feasible shifts, it suffices to approximate $\mathsf{ED}(Y_{v,s}, Y_{v,s'})$ for all shifts $s \in S_v$ and $s' \in \{-k \cdot n^{o(1)}, \dots, k \cdot n^{o(1)}\}$. (We could also discretize the range of s', but this does not improve the performance here.) In this way, we incur an additive error of at most $O(t_v)$. The computation per node v takes time $|Y_v| \cdot k \cdot k \cdot n^{o(1)}/t_v$, which becomes $kn^{1+o(1)}$ in expectation by Lemma 5 and by summing over all nodes v.

We then show that we can improve the preprocessing time to $n^{1+o(1)}$ by applying the non-oblivious embedding of edit distance into ℓ_1 by Andoni and Onak [7]. Formally we obtain the following lemma, which we prove in Section 4.3.

▶ **Lemma 7** (Shifted-Distance Queries). *We can preprocess Y in time $n^{1+o(1)}$ to answer $\textsc{ShiftedDistance}(Y, v, s, s')$ queries in time $n^{o(1)}$ with high probability.*

4 Our Algorithm in Detail

In this section we give a detailed proof of our main theorems by analyzing Algorithms 1 and 2 (see Sections 4.4 and 4.5). Algorithm 2 is the previously discussed reformulation of the Andoni-Krauthgamer-Onak algorithm, with the improvement that the recursive computation can be avoided whenever Algorithm 1 succeeds. Algorithm 1 implements the pruning rule which, as we will argue, triggers often enough to improve the running time.

Throughout we fix the initial tolerance of the precision tree T to $t = t_{\mathsf{root}(T)} = k$. Moreover, we set $S = \{-k \cdot 3^{\log_B(n)}, \dots, k \cdot 3^{\log_B(n)}\}$ and $S_v = S \cap \lfloor t_v/2 \rfloor \mathbf{Z}$. Observe that $|S| = k \cdot 3^{\log_B(n)} = k \cdot n^{o(1)}$ for our choice of $B = \exp(\widetilde{O}(\sqrt{\log n}))$.

Outline. Compared to the technical overview, we present the details in the reverse order: Starting with the implementation of the data structures for MATCHING and SHIFTED-DISTANCE queries (in Section 4.1), we first analyze Algorithm 1 (in Section 4.4). Then we analyze Algorithm 2 (in Section 4.5) and put the pieces together for our main theorems (in Section 4.6).

4.1 Matching Queries

In this section we show to implement data structures that answer MATCHING queries. We start with a formal definition.

▶ **Definition 8** (Matching Queries). *Let X, Y be strings of length n, and let v be a node in the precision tree. A* MATCHING(X, Y, v) *query is correctly answered by one of two outputs:*

- CLOSE(s^*) *for some shift $s^* \in S$ satisfying $\mathsf{HD}(X_v, Y_{v,s^*}) \leq t_v/2$, or*
- FAR *if there exists no shift $s^* \in S$ with $X_v = Y_{v,s^*}$.*

▶ **Lemma 6** (Matching Queries). *We can preprocess Y in expected time $n^{1+o(1)}$ to answer* MATCHING(X, Y, v) *queries in time $\widetilde{O}(|X_v|/t_v)$ with high probability. Moreover, we can separately preprocess both X and Y in expected time $n^{1+o(1)}$ to answer* MATCHING(X, Y, v) *queries in time $O(1)$ with high probability.*

Proof. We first explain the general idea behind the data structure and then point out the specifics for the one-sided and two-sided preprocessing. For each node v in the precision tree, we subsample a set $H_v \subseteq [\,|Y_v|\,]$ with rate $\Theta(\log n/t_v)$ and sample a hash function $h : \Sigma^{|H_v|} \to [\,\mathrm{poly}(n)\,]$ from any universal family of hash functions. (Take for instance the function $h(\sigma_1, \ldots, \sigma_{|H_v|}) = \sum_i a_i\sigma_i \bmod p$ for some prime $p = \mathrm{poly}(n)$ and random a_i's).

We associate to a string $A \in \Sigma^{|Y_v|}$ the *fingerprint* $h(A[\,H_v\,])$, where we write $A[\,H_v\,]$ for the subsequence of A with indices in H_v. We claim that these fingerprints can distinguish any two strings A, B with $\mathsf{HD}(A, B) > t_v/2$ with high probability. Indeed, the probability that H_v contains an index i with $A[\,i\,] \neq B[\,i\,]$ is at least

$$1 - \left(1 - \frac{\Omega(\log n)}{t_v}\right)^{\mathsf{HD}(A,B)} \geq 1 - \exp(-\Omega(\log n)) = 1 - \frac{1}{\mathrm{poly}(n)}.$$

Hence, with high probability we have that $A[\,H_v\,] \neq B[\,H_v\,]$. Moreover, since h is sampled from a universal family of hash functions, we also have that $h(A[\,H_v\,]) \neq h(B[\,H_v\,])$ with high probability. On the other hand, if $A = B$ then clearly $h(A[\,H_v\,]) = h(B[\,H_v\,])$. We now turn to the implementation details for one-sided and two-sided preprocessing.

One-Sided Preprocessing. In the preprocessing phase, we prepare for every node v the fingerprints of $Y_{v,s}$ for all shifts $s \in S$, and store these fingerprints in a lookup table. In the query phase, given some string X we compute the fingerprint of X_v and check whether it appears in the lookup table. If so, we return CLOSE(s^*) for the shift s^* corresponding to the precomputed fingerprint. Otherwise, we return FAR. The correctness follows from the previous paragraph.

The expected running time of the preprocessing phase is bounded by $\sum_v \widetilde{O}(|Y_v|/t_v \cdot |S|)$ (for every node we have to prepare $|S|$ fingerprints, each taking time $\widetilde{O}(|Y_v|/t_v)$). This becomes $\sum_v O(|Y_v| \cdot n^{o(1)}/t_{\mathsf{root}(T)} \cdot k) = \sum_v O(|Y_v| \cdot n^{o(1)}) = n^{1+o(1)}$ in expectation over the tolerances t_v; see Lemma 5. The query time is dominated by computing the fingerprint of X_v which takes expected time $\widetilde{O}(|X_v|/t_v)$ (even with high probability by an application of the Chernoff bound).

Two-Sided Preprocessing. For two-sided preprocessing, we can also prepare the fingerprints of X_v for all nodes v in the preprocessing phase. The expected preprocessing time is still $n^{1+o(1)}$, but for queries it only takes constant time to perform the lookup. ◀

4.2 Simple Shifted-Distance Queries

We next demonstrate how to deal with ShiftedDistance queries, formally defined as follows:

▶ **Definition 9** (Shifted-Distance Queries). *Let Y be a string of length n, let v be a node in the precision tree and let $s, s' \in S$. A ShiftedDistance(Y, v, s, s') query computes a number satisfying:*

$$\mathsf{ED}(Y_{v,s}, Y_{v,s'}) - t_v/2 \leq \textsc{ShiftedDistance}(Y, v, s, s') \leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s}, Y_{v,s'}) + t_v/2).$$

We will first show how to implement a simpler version of Lemma 7 at the cost of worsening the preprocessing time to $kn^{1+o(1)}$ instead of $n^{1+o(1)}$. The benefit of this weaker version is that it is a black-box reduction to any almost-linear time $n^{o(1)}$-approximation algorithm for edit distance, while the improved version crucially relies on the properties of the particular algorithm by Andoni and Onak [7]. In the next section we show how to obtain the speed-up.

▶ **Lemma 10** (Slower Shifted-Distance Queries). *We can preprocess Y in time $kn^{1+o(1)}$ to answer ShiftedDistance(Y, v, s, s') queries in time $n^{o(1)}$ with high probability.*

Proof. We will use the result that an $n^{o(1)}$-approximation for the edit distance of two length-n strings can be computed in time $n^{1+o(1)}$ [4, 7, 6]. In the preprocessing phase, we compute $n^{o(1)}$-factor approximations $\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'})$ for all nodes v, all $\tilde{s} \in S_v$ and all $s' \in S$. Then, to answer a query ShiftedDistance(Y, v, s, s') we let

$$\tilde{s} := \operatorname*{argmin}_{\tilde{s} \in S_v} |s - \tilde{s}|$$

and output $\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'})$.

First we argue that this gives a good approximation. Indeed, we have that $|s - \tilde{s}| \leq t_v/4$ by the definition of S_v. Therefore:

$$\begin{aligned}\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'}) &\leq n^{o(1)} \cdot \mathsf{ED}(Y_{v,\tilde{s}}, Y_{v,s'}) \\ &\leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s}, Y_{v,s'}) + \mathsf{ED}(Y_{v,s}, Y_{v,\tilde{s}})) \\ &\leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s}, Y_{v,s'}) + t_v/2),\end{aligned}$$

where second inequality is an application of the triangle inequality, and the last inequality follows since we can transform $Y_{v,\tilde{s}}$ into $Y_{v,s}$ by adding and removing $t_v/4$ symbols. A symmetric argument shows the claimed lower bound $\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'}) \geq \mathsf{ED}(Y_{v,s}, Y_{v,s'}) - t_v/2$.

Next we analyze the running time. For the preprocessing, we compute $|S_v| \cdot |S| = O^*(k^2/t_v)$ many approximations for each node v, each in time $|Y_v|^{1+o(1)}$. We can bound $\mathbf{E}(1/t_v) = n^{o(1)}/k$ by Lemma 5, so the expected total time is $\sum_v |Y_v|^{1+o(1)} n^{o(1)} \cdot k = kn^{1+o(1)}$. Answering a query takes constant time since we only need to compute s^* as stated above and perform a constant-time lookup. ◀

4.3 Faster Shifted-Distance Queries

In this section we show how to improve the preprocessing time for ShiftedDistance queries to $n^{1+o(1)}$. We thank the anonymous reviewer who suggested this improvement. The key technical tool to obtain this improvement is the following result by Andoni and Onak [7].

Algorithm 1 Approximates the edit distance of preprocessed strings, or fails.

Input: Strings X (un- or preprocessed) and Y (preprocessed), a node v in the precision tree
Output: An approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ of $\mathsf{ED}(X_v, Y_{v,s})$ for all $s \in S_v$, or FAIL

1: **if** MATCHING$(X, Y, v) =$ CLOSE(s^*) **then**
2: **return** $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) =$ SHIFTEDDISTANCE(Y, v, s^*, s) for all $s \in S_v$
3: **else**
4: **return** FAIL

▶ **Theorem 11** (Embedding Substrings into ℓ_1 [7]). *Let X be a string of length n. Then, for each integer m of the form $m = \lfloor n/B^i \rfloor$ for $0 \le i \le \log_B(n)$ and $B = 2^{\Theta(\sqrt{\log n \log\log n})}$, we can embed all length-m substrings $X_0, \dots, X_{n-m}$ of X into vectors $v_0^m, \dots, v_{n-m}^m$ of dimension $n^{o(1)}$ such that for every $j, j' \in [\, n - m + 1\,]$, with high probability it holds that*

$$\mathsf{ED}(X_j, X_{j'}) \le \|v_j^m - v_{j'}^m\|_1 \le n^{o(1)} \cdot \mathsf{ED}(X_j, X_{j'}).$$

The time to compute all vectors is $n^{1+o(1)}$.

The main application of this embedding in [7] is an $n^{o(1)}$-approximation for the edit distance of two length-n strings in time $n^{1+o(1)}$ (in [7], Theorem 11 is applied to the concatenation of two strings to compute the ℓ_1-distance between their corresponding vectors). We remark that the guarantee of the embedding in Theorem 11 is *non-oblivious*, in the sense that the algorithm needs to have access to all the substrings it is embedding. In particular, this means that it cannot be directly applied in the two-sided preprocessing setting where we would like to embed the strings separately.

▶ **Lemma 7** (Shifted-Distance Queries). *We can preprocess Y in time $n^{1+o(1)}$ to answer SHIFTEDDISTANCE(Y, v, s, s') queries in time $n^{o(1)}$ with high probability.*

Proof. We assume without loss of generality that $n = |Y| = |X|$ is a power of B (we can pad both X and Y so that this holds, which has no impact on the running time or the approximation guarantee of our algorithms). We apply Theorem 11 to the string Y and store all the $n^{1+o(1)}$ embedded vectors. To answer a query SHIFTEDDISTANCE(Y, v, s, s'), note that $Y_{v,s}$ and $Y_{v,s'}$ are substrings of Y of length $m := n/B^{\mathsf{depth}(v)}$. Thus, the ℓ_1 distance between their corresponding vectors $v^m_{j(v,s)}, v^m_{j(v,s')}$ given by Theorem 11 gives the desired approximation (with no additive error). Since these vectors have dimension $n^{o(1)}$, the time to compute $\|v^m_{j(v,s)} - v^m_{j(v,s')}\|_1$ and answer the query is $n^{o(1)}$, as desired. ◀

4.4 Pruning Rule for Preprocessed Strings

In this section we analyze Algorithm 1. We always assume that either only Y or both X and Y have been preprocessed by Lemmas 6 and 7 to efficiently answer MATCHING and SHIFTEDDISTANCE queries.

▶ **Lemma 12** (Correctness of Algorithm 1). *Whenever Algorithm 1 does not return FAIL, it returns approximations $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ satisfying with high probability*

$$\mathsf{ED}(X_v, Y_{v,s}) - t_v \le \widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \le n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v).$$

Proof. Algorithm 1 only succeeds if the MATCHING query in Algorithm 1 successfully identified some shift s^* with $\mathsf{HD}(X_v, Y_{v,s^*}) \leq t_v/2$ by Lemma 6. In this case, we report $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) := \textsc{ShiftedDistance}(Y, v, s^*, s)$. By Lemma 7, this is an approximation of $\mathsf{ED}(Y_{v,s^*}, Y_{v,s})$ with additive error $t_v/2$ and multiplicative error $n^{o(1)}$. Combining both facts, and using the triangle inequality we obtain that

$$\begin{aligned}\widetilde{\mathsf{ED}}(Y_{v,s^*}, Y_{v,s}) &\leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s^*}, Y_{v,s}) + t_v/2)\\ &\leq n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + \mathsf{ED}(X_v, Y_{v,s^*}) + t_v/2)\\ &\leq n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v),\end{aligned}$$

and

$$\begin{aligned}\widetilde{\mathsf{ED}}(Y_{v,s^*}, Y_{v,s}) &\geq \mathsf{ED}(Y_{v,s^*}, Y_{v,s}) - t_v/2\\ &\geq \mathsf{ED}(X_v, Y_{v,s}) - \mathsf{ED}(X_v, Y_{v,s^*}) - t_v/2\\ &\geq \mathsf{ED}(X_v, Y_{v,s}) - t_v.\end{aligned}$$

◀

▶ **Lemma 13** (Running Time of Algorithm 1)**.** *If only the string Y is preprocessed, then Algorithm 1 runs in expected time $O^*(|X_v|/t_v + k/t_v)$. If both strings X, Y are preprocessed, then Algorithm 1 runs in expected time $O^*(k/t_v)$.*

Proof. The expected running time of the MATCHING query is bounded by $\widetilde{O}(|X_v|/t_v)$ (for one-sided preprocessing) or by $O(1)$ (for two-sided preprocessing). The running time of a single SHIFTEDDISTANCE query is bounded by $n^{o(1)}$, and we make $|S_v| = \widetilde{O}(k/t_v)$ SHIFTEDDISTANCE queries. Hence, the total expected time is $O^*(|X_v|/t_v + k/t_v)$ (for one-sided preprocessing) or $O^*(k/t_v)$ (for two-sided preprocessing). ◀

▶ **Lemma 14** (Efficiency of Algorithm 1)**.** *For any two strings X, Y, there are at most $O(k \log n)$ many nodes v in the precision tree for which Algorithm 1 fails, assuming that $\mathsf{ED}(X, Y) \leq k$.*

Proof. It suffices to argue that there are at most $O(\mathsf{ED}(X, Y) \log n)$ nodes for which the MATCHING query in Algorithm 1 fails. We start with a fixed level in the precision tree, and enumerate all nodes on that level as $v_1, \ldots, v_m$. By definition we have that $X = \bigcirc_{i=1}^{m} X_{v_i}$, hence there exist indices $0 = j_0 < \cdots < j_m = n$ such that $X_{v_i} = X[\, j_i \,..\, j_{i+1}\,]$. Now consider an optimal alignment $A : \{\, 0, \ldots, n \,\} \to \{\, 0, \ldots, n \,\}$ between X and Y. In particular, A satisfies

$$\mathsf{ED}(X, Y) = \sum_{i=1}^{m} \mathsf{ED}(X[\, j_i \,..\, j_{i+1}\,], Y[\, A(j_i) \,..\, A(j_{i+1})\,]).$$

There can be at most $\mathsf{ED}(X, Y) \leq k$ many nonzero terms in the sum, and we claim that every zero term corresponds to a node v_i for which Algorithm 1 succeeds. Indeed, for any zero term we have $X[\, j_i \,..\, j_{i+1}\,] = Y[\, A(j_i) \,..\, A(j_{i+1})\,]$ and therefore $X_{v_i} = Y_{v_i,s^*}$ where $s^* = A(j_i) - j_i$. It remains to argue that $|s^*| \leq k$ (since otherwise the index s^* would be out-of-bounds and could not be detected by a MATCHING query). To see this, observe that

$$|s^*| \leq \mathsf{ED}(X[\, j_0 \,..\, j_i\,], Y[\, A(j_0) \,..\, A(j_i)\,]) \leq \mathsf{ED}(X, Y) \leq k,$$

exploiting again that A is an optimal alignment.

Finally, recall that there are only $\log_B(n) \leq \log n$ levels in the precision tree, hence the total number of nodes for which Algorithm 1 fails is bounded by $O(k \log n)$. ◀

Algorithm 2 Approximates the edit distance of preprocessed strings.

Input: Strings X (un- or preprocessed), Y (preprocessed), a node v in the precision tree
Output: An approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ of $\mathsf{ED}(X_v, Y_{v,s})$ for all $s \in S_v$

1: **if** Algorithm 1 succeeds and reports $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ for all $s \in S_v$ **then**
2: **return** $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ for all $s \in S_v$
3: **if** v is a leaf **then**
4: **return** $\mathsf{ED}(X_v, Y_{v,s})$ for all $s \in S_v$
5: **for** all children v_i of v **do**
6: Recursively compute $\widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i})$ for all $s_i \in S_{v_i}$
7: Compute $\widetilde{a}_{i,s} = \min_{s_i \in S_{v_i}} \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2 \cdot |s - s_i|$ for all $s \in S_v$
8: **return** $\textsc{Recover}(\widetilde{a}_{1,s}, \dots, \widetilde{a}_{B,s}, u_{v_1}, \dots, u_{v_B})$ for all $s \in S_v$

4.5 The Complete Algorithm

Our complete Algorithm 2 is essentially what we described in Section 4.5 with the additional pruning rule of applying Algorithm 1. We start with the correctness of Algorithm 2. We need the following lemma, which has previously been referred to as a *Precision Sampling Lemma* [4, 5, 3, 15]. The lemma was introduced by Andoni, Krauthgamer and Onak [4], and was refined in [5, 3].

Intuitively, the lemma serves the following purpose: For fixed numbers $a_1, \dots, a_B$, say that we have access to approximations $\widetilde{a}_1, \dots, \widetilde{a}_B$ with multiplicative error α and additive approximation error β. Then we can naively approximate $\sum_i a_i$ by $\sum_i \widetilde{a}_i$ with multiplicative error α and additive error $B \cdot \beta$. The Precision Sampling Lemma states that the blow-up in the additive error can be avoided if the approximations $\widetilde{a}_i$ instead have additive error $\beta \cdot u_i$ for some *non-uniformly sampled precisions* u_i.

▶ **Lemma 15** (Precision Sampling Lemma [3]). *Fix parameters $\delta > 0$, $\alpha \geq 1$ and $\beta \geq 0$. Let $a_1, \dots, a_B \geq 0$ be reals, and independently sample $u_1, \dots, u_B \sim \mathrm{Exp}(O(\log(\delta^{-1})))$ (for some sufficiently large hidden constant). There is an $O(B \log(\delta^{-1}))$-time algorithm $\textsc{Recover}$ satisfying for all $\widetilde{a}_1, \dots, \widetilde{a}_B$, with success probability at least $1 - \delta$:*

- *If $\widetilde{a}_i \geq \frac{1}{\alpha} \cdot a_i - \beta \cdot u_i$ for all i, then $\textsc{Recover}(\widetilde{a}_1, \dots, \widetilde{a}_B, u_1, \dots, u_B) \geq \frac{1}{2\alpha} \cdot \sum_i a_i - \beta$.*
- *If $\widetilde{a}_i \leq \alpha \cdot a_i + \beta \cdot u_i$ for all i, then $\textsc{Recover}(\widetilde{a}_1, \dots, \widetilde{a}_B, u_1, \dots, u_B) \leq 2\alpha \sum_i a_i + \beta$.*

▶ **Lemma 16** (Correctness of Algorithm 2). *Algorithm 2 computes values $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ (for all $s \in S_v$) satisfying the following bounds with high probability:*

- $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \geq n^{-o(1)} \cdot \mathsf{ED}(X_v, Y_{v,s}) - t_v$, *and*
- $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \leq n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v)$ *assuming that* $\mathsf{ED}(X_v, Y_{v,s}) \leq k$ *and* $2|s| \leq k$.

Proof. The recursion of Algorithm 2 terminates in one of three cases: If v is a leaf, then the output is exact and the claim is obvious. If v is directly solved by Algorithm 1, then by the guarantee of Lemma 12 the claim is true. It remains to analyze the case when the algorithm recurs, so assume that v is an internal node with children $v_1, \dots, v_B$. We prove the lower and upper bounds separately.

Lower Bound. Fix some shift $s \in S_v$, and let $s_1, \dots, s_B$ be the corresponding shifts selected in Algorithm 2 of the algorithm. We prove that $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \geq \mathsf{ED}(X_v, Y_{v,s})/\alpha(\mathsf{height}(v)) - t_v$ by induction, where $\alpha(\mathsf{height}(v)) = 2^{\mathsf{height}(v)}$. To this end, we apply the Precision Sampling Lemma with $a_i = \mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i|$ and the following parameters:

- $\delta = 1/\operatorname{poly}(n)$,
- $\alpha = \alpha(\mathsf{height}(v) - 1)$,
- $\beta = t_v$.

Recall that the algorithm computes $\widetilde{a}_{i,s} = \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i|$, where the approximation $\widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i})$ is computed recursively. Hence, it satisfies $\widetilde{a}_{i,s} \geq a_i/\alpha - \beta \cdot u_{v_i}$ by the induction hypothesis (recall that $t_{v_i} \leq t_v \cdot u_{v_i}$). Then $\textsc{Recover}(\widetilde{a}_{1,s}, \ldots, \widetilde{a}_{B,s}, u_{v_1}, \ldots, u_{v_B})$ computes, with high probability, a number satisfying

$$\begin{aligned}
\textsc{Recover}(\cdot) &\geq \frac{1}{2\alpha(\mathsf{height}(v)-1)} \cdot \left(\sum_{i=1}^{B} a_i \right) - \beta \\
&= \frac{1}{2\alpha(\mathsf{height}(v)-1)} \cdot \left(\sum_{i=1}^{B} \mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i| \right) - t_v \\
&\geq \frac{1}{2\alpha(\mathsf{height}(v)-1)} \cdot \mathsf{ED}(X_v, Y_{v,s}) - t_v \\
&\geq \frac{1}{\alpha(\mathsf{height}(v))} \cdot \mathsf{ED}(X_v, Y_{v,s}) - t_v,
\end{aligned}$$

where in the third step, we applied the lower bound from Lemma 3 and in the last step we used the definition of $\alpha(\cdot)$. Finally, recall that $\mathsf{height}(T) \leq \log_B(n)$ and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$. Hence the total multiplicative error is bounded by $\alpha(\log_B(n)) \leq 2^{\log_B(n)} = n^{o(1)}$.

Upper Bound. This proof is similar to the previous paragraph, but requires a more careful application of Lemma 3. We prove by induction the algorithm computes an approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \leq \alpha(\mathsf{height}(v)) \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v)$ where this time the multiplicative error is bounded by $\alpha(\mathsf{height}(v)) \leq O(B) \cdot 2^{O(\mathsf{height}(v))}$, provided that $\mathsf{ED}(X_v, Y_{v,s}) \leq k \cdot 3^{\mathsf{depth}(v)}$ and $2|s| \leq k \cdot 3^{\mathsf{depth}(v)}$. Note that this implies the lemma statement.

Throughout, fix some shift $s \in S_v$. The idea is to first use Lemma 3 to find "optimal" shifts $s_1^*, \ldots, s_B^* \in \mathbf{Z}$, which we use for the recursive computation. Unfortunately these shifts s_i^* may not fall into the restricted set of feasible shifts S_{v_i}. We therefore argue that picking shifts $s_i \in S_{v_i}$ closest possible to s_i^* is sufficient to obtain the claimed guarantee. Formally, by Lemma 3 there exist shifts $s_1^*, \ldots, s_B^*$ satisfying the following two properties:

$$\sum_{i=1}^{B} \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) \leq 2\mathsf{ED}(X_v, Y_{v,s}), \tag{2}$$

$$2|s - s_i^*| \leq \mathsf{ED}(X_v, Y_{v,s}). \tag{3}$$

We now pick $s_1 \in S_{v_1}, \ldots, s_B \in S_{v_B}$ to be the closest values to the optimal shifts $s_1^*, \ldots, s_B^*$. As a first insight, observe that:

$$2|s_i^*| \leq 2|s - s_i^*| + 2|s| \leq \mathsf{ED}(X_v, Y_{v,s}) + 2|s| \leq 2k \cdot 3^{\mathsf{depth}(v)}.$$

Recall that we set $S = \{ -k \cdot 3^{\log_B(n)}, \ldots, k \cdot 3^{\log_B(n)} \}$ and we therefore have $s_i^* \in S$. It follows that $|s_i - s_i^*| \leq t_{v_i}/2$ and thus

$$2|s_i| \leq 2|s_i^*| + t_{v_i} \leq 2|s_i^*| + k \leq 2k \cdot 3^{\mathsf{depth}(v)} + k \leq k \cdot 3^{\mathsf{depth}(v_i)}.$$

Next, we claim that $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) \leq k \cdot 3^{\mathsf{depth}(v_i)}$, which we will use to guarantee that the recursive calls of the algorithm succeed. Indeed, we have that

$$\mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) \leq \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) + t_{v_i} \leq 2 \cdot \mathsf{ED}(X_v, Y_{v,s}) + k \leq k \cdot 3^{\mathsf{depth}(v_i)}.$$

We claim that if the algorithm was to choose the shifts s_i specified in the previous paragraph in Algorithm 2, then the output is bounded as claimed. (This is sufficient, since Algorithm 2 in fact minimizes over all possible shifts s_i.) In this case, we inductively have that

$$\begin{aligned}\widetilde{a}_{i,s} &\leq \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i| \\ &\leq \alpha(\mathsf{height}(v) - 1) \cdot (\mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) + t_{v_i}) + 2|s - s_i| \\ &\leq \alpha(\mathsf{height}(v) - 1) \cdot (\mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) + 3t_{v_i}) + 2|s - s_i^*|.\end{aligned}$$

In the last step we used that $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*})$ differs from $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i})$ by an additive error of t_{v_i}, and the same is true for $2|s - s_i^*|$ and $2|s - s_i|$. Next, we apply the Precision Sampling Lemma with $a_i = \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) + 2\alpha^{-1}|s - s_i^*|$ and parameters

- $\delta = 1/\operatorname{poly}(n)$,
- $\alpha = \alpha(\mathsf{height}(v) - 1)$,
- $\beta = \alpha t_v$.

Recall that $t_{v_i} = t_v \cdot u_{v_i}/3$, thus by definition we have that $\widetilde{a}_{i,s} \leq \alpha \cdot a_i + \beta \cdot u_{v_i}$. Therefore, the Precision Sampling Lemma states that with high probability the recovery algorithm returns

$$\begin{aligned}\textsc{Recover}(\cdot) &\leq 2\alpha \cdot \left(\sum_{i=1}^{B} a_i\right) + \beta \\ &\leq 2\alpha \left(\sum_{i=1}^{B} \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*})\right) + 2\left(\sum_{i=1}^{B} 2|s - s_i^*|\right) + \alpha t_v \\ &\leq 4\alpha \cdot \mathsf{ED}(X_v, Y_{v,s}) + 2B \cdot \mathsf{ED}(X_v, Y_{v,s}) + \alpha t_v \\ &\leq 4(\alpha + B) \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v) \\ &\leq \alpha(\mathsf{height}(v)) \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v),\end{aligned}$$

where for the third inequality we applied the bounds in (2) and (3) and in the last step we used the definition of $\alpha(\mathsf{height}(v)) = O(B) \cdot 2^{O(\mathsf{height}(v))}$ (for sufficiently large hidden constants). Since the tree has height $\log_B(n)$ and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$, the overall approximation factor is bounded by $\alpha(\log_B(n)) = O(B) \cdot 2^{O(\log_B(n))} = n^{o(1)}$, as claimed. ◀

Next, we analyze the running time of Algorithm 2. It turns out that the bottleneck is the computation in Algorithm 2, and a naive implementation would be too slow for our purposes. For that reason, we use the following lemma for an improved implementation of Algorithm 2. The lemma is a generalization of [15, Lemma 10].

▶ **Lemma 17** (Range Minimum Problem). *Let T, T' be sets of integers (given in sorted order), and let $(b_{s'})_{s' \in T'}$ be given. There is an $O(|T| + |T'|)$-time algorithm to compute $(a_s)_{s \in T}$ defined by*

$$a_s = \min_{s' \in T'} b_{s'} + 2 \cdot |s - s'|.$$

Proof. The idea is to compute auxiliary values

$$a_s^{\leq} = \min_{\substack{s' \in T' \\ s' \leq s}} b_{s'} + 2s - 2s',$$

$$a_s^{\geq} = \min_{\substack{s' \in T' \\ s' \geq s}} b_{s'} - 2s + 2s',$$

as then returning $a_s = \min(a_s^{\leq}, a_s^{\geq})$ is correct. We show how to compute $a_s^{\leq}$ (for all s); the values $a_s^{\geq}$ are symmetric. Let $T = \{ s_1 < \cdots < s_{|T|} \}$. We evaluate the base case $a_{s_1}^{\leq}$ naively. We then compute $a_{s_i}^{\leq}$ for all $i = 2, \ldots, |T|$ as follows:

$$a_{s_i}^{\leq} = \min \left\{ a_{s_{i-1}}^{\leq} + 2s_i - 2s_{i-1}, \min_{s_{i-1} < s' \leq s_i} b_{s'} + 2s_i - 2s' \right\}.$$

For the correctness, we distinguish two cases. Let $s' \leq s_i$ be the index which attains the minimum in the definition of $a_{s_i}^{\leq}$. On the one hand, if $s' \leq s_{i-1}$, then $a_{s_i}^{\leq} = a_{s_{i-1}}^{\leq} + 2s_i - 2s_{i-1}$ and thus the first term in the minimum is correct. On the other hand, if $s' > s_{i-1}$, then the second term in the minimum is correct by definition. In order to compute $(a_s^{\leq})_s$ we sweep from left to right over all values in T and T' exactly once, hence the running time can be bounded by $O(|T| + |T'|)$. ◀

▶ **Lemma 18** (Running Time of Algorithm 2). *Assume that* $\mathsf{ED}(X, Y) \leq k$. *If only the string* Y *is preprocessed, then Algorithm 2 runs in expected time* $n^{1+o(1)}/k + kn^{o(1)}$. *If both strings* X, Y *are preprocessed, then Algorithm 2 runs in expected time* $kn^{o(1)}$.

Proof. We first bound the running time of a single execution of Algorithm 2 (ignoring the cost of recursive calls). The computation in Algorithm 2 merely compares single characters and therefore takes time $|S_v| = O^*(k/t_v)$. By Lemma 13, the call to Algorithm 1 takes expected time $O^*(|X_v|/t_v + k/t_v)$ (for one-sided preprocessing) or $O^*(k/t_v)$ (for two-sided preprocessing). Each iteration of the loop in Algorithms 2–2 is dominated by the computation in Algorithm 2 (ignoring the recursive calls in Algorithm 2). Using Lemma 17 with $T = S_v$ and $T' = S_{v_i}$, this step takes time $O(|S_v| + |S_{v_i}|)$, and thus the loop runs in time $O(B \cdot |S_v| + \sum_i |S_{v_i}|) = O^*(k/t_v + \sum_i k/t_{v_i})$. In all of these bounds we can bound $1/t_v$ by $n^{o(1)}/k$ in expectation, according to Lemma 5, so the total expected time per node becomes $n^{o(1)} \cdot (|X_v|/k + 1)$ (for one-sided preprocessing) or $n^{o(1)}$ (for two-sided preprocessing).

To account for the recursive calls, we first use Lemma 14 to bound the number of recursive calls by $O(k \log n)$. Let $v_1, \ldots, v_m$ (with $m = O(k \log n)$) denote all nodes for which Algorithm 2 is recursively called. Then the expected running time for one-sided preprocessing can be bounded by

$$\begin{aligned} n^{o(1)} \cdot \sum_{i=1}^{m} \left(\frac{|X_{v_i}|}{k} + 1 \right) &\leq kn^{o(1)} + n^{o(1)} \cdot \sum_{d=1}^{\log_B(n)} \sum_{\substack{i=1 \\ \mathsf{depth}(v_i)=d}}^{m} \frac{|X_{v_i}|}{k} \\ &\leq kn^{o(1)} + n^{o(1)} \cdot \sum_{d=1}^{\log_B(n)} \frac{n}{k} \\ &\leq kn^{o(1)} + \frac{n^{1+o(1)}}{k}. \end{aligned}$$

Here we used that across any level in the precision tree, the strings X_v form a partition of X into consecutive substrings. In the same way we can bound the running time for two-sided preprocessing by $kn^{o(1)}$. ◀

4.6 Proof of the Main Theorems

We are finally ready to prove our main theorems.

Proof of Theorems 1 and 2. We assume that either only Y (for Theorem 1) or both X and Y (for Theorem 2) are preprocessed by Lemmas 6 and 7. We will solve the (k, K)-gap edit distance problem, for some parameter K to be picked later, by running Algorithm 2 (with input $v = \mathsf{root}(T)$ and $s = 0$, so in particular $2|s| \leq k$). For the correctness proof, we apply Lemma 16 to the following two cases:

- If $\mathsf{ED}(X,Y) \le k$, then $\widetilde{\mathsf{ED}}(X,Y) \le n^{o(1)} \cdot (\mathsf{ED}(X,Y)+t) \le n^{o(1)} \cdot k$.
- If $\mathsf{ED}(X,Y) \ge K$, then $\widetilde{\mathsf{ED}}(X,Y) \ge n^{-o(1)} \cdot \mathsf{ED}(X,Y) - t \ge n^{-o(1)} \cdot K$.

By setting $K = k \cdot n^{o(1)}$ for a sufficiently large subpolynomial factor, we can distinguish the two cases based on the outcome $\widetilde{\mathsf{ED}}(X,Y)$.

Next, we analyze the running time. If $\mathsf{ED}(X,Y) \le k$, then the algorithm runs in expected time $n^{1+o(1)}/k + kn^{o(1)}$ or $kn^{o(1)}$, respectively; see Lemma 18. By Markov's inequality, the algorithm respects these time bounds with constant probability. We may therefore interrupt the algorithm after it exceeds its time budget and report "$\mathsf{ED}(X,Y) \ge K$" in case of an interruption. We can boost the success probability to $1 - 1/\operatorname{poly}(n)$ by running the algorithm $O(\log n)$ times in parallel and reporting the majority answer. (This also means that the preprocessing is repeated $O(\log n)$ times with independently sampled precision trees.) ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '15, pages 59–78. IEEE Computer Society, 2015.

2 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th ACM Symposium on Theory of Computing*, STOC '16, pages 375–388. ACM, 2016.

3 Alexandr Andoni. High frequency moments via max-stability. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, ICASSP '17, pages 6364–6368. IEEE, 2017.

4 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 377–386. IEEE Computer Society, 2010.

5 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 363–372. IEEE Computer Society, 2011.

6 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '20, pages 990–1001. IEEE Computer Society, 2020.

7 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.

8 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the 47th ACM Symposium on Theory of Computing*, STOC '15, pages 51–58. ACM, 2015.

9 Ziv Bar-Yossef, S. Thathachar Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '04, pages 550–559. IEEE Computer Society, 2004.

10 Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, STOC '03, pages 316–324. ACM, 2003.

11 Tugkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, SODA '06, pages 792–801. SIAM, 2006.

12 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '16, pages 51–60. IEEE Computer Society, 2016.

13 Joshua Brakensiek, Moses Charikar, and Aviad Rubinstein. A simple sublinear algorithm for gap edit distance, 2020. `arXiv:2007.14368`.

14 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd ACM Symposium on Theory of Computing*, STOC '20, pages 685–698. ACM, 2020.

15 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. Almost-optimal sublinear-time edit distance in the low distance regime. In *Proceedings of the 54rd ACM Symposium on Theory of Computing*, STOC '22. ACM, 2022.

16 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '15, pages 79–97. IEEE Computer Society, 2015.

17 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *J. ACM*, 67(6), 2020.

18 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th ACM Symposium on Theory of Computing*, STOC '16, pages 712–725. ACM, 2016.

19 Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. Gap edit distance via non-adaptive queries: Simple and optimal, 2021. `arXiv:2111.12706`.

20 Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '20, pages 1101–1120. IEEE Computer Society, 2019.

21 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *Proceedings of the 52nd ACM Symposium on Theory of Computing*, STOC '20, pages 657–670. ACM, 2020.

22 Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '19, pages 334–347. IEEE Computer Society, 2019.

23 Hossein Jowhari. Efficient communication protocols for deciding edit distance. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA '12, pages 648–658. Springer, 2012.

24 Tomasz Kociumaka and Barna Saha. Sublinear-time algorithms for computing & embedding gap edit distance. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '20, pages 1168–1179. IEEE Computer Society, 2020.

25 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd ACM Symposium on Theory of Computing*, STOC '20, pages 699–712. ACM, 2020.

26 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *J. ACM*, 54(5):23, 2007.

27 Sebastian Wandelt, Dong Deng, Stefan Gerdjikov, Shashwat Mishra, Petar Mitankin, Manish Patil, Enrico Siragusa, Alexander Tiskin, Wei Wang, Jiaying Wang, and Ulf Leser. State-of-the-art in string similarity search and join. *SIGMOD Rec.*, 43(1):64–76, 2014.

A Proofs of Lemmas 3 and 5

In this section we provide proofs of Lemmas 3 and 5.

▶ **Lemma 3** (Divide and Conquer). *Let X, Y be length-n strings, and let $0 = j_0 < \cdots < j_B = n$. We write $X_i = X[\, j_{i-1} \,..\, j_i \,]$ and $Y_{i,s} = Y[\, j_{i-1} + s \,..\, j_i + s \,]$.*

- *For all shifts $s_1, \ldots, s_B$ we have that $\mathsf{ED}(X,Y) \le \sum_i \mathsf{ED}(X_i, Y_{i,s_i}) + 2|s_i|$.*
- *There are shifts $s_1, \ldots, s_B$ with $\sum_i \mathsf{ED}(X_i, Y_{i,s}) \le 2\mathsf{ED}(X,Y)$ and $2|s_i| \le \mathsf{ED}(X,Y)$ for all i.*

Proof. Lower Bound. Let us write $Y_i = Y_{i,0} = Y[\, j_{i-1} \,..\, j_i \,]$ (in analogy to the notation X_i). It is clear that $\mathsf{ED}(Y_{i,s_i}, Y_i) \le 2|s_i|$ by deleting and inserting at most $|s_i|$ symbols. Therefore, by several applications of the triangle inequality we have

$$\begin{aligned}\sum_{i=1}^{B} \mathsf{ED}(X_i, Y_{i,s_i}) + 2|s_i| &\ge \sum_{i=1}^{B} \mathsf{ED}(X_i, Y_{i,s_i}) + \mathsf{ED}(Y_{i,s_i}, Y_i) \\ &\ge \sum_{i=1}^{B} \mathsf{ED}(X_i, Y_i) \\ &\ge \mathsf{ED}(X,Y).\end{aligned}$$

Upper Bound. For the upper bound, let $A : \{0, \ldots, n\} \to \{0, \ldots, n\}$ denote an optimal alignment between X and Y (as defined in Section 2). Then

$$\mathsf{ED}(X,Y) = \sum_{i=1}^{B} \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]). \tag{4}$$

We pick $s_i = A(j_{i-1}) - j_{i-1}$. The first step is to prove that $2|s_i| \le \mathsf{ED}(X,Y)$, thereby proving the second item of the upper bound. To see this, we first express $\mathsf{ED}(X,Y)$ as the sum of two edit distances $\mathsf{ED}(X[\, 0 \,..\, j_{i-1} \,], Y[\, 0 \,..\, A(j_{i-1}) \,])$ and $\mathsf{ED}(X[\, j_{i-1} \,..\, n \,], Y[\, A(j_{i-1}) \,..\, n \,])$, using that A is an optimal alignment. Now, since the edit distance of two strings A, B is always at least $|\,|A| - |B|\,|$, we conclude that $\mathsf{ED}(X,Y) \ge 2|s_i|$.

The next and final part is to prove that $\sum_i \mathsf{ED}(X_i, Y_{i,s_i}) \le 2\mathsf{ED}(X,Y)$.

$$\begin{aligned}\mathsf{ED}(X_i, Y_{i,s_i}) &= \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_{i-1}) + j_i - j_{i-1} \,]) \\ &\le \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]) + |(A(j_i) - A(j_{i-1})) - (j_i - j_{i-1})| \\ &\le 2 \cdot \mathsf{ED}(X[\, i_{j-1} \,..\, i_j \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]),\end{aligned}$$

where in the last step we again used that the edit distance between two strings is at least their length difference. It follows that

$$\sum_{i=1}^{B} \mathsf{ED}(X_i, Y_{i,s_i}) \le 2 \sum_{i=1}^{B} \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]) \le 2\mathsf{ED}(X,Y). \qquad \blacktriangleleft$$

▶ **Lemma 5** (Expected Precision). *Let T be a B-ary precision tree with initial tolerance t and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$, and let v be a node in T. Then, conditioned on a high-probability event E, it holds that*

$$\mathbf{E}\left(\frac{1}{t_v} \,\middle|\, E \right) \le \frac{(\log n)^{O(\mathsf{depth}(v))}}{t} \le \frac{n^{o(1)}}{t}.$$

Proof. Recall that we assign $t_v = t_{\mathsf{parent}(v)} \cdot u_v/3$ for all non-root nodes in the precision tree, where the samples u_v are independent of each other and sampled from $\mathrm{Exp}(\lambda)$, the exponential distribution with parameter $\lambda = O(\log n)$. Recall that the exponential distribution has probability density function $f(x) = \lambda e^{-\lambda x}$, and thus

$$\mathop{\mathbf{P}}_{u\sim\mathrm{Exp}(\lambda)}(u \le x) = \int_{u=0}^{x} \lambda e^{-\lambda x} du = 1 - e^{-\lambda x} \le \lambda x.$$

We let E denote the event that $u_v \ge 1/\operatorname{poly}(n)$ for all nodes v. For any specific node v we have $u_v \ge 1/\operatorname{poly}(n)$ with high probability, and thus by a union bound E happens with high probability as well.

Next, we prove that conditioned on E, the expectation of $1/u$ for $u \sim \mathrm{Exp}(\lambda)$ is small:

$$\begin{aligned}
\mathop{\mathbf{E}}_{u\sim\mathrm{Exp}(\lambda)}(1/u \mid E) &\le \frac{1}{\mathbf{P}(E)} \cdot \int_{u=1/\operatorname{poly}(n)}^{\infty} \frac{1}{u} \cdot \lambda e^{-\lambda u} du \\
&\le \frac{1}{\mathbf{P}(E)} \cdot \left(\int_{u=1/\operatorname{poly}(n)}^{1} \frac{1}{u} \cdot \lambda e^{-\lambda u} du + \int_{u=1}^{\infty} \frac{1}{u} \cdot \lambda e^{-\lambda u} du \right) \\
&\le \frac{1}{\mathbf{P}(E)} \cdot O(\lambda \log n + 1) \\
&\le O(\log^2 n).
\end{aligned}$$

We now prove the claimed bound. Fix some node v and let $\mathsf{root}(T) = v_1, \ldots, v_{\mathsf{depth}(v)} = v$ denote the root-to-v path in the precision tree. Recall that $t_v = t \cdot (u_{v_1}/3) \cdots (u_{v_{\mathsf{depth}(v)}}/3)$. We have

$$\begin{aligned}
\mathbf{E}\left(\frac{1}{t_v} \,\middle|\, E\right) &= \frac{1}{t} \cdot \left(3 \cdot \mathop{\mathbf{E}}_{u\sim\mathrm{Exp}(\lambda)}\left(\frac{1}{u} \,\middle|\, E\right)\right)^{\mathsf{depth}(v)} \\
&\le \frac{(\log n)^{O(\mathsf{depth}(v))}}{t} \\
&\le \frac{n^{o(1)}}{t}.
\end{aligned}$$

For the last inequality, we used that $\mathsf{depth}(v) \le \log_B(n) = \widetilde{O}(\sqrt{\log n})$, and therefore the overhead becomes $\exp(\widetilde{O}(\sqrt{\log n})) = n^{o(1)}$. ◀

Polynomial Delay Algorithm for Minimal Chordal Completions

Caroline Brosse
Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, Limos, F-63000 Clermont-Ferrand, France

Vincent Limouzy
Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, Limos, F-63000 Clermont-Ferrand, France

Arnaud Mary
Université de Lyon, Université Lyon 1, CNRS, Laboratoire de Biométrie et Biologie Evolutive UMR 5558, 69622 Villeurbanne, France
ERABLE team, Inria Grenoble Rhône-Alpes, Villeurbanne, France

Abstract

Motivated by the problem of enumerating all tree decompositions of a graph, we consider in this article the problem of listing all the minimal chordal completions of a graph. In [8] (Pods 2017) Carmeli et al. proved that all minimal chordal completions or equivalently all proper tree decompositions of a graph can be listed in incremental polynomial time using exponential space. The total running time of their algorithm is quadratic in the number of solutions and the existence of an algorithm whose complexity depends only linearly on the number of solutions remained open. We close this question by providing a polynomial delay algorithm to solve this problem which, moreover, uses polynomial space.

Our algorithm relies on *Proximity Search*, a framework recently introduced by Conte and Uno [12] (Stoc 2019) which has been shown powerful to obtain polynomial delay algorithms, but generally requires exponential space. In order to obtain a polynomial space algorithm for our problem, we introduce a new general method called *canonical path reconstruction* to design polynomial delay and polynomial space algorithms based on proximity search.

2012 ACM Subject Classification Mathematics of computing → Enumeration; Mathematics of computing → Graph algorithms

Keywords and phrases Graph Algorithm, Algorithmic Enumeration, Minimal chordal completions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.33

Category Track A: Algorithms, Complexity and Games

Related Version *Previous Version*: `https://arxiv.org/abs/2107.05972`

Funding *Caroline Brosse*: ANR project GRALMECO (ANR-21-CE48-0004-01).
Vincent Limouzy: ANR project GRALMECO (ANR-21-CE48-0004-01) and by H2020 Marie Skłodowska-Curie Research and Innovation Staff Exchange European project 691161 "GEO-SAFE".
Arnaud Mary: ANR Project GrR (ANR-18-CE40-0032).

1 Introduction

Since its introduction by Dirac [13], the class of chordal graphs received a lot of attention. The many interesting properties of chordal graphs (*e.g.* a linear number of maximal cliques and minimal separators, a useful intersection model, a specific elimination ordering to cite a few) lead to the design of efficient algorithms for problems that are usually difficult on general graphs. On top of that, chordal graphs are closely connected to an important graph parameter called *treewidth* and its associated *tree-decomposition* introduced independently by Halin [14] and Robertson and Seymour [22].

Treewidth has played an important role in algorithmics for the last forty years, since its introduction and popularization by Robertson and Seymour. This popularity is deserved: given a tree decomposition of small width, many problems that are usually hard become solvable in polynomial time.

From the structure of maximal cliques known for chordal graphs, it is well-known that an optimal tree-decomposition can be computed in linear time on this class. For general graphs, one way to define the treewidth is to find the smallest value k such that the graph is a subgraph of a k-tree, that is to say, a subgraph of a chordal graph with maximum clique size at most $k+1$. Unfortunately, it was shown by Arnborg, Corneil & Proskurowski [1] that determining whether a graph is a partial k-tree (*i.e.* a subgraph of a k-tree) is NP-complete. In a different direction, the *minimum fill-in* problem asks to find the minimum number of edges to add to the graph in order to turn it into a chordal graph. Yannakakis [27] proved that the minimum fill-in problem is NP-hard.

Since the aforementioned problems are intractable, relaxations of these problems have been considered. The problem of computing chordal completion that are *inclusion-wise minimal*, has been intensively studied, either to find an optimal tree decomposition with an exponential-time algorithm or to find one tree decomposition in polynomial-time. Over the past decades, numerous polynomial algorithms have been provided to compute one minimal chordal completion [3, 5, 24, 15, 19, 20, 23].

However, from an application point of view, computing only one tree decomposition might not be satisfactory. For that reason, Carmeli et al. [7, 8] considered the problem of listing all the minimal chordal completions of a graph, and hence obtaining all the minimal tree decompositions of a graph. In the same line of research, Ravid et al. [21] considered the same problem by adding a requirement on the order in which solutions are produced.

An enumeration algorithm lists every solution of a given problem exactly once. Since the considered problem can have exponentially (in the input size) many solutions, the traditional complexity measures are no longer relevant. Instead, the common approach called *output sensitive* analysis is to bound the time complexity by a function of the input and the output size. Johnson et al. adapted in [17] the notion of polynomial time algorithm for enumeration algorithms. An algorithm is said to be *output polynomial* if its complexity can be bounded by a polynomial function expressed in the size of both the input and the output. As the number of solutions might be huge, this notion is not fine enough to capture the efficiency of the algorithm. For that reason, Johnson et al. further refined this notion by introducing the *incremental polynomial time*, meaning that the time used to produce a new solution of the problem is bounded by a polynomial in the size of the input and the size of all the already produced solutions. They strengthened the concept with the notion of *polynomial delay*: the time between the generation of two consecutive solutions is bounded by a polynomial in the size of the input only. The total running time of a polynomial delay algorithm depends only linearly on the size of the output, and since all solutions have to be outputted, this dependency is optimal.

For the enumeration of minimal triangulations of a graph, Carmeli et al. in [7, 8] presented a highly non-trivial algorithm. Their algorithm runs in *incremental polynomial time*; its total complexity is quadratic in the number of solutions, and to avoid duplication of solutions, requires exponential space. The algorithm is based on a result by Parra & Scheffler [20], according to which there is a bijection between the minimal triangulations of a graph and the maximal independent sets of a special graph of minimal separators of the input graph. In [8], they proved under *Exponential Time Hypothesis* that their approach cannot be improved to achieve polynomial delay. This intractability result also holds for the exponential space.

In [7, 8] and at a Dagstuhl seminar [4], it was left as an open problem whether a polynomial delay and/or a polynomial space algorithm for this problem could be obtained. We answer this question by the affirmative. Our main result is the following theorem.

▶ **Theorem 1.** *The minimal chordal completions of a graph can be computed in polynomial delay and polynomial space.*

In addition, our algorithm is simple and can be easily implemented. Since it is a polynomial delay algorithm, its total complexity is linear in the number of solutions. It contrasts with the quadratic one of [7, 8].

The paper is organised as follows. In Section 2 we recall the results and techniques used in algorithmic enumeration. In Section 3, we present the concepts that will be used for minimal chordal completions, and we present a first polynomial delay and exponential space algorithm. Then in Section 4 we present our main result, namely a polynomial delay and polynomial space algorithm to list all the minimal chordal completions of a graph without duplication. Finally, in Section 5 we formalize the framework used in Section 4 by introducing a new general method called *canonical path reconstruction* to design polynomial space enumeration algorithms.

2 Definitions and preliminary results

Throughout the article, standard graph theory notations will be used. A graph, always assumed to be simple, finite and undirected, is denoted by $G = (V, E)$ where V is the set of vertices and E is the set of edges. When the context is ambiguous, notations $V(G)$ and $E(G)$ can be used. For any $k \geq 3$, C_k denotes a cycle of length k.

We denote by E^c the set of *non-edges* of G, that is, the complement of the set E in the larger set of all two-element subsets of V. Then, for a subset $F \subset E^c$ we denote by $\bar{F}$ the set $E^c \setminus F$.

A graph is *chordal*, or *triangulated*, if it does not contain any induced cycle of length 4 or more. In other words, every cycle of length more than 3 of a chordal graph has at least one chord (*i.e.* an edge that connects non-consecutive vertices of the cyle).

Given a graph G and a supergraph H of G on the same vertex set, H is called a *triangulation* of G if H is chordal, and the edges of H which are not edges of G are called *fill edges*. In the whole paper, triangulations, or chordal completions, will be identified with the set of fill edges they induce. This is why for a graph $G = (V, E)$, we call a set $F \subseteq E^c$ a *chordal completion of* G if the supergraph $G_F = (V, E \cup F)$ is chordal. Let us denote by $\mathcal{F}$ the set of all chordal completions of G and by $\mathcal{F}_{\min}$ the set of minimal chordal completions of G, with respect to inclusion. A characterisation of minimal chordal completions is given in [23, Theorem 2]: a chordal completion F of a graph G is minimal *if and only if* for any $f \in F$, the graph $G_F \setminus \{f\}$ has an induced C_4.

From now on, $G = (V, E)$ is considered to be an arbitrary input graph that has no particular property and is therefore not assumed to be chordal. In the whole article, notation n is used for the number of vertices of G, that is, $n = |V|$. Finally, as our goal is to enumerate all minimal chordal completions of G, we may simply refer to them as "minimal completions".

The enumeration of minimal chordal completion takes place in the more general task of enumerating the minimal or the maximal subsets of a set system. A *set system* is a couple $(\mathcal{U}, \mathcal{F})$ where $\mathcal{U}$ is called the ground set and $\mathcal{F} \subseteq 2^{\mathcal{U}}$ is a family of subsets of $\mathcal{U}$. For a set system $(\mathcal{U}, \mathcal{F})$ we denote by $\mathcal{F}_{\min}$ (resp. $\mathcal{F}_{\max}$) the inclusion-wise minimal (resp. maximal) sets in $\mathcal{F}$. Many enumeration problems consist in enumerating the set $\mathcal{F}_{\max}$ or $\mathcal{F}_{\min}$ of a set

system $(\mathcal{U}, \mathcal{F})$. Of course the set $\mathcal{F}$ is usually not part of the input and we simply assume that a polynomially computable oracle membership is given, *i.e.* one can check whether a subset $F \subseteq \mathcal{U}$ belongs to $\mathcal{F}$ in time polynomial in $\mathcal{U}$.

In [12], the authors describe a method called *Proximity Search* (by canonical reconstruction) to design polynomial delay algorithms to enumerate $\mathcal{F}_{\max}$ of a set system $(\mathcal{U}, \mathcal{F})$. Given a set family $\mathcal{F}$ on a ground set $\mathcal{U}$, an *ordering scheme* π is a function which associates for every $F \in \mathcal{F}$ a permutation $\pi(F) = f_1, ..., f_{|F|}$ of the elements of F such that for all $i < |F|$, the ith prefix $\{f_1, ..., f_i\}$ of $\pi(F)$ is in $\mathcal{F}$. Notice that a set system $(\mathcal{U}, \mathcal{F})$ has an ordering scheme if and only if for every $F \in \mathcal{F}$, there exists $f \in F$ such that $F \setminus \{f\} \in \mathcal{F}$. Set systems having this property are called *accessible*.

The method described in [12] is based on the *proximity* between 2 solutions. While this notion has been defined in a very general context, most of its use cases are based on an ordering scheme. Given an ordering scheme π, and given $F_1, F_2 \in \mathcal{F}$ with $\pi(F_2) = f_1, ..., f_{|F_2|}$, the *$\pi$-proximity* between F_1 and F_2, denoted by $F_1 \tilde{\cap} F_2$, is the largest $i \leq |F_2|$ such that $\{f_1, ..., f_i\} \subseteq F_1$. It is worth noticing that the proximity relation between two solutions is not necessarily symmetric.

A polynomial-time computable function $\textsc{Neighbours} : \mathcal{F}_{\max} \to 2^{\mathcal{F}_{\max}}$ is called *π-proximity searchable*, if for every $F_1, F_2 \in \mathcal{F}_{\max}$ there exists $F' \in \textsc{Neighbours}(F_1)$ such that $F' \tilde{\cap} F_2 > F_1 \tilde{\cap} F_2$. Finally, we say by extension that an ordering scheme π of a set system $(\mathcal{U}, \mathcal{F})$ is *proximity searchable* if there exists a π-proximity searchable function Neighbours.

One of the major results of [12] is the following theorem.

▶ **Theorem 2** ([12]). *Let $(\mathcal{U}, \mathcal{F})$ be a set system and assume that one can find in polynomial time a maximal set $F_0 \in \mathcal{F}_{max}$. If $\mathcal{F}$ has a proximity searchable ordering scheme, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay.*

The proof of this theorem is based on the analysis of the *supergraph of solutions*. The supergraph of solutions is the directed graph having $\mathcal{F}_{\max}$ as vertex set and there is an arc from F_1 to F_2 if $F_2 \in \textsc{Neighbours}(F_1)$ (where Neighbours is the π-proximity searchable function). The proximity searchability of the ordering scheme implies that this supergraph of solutions is strongly connected. Then, the polynomial delay algorithm consists in starting from an arbitrary solution $F_0 \in \mathcal{F}_{\max}$ and performing a traversal of the supergraph of solutions, following at each step the arcs computed on the fly by Function Neighbours. The strong connectivity of the supergraph of solutions ensures that all solutions will be found. However, with this method, one needs to store in memory the solutions already visited, which in general results in the need of an exponential space.

One of the classical methods in enumeration to avoid the storage of the already outputted solutions is to define a parent-child relation over the set of solutions. It consists in associating to each $F \in \mathcal{F}_{\max} \setminus F_0$ a parent solution $\textsc{Parent}(F) \in \mathcal{F}_{\max}$ such that $F \in \textsc{Neighbours}(\textsc{Parent}(F))$. Given the Parent function, one can finally define the Children as $\textsc{Children}(F) := \{F' \in \textsc{Neighbours}(F) \mid \textsc{Parent}(F') = F\}$. The goal is to define the function Parent in such a way that the arcs of the supergraph of solutions defined by Function Children form a spanning arborescence of the supergraph of solutions, because in such a situation, one does not need to store the already visited solutions in a traversal of the supergraph of solutions. This method and the way to traverse the spanning arborescence of the supergraph is called *Reverse Search* [2] and has been used in many contexts.

Most applications of Reverse Search use a classical parent-child relation originally introduced in [25] for the enumeration of maximal independent sets of a graph. This specific parent-child relation has been used in general enumeration frameworks to obtain polynomial delay and polynomial space algorithms [18, 9]. Unfortunately, this method can only be used for hereditary set systems and it is not compatible with Proximity Search in general.

In [10] the authors show how to adapt this parent-child relation to commutable set systems (a class that strictly contains hereditary set systems) and in [11], it has been shown that the same method can be combined with Proximity Search to obtain polynomial delay and polynomial space algorithms for commutable set systems.

A set system is *commutable* if for any $X, Y \in \mathcal{F}$ with $X \subseteq Y$, the following two conditions hold:

- **Strong accessibility:**
  There exists $f \in Y \setminus X$ such that $X \cup \{f\} \in \mathcal{F}$
- **Commutability:**
  For any $a, b \in Y \setminus X$, if $X \cup \{a\} \in \mathcal{F}$ and $X \cup \{b\} \in \mathcal{F}$ then $X \cup \{a, b\} \in \mathcal{F}$

More formally, the authors introduce the notion of *prefix-closed* ordering schemes (*cf.* Section 5 for a definition) and they show the following theorem.

▶ **Theorem 3** ([11]). *Let $(\mathcal{U}, \mathcal{F})$ be a commutable set system and assume that one can find in polynomial time a maximal set $F_0 \in \mathcal{F}_{max}$. If $\mathcal{F}$ has a proximity searchable prefix-closed ordering scheme, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay and polynomial space.*

The polynomial space complexity comes from the definition of a parent-child relation that strongly relies on the commutability property. In the current paper, we introduce a new way of defining parent-child relations called *canonical path reconstruction* that does not require the commutability property. Since the set of chordal completions is not a commutable set system, this new method will be used in Section 4 to obtain a polynomial delay algorithm. Then, in Section 5 we improve Theorem 3 by proving the following which applies to non-commutable set systems.

▶ **Theorem 4.** *Let $(\mathcal{U}, \mathcal{F})$ be a set system and assume that one can find in polynomial time a maximal set $F_0 \in \mathcal{F}_{max}$. If $\mathcal{F}$ has a proximity searchable prefix-closed ordering scheme, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay and polynomial space.*

3 Enumeration of minimal chordal completions: polynomial delay

In this section we present a polynomial delay and exponential space algorithm to list all minimal chordal completions of a graph. In Section 3.1 we present all the concepts necessary to define an ordering scheme that will suit to minimal chordal completions. Then in Section 3.2, we present the neighbouring function and prove that this function is proximity searchable. As a consequence, together with Theorem 2, we obtain a polynomial delay algorithm.

3.1 Ordering scheme for chordal graphs

A class $\mathcal{C}$ of graphs is called *sandwich-monotone* [16] if for any two graphs H_1 and H_2 in $\mathcal{C}$ such that $E(H_1) \subsetneq E(H_2)$, there exists an edge $e \in E(H_2) \setminus E(H_1)$ such that $H_2 \setminus \{e\}$ is also in $\mathcal{C}$. This property is equivalent to being *strongly accessible* in terms of set systems (see for example [10]).

In [23, Lemma 2], the authors proved that the class of chordal graphs is sandwich-monotone. This immediately implies that if $F_1 \subsetneq F_2$ are two chordal completions of any graph, there exists $e \in F_2 \setminus F_1$ such that $F_2 \setminus \{e\}$ is a chordal completion, or equivalently there exists $e \in F_2 \setminus F_1$ such that $F_1 \cup \{e\}$ is a chordal completion. That rephrases as the following lemma.

▶ **Lemma 5.** *Chordal graphs are sandwich-monotone. In particular, chordal completions of G are sandwich-monotone.*

The general idea is then to use the sandwich-monotonicity of chordal completions to derive a suitable ordering scheme. The Proximity Search framework introduced in [10] has been designed to enumerate inclusion-wise maximal subsets of a set system. However, the same framework could be applied to the enumeration of inclusion-wise minimal subsets of a set system, by considering the complements of the solutions. To this effect, we will not work directly with the edge sets of the completions but rather with the sets of their *non-edges*. That is to say, we will consider for any completion F its complement $\bar{F} := E^c \setminus F$. The goal of this section is then to apply Proximity Search to the set $\bar{\mathcal{F}} := \{\bar{F} \mid F \in \mathcal{F}\}$ in order to enumerate the set $\bar{\mathcal{F}}_{\max} := \max(\bar{\mathcal{F}}) = \{\bar{F} \mid F \in \mathcal{F}_{min}\}$.

From now on, the elements of E^c (that is to say, the non-edges of G) are assumed to be arbitrarily ordered. The following definitions are similar to the ones given in [10] but are defined on $\mathcal{F}$ instead of $\bar{\mathcal{F}}$. Let F be a chordal completion of G and X be any subset of E^c. Consider the set X as the set from which we are allowed to remove elements. We define:

- $\textsc{Candidates}(F, X) := \{e \in X \cap F : F \setminus \{e\} \in \mathcal{F}\}$
- $\textsc{Candidates}(F) := \textsc{Candidates}(F, F)$
- $c(F, X) := \min(\textsc{Candidates}(F, X))$
- $c(F) := \min(\textsc{Candidates}(F))$

Now, given a chordal completion F and a set $X \subseteq E^c$, we denote by $\textsc{Del}(F, X)$ the chordal completion included in F by iteratively removing $c(F, X)$ from F at each step. Finally, we define $\textsc{Del}(F) := \textsc{Del}(F, F)$. Note that, for any F, X, computing $\textsc{Del}(F, X)$ corresponds to the following procedure.

Function Del(F, X).

while $\textsc{Candidates}(F, X) \neq \emptyset$ **do**
  remove $c(F, X)$ from F;
return F

▶ **Remark 6.** *By Lemma 5, if $F \in \mathcal{F}$, then $\textsc{Del}(F) \in \mathcal{F}_{min}$. That is to say, Del can be used to turn a chordal completion into a minimal one in a canonical way.*

Also, as E^c is a chordal completion (it corresponds to the clique completion) of G, the next lemma holds.

▶ **Lemma 7.** *If F is a minimal chordal completion of G, then $\textsc{Del}(E^c, \bar{F}) = F$.*

Proof. By Remark 6, it holds $\textsc{Del}(E^c, \bar{F}) \in \mathcal{F}_{\min}$, and $F \subset \textsc{Del}(E^c, \bar{F})$. As a result, since the only minimal solution containing F is F itself, then $\textsc{Del}(E^c, \bar{F}) = F$. ◀

The procedure followed to compute $\textsc{Del}(E^c, \bar{F})$ provides an ordering on the elements of $\bar{F}$ by considering the order in which the elements of $\bar{F}$ are removed to obtain F. From this ordering, we can define for any chordal completion F the canonical ordering $\textsc{Can}(\bar{F}) := s_1, ..., s_{|\bar{F}|}$ of $\bar{F}$ as follows:

- $s_1 := c(E^c, \bar{F})$;
- for all $1 \leq i < |\bar{F}|$, $s_{i+1} := c(E^c \setminus \{s_1, ..., s_i\}, \bar{F})$.

For a set $\bar{F}$ and its canonical ordering $\textsc{Can}(\bar{F}) := s_1, ..., s_{|\bar{F}|}$ we define $\bar{F}^i$ as the set of elements $\{s_1, ..., s_i\}$ of $\bar{F}$ and we define $F^i := E^c \setminus \bar{F}^i$. By definition of $\textsc{Candidates}(E^c, \bar{F})$, any prefix $\bar{F}^i$ of this ordering belongs to $\bar{\mathcal{F}}$. In other words, F^i denotes the chordal completion of G, not necessarily minimal, obtained by removing the ith prefix of $\textsc{Can}(\bar{F})$ from the clique completion. Thus, the canonical ordering Can is an ordering scheme of $\bar{\mathcal{F}}$.

This ordering will be used to measure the proximity between two solutions in order to call the Proximity Search algorithm for minimal chordal completions. Note that it can be computed in polynomial time for any solution, the complexity being essentially this of calling Function Del.

We now define the notion of proximity between two solutions that will be used in the sequel. For F_1 and F_2 two minimal completions of G, let $\textsc{Can}(\bar{F_2}) = f_1, ..., f_k$ be the canonical ordering of $\bar{F_2}$. The *proximity* $\bar{F_1} \tilde{\cap} \bar{F_2}$ between $\bar{F_1}$ and $\bar{F_2}$ is defined as the largest $i \leq k$ such that $\{f_1, ..., f_i\} \subseteq \bar{F_1}$. It is worth noticing that the proximity relation between two solutions is not necessarily symmetric. The notion defined here corresponds to the "standard" proximity measure that is usually adopted when using the Proximity Search framework [12, 10, 6]. Note that the proximity is defined here on the set of elements of E^c (that is to say, non-edges of G) which do *not* belong to the chordal completion. Since we are working on both the completions and their complement, by a slight abuse of language, when we speak about the proximity between two minimal completions, we actually mean the proximity between their complement sets.

3.2 Polynomial delay algorithm

We show in this section that the ordering scheme Can defined above is proximity searchable. Since the idea is to consider the complement sets of minimal completions rather than the completions themselves, we will seek to maximise the set of common *non-edges* between two minimal completions in order to increase the proximity. So, we actually show that Can is a proximity searchable ordering scheme of $\bar{\mathcal{F}}$.

The first goal is to define a suitable neighbouring function on the class of chordal graphs. Any solution must have a polynomial number of neighbours, each of them being computable in polynomial time in order to guarantee the polynomial delay when applying Proximity Search.

Given a chordal graph H and an edge $e = \{x, y\} \in E(H)$, the *flip operation* $\textsc{Flip}(H, e)$ consists in removing e from H, and turning the common neighbourhood of x and y into a clique. More formally, if $H' = \textsc{Flip}(H, e)$, then $V(H') = V(H)$ and $E(H') := (E(H) \setminus \{e\}) \cup \{uv \mid u, v \in N_H(x) \cap N_H(y)\}$. The flip operation is illustrated in Figure 1.

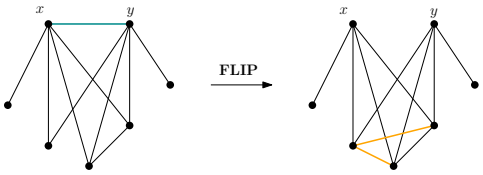


Figure 1 The flip operation on $e = \{x, y\}$. The common neighbourhood of x and y is turned into a clique.

Since H is chordal, the removal of e can create several chordless C_4s of which e was the only chord. We will see that completing the common neighbourhood of x and y into a clique adds the missing chords to all these C_4. In other words, the flip operation preserves the class of chordal graphs, as stated next.

▶ **Lemma 8.** *Let H be a chordal graph, and $e = \{x, y\}$ be an edge of H. Then the graph $H' := \textsc{Flip}(H, e)$ is also chordal.*

Proof. Assume (for contradiction) that H' contains a chordless cycle C of length $\ell > 3$.

Suppose C contains only edges of H. Since H is chordal, it does not contain long induced cycles. Then necessarily C is a C_4 of H of which e was the unique chord, otherwise there would be an induced cycle of length at least 4 (either C or a cycle made from e and edges of C) in the chordal graph H. By definition, the flip operation adds an edge between the other two vertices of C, meaning that C has a chord in H'. This cannot happen.

Therefore, C contains an edge $e' = \{z, t\}$ added by the flip operation (*i.e.* e' is an edge of H' but not of H). Since C contains the edge e', it does not contain any other fill edge added by the flip operation, otherwise a chord of C would also be added by the flip. Plus, we assumed that e' is added by the flip operation, so necessarily both z and t belong to $N_H(x) \cap N_H(y)$. Therefore, $P := C - \{e'\}$ is a $z - t$ path of H', disjoint from x and y. Remark that P is an induced path of H since it contains only edges of H, that is, z and t are in the same connected component of $H \setminus \{x, y\}$.

▷ Claim 9. Every vertex of P is in $N_H(x) \cap N_H(y)$.

Proof. Suppose that there exists a vertex s of P which is not a neighbour of x. We will show that there exists in H a chordless cycle containing s and x.

As s is not a neighbour of x, and z is by hypothesis, there exists $q \in N(x)$ a vertex of P that is between z and s, and closest to s on P. Similarly, let $r \in N(x)$ be a vertex of P that is between t and s, and closest to s.

The $q - r$ subpath of P induced by all vertices between q and r is therefore a chordless path of H of which all internal nodes are non-neighbours of x. It follows that adding x to this $q - r$ subpath creates a chordless cycle of length at least 4 in H. This is excluded, so all vertices of P are neighbours of x in H.

By symmetry in x and y, we also deduce that all vertices of P are neighbours of y.

The claim is proved: every vertex of P is in $N_H(x) \cap N_H(y)$. ◁

By the previous Claim, the flip operation turns the cycle C into a clique. Hence C has length 3, a contradiction. ◀

When dealing with chordal completions, the notation of the flip operation will be slightly adapted: for $F \in \mathcal{F}$, and $e \in F$ (e is always chosen among the fill edges of the completion), we write $\textsc{Flip}(F, e)$ instead of $\textsc{Flip}(G_F, e)$.

From Lemma 8, we are then able to deduce the following.

▶ **Lemma 10.** *Let F be a chordal completion of a graph G. If $F \in \mathcal{F}_{min}$, and $e \in F$, then $\textsc{Flip}(F, e) \in \mathcal{F}$.*

Proof. The only edge that is in F but not in $\textsc{Flip}(F, e)$ is e. Since $e \in F$, it implies that all edges of G are still in $\textsc{Flip}(F, e)$. As $\textsc{Flip}(F, e)$ is chordal by Lemma 8. ◀

Observe that $\textsc{Flip}(F, e)$ is a chordal completion of G that is not necessarily minimal.

We are now ready to explain the neighbouring function used to enumerate $\mathcal{F}_{\min}$. Given $F \in \mathcal{F}_{\min}$ and $e \in F$, we define the successor of F according to e as the minimal completion $\textsc{Succ}(F, e) := \textsc{Del}(\textsc{Flip}(F, e))$. We now define the neighbours of a solution F as $\textsc{Neighbours}(F) := \{\textsc{Succ}(F, f) | f \in F\}$. As a corollary of Remark 6 and Lemma 10, it is true that $\textsc{Succ}(F, e) \in \mathcal{F}_{\min}$. Observe also that each solution has a polynomial number of neighbours since $\textsc{Neighbours}(F) \leq |F|$.

Now the neighbouring function is properly defined, it is easy to notice that it can be computed in polynomial time. It remains to prove that the function NEIGHBOURS is CAN-proximity searchable. This will allow us to use Proximity Search on the set of minimal chordal completions of a graph G.

▶ **Lemma 11.** *Let F_1 and F_2 be two minimal chordal completions of a graph G. Let $f_1, \dots, f_k = \text{CAN}(\bar{F}_2)$ and let $i := \bar{F}_1 \tilde{\cap} \bar{F}_2$. Then the following statements hold:*
1. *$f_{i+1} \notin \bar{F}_1$;*
2. *$\overline{\text{SUCC}(F_1, f_{i+1})} \tilde{\cap} \bar{F}_2 > i$.*

Proof. 1. By definition of i as the length of the largest prefix of $\text{CAN}(\bar{F}_2)$ included in $\bar{F}_1$, it holds $f_{i+1} \notin \bar{F}_1$.

2. Let $F' := \text{FLIP}(F_1, f_{i+1})$, we will show that $\{f_1, ..., f_{i+1}\} \subseteq \bar{F'}$. Since $\text{SUCC}(F_1, f_{i+1}) \subseteq F'$, it will imply that $\{f_1, ..., f_{i+1}\} \subseteq \overline{\text{SUCC}(F_1, f_{i+1})}$, and finally $\overline{\text{SUCC}(F_1, f_{i+1})} \tilde{\cap} \bar{F}_2 \geq i+1$.

First, notice that by definition of FLIP, $f_{i+1} \notin F'$. Let x and y be the two endpoints of f_{i+1}. Assume for contradiction that an edge f_j, $j \leq i$ is added when completing $N_{G_{F_1}}(x) \cap N_{G_{F_1}}(y)$ into a clique. As stated earlier, the notation $\bar{F}_2^{\,i+1}$ is used to denote the set $\{f_1, ..., f_{i+1}\}$, which is included in $\bar{F}_2$, and F_2^{i+1} represents the associated chordal completion.

Let u and v be the two endpoints of f_j. Then x, u, y, v form a C_4 of which f_{i+1} was the unique chord in G_{F_1}. By definition of $\text{CAN}(\bar{F}_2)$, F_2^{i+1} is a chordal completion of G, and since neither the chords f_j nor f_{i+1} belong to it, at least one of the edges xu, uy, yv or vx does not belong to F_2^{i+1} since otherwise x, u, y, v would form a chordless C_4 in F_2^{i+1}. Assume without loss of generality that $xu \notin F_2^{i+1}$. Since $\bar{F}_2^{\,i+1} = \{f_1, ..., f_{i+1}\}$, there exists $\ell \leq i$ such that $xu = f_\ell$. But then we have found an edge $f_\ell \notin \bar{F}_1$ with $\ell \leq i$, which contradicts the assumption $\bar{F}_1 \tilde{\cap} \bar{F}_2 = i$.

Consequently, $\overline{\text{SUCC}(F_1, f_{i+1})} \tilde{\cap} \bar{F}_2 > i$. ◀

As a consequence, since the polynomial computable function SUCC is able to increase the proximity between 2 solutions, it proves that the ordering scheme defined by CAN is proximity searchable. More formally, the function $\text{NEIGHBOURS}(\bar{F}) := \{\bar{X} \mid X \in \text{NEIGHBOURS}(F)\}$ is a CAN-proximity searchable function and CAN is then a proximity searchable ordering scheme for $\bar{\mathcal{F}}_{max}$.

Therefore, by Theorem 2, $\bar{\mathcal{F}}_{max}$ or equivalently $\mathcal{F}_{\min}$ can be enumerated with polynomial delay. This is summarized in the following theorem.

▶ **Theorem 12.** *There exists a polynomial delay algorithm for the enumeration of minimal chordal completions of a graph.*

To prove that CAN is proximity searchable, we only used the fact that it is an ordering scheme of $\bar{\mathcal{F}}$. Thus any ordering scheme of $\bar{\mathcal{F}}$ is actually proximity searchable.

Yet, applying Proximity Search usually gives polynomial delay with exponential space: to avoid duplication it is necessary to store all the generated solutions in a lookup table. Each newly generated solution is then searched in the table in polynomial time.

4 Polynomial space

We would like to make the enumeration process work in polynomial space, since this could lead to an algorithm that is more usable in practice. As stated in Theorem 3, it is proven in [10] that if the ordering scheme is prefix-closed and $\mathcal{F}$ is a *commutable* set system, one can design a polynomial delay and polynomial space algorithm for the enumeration of $\mathcal{F}_{\max}$.

An ordering scheme π over $\mathcal{F}$ is called *prefix-closed*, if for all $F \in \mathcal{F}$, there exists an ordering $<_F$ of the elements of $\textsc{Candidates}(F)$ such that $\pi(\bar{F}) = f_1, ..., f_\ell$ *if and only if* for any $i < \ell$, it holds $f_{i+1} = \min_{<_{F^i}}(\textsc{Candidates}(F^i) \cap \bar{F})$. We know by definition of $\textsc{Can}$ that this ordering is prefix-closed (with the ordering of E^c). By Section 3, $\textsc{Can}$ is also proximity searchable. As stated earlier, chordal completions form a strongly accessible set system since they are sandwich-monotone, and so are their complements. However, the set of chordal completions is not a commutable set system, and neither is the set of their complements.

This section is devoted to the definition of a suitable parent-child relation over $\mathcal{F}_{\min}$ which does not require the system to be commutable, in order to obtain a polynomial space algorithm. In the supergraph of solutions, we identify a reference solution named F_0 and we manage to relate any other solution F to F_0. To uniquely determine the parent of a solution, we introduce a new concept called *canonical path reconstruction*. The idea is, for any solution F distinct from F_0, to identify a path in the supergraph of solutions from F_0 to F in a unique manner. Then the parent of solution F is defined as the immediate predecessor of F on this path. In addition, we are able to compute this path in polynomial time for any solution.

To ensure that such a path can be uniquely determined and computed in polynomial time we rely on the specific structure of chordal completions and more specifically on the fact that the canonical ordering we defined on chordal completions is prefix-closed. To the best of our knowledge, it is the first algorithm to define a parent-child relation in this manner.

The parent-child relation will define a spanning arborescence of the supergraph of solutions rooted at F_0. This way, the enumeration algorithm sums up in performing a traversal of the tree in a depth-first manner.

It is somehow counter-intuitive to observe that the canonical path of a solution is not necessarily part of the final arborescence rooted at F_0. This canonical path is only computed to find the last solution in the path before F to define it as the parent of F. We will prove that while the so defined parent-child relation does not exactly follow the canonical paths, it still forms a spanning arborescence of the solution set rooted at F_0.

Let us consider the total ordering $\prec$ over $\bar{\mathcal{F}}_{min}$ defined as $\bar{F}_1 \prec \bar{F}_2$ if $\textsc{Can}(\bar{F}_1)$ is lexicographically smaller than $\textsc{Can}(\bar{F}_2)$.

▶ **Lemma 13.** *Let $F_1, F_2 \in \mathcal{F}_{min}$ with $\textsc{Can}(\bar{F}_1) = f_1, ..., f_\ell$ and $\textsc{Can}(\bar{F}_2) = t_1, ..., t_k$. Assume furthermore that there exists $j \leq \ell$ such that $\{f_1, ..., f_j\} \subset \bar{F}_2$. Then one of the two possibilities holds:*

1. *$\bar{F}_2 \prec \bar{F}_1$;*
2. *$f_i = t_i$ for all $i \leq j$.*

Proof. Let i^* be the smallest index such that $t_{i^*} \neq f_{i^*}$. If $i^* > j$, then 2 holds. Else, $i^* \leq j$. In this case, we deduce from the minimality of i^* that

$$\bar{F_1}^{i^*-1} = \{f_1, ..., f_{i^*-1}\} = \{t_1, ..., t_{i^*-1}\} = \bar{F_2}^{i^*-1} .$$

Since $f_{i^*} \in \bar{F}_2$ and since $f_{i^*} \in \textsc{Candidates}(F_1^{i^*-1}, \bar{F}_1)$ by definition of $\textsc{Can}(\bar{F}_1)$, we deduce $f_{i^*} \in \textsc{Candidates}(F_2^{i^*-1}, \bar{F}_2)$. The canonical ordering $\textsc{Can}$ is prefix-closed, therefore we know that $t_{i^*} = \min(\textsc{Candidates}(F_2^{i^*-1}, \bar{F}_2))$. We deduce from it that $t_{i^*} \leq f_{i^*}$ and since $t_{i^*} \neq f_{i^*}$, we have $t_{i^*} < f_{i^*}$ which proves $\bar{F}_2 \prec \bar{F}_1$. ◀

The algorithm starts by computing $F_0 := \textsc{Del}(E^c)$ in polynomial time. This solution F_0 will be used as the reference solution. Note that we could start from any solution and the results would still be valid, but for simplicity we start from a solution that can easily be identified.

For a minimal chordal completion F with canonical ordering $\textsc{Can}(\bar{F}) := f_1, ..., f_\ell$, we define the *canonical path of* F as the sequence of minimal completions $F_0, ..., F_k = F$ starting at F_0 such that for all $1 \leq i \leq k$, $F_i = \textsc{Succ}(F_{i-1}, f_{(\bar{F}_{i-1} \tilde{\cap} \bar{F})+1})$. Remark that the edge $f_{(\bar{F}_{i-1} \tilde{\cap} \bar{F})+1}$ is part of F_{i-1} and the call to Succ is correct. Let us also note that by construction, the proximity strictly increases along the path. Then, $\textsc{Parent}(F)$ is defined for any $F \neq F_0$ as the last minimal completion F_{k-1} in the canonical path of F.

As remarked before, except F_{k-1}, the other ancestors of F may not be part of its canonical path. For instance F_{k-2} may not be the parent of F_{k-1}.

The detailled Parent function is presented 11.

Function Parent(F).

Compute $f_1, \ldots, f_\ell := \textsc{Can}(\bar{F})$;
Compute $F_{current} := \textsc{Del}(E^c)$;
while $F_{current} \neq F$ **do**
  $f := f_{(\overline{F_{current}} \tilde{\cap} \bar{F})+1}$;
  $F_{prec} := F_{current}$;
  $F_{current} := \textsc{Succ}(F_{prec}, f)$;
return F_{prec}

Then we define $\textsc{Children}(F) := \{F' \in \textsc{Neighbours}(F) \mid \textsc{Parent}(F') = F\} = \{\textsc{Succ}(F, e) \mid e \in F,\ \textsc{Parent}(\textsc{Succ}(F, e)) = F\}$.

▶ **Lemma 14.** *Let F be a minimal chordal completion of G, and let $F_0, \ldots, F_k = F$ be the canonical path of F. For all $j \leq k$, $\bar{F}_j \tilde{\cap} \bar{F} \geq j$.*

Proof. By construction of the canonical path of F, the proximity strictly increases at each step by at least one. ◀

By Lemma 14 the length of the canonical path of F is smaller than $|\bar{F}|$. Since the construction of the canonical path of F is done by applying at most $|\bar{F}|$ times Function Succ, $\textsc{Parent}(F)$ is computable in polynomial time for any $F \in \mathcal{F}_{\min}$.

Now, since the computation of $\textsc{Children}(F)$ is done by applying Function Succ followed by Function Parent at most $|F|$ times, it is also computable in polynomial time.

Once the Children function is defined, one just has to apply the classical algorithm to visit and output all solutions whose high-level description is presented in Algorithm 1. To prove the correctness of the algorithm, we just have to prove that the function Children defines a spanning arborescence on the set of solutions rooted at F_0.

Observe that the naive implementation of Algorithm 1 may use exponential space, since the height of the recursion tree might be exponential. Indeed the recursion tree corresponds to the arborescence defined by the parent-child relation and we are not able to guarantee that its height is polynomial. However, since the functions Parent and Children are computable in polynomial time we don't need to store the state of each recursion call. The classical trick introduced in [25] and formalized in [2] as part of the Reverse Search algorithm is to use Function Parent to perform the backtrack operation on the fly. Indeed this function is able to navigate backward in the tree, removing the need of keeping in memory all recursion calls. Thus, the algorithm only needs to keep in memory the current solution.

▶ **Theorem 15.** *Algorithm 1 outputs all minimal chordal completions of the input graph without duplication, with polynomial delay and using polynomial space.*

Algorithm 1 Efficient enumeration of minimal chordal completions.

input : A graph $G = (V, E)$
output : All minimal chordal completions of G

$F_0 := \textsc{Del}(E^c)$;
Call $\textsc{enum}(F_0)$;

Function `enum` *(F)*:
    `/* Output` F `if recursion depth is even */`;
    foreach $F' \in \textsc{Children}(F)$ **do**
        `enum`(F');
    `/* Output` F `if recursion depth is odd */`;

Proof. Assume that some solutions of $\mathcal{F}_{\min}$ are not outputted by the algorithm, and let F be the smallest one with respect to $\prec$ with $\textsc{Can}(\bar{F}) := f_1, ..., f_\ell$. Let $F_0, ..., F_k = F$ be the canonical path of F. We prove that all F_i, $i < k$ will be outputted by the algorithm. This way, $\textsc{Children}(F_{k-1})$ will be produced, contradicting the fact that F has not been outputted.

Let i be the smallest index such that F_i has not been processed by the algorithm. By minimality of $\bar{F}$ with respect to $\prec$, we know that $\bar{F} \prec \bar{F}_i$. Hence by Lemmas 13 and 14, $f_1, ..., f_i$ is a prefix of $\textsc{Can}(\bar{F}_i)$. So the canonical path of F_i is precisely $F_1, ..., F_i$ and $\textsc{Parent}(F_i) = F_{i-1}$. Now, by minimality of i, we know that F_{i-1} has been outputted by the algorithm, and $F_i \in \textsc{Children}(F_{i-1})$ will be outputted during the processing of F_i.

To obtain a polynomial delaya and polynomial space algorithm, we rely on the fact that the Parent and Childrenfunctions admit a polynomial time complexity. Then with this conditions fulfilled, it remains to show that Algorithm 1 can be implemented so that the delay required to produce a new solution is polynomial in the size of the input and that the memory space is polynomial. Algorithm 1 can be seen as a traversal of the tree of solutions defined by the Parent-Child relation. The time a solution is produced will depend of the height of the solution in the solution tree. It is either produced at the beginning of the call or at the end wheter the height is odd or even. This classic technique in enumeration grant the delay. For this same traversal of the solution tree, Uno [26] developped techniques, that provided the Parent and Children functions admit a polynomial time complexity, will be able to traverse the tree of solutions only using polynomial space. ◀

5 Canonical path reconstruction: a general approach

In fact, the result we proved for chordal completions is part of a more general framework. In this section, we show how to extend the algorithm obtained for chordal graphs, in order to apply it to other graph classes.

As stated in Theorem 3, the authors of [11] proved that if a proximity searchable ordering scheme is prefix-closed, then one can design a polynomial delay and polynomial space algorithm to enumerate $\mathcal{F}_{\max}$ whenever $\mathcal{F}$ is a commutable set system. We prove here that the same remains true even if the system is not commutable. To prove that, we adapt the concept of defined earlier *canonical path reconstruction* in a more general setting to define parent-child relations over $\mathcal{F}_{\max}$ which does not require the commutability condition.

Given $F \in \mathcal{F}$ we denote by F^+ the set $\{x \in \mathcal{U} : F \cup \{x\} \in \mathcal{F}\}$ of candidates for F. Assuming that an ordering scheme π is fixed, we denote by F^i the ith prefix of F according to π.

The notion of prefix-closed ordering has been introduced in [10] and it appears to be a key property to design parent-child relations. Intuitively assume that for each non maximal element of $F \in \mathcal{F}$ we have a preference given by a total ordering $<_F$ over the potential elements that can be added to F (i.e. a total ordering over F^+). This preference depends on the set F and the preference among two elements may vary from one F to another. Then an ordering scheme π is said to be prefix-closed if for each prefix F_i of F the next element f_{i+1} corresponds to the most preferred element that remains in F according to the preference relation $<_{F^i}$. More formally :

▶ **Definition 16.** *An ordering scheme π is* prefix-closed*, if there exists an ordering $<_F$ of the elements of F^+ for each $F \in \mathcal{F}$ that verify the following condition : For every $F \in \mathcal{F}$, $\pi(F) = f_1, ..., f_\ell$ if and only if $f_{i+1} = \min_{<_{F^i}}(F^{i^+} \cap F)$ for any $i < \ell$.*

The goal of this section is to prove the following theorem.

▶ **Theorem 17.** *If π is a polynomial time computable ordering scheme for $\mathcal{F}$ which is both proximity searchable and prefix-closed, then $\mathcal{F}_{max}$ can be enumerated with polynomial delay and polynomial space.*

Notice that any strongly accessible set system has a prefix-closed ordering scheme. Similarly as the one shown for chordal completions one can choose an arbitrary ordering on the elements of $\mathcal{U}$ and simply define $\pi(F) := f_1, ..., f_\ell$ with $f_{i+1} := \min(F^{i^+} \cap F)$. The strong accessibility ensures that this ordering is well defined.

To prove Theorem 17, we define the general parent-child relation over $\mathcal{F}_{\max}$ based on the *canonical path reconstruction method*. Let then π be a prefix-closed, proximity searchable ordering scheme on the set family $\mathcal{F}$.

Whenever an ordering scheme π is proximity searchable, the supergraph of solutions defined by the π-proximity searchable function Neighbours is strongly connected. The goal will be to choose a reference solution F_0 and to identify for each other solution F a canonical path from F_0 to F. The parent of F will be defined as the last solution of this path.

For a prefix-closed ordering scheme π, let us define a total ordering $\prec_\pi$ over $\mathcal{F}_{\max}$. Let $F_1, F_2 \in \mathcal{F}_{\max}$, $\pi(F_1) = t_1, ..., t_{|F_1|}$, $\pi(F_2) = f_1, ..., f_{|F_2|}$ and let $j \geq 0$ be the largest index such that $F_1^j = F_2^j$. Let us denote $F := F_1^j = F_2^j$. Then we have $F_1 \prec_\pi F_2$ if $t_{j+1} <_F f_{j+1}$.

We can now define what will be the canonical path of a solution $F \in \mathcal{F}_{\max}$. To this end, we need to define a function Next : $\mathcal{F}_{\max} \times \mathcal{F}_{\max} \to \mathcal{F}_{\max}$ such that given $F_1, F_2 \in \mathcal{F}_{\max}$ with $F_1 \tilde{\cap} F_2 = i$, $\textsc{Next}(F_1, F_2) \in \{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$. Since we assumed that π is proximity searchable, we know that the set $\{F \in \textsc{Neighbours}(F_1) \mid F \tilde{\cap} F_2 > i\}$ is not empty. Hence, to define $\textsc{Next}(F_1, F_2)$, one just has to choose deterministically an element of the set $\{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$. If the function Neighbours produces solutions in a deterministic order, then $\textsc{Next}(F_1, F_2)$ can be chosen as the first $F \in \textsc{Neighbours}(F_1)$ such that $F \tilde{\cap} F_2 > i$. Otherwise and to be as general as possible, let us define $\textsc{Next}(F_1, F_2) := \min_{Lex}\{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$ as the lexicographically smallest set of $\{F \in \textsc{Neighbours}(F_1) : F \tilde{\cap} F_2 > i\}$.

Given a reference solution F_0, the *canonical path* of $F \in \mathcal{F}_{\max}$ is then defined as the sequence $F_0, ..., F_k$ of elements of $\mathcal{F}$ such that:

- $F_k = F$
- $F_{i+1} = \textsc{Next}(F_i, F)$ for all $i < k$

Each $F \in \mathcal{F}_{\max}$ has a canonical path. Indeed, applying Function Next increases the proximity with F, until F is eventually reached. We define the parent of a solution F of canonical path $F_0, ..., F_k$ as $\textsc{Parent}(F) := F_{k-1}$. The set of children of F is then defined as $\textsc{Children}(F) := \{F' \in \textsc{Neighbours}(F) \mid \textsc{Parent}(F') = F\}$.

▶ **Lemma 18.** *Let $F \in \mathcal{F}_{max}$ and let $F_0, ..., F_k$ be its canonical path. Then for all $i < j \leq k$, $F_j \tilde{\cap} F > F_i \tilde{\cap} F$.*

Proof. By construction of the canonical path of F the proximity increases at each step by at least 1. ◀

The immediate following Corollary ensures that canonical paths are of polynomial size.

▶ **Corollary 19.** *If $F \in \mathcal{F}_{max}$, then its canonical path is of size at most $|F| + 1$.*

▶ **Proposition 20.** *The function PARENT can be computed in time $O(\mathcal{P} + f^2\mathcal{N})$ where f is a maximum size of a solution in $\mathcal{F}_{max}$, $\mathcal{N}$ is the complexity of function NEIGHBOURS and $\mathcal{P}$ is the time needed to compute the function π.*

Proof. To compute the PARENT(F) we first need to compute $\pi(F)$ and then to compute the canonical path of F. By Corollary 19, the path is of length at most f, so we only need to call f times the function NEXT. To compute NEXT(T, F), we compute NEIGHBOURS(T) and for each $F' \in$ NEIGHBOURS(T) we compute the proximity between F' and F and check whether F' is lexicographically smaller than the current best candidate (notice that we don't need to compute $\pi(F')$). Since both operations can be done in $O(f)$, and since |NEIGHBOURS(T)| $< \mathcal{N}$ the function NEXT can be computed in time $O(f\mathcal{N})$. ◀

Since the computation of CHILDREN(F) is done by computing first the set NEIGHBOURS(F) and then filtering it according to the function PARENT, we obtain the following complexity for the function CHILDREN.

▶ **Corollary 21.** *The function CHILDREN can be computed in time $O(\mathcal{P} + f^2\mathcal{N}^2)$ where f is a maximum size of a solution in $\mathcal{F}_{max}$, $\mathcal{N}$ is the complexity of function NEIGHBOURS and $\mathcal{P}$ is the time needed to compute the function π.*

▶ **Lemma 22.** *Let $F \in \mathcal{F}_{max}$ with $\pi(F) = f_1, ..., f_\ell$, let $F_0, ..., F_k$ be its canonical path and let $i \leq k$. If $f_1, ..., f_{F_i \tilde{\cap} F}$ is a prefix of $\pi(F_i)$, then the canonical path of F_i is $F_0, ..., F_i$.*

Proof. Let $T_0, ..., T_h$ be the canonical path of F_i. By definition we have $T_0 = F_0$. Assume that $T_0, ..., T_j = F_0, ..., F_j$ for some $j < i$. By Lemma 18, we know $F_j \tilde{\cap} F < F_i \tilde{\cap} F$, so there exists $r < F_i \tilde{\cap} F$ such that $\{f_1, ..., f_r\} \subseteq F_j$ and $f_{r+1} \notin F_j$.

Hence, since $f_1, ..., f_{r+1}$ is a prefix of $\pi(F_i)$, we have $r = F_j \tilde{\cap} F = F_j \tilde{\cap} F_i < F_i \tilde{\cap} F$. Thus

$$\begin{aligned} F_{j+1} = \text{NEXT}(F_j, F) &= \min_{<_{Lex}} \{F' \in \text{NEIGHBOURS}(F_j) : \{j_1, ..., j_{r+1}\} \subseteq F'\} \\ &= \min_{<_{Lex}} \{F' \in \text{NEIGHBOURS}(T_j) : \{j_1, ..., j_{r+1}\} \subseteq F'\} = \text{NEXT}(T_j, F_i) = T_{j+1}. \end{aligned}$$

This concludes the proof. ◀

The following key lemma is the generalisation of Lemma 13 in the case of chordal completions.

▶ **Lemma 23.** *Let $F_1, F_2 \in \mathcal{F}$ with $\pi(F_1) = f_1, ..., f_\ell$ and $\pi(F_2) = t_1, ..., t_k$. Assume furthermore that there exists $j \leq \ell$ such that $\{f_1, ..., f_j\} \subseteq F_2$, then one of the two possibilities holds:*

1. *$F_2 \prec_\pi F_1$;*
2. *$f_i = t_i$ for all $i \leq j$.*

Proof. Let i be the smallest index such that $t_i \neq f_i$. If $i > j$, then 2 holds and we are done.

Assume that $i \leq j$. By minimality of i, we have $F_1^{i-1} = F_2^{i-1} =: L$. Since $f_i \in F_2$ and since $f_i \in F_1^{i-1^+}$, we know that $f_i \in L^+ \cap F_2$. Therefore, as $t_i = \min_{<_L}(L^+ \cap F_2)$, it holds $t_i \leq_L f_i$, and since $t_i \neq f_i$, we have $t_i <_L f_i$. It proves $F_2 \prec_\pi F_1$. ◀

The previous lemma is the one that makes the canonical path reconstruction method possible. Independently of the prefix-closed property, if one can find a total ordering $\prec$ over $\mathcal{F}_{\max}$ which satisfies Lemma 23, then the canonical path reconstruction method is applicable.

Finally, to prove Theorem 17, it remains to show that the polynomial time computable function Children defines a spanning arborescence on $\mathcal{F}_{\max}$. As already mentioned, the application of the Reverse Search algorithm in [2] would output all solutions without repetition with polynomial delay and polynomial space which will conclude the proof of Theorem 17.

▶ **Theorem 24.** *Children defines a spanning arborescence on $\mathcal{F}_{max}$ rooted at F_0.*

Proof. Recall that the supergraph of solution defined by Function Children is the directed graph on $\mathcal{F}_{\max}$ with arcs set $\{(F_i, F_j) \mid F_j \in \textsc{Children}(F_i)\}$.

Since each $F \in \mathcal{F}_{\max}$, $F \neq F_0$ has only one in-neighbour $\textsc{Parent}(F)$ it is sufficient to show that for all $F \in \mathcal{F}_{\max}$, there exists a path from F_0 to F. Let's denote by $\mathcal{R}(F_0) \subseteq \mathcal{F}_{\max}$ the sets $F \in \mathcal{F}_{\max}$ for which there exists a path from F_0 to F.

Assume for contradiction that $\mathcal{R}(F_0) \neq \mathcal{F}_{\max}$ and let $F := \min_{\prec_\pi}(\mathcal{F}_{\max} \setminus \mathcal{R}(F_0))$. Let $\pi(F) = f_1, ..., f_\ell$ and let $F_0, ..., F_k$ be the canonical path of F. Now, consider the the smallest index $i^* \leq k$ such that $F_{i^*} \notin \mathcal{R}(F_0)$. If $i^* = k$ then $F_{k-1} \in \mathcal{R}(F_0)$ and since $F_{k-1} = \textsc{Parent}(F)$, F would belong to $\textsc{Children}(F_{k-1})$ and then F would belong to $\mathcal{R}(F_0)$. So assume that $i^* < k$.

Let $j := F_{i^*} \tilde{\cap} F$. By minimality of F, we have $F \prec_\pi F_{i^*}$ and since $\{f_1, ..., f_j\} \subseteq F_{i^*}$, by Lemma 23, the $j^{(th)}$ prefix of $\pi(F_{i^*})$ is $f_1, ..., f_j$. Now by Lemma 22 the canonical path of F_{i^*} is $F_0, ..., F_{i^*}$, thus F_{i^*} has F_{i^*-1} as parent. By minimality of i^*, we know that $F_{i^*-1} \in \mathcal{R}(F_0)$ and since $F_{i^*} \in \textsc{Children}(F_{i^*-1})$, we deduce $F_i^* \in \mathcal{R}(F_0)$. ◀

References

1. Stefand Arnbog, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, 1987. doi:10.1137/0608024.
2. David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete applied mathematics*, 65(1-3):21–46, 1996.
3. Anne Berry, Jean RS Blair, Pinar Heggernes, and Barry W Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.
4. Endre Boros, Benny Kimelfeld, Reinhard Pichler, and Nicole Schweikardt. Enumeration in data management (dagstuhl seminar 19211). Technical report, Dagstuhl Seminar, 2019. doi:10.4230/DagRep.9.5.89.
5. Vincent Bouchitté and Ioan Todinca. Minimal triangulations for graphs with "few" minimal separators. In *European Symposium on Algorithms*, pages 344–355. Springer, 1998.
6. Caroline Brosse, Aurélie Lagoutte, Vincent Limouzy, Arnaud Mary, and Lucas Pastor. Efficient enumeration of maximal split subgraphs and sub-cographs and related classes. *arXiv preprint*, 2020. arXiv:2007.01031.
7. Nofar Carmeli, Batya Kenig, and Benny Kimelfeld. Efficiently enumerating minimal triangulations. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 273–287. ACM, 2017. doi:10.1145/3034786.3056109.

8 Nofar Carmeli, Batya Kenig, Benny Kimelfeld, and Markus Kröll. Efficiently enumerating minimal triangulations. *Discrete Applied Mathematics*, In Press, 2020.

9 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147–1159, November 2008. doi:10.1016/j.jcss.2008.04.003.

10 Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Listing maximal subgraphs satisfying strongly accessible properties. *SIAM Journal on Discrete Mathematics*, 33(2):587–613, 2019.

11 Alessio Conte, Andrea Marino, Roberto Grossi, Takeaki Uno, and Luca Versari. Proximity Search For Maximal Subgraph Enumeration. *arXiv:1912.13446 [cs]*, July 2020. arXiv:1912.13446.

12 Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1179–1190, 2019.

13 G. A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961. doi:10.1007/BF02992776.

14 Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976. doi:10.1007/BF01917434.

15 Pinar Heggernes. Minimal triangulations of graphs: a survey. *Discrete Mathematics*, 306(3):297–317, 2006.

16 Pinar Heggernes and Charis Papadopoulos. Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. *Theoretical Computer Science*, 410(1):1–15, 2009.

17 David S Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

18 E. Lawler, J. Lenstra, and A. Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM J. Comput.*, 9:558–565, 1980.

19 Tatsuo Ohtsuki. A fast algorithm for finding an optimal ordering for vertex elimination on a graph. *SIAM Journal on Computing*, 5(1):133–145, 1976.

20 Andreas Parra and Petra Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1-3):171–188, 1997.

21 Noam Ravid, Dori Medini, and Benny Kimelfeld. Ranked enumeration of minimal triangulations. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, pages 74–88, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3294052.3319678.

22 Neil Robertson and P.D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

23 Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

24 Ioan Todinca. *Aspects algorithmiques des triangulations minimales des graphes*. PhD thesis, École normale supérieure (Lyon), 1999.

25 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM J. Comput.*, 6(3):505–517, September 1977. doi:10.1137/0206036.

26 Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. *National Intsitute of Informatics, Tech. report*, E 4, 2003. URL: http://www.nii.ac.jp/TechReports/public_html/03-004E.pdf.

27 Mihalis Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2:77–79, 1981. doi:10.1137/0602010.

Unique Assembly Verification in Two-Handed Self-Assembly

David Caballero
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Timothy Gomez
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Robert Schweller
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Tim Wylie
Department of Computer Science, University of Texas Rio Grande Valley, TX, USA

Abstract

One of the most fundamental and well-studied problems in tile self-assembly is the Unique Assembly Verification (UAV) problem. This algorithmic problem asks whether a given tile system uniquely assembles a specific assembly. The complexity of this problem in the 2-Handed Assembly Model (2HAM) at a constant temperature is a long-standing open problem since the model was introduced. Previously, only membership in the class coNP was known and that the problem is in P if the temperature is one ($\tau = 1$). The problem is known to be hard for many generalizations of the model, such as allowing one step into the third dimension or allowing the temperature of the system to be a variable, but the most fundamental version has remained open.

In this paper, we prove the UAV problem in the 2HAM is hard even with a small constant temperature ($\tau = 2$), and finally answer the complexity of this problem (open since 2013). Further, this result proves that UAV in the staged self-assembly model is coNP-complete with a single bin and stage (open since 2007), and that UAV in the q-tile model is also coNP-complete (open since 2004). We reduce from Monotone Planar 3-SAT with Neighboring Variable Pairs, a special case of 3SAT recently proven to be NP-hard. We accompany this reduction with a positive result showing that UAV is solvable in polynomial time with the promise that the given target assembly will have a tree-shaped bond graph, i.e., contains no cycles. We provide a $\mathcal{O}(n^5)$ algorithm for UAV on tree-bonded assemblies when the temperature is fixed to 2, and a $\mathcal{O}(n^5 \log \tau)$ time algorithm when the temperature is part of the input.

2012 ACM Subject Classification Theory of computation → Computational geometry; Applied computing → Computational biology; Theory of computation → Problems, reductions and completeness; Theory of computation → Self-organization

Keywords and phrases self-assembly, unique assembly verification, 2-handed assembly model

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.34

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2112.05070

Funding This research was supported in part by National Science Foundation Grant CCF-1817602.

1 Introduction

Since the inception of tile self-assembly [28], one of the most important algorithmic questions has been determining if a given tile system uniquely self-assembles into a specific assembly structure. This basic algorithmic question, termed the Unique Assembly Verification (UAV) problem, is fundamental for efficiently checking if a designed tile system acts as intended, and is tantamount to the design of an efficient simulator for a tile self-assembly model. Thus, UAV has been a central question for every self-assembly model.

Although many different self-assembly models have been proposed in order to simulate different laboratory or experimental setups, two premiere models have emerged as the primary foci of study. First, is the seeded Abstract Tile Assembly Model (aTAM) [28], in which singleton tiles attach one by one to a growing seed if sufficient bonding strength exists based on glue types of attaching tiles. This model has had many foundational results in recent years showing the limits related to intrinsic universality and program-size complexity [20, 21]. The second model is the hierarchical Two-Handed Tile Assembly Model (2HAM) [9], where any two producible assemblies may be combined (one in each of two hands) to create a new producible assembly provided there is sufficient bonding strength between the two pieces. Many foundational results that are known for the aTAM are still open for the 2HAM.

The 2HAM has been shown to be more powerful than the aTAM in its ability to build infinite fractal patterns [11, 18], its program-size efficiency for finite shapes [8], and its running-time efficiency for the self-assembly of finite shapes [12]. While the aTAM has a polynomial time solution to the UAV problem [2], allowing for the production of efficient simulators [17, 22, 29], the complexity of UAV in the 2HAM has remained a long-standing open problem in the field. The 2HAM appeared formally in 2013 [9], but was essentially defined in staged self-assembly [13] (2007), and a seeded version of the 2HAM appears as the *multiple tile* model in [4] (2004). UAV has been open for all of these models, and our coNP-complete result for UAV in the 2HAM proves that UAV with a single bin and single stage in the staged model is coNP-complete, and that UAV in the multiple tile model is also coNP-complete with polynomial-sized pieces, thus answering both of these long-standing open questions. See [15, 23, 30, 31] for surveys and applications of self-assembly theory.

Previous work on UAV. A number of results have pushed closer to resolving the complexity of UAV in the 2HAM. One of the first results showed that the simpler problem of determining if a given assembly was at least produced (i.e., built but possibly along with other different assemblies) is polynomial time solvable [16], which serves as a key step in showing that UAV resides within the class coNP. Another result augmented the basic 2HAM model to 3 dimensions and showed coNP-hardness for the 3D 2HAM [9]. A recent result focused on 2D, but allowed the temperature threshold, a parameter that determines how much glue strength is required for assemblies to stick together, to be a variable input to the UAV problem (as opposed to a fixed constant value), and showed coNP-completeness in this scenario [26].

Other approaches considered the allowance of initial assemblies consisting of small prebuilt assemblies, as opposed to only initial singleton tiles, and showed UAV becomes coNP^{NP}-complete with this extension [7]. Alternately, the inclusion of a negative force glue, even without detachments, has also been shown to imply coNP-completeness in the aTAM [10]. Another generalization of the 2HAM allows for up to k hands to create new assemblies, instead of just two, causing the problem to become either coNP-complete or PSPACE-complete, depending on the encoding of the variable k [5]. An even more powerful generalization of the 2HAM is the *staged* model [13], in which multiple distinct stages of self-assembly are considered. Within the staged model, UAV becomes coNP^{NP}-hard after 3 stages, and PSPACE-complete in general [6, 27]. Thus, for nearly every way in which the 2HAM has been extended, a corresponding hardness reduction has been found. Yet, the original question of UAV in the 2HAM has remained open.

Our Contributions. We show that UAV in the 2HAM is coNP-complete within the original model (2-dimensional, constant bounded temperature parameter, singleton tile initial assemblies), thus resolving the long-standing open problem of UAV in the 2HAM. Further, this

Table 1 Known Results for the Unique Assembly Verification Problem in the 2HAM and the results presented in this paper. $|A|$ is the size of the target assembly, τ is the temperature of the system, and $|T|$ is the number of tile types in the system. Under the **Temperature** column, τ indicates that the temperature may be included as part of the input.

Shape	Dimensions	Temperature	Complexity	Reference
General	2	1	$\mathcal{O}(\|A\|\|T\| \log \|T\|)$	[16]
General	3	2	coNP-complete	[9]
General	2	τ	coNP-complete	[26]
General	**2**	**2**	**coNP-complete**	**Thm. 7**
Tree	**2**	$\boldsymbol{\tau}$	$\mathcal{O}(\|A\|^5 \log \tau)$	**Thm. 12**

proves that UAV in the staged model with a single bin and stage is coNP-complete, and that UAV in the q-tile/multiple-tile model with polynomial-sized pieces is coNP-complete. We augment this result with a positive result for the special case of tree-shaped assemblies, providing a $\mathcal{O}(|A|^5 \log \tau)$ time solution for UAV in this case (where $|A|$ is the size of the assembly) even if τ is included as part of the input.

Our results are highlighted in Table 1 along with other known results for UAV in the 2HAM. To show coNP-hardness for UAV we construct an explicit polynomial-time reduction from Monotone Planar 3-SAT with Neighboring Variable Pairs (MP-3SAT-NVP). This reduction takes inspiration from the recent break-through proof that MP-3SAT-NVP is NP-hard and its use to prove that the connected-assembly-partitioning problem with unit squares is NP-hard [3]. For our tree UAV algorithm, we utilize a cycle decomposition approach over possible produced assemblies combined with dynamic programming.

Overview. The paper is structured as follows. Section 2 formally defines the model, the UAV problem, important definitions, and some small examples. Section 3 has the reduction proving UAV in the 2HAM is coNP-hard. Due to the numerous intricate details related to the proof, the section is broken up into several subsections explaining different aspects of the reduction. Section 4 then gives the algorithms for solving UAV for tree-bonded assemblies. Section 5 then concludes the paper with a summary and future work.

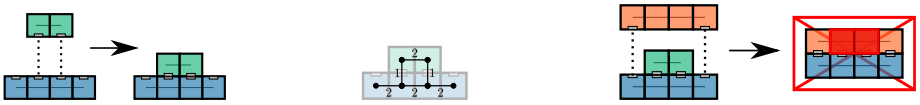
2 Definitions

In this section we overview the basic definitions related to the two-handed self-assembly model and the verification problems under consideration.

Tiles. A *tile* is a non-rotating unit square with each edge labeled with a *glue* from a set Σ. Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength* $\text{str}(g_1, g_2)$.

Configurations. A *configuration* is a partial function $\tilde{A} : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e. an arrangement of tiles on a square grid. For a configuration $\tilde{A}$ and vector $\vec{u} = \langle u_x, u_y \rangle$ with $u_x, u_y \in \mathbb{Z}^2$, $\tilde{A} + \vec{u}$ denotes the configuration $\tilde{A} \circ f$, where $f(x, y) = (x + u_x, y + u_y)$. For two configurations $\tilde{A}$ and $\tilde{B}$, $\tilde{B}$ is a *translation* of $\tilde{A}$, written $\tilde{B} \simeq \tilde{A}$, provided that $\tilde{B} = \tilde{A} + \vec{u}$ for some vector $\vec{u}$.

Bond graphs, and stability. For a given configuration $\tilde{A}$, define the *bond graph* $G_{\tilde{A}}$ to be the weighted grid graph in which each element of $\text{dom}(\tilde{A})$ is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A



(a) Cooperative Binding. **(b)** Bond Graph. **(c)** Geometric Blocking.

Figure 1 (a) Example of an attachment that takes places using cooperative binding at $\tau = 2$. We denote a glue strength of 1 with a rectangle and a glue of strength 2 with a solid line through the two tiles. Dotted lines between glues indicate that these tiles may attach to each other with the respective strength. Assume assemblies shown are τ-stable unless stated otherwise. (b) The bond graph of the assembly showing that it is τ-stable. (c) These two assemblies are not τ-combinable since this would place two tiles at the same location. We say this is due to geometric blocking.

configuration is said to be *τ-stable* for a positive integer τ if $G_{\tilde{A}}$ is connected and if every edge cut of $G_{\tilde{A}}$ has a weight of at least τ. This means that the sum of the glue strengths along each cut is greater or equal to τ. A small example bond graph is shown in Figure 1b.

Assemblies. For a configuration $\tilde{A}$, the *assembly* of $\tilde{A}$ is the set $A = \{\tilde{B} : \tilde{B} \simeq \tilde{A}\}$. Informally an assembly A is a set containing all translations of a configuration $\tilde{A}$. An assembly A is a *subassembly* of an assembly B, denoted $A \sqsubseteq B$, provided that there exists an $\tilde{A} \in A$ and $\tilde{B} \in B$ such that $\tilde{A} \subseteq \tilde{B}$. We define $|A|$ to be the number of tiles in a configuration of A.

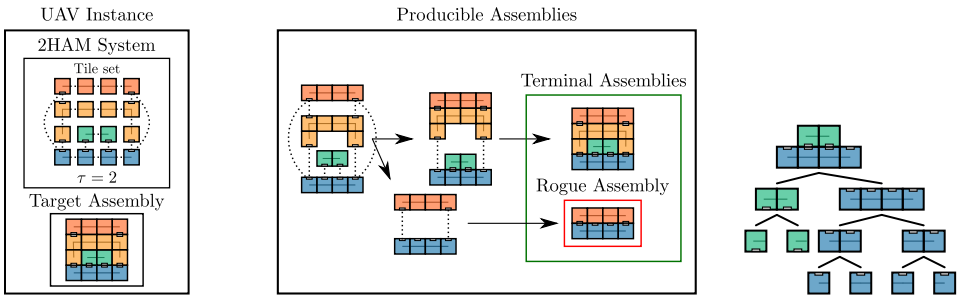
An assembly is *τ-stable* if the configurations it contains are τ-stable. Assemblies A and B are *τ-combinable* into an assembly C if there exist $\tilde{A} \in A$, $\tilde{B} \in B$, and $\tilde{C} \in C$ such that 1) $\tilde{A} \cup \tilde{B} = \tilde{C}$, 2) $\tilde{A} \cap \tilde{B} = \varnothing$, and 3) $\tilde{C}$ is τ-stable. Informally, two assemblies are τ-combinable if there exist two configurations of the assemblies that may be combined resulting in a τ-stable assembly without placing two tiles in the same location.

Two assemblies combining or binding together is called an attachment. An attachment takes place using *cooperative binding* if the two assemblies do not share a τ-strength glue and instead use multiple weaker glues summing to τ. An example of an attachment that takes place using cooperative binding can be seen in Figure 1a. If an attachment cannot take place because the two tiles would be placed in the same position, it is *geometrically blocked*. Two assemblies whose attachment is geometrically blocked is shown in Figure 1c.

Two-handed Assembly. A two-handed assembly system (2HAM) is an ordered tuple $\Gamma = (T, \tau)$ where T is a set of tiles and τ is a positive integer parameter called the *temperature*. For a system Γ, the set of *producible* assemblies P'_Γ is defined recursively as follows: 1) $T \subseteq P'_\Gamma$, and 2) If $A, B \in P'_\Gamma$ are τ-combinable into C, then $C \in P'_\Gamma$. We are naturally extending the concept of τ-combinable to single tiles by considering them assemblies of size 1.

A producible assembly is *terminal* provided it is not τ-combinable with any other producible assembly. Denote the set of all terminal assemblies of a system Γ as P_Γ. Intuitively, P'_Γ represents the set of all possible assemblies that can self-assemble from the initial set T, whereas P_Γ represents only the set of assemblies that cannot grow any further. Figure 2 shows a small 2HAM example.

An *Assembly Tree* for a 2HAM system $\Gamma = (T, \tau)$ is any rooted binary tree whose nodes are elements of P'_Γ, the leaves are single tiles from the set T, and the two children of any non-leaf node are τ-combinable into their parent. An assembly tree with root A is said to be an assembly tree for assembly A. A small example is shown in Figure 2c.



(a) UAV Instance. **(b)** Producible Assemblies. **(c)** Small Assembly Tree.

Figure 2 (a) An example instance of the Unique Assembly verification problem. The input is the 2HAM system (tile set and temperature) and the target assembly. (b) The main producible assemblies of the 2HAM system (for clarity, not all subassemblies are shown). The target assembly is producible and terminal. However, there is also a produced assembly that is a rogue assembly (highlighted) since it not a subassembly of our target and it is terminal. (c) A small example of an assembly tree for one of the producibles.

Unique Assembly. Intuitively, the unique assembly of A means that any produced assembly can continue to grow until it becomes A, thus making A the *uniquely produced* assembly if the process is provided sufficient time to assemble. This means A is the unique terminal assembly and all produced assemblies are subassemblies of A. Formally, we say a system Γ uniquely produces an assembly A if: 1) $P_\Gamma = \{A\}$, and 2) For all $B \in P'_\Gamma$, $|B| \leq |A|$.

▶ **Problem 1** (Unique Assembly Verification)**. Input:** A 2HAM system Γ, an assembly A. **Output:** Does Γ uniquely produce the assembly A?

The Unique Assembly Verification problem (UAV) is the computational problem that asks to verify if an assembly is uniquely produced. A key concept used throughout this paper is a *rogue assembly*, which is any producible assembly that breaks one of the conditions of unique assembly and serves as a proof that the instance of the UAV problem is false.

▶ **Definition 2** (Rogue Assembly)**.** *Given an instance of UAV (Γ, A), an assembly $R \sqsubseteq P'_\Gamma$ is a rogue assembly if $R \neq A$ and R is not a subassembly of A.*

We prove the following Lemma, which is used in the hardness reduction and the positive result. This lemma states that if the instance of UAV is false and all the tiles in Γ are used to build A, then any rogue assembly is made of combinable subassemblies of A.

▶ **Lemma 3.** *For an instance of UAV (Γ, A) that is false the following statement must be true: there exists two producible assemblies B, C such that $B, C \sqsubseteq A$ and B and C are τ-combinable into a rogue assembly R, or there exists a rogue assembly R that is composed of a single tile.*

Proof. First, since the instance of UAV is false, there must exist some rogue assembly R. If R is composed of a single tile, the Lemma is true. If R is not composed of a single tile, we walk through its assembly tree to find the assemblies B and C that are both subassemblies of our target A. Consider an assembly tree of R, Υ_R. Start at the root- if its two children are both subassemblies of A, then the rogue assembly R satisfies the Lemma. If either of the children is also a rogue assembly (not a subassembly of the target), then follow that node and do the same thing. If both are rogue assemblies, it does not matter which we follow.

Since this is an assembly tree all the leaves represent assemblies composed of one tile. Since we know none of the leaf assemblies are rogue assemblies (if it was the lemma would already be satisfied) we know at some point we must reach a node representing a rogue assembly that can be built from two subassemblies of our target A. ◀

3 Unique Assembly Verification Hardness

In this section, we show coNP-hardness of the Unique Assembly Verification problem in the 2HAM with constant temperature by a reduction from Monotone Planar 3-SAT with Neighboring Variable Pairs.

▶ **Problem 4** (Monotone Planar 3-SAT with Neighboring Variable Pairs (MP-3SAT-NVP)). **Input:** Boolean formula $\phi = C_1 \wedge \cdots \wedge C_m$ in 3-CNF form where each clause only contains positive or negated literals from $X = \{x_1, \ldots, x_n\}$. Further, any clause of ϕ with 3 variables is of the form (x_i, x_{i+1}, x_j) or $(\neg x_i \vee \neg x_{i+1} \vee \neg x_j)$, i.e., at least two of the literals are neighbors. **Output:** Does there exist a satisfying assignment to ϕ?

Monotone Planar 3-SAT with Neighboring Variable Pairs was recently shown to be NP-hard in [3]. We assume the instance of the problem is a rectilinear planar embedding where each variable is represented by a unit height rectangle arranged in the *variable row*. Any planar 3SAT formula has a rectilinear encoding [19]. We also assume that every clause is a unit-height rectangle with edges connecting the clauses and the contained variables. The monotone property ensures that each clause contains either only positive or only negative literals. Thus, the clauses may be separated with all positive clauses above the variable row, and all the negative clauses below. The final restriction is neighboring variable pairs, which states that for the three variables in each clause, at least two of the variables are neighbors in the variable row. An example instance is shown in Figure 3a.

3.1 Overview

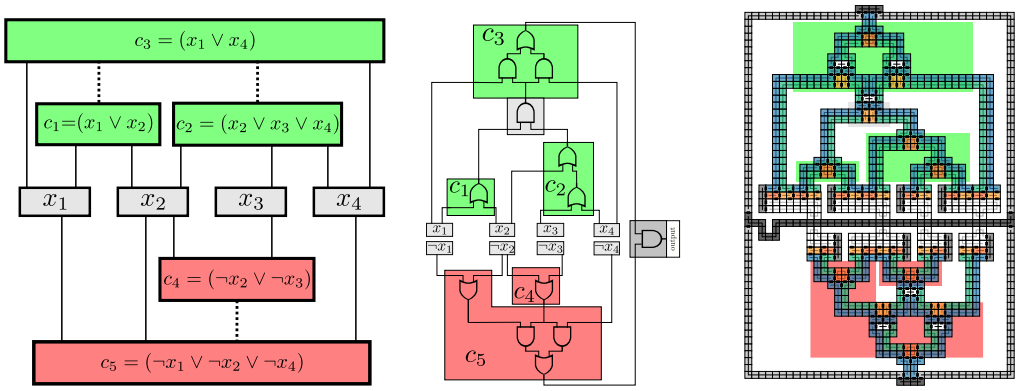
Given an instance of MP-3SAT-NVP ϕ, we build an assembly A and a 2HAM system Γ that uniquely assembles A if and only if ϕ does not have a satisfying assignment. An example instance is shown in Fig. 3a and 3c. Alternatively, Γ produces a rogue assembly if and only if there exists a satisfying assignment to ϕ.

The ability to place all positive clauses above the variables and negative clauses below, along with the neighboring variable pairs, allows the clauses to be built hierarchically from the variables up. These properties allow us to require all nested clauses be evaluated and built before the outer clause is built. Thus, we define *parent* and *child* clauses as well as *root* clauses. In Figure 3a, dotted lines connect child clauses c_1 and c_2 with their parent c_3. The root clauses are c_3 and c_5.[1]

▶ **Definition 5** (Parent/Child/Root Clause). *Given a rectilinear encoding of Monotone Planer 3-SAT, a clause C_p is a* parent *clause of* child *clause C_c, if C_p fully encloses C_c, and any other clause that encloses C_c also encloses C_p. A* root *clause is a clause without a parent.*

Since ϕ is monotone, the positive and negative clauses may be separated across the variable row. The assembly A is also separated by a horizontal bar that splits the assembly in two. This bar partially extends downward to prevent this assembly from attaching to itself. Above this bar is a subassembly that encodes the positive clauses and below the bar is a subassembly that encodes the negative clauses, which we call the positive and negative circuit, respectively.

[1] While a formula may have multiple clauses without a parent, the authors of [3] show that by adding additional variables, an instance may be constructed with only a single root clause. For MP-3SAT-NVP, we need at least two root clauses (one for the positive and negative sides).



(a) MP-3SAT-NVP Instance. **(b)** Target Assembly Circuit. **(c)** UAV Target Assembly.

Figure 3 (a) Example instance of Monotone Rectilinear 3SAT with Neighboring Variable Pairs. Dotted lines are drawn between parent and child clauses. In this example c_3 and c_5 are the positive and negative root clauses respectively. (b) A circuit view of our example instance with gates divided into the clauses they compute. We add AND gates (shown in grey) between child clauses that have the same variables. (c) Target assembly constructed from instance on left. Each tile in the assembly is a unique tile type. Each glue is unique except for the strength 1 glues connecting the horizontal bar and the arms of each circuit. The parts of the assembly that represent each clause are boxed in.

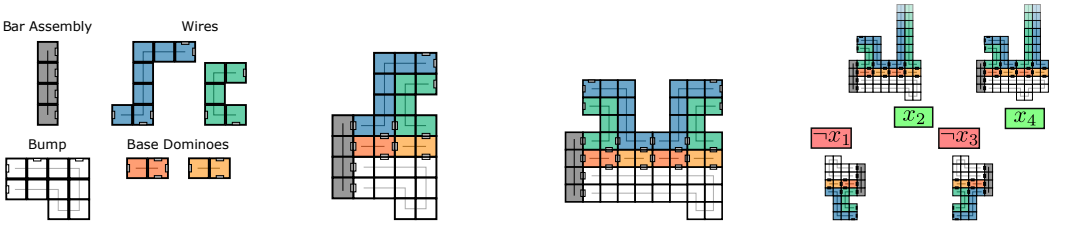
The target assembly is designed so that it must be built from the variables up to the clauses. The clause gadgets can only be built if they are satisfied. Thus, parent clauses require that their variables or child clauses be satisfied to build the gadget. We will ensure this by using AND and OR gadgets between the variable and clause gadgets. We cover the parts of the system and gadgets in the order they must assemble:

- Section 3.2: variable gadgets
- Section 3.3: OR gates and non-parent clause gadgets
- Section 3.4: AND gates and parent clauses
- Section 3.5: the root clauses and the horizontal bar
- Section 3.6: how a rogue assembly may form if and only if ϕ is satisfiable

3.2 Variable Gadget

For each variable gadget we use $(2 + 4d)$ subassemblies (Figure 4a) to build the variable gadget where d is the number of times the variable is used or its outdegree. An example is shown in Figure 4b where $d = 1$ and Figure 4c where $d = 2$. In the figures, the lines are strength-2 glues, and the rectangle glues are all strength-1, thus requiring cooperative binding for the subassemblies to attach in a specific build order. We draw our gadgets separated into subassemblies but we construct our tile set using the single tiles which will self-assemble into these subassemblies. Every variable gadget is built as follows.

- The **Bar Assembly** acts as a backbone (or separator) for the completed circuit subassemblies to connect to each other.
- The **Bump** is the first assembly to attach to the Bar Assembly. The Bump is a height 2 rectangle with an extra domino below it that is used for geometric blocking and encoding the assignment to that variable. The position of this domino is dependent on the position of the variable gadget on the opposite (negative) side.
- The **Base Dominoes** are used as part of the process of duplicating a variable path to multiple clauses. For each clause a variable is in, we use four subassemblies to connect to the next gadget. The first two gadgets are the Base dominoes. Once the Bump attaches



(a) Subassemblies of a variable gadget.

(b) A variable gadget with outdegree 1.

(c) A variable gadget with outdegree 2.

(d) Assignment to the example MP-3SAT-NVP instance.

Figure 4 (a) The smaller assemblies used for building a variable gadget. (b) A variable gadget for a variable that is used in a single clause. (c) A variable gadget with outdegree 2. For each additional output more base dominoes and wires are added. (d) A set of producible subassemblies representing variables that satisfy the example instance. We will walk through how these assemblies grow into a rogue assembly.

to the Bar Assembly, the Base Dominoes can attach cooperatively to both. Once the first Base Domino attaches the next can attach using the glue from the previous domino and the other from the Bump.[2]

- The **Wires** attach the variable gadgets to the clauses, and are the final two subassemblies for connecting to the next gadget. The first wire attaches cooperatively to the Bar Assembly and subsequent ones attach to the previous wire. The wires in our system are all built from two assemblies. When both halves of the wire are connected, the next gadget may attach. We call this a completed wire.

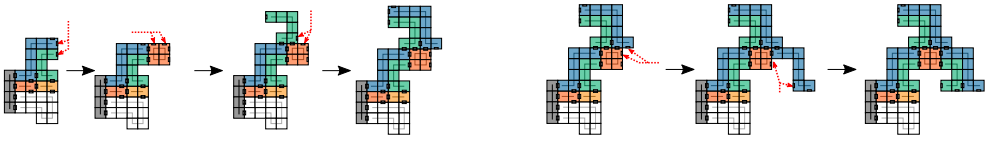
Variable gadgets in the negative circuit are built symmetrically rotated 180 degrees. We adjust the position of dominoes on the Bumps of the gadgets so that variable gadgets on opposite circuits that represent the same variable have their domino in the same column. We may generalize these gadgets to out degree d (the variable appears in d clauses) by increasing the width of the bump, and adding additional dominoes and wires.

3.3 OR Gates and Clause Gadgets

In CNF form, every variable in a clause is separated by a logical OR, thus, as part of our clause gadgets, we create OR gates to bring the variables together to ensure that the clause only forms if there is at least one variable assignment that satisfies the clause.

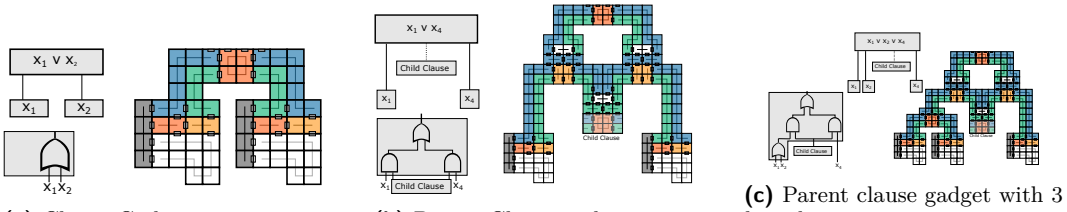
OR Gates. An example of how the OR gate grows off of a variable gadget is shown in Figure 5a. The OR gate consists of a single 2×2 square with strength-1 glues on the west, north, and east facing tile edges. The west and east glues each connect to the wires that input to the gate. The north glues are used cooperatively with glues on the incoming wire gadgets to attach another wire gadget going to the clause gadget. To complete the new wire gadget it must also cooperatively use the other incoming wire. Note the wires from the other input can backfill, but this does not cause an issue as the "backward" growth stops after building the wire. Figure 5b shows an example with only one variable used in the OR gate.

[2] Without these base dominoes, variable gadgets could build without the full Bump due to backfilling or backwards growth. The dominoes ensure this can not happen.



(a) Build OR gate. **(b)** Wire backing filling.

Figure 5 (a) The process of a variable gadget growing the OR gate used for clauses. Glues used for attachment in the next step are denoted by arrows. If one of the variable gadgets is constructed the 2×2 square assembly may attach. The output wires of the gate then attach cooperatively with the wire from the variable assembly and the square. Note that only one of the variable assemblies needs to be constructed for the OR gate to build its output wire. (b) Once the output wires of the OR gate have attached the wire for the other variable may "Backfill" or grow backwards.



(a) Clause Gadget. **(b)** Parent Clause gadget. **(c)** Parent clause gadget with 3 literals.

Figure 6 (a) A clause gadget with 2 neighboring variables. (b) A clause gadget with two variables and a child clause. (c) When a parent clause has 3 literals we know two of them must be neighbors. Using an additional OR gate we may use the same gadget as the clause with 2 literals.

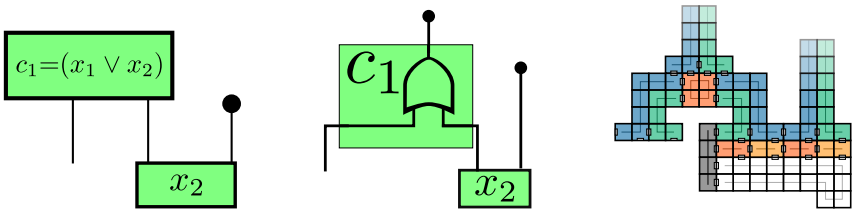
Non-parent Clauses. We first cover clauses without children, or clauses at the bottom of the circuit. The simplest type of this gadget are clauses with only 2 literals as in Figure 6a. This gadget is fairly straightforward to implement as we only need to use a single OR gate. An example of this type of clause is in Figure 7a, and its implementation is in Figure 7c. Note that both variables appear in other clauses so those variable gadgets have additional wires. For non-parent clauses with 3 literals (Figure 10a), we use 2 OR gates (Figure 8c).

3.4 AND Gates and Parent Clauses

Since every clause in CNF form is separated by a logical AND, we create AND gates that compare clauses. Thus, we need to know which clauses are parent clauses since they have child clauses underneath them with wires coming into the gates. We also build a FANOUT gate for connecting clauses.

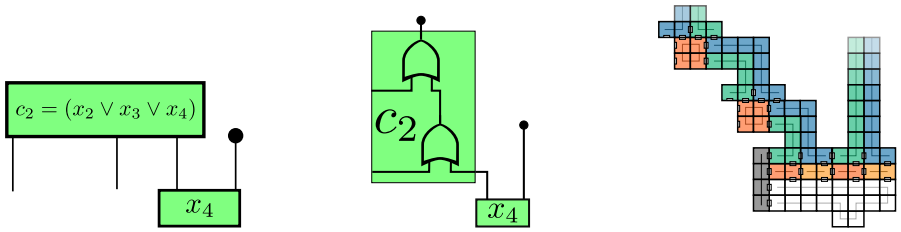
AND Gates. The AND gate uses 2 vertical dominoes that share a single strength-1 glue between them. Figure 9a shows an example AND gate being constructed. Once a wire that inputs to the gate is completed, one of the dominoes can cooperatively attach. The domino has another strength-1 glue on its north side that allows a horizontal domino to cooperatively attach using the glue exposed on the wire.

Using the glues from the newly attached dominoes, the two halves of the gate are able to attach to each other. This allows for the two glues on the horizontal dominoes to be used to cooperatively bind the white center domino. From here, the two halves of the wire that outputs from the AND gate can attach.



(a) Clause c_1. **(b)** Circuit for c_1. **(c)** Gadgets for c_1.

Figure 7 (a) The clause c_1 in Figure 3a is satisfied by $x_2 = 1$. (b) The OR gate grows off of x_2. The other wires on the variable gadget are used to connect to other clause gadgets. (c) The gadget constructed for the clause c_1. Note the other wire from the OR gate has backfilled.



(a) Clause c_2. **(b)** Circuit for c_2. **(c)** Gadgets c_3.

Figure 8 (a) c_2 from our example. This clause has 3 variables and no children. (b) The clause is computed using two OR gates. The gates are able to grow from x_4. (c) x_4 variable gadget allows for the two OR gates to attach.

FANOUT Gates. In order to build the parent clause, we also need a way to "fan-out" and copy the signal from an AND gate to two other gadgets. We do this by adding glues to the north side of the center domino and having two wires grow off of the gadget. This process is shown in Figure 9b.

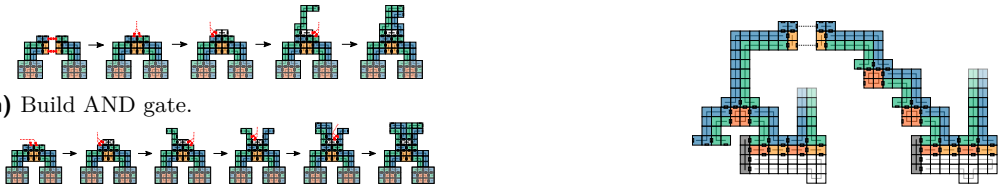
Parent Clauses. Consider a parent clause $C_p = (x_1 \vee x_4)$. Let C_c be the child clause. Since we want this gadget to build only if its own clause and its child are both satisfied, we can view this statement as $(x_1 \vee x_4) \wedge C_c$. However, we can modify the statement to be $(x_1 \wedge C_c) \vee (C_c \wedge X_4)$, which we can build since we have planar circuits. An example of the circuit and gadgets are shown in Figure 6b.

By the neighboring variable pairs restriction, we know that any clause with three variables has at least a pair of them being neighbors. This means that there cannot be any child clauses beneath that neighboring pair, so we may use an OR gate between those two variables and then build the rest of the gadget in the same way as the two literal version (Figure 6c).

In our example instance, the root clause of the positive circuit has two children. For these cases we may use the AND gadget to verify that both child clauses have been satisfied before allowing the parent clause to build. The root clause of the negative circuit in our example instance (Figure 11a) has three literals. The constructed gadget can be seen in Figure 11a.

3.5 Root Clauses and Horizontal Bar

Root Clauses and Arms. The root clause is the outermost clause on either side of the variables. Although it functions similar to the other clauses, instead of outputting a wire, a horizontal 4×1 rectangle can attach after it finishes assembling. The arms may then cooperatively bind to the rectangle and the wires of the root clause forming the top of the

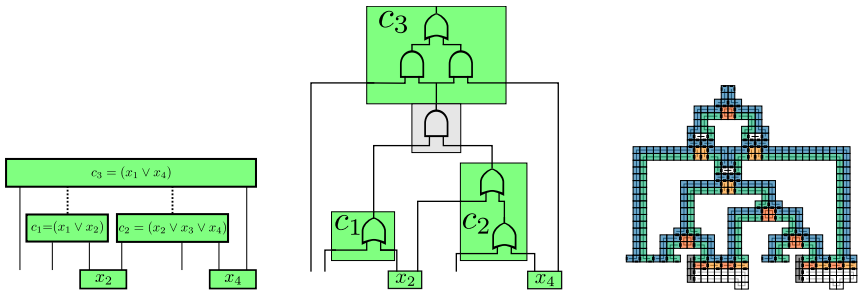


(a) Build AND gate.

(b) Build FANOUT.

(c) Attachment of an AND gate.

Figure 9 (a) The process of an AND gadget assembling. The output wires can only grow from the combined halves of the AND gate. (b) By modifying the center domino, two wires may be output from a single AND gate, which works as a FANOUT. (c) The two clauses c_1 and c_2 both have the same parent clause so they are joined by an AND gate. Once the dominoes attach to the output wire of the clauses the two assemblies may attach to each other.



(a) Clause c_3.

(b) Circuit for c_3.

(c) Gadget for c_3.

Figure 10 (a) The root clause of the example instance. This clause has two literals and two children. (b) Since the AND gate has built and x_4 satisfies c_3 the clause may grow. (c) The two child clause's output are connected by an AND gate and then used as the middle input to the gadget.

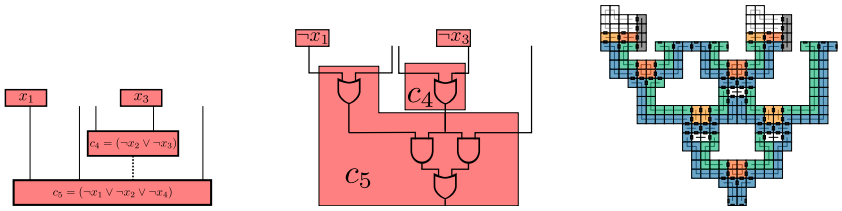
circuit. The glues on the ends of these arms allow for the circuit to attach to the horizontal bar. A high-level view of the root clauses and arms attached is shown in Figure 12a as well as a detail of the assembly process of the root clause in Figure 12b.

Horizontal Bar. The horizontal bar (Figure 12a) is a width-1 assembly that extends the width of both circuits with strength-1 glues on the north and south side of the outer tiles. Since the arms must also be able to attach to each other to form a rogue assembly the glues on the ends of the horizontal bar must be the same. In order to prevent the horizontal bar from attaching to another instance of itself, we extend the bar partially downward so it will geometrically block copies from attaching.

3.6 Rogue Assemblies

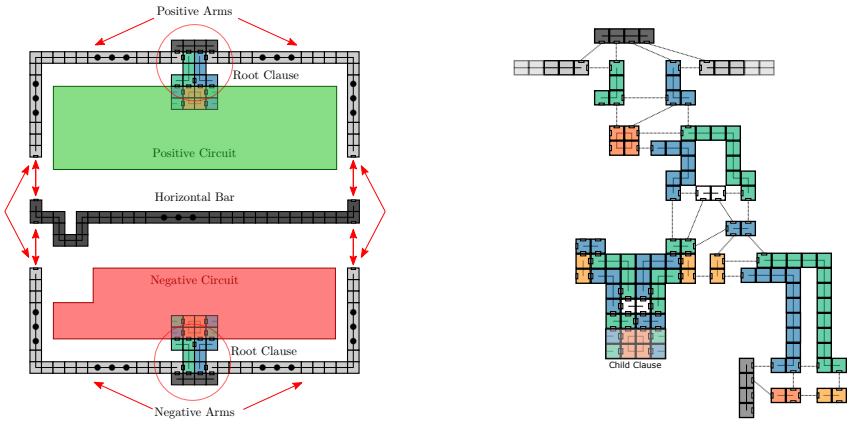
For the construction of the target assembly, each piece is built from the variables up to the root clause. However, the nondeterministic build order means that not all parts of each circuit need to be built in order for the root clause to be satisfied. For instance, if one of the variables in a clause attaches, the OR gates will still allow the wires to attach. Thus, using a variable constitutes setting it to true (and in the negative circuit using a variable is setting the negation to true).

With root clauses satisfied and the arms attaching, a rogue assembly may occur as shown in Figure 13c. The corresponding circuit is shown in Figure 13b. This can occur because the arms can attach to each other without the horizontal bar. Normally, the variable gadgets



(a) Clause c_5. **(b)** Circuit for c_5. **(c)** Gadget for c_5.

Figure 11 (a) In the example instance the negative circuit has c_5 which is a parent clause with 3 literals. (b) The negative circuit draw with gates. The variables x_1 and x_3 being false satisfies all the clauses. (c) The variable assemblies we selected at the beginning also grow into a circuit with the root clause built.



(a) Horizontal Bar and Completed Circuits. **(b)** Glues on root clause and variable gadgets.

Figure 12 (a) The root clauses and arms joining the positive and negative assemblies with the horizontal bar. The root clause allows a short wire to attach where the arms can then attach. Each arm has a strength-1 glue at the end. The horizontal bar separates the positive circuit from the negative circuit. The small bump is so the bar can not attach to other horizontal bars. (b) Each subassembly of the root clause cannot attach to each other without being satisfied from the child clauses since each subassembly only shares a strength-1 glue with adjacent assemblies.

would overlap and prevent this attachment if both the positive and the negative circuit used the same variable (which is setting a variable to both true and false). Thus, the positive and negative side each have their own set of variables that make all clauses on their respective sides true. This rogue assembly can only happen if there is a subset for each side that allows all clauses to be true, and thus satisfies the original MP-3SAT-NVP formula.

For an MP-3SAT-NVP instance ϕ and an assignment X_s to the variables in X, let A_p and A_n be the positive and negative circuit assemblies, respectively, created from ϕ. We say an assembly $A'_p \sqsubseteq A_p$ represents the assignment X_s if it has attached variable gadgets for the variables in X_s that equal 1, and has built its root clause. For negated circuits, it must have variable gadgets attached for variables set to 0 in X_s.

▶ **Lemma 6.** *For a rectilinear encoding of Monotone Planar 3SAT ϕ with neighboring variable pairs and 2HAM system Γ_ϕ as described above, there exist two producible assemblies $A'_p \sqsubseteq A_p$ and $A'_n \sqsubseteq A_n$ that both represent the same assignment X_s to the variables X, if and only if X_s satisfies ϕ.*

Proof. If there exists a satisfying assignment X_s to X, we may build A'_p by taking the variable gadgets for variables assigned to 1 and grow the circuit off of them. Since we know all the clauses are satisfied, each clause gadget (including the root clause) may grow resulting in an assembly A'_p that represents X_s. By the same argument we know A'_n is producible since X_s satisfies ϕ, which includes the negated clauses.

We prove these assemblies are producible only if X_s satisfies ϕ via contradiction. Assume X_s does not satisfy ϕ, but both assemblies A'_p and A'_n are producible. Since X_s does not satisfy ϕ, there must exist at least one unsatisfied clause c_i. W.L.O.G., assume c_i is a positive clause. We show the assembly A'_p cannot be produced.

If c_i is the root clause, assume all of the children of c_i are satisfied. The center input of the clause is a producible subassembly of A'_p since variable gadgets are allowed that satisfy the clauses below it. We can see in Figure 12b the other producible subassemblies of the gadgets only have a strength-1 glue between them. This means none of the subassemblies are able to attach to each other on their own. In order for the arms to attach to the output wire of the root clause, at least one of the AND gates must be fully constructed. The AND gate cannot assemble unless both halves of the gate have been constructed. The middle input is built, but the other half of the AND gadget must grow off a completed wire from the variable gadget. However, since c_i is not satisfied the variables gadgets which satisfy the formula have not attached so the assembly A'_p cannot build the clause gadget.

If c_i is another parent clause that is not the root. Let the clause c_j be the parent clause of c_i. If the clause gadget for c_i is not constructed then the gadget for c_j is not buildable. Since the gadgets used are the same as the root clause, the output wires of the clause gadget for c_i cannot be built without a variable gadget which satisfies the formula. The middle input of clause gadget representing c_j will not be buildable since this would be the output wire of c_i. The middle input goes to two AND gates that cannot construct unless both wires have been built. Thus, the output wire of c_j cannot be built without its children clauses satisfying it. In the case c_j has multiple children, the output wires of all its children are joined by AND gates that will not construct without both inputs.

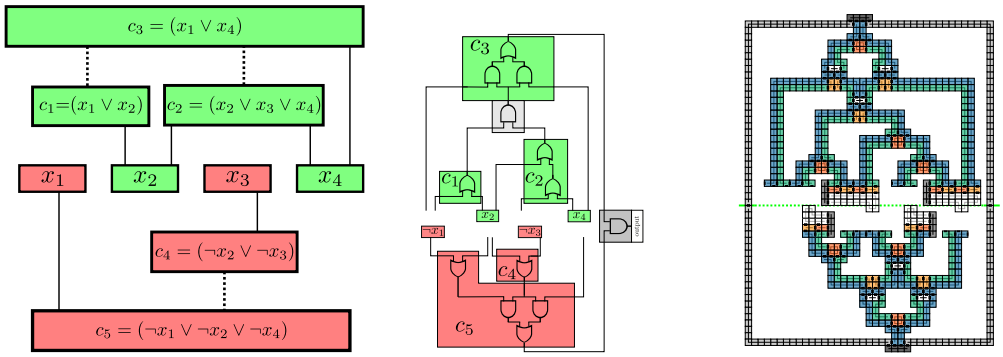
Finally, consider the case where c_i is a clause without children. In order for the clause's output wire to complete, it must be attached to an OR gadget and the outer wire of the variable gadget. The OR gadget may only attach to a completed wire from a variable gadget (or another OR). The variable gadget cannot be completed without placing the bump, so we cannot have built the outwire of c_i. By the same argument as the previous case, this clause not being built results in its parent not being built. If c_i is not satisfied, the clause gadget for c_i cannot be constructed, which means the assembly A'_p is not producible. ◀

▶ **Theorem 7.** *The Unique Assembly Verification problem in the 2HAM is coNP-Complete with $\tau = 2$.*

Proof. Given an instance of a rectilinear encoding of Monotone Planar 3SAT with neighboring variable pairs ϕ, we provide an explicit polynomial time reduction by creating a 2HAM system $\Gamma = (T, 2)$ and an assembly A such that Γ uniquely produces A if and only if there does not exist a satisfying assignment to ϕ. We create the assembly A by taking the rectilinear encoding of ϕ, arranging the rectangles on a grid graph, and replacing the rectangles with the given variable and clause gadgets. We also add the arms and horizontal bar.

Assume there exists a satisfying assignment X_s to the variables X, for ϕ. We know by Lemma 6, there exist two producible assemblies A'_p and A'_n that both contain the arms and have complementary bump positions[3]. These two assemblies can cooperatively bind to one another using the two glues on their arms, and thus produce a rogue assembly as in Figure 13c. This means a satisfying assignment to ϕ implies Γ does not uniquely construct A.

[3] Having complimentary bump positions is equivalent to both representing the same assignment.



(a) Satisfying Assignment. **(b)** Rogue Assembly Circuit. **(c)** Rogue Assembly.

Figure 13 (a) There exists a satisfying assignment for the example instance with green blocks representing variables which equal 1 and red blocks representing 0. (b) Rogue assembly drawn as a circuit with selected variables. (c) The 2HAM system will produce a Rogue Assembly from the two circuit assemblies which represent the satisfying assignment.

(a) Tree-bonded and Shaped. **(b)** Tree-bonded. **(c)** Not Tree-bonded.

Figure 14 (a) Both the shape and bond graph of this assembly are trees. (b) Even though the shape of this assembly is a square, its bond graph is acyclic, and thus this assembly is tree-bonded. (c) This assembly is not tree-bonded due to the cycle in its bond graph.

Now assume Γ does not uniquely produce A, so there exists some rogue assembly B. The only repeated glues in the tile set of Γ are the exposed glues on the arms. Any rogue assembly must use these two glues to assemble, and they must be assembled from two subassemblies of the target by Lemma 3. Let B be producible by combining two assemblies b and b'. Since both b and b' are producible assemblies with both their arms, and they can attach to each other, they are not geometrically blocked. This implies they must represent the same assignment and by Lemma 6, this can only be true if the assignment satisfies ϕ. By viewing which variable gadgets are included in the two assemblies, we can identify the satisfying assignment to ϕ. Thus, Γ will uniquely produce A if and only if there does not exist a satisfying assignment to ϕ. ◀

4 Verification of Tree-Bonded Assemblies

In this section, we investigate the problem of Unique Assembly Verification with the promise that the target assembly A is *tree-bonded*, meaning the bond graph of the target assembly forms a tree. Figures 14a and 14b show examples of tree-bonded assemblies. Figure 14c shows an assembly whose bond graph contains a cycle and thus is not a tree-bonded assembly. We first present a $\mathcal{O}(|A|^5)$ algorithm for temperature 2 systems, and then extend this method to provide a $\mathcal{O}(|A|^5 \log \tau)$ time dynamic programming algorithm for the case where the temperature τ of the system can be passed as a parameter. Before describing the algorithms, we first introduce some required definitions and the problem formulations.

Tree-Bonded Assemblies. An assembly A is a *tree-bonded assembly* if and only if the induced bond graph $\mathcal{G}_A$ is acyclic.

Binding Sites. For two configurations C_1 and C_2, we say a *binding site* $\mathcal{B}$ is a pair of points (p_a, p_b), such that their distance is $||p_a - p_b||_2 = 1$, and the tiles $C_1(p_a)$ and $C_2(p_b)$ have nonzero glue strength between each other. The set of binding sites for two configurations is the set of pairs of points that meet this requirement. We also define an *inner binding site*. For two configurations, C_1 and C_2, and a pair of binding sites $a = (a_1, a_2'), b = (b_1, b_2')$, let $I(a, b)$ be the set of binding sites that occur on the inside of the loop formed by a, b (inner binding sites). An example of the area enclosed by a loop is seen in Figure 15c.

Simple Loops. For two configurations, C_1 and C_2, and a pair of binding sites $a = (a_1, a_2'), b = (b_1, b_2')$, we say the loop formed by a, b is a *simple loop* if $|I(a, b)| = 0$.

Origin Configuration In discussing different configurations and assemblies, it is useful to anchor a configuration to a fixed point. For an assembly A, the *origin configuration* A_0 is the translation of $A's$ configuration such that the bottom left vertex of the bounding box of elements in $dom(A_0)$ is at the origin $(0, 0)$.

▶ **Problem 8** (Tree-UAV). **Input:** A 2HAM system Γ and a tree-bonded assembly A. **Output:** Does Γ uniquely produce the assembly A?

4.1 High-level Overview

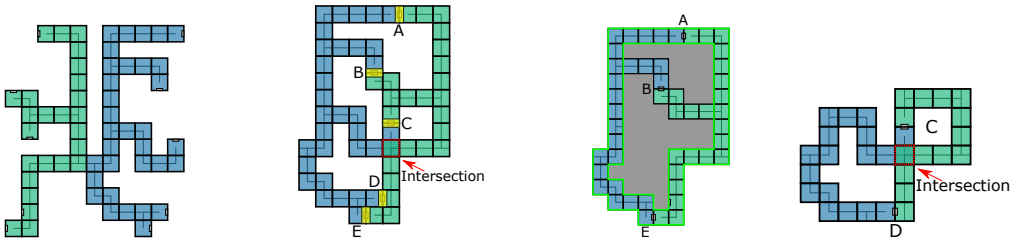
The high-level goal of this algorithm is to find a rogue assembly that acts as a witness that the instance of UAV is false. We note that a given instance, $P = (\Gamma, A)$, of Temp2-Tree-UAV, where $\Gamma = (T, 2)$, can be broken down into three possible cases. An example tree-bonded assembly is shown in Figure 15a.

1. The instance P is false, and Γ produces a tree-bonded rogue assembly.
2. The instance P is false, and the only rogue assemblies producible in Γ are non tree-bonded.
3. The instance P is true.

At a high level, the algorithm checks if either Case 1 or Case 2 as true, and f so, the algorithm rejects, otherwise it accepts. Case 1 can be checked efficiently by modifying Γ to function as a noncooperative system Γ' and utilizing the algorithm for temperature-1 UAV provided in [16]. To check the second case, Lemma 3 states that if the instance is false, it suffices to check pairs of subassemblies of the target assembly A in order to find a witness rogue assembly. Thus, we take two copies of the target assembly and attempt to find possible ways they may bond, even if the resulting assembly places two tiles at the same position. We call the pairs of tiles that contribute glue strength *binding sites* Tiles that are in the same position are called *intersections*. An example of both may be seen in Figure 15b.

We first analyze the case of temperature-2 systems where only two binding sites that do not intersect are needed. We then generalize this algorithm by using dynamic programming to find the set of binding sites to maximize the binding strength between the assemblies without any intersections.

▶ **Problem 9** (Temp2-Tree-UAV). **Input:** A $\tau = 2$ 2HAM system Γ and a tree-bonded assembly A. **Output:** Does Γ unique produce the assembly A?



(a) Tree-bonded Assembly. **(b)** Overlap Subassemblies. **(c)** Outer Loop. **(d)** Intersection loop.

Figure 15 (a) An example tree-bonded target assembly. (b) One possible overlap configuration formed by two subassemblies with 5 binding sites that are highlighted. (c) The loop formed by binding sites A and E is outlined in green. Any binding site that occurs in the grey shaded area, such as B, is in the set of inner binding sites for (A, E). (d) The loop formed using binding sites (C, D) intersects itself and cannot be used.

Algorithm 1 NonCoop-UAV(Γ, A). The runtime of Temp1-UAV is $\mathcal{O}(|A||T| \log |T|)$ [16].

Data: 2HAM System $\Gamma = (T, \tau)$, an assembly A
Result: Does Γ uniquely assembly A if it can only utilize strength $\geq \tau$ glues?
Modify T to create T' by removing all glues of strength less than τ, and setting the strength of all glues of strength $\geq \tau$ to 1;
if *Temp1-UAV*$(\Gamma' = (T', \tau = 1), A)$ **then** accept;
else reject;

4.2 Tree-Bonded Rogue Assemblies

The following algorithm checks if a system uniquely assembles a given shape provided the system is restricted to behaving in a noncooperative manner. This means that two assemblies can only attach if they share one or more strength-τ glues between them. This system functions equivalently to a temperature-1 system where all glues less than strength-τ are removed and all glues greater than strength-τ are set to strength-1. We modify the system in this way and then use the known polynomial time algorithm for temperature-1 UAV [16].

▶ **Lemma 10.** *For any 2HAM system $\Gamma = (\Sigma, \tau)$ and tree-bonded assembly A, if NonCoop-UAV(Γ, A) (Algorithm 1) is true, and Γ does not uniquely assemble A, then there exists assemblies B, B_1, B_2, s.t. $B \not\sqsubseteq A$, $B_1, B_2 \sqsubseteq A$, B_1 and B_2 combine to form B by utilizing cooperative binding.*

Proof. Since NonCoop-UAV(Γ, A) is true, but Γ does not uniquely assemble A, there must exist a rogue assembly $B' \not\sqsubseteq A$ since any subassembly of A would be tree-bonded. Consider an assembly tree Υ'_B for B'. Since NonCoop-UAV(Γ, A) is true, the singleton tile leaves of Υ'_B must be subassemblies of A. We will show there exists a node $B \in \Upsilon'_B$, with children B_1 and B_2, respectively, such that $B' \not\sqsubseteq A$, and $B_1, B_2 \sqsubseteq A$.

Let the root node of the tree Υ_B be the candidate node B, and let assemblies B_1 and B_2 be the two children of B. If B_1 and B_2 are both subassemblies of A, then the conditions are met. Otherwise, W.L.O.G. assume $B_1 \not\sqsubseteq A$. Now set the candidate node B to B_1 and repeat. Since all leaves of Υ_B represent subassemblies of A, there must be a point in which the candidate B node is some assembly $B \not\sqsubseteq A$, and its children are assemblies $B_1, B_2 \sqsubseteq A$. ◀

Algorithm 2 Algorithm to solve Temp2-Tree-UAV in $\mathcal{O}(|A|^5)$ time.

Data: 2HAM System $\Gamma = (\Sigma, 2)$, Tree-Bonded Assembly A of height h and width w
Result: Does Γ uniquely produce A?
if *NonCoop-UAV*(Γ, A) *rejects* **then** reject;
Let A_0 be the origin configuration of assembly A;
for $i \leftarrow -w$ **to** w **do**
  for $j \leftarrow -h$ **to** h **do**
    $A' \leftarrow A_0 + \langle i, j \rangle$;
    Let $\mathcal{B}$ be the set of binding sites between A_0 and A';
    for *each pair of binding sites* $a, b \in \mathcal{B}$ **do**
      if *The loop formed using* a, b *does not intersect itself* **then** reject;
accept;

4.3 Temperature-2

With respect to the given instance of Temp2-Tree-UAV P, if P is false, and the algorithm for NonCoop-UAV(Γ, A) returns "accept", then Lemma 10 implies there exist two subassemblies of the target, B_1 and B_2, that attach to each other using cooperative binding.

To find these two subassemblies, we take two "copies" of the target assembly and find all $|A|^2$ possible ways to combine them- even if it results in intersections. If any combination results in at least two binding sites, we attempt to find 2-combinable subassemblies. Since these subassemblies are also tree-bonded, there only exists one path between each pair of tiles- including the binding sites. So for each pair of binding sites, we take the loop formed by the two binding sites and check if it intersects itself. An example of a loop that intersects itself is shown in Figure 15d. If there ever exists a pair of binding sites whose paths do not intersect, then those two subassemblies will form a rogue assembly and we reject.

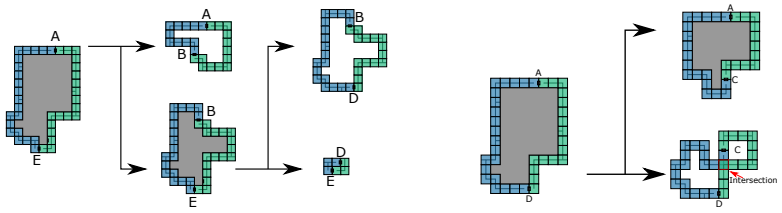
▶ **Theorem 11.** *There is a $\mathcal{O}(|A|^5)$ time algorithm that decides Temp2-Tree-UAV.*

Due to space constraints, the analysis of Alg. 2 and proof of Thm. 11 have been omitted.

4.4 Variable Temperature

We now present an algorithm for Tree-UAV as a generalization of the previous problem where the temperature of the system τ is given as input. The algorithm is similar except it does not suffice to only find a single loop since the temperature requirement attachment may be greater than 2. We must find multiple loops between binding sites that do not intersect. Once we find a way to combine the assemblies, we view binding sites and loops hierarchically using inner binding sites. An example of an inner binding site is in Figure 15c. We recursively calculate the max binding strength when taking each pair of binding sites as the outer loop. After calling NonCoop-UAV(Γ, A), we check each possible way to attach A to itself. For each of these ways, we build a $b \times b$ table where b is the total number of induced binding sites. For each pair of binding sites, we calculate the maximum value recursively augmented with the table. Thus, we only compute the maximum value once for each loop.

First, if the created loop intersects itself, we cannot use it, so the value is set to -1. Next, we check if the binding sites form a simple loop with no inner binding sites. Here, the max value is the sum of the glue strengths between the binding sites. For loops with inner binding sites, we do a *loop decomposition*, which is the process of breaking a loop into two smaller loops along one of the inner binding sites. An example is shown in Figure 16a.



(a) Loop Decomposition. **(b)** Invalid Loop Decomposition.

Figure 16 (a) One possible way to decompose loops into simple loops based on inner binding sites. (b) Decomposing the loop (A, D) along binding site C results in the loop (C, D) which intersects itself. This means we cannot decompose the loop (A, D) along C.

Algorithm 3 Algorithm to solve Tree-UAV in $\mathcal{O}(|A|^5 \log \tau)$ time.

Data: 2HAM System $\Gamma = (\Sigma, \tau)$, Tree-Bonded Assembly A of height h and width w
Result: Does Γ uniquely produce A?
if *NonCoop-UAV*(Γ, A) *rejects* **then** reject;
Let A_0 be the origin configuration of assembly A;
for $x \leftarrow -w$ **to** w **do**
  for $y \leftarrow -h$ **to** h **do**
    $A' \leftarrow A_0 + \langle x, y \rangle$;
    Let $\mathcal{B}$ be the set of binding sites between A_0 and A' and let $b = |\mathcal{B}|$;
    Create a $b \times b$ table $T_{\mathcal{B}}$ indexed by the elements of $\mathcal{B}$ with all cells empty;
    for *each pair of binding sites* $b_1, b_2 \in \mathcal{B}$ **do**
      if $maxStr(C = (A_0 \bigcup A'), T_{\mathcal{B}}, b_1, b_2) \geq \tau$ **then** reject;
accept;

To find the max binding strength of the outer loop, we break the loop up along each inner binding site and recursively get the max strength of the two resulting loops (subtracted by the glue strength of the inner binding site since it would be counted twice). If either of the smaller loops intersects itself, it returns -1 and we know not to use that inner binding site. The max binding strength of the outer loop is then the maximum of these computed values over all choices of inner binding sites. The recursive checks are implemented with a dynamic programming/memoization table to eliminate repeated recursive calls.

▶ **Theorem 12.** *There is a* $\mathcal{O}(|A|^5 \log \tau)$ *time algorithm that decides* Tree-UAV.

Due to space constraints, the analysis of Alg. 3 and proof of Thm. 12 have been omitted.

5 Conclusion

In this paper, we have addressed the long-standing open problem of the complexity of verifying unique assembly within the 2-handed tile self-assembly model and shown that the problem is coNP-complete even at temperature $\tau = 2$ and in two dimensions. These are the smallest possible values for which this problem can be hard, as both temperature-1 self-assembly and 1-dimensional self-assembly have established polynomial time verification solutions. Given this hardness, we explored a natural scenario where this problem might be more tractable, and showed that restricting the input assemblies to tree-bonded assemblies allows for an efficient $\mathcal{O}(|A|^5 \log \tau)$-time unique assembly verification algorithm.

Algorithm 4 $maxStr(C, T_{\mathcal{B}}, b_1, b_2)$. The subroutine that calculates the max strength when using two binding sites as the outer loop. The method $glueStr(b)$ takes in a binding site and returns the strength of the glue connecting the two tiles.

Data: Union of two assemblies C, Table T_B, and Binding sites b_1, b_2
Result: The maximum binding strength used to build a stable subassembly of C.
if $T_{\mathcal{B}}(b_1, b_2)$ *is empty* **then**
  if *The loop formed by* b_1, b_2 *intersects itself* **then** return -1;
  if $|I(b_1, b_2)| = 0$ **then** return $glueStr(b_1) + glueStr(b_2)$;
  Let $T_B(b_1, b_2) = 0$;
  for $b_i \in I(b_1, b_2)$ **do**
    if $maxStr(b_1, b_i) \neq -1$ *AND* $maxStr(b_i, b_2) \neq -1$) **then**
      $s \leftarrow maxStr(b_1, b_i) + maxStr(b_i, b_2) - glueStr(b_i)$;
      if $s > T_{\mathcal{B}}(b_1, b_2)$ **then** $T_{\mathcal{B}}(b_1, b_2) \leftarrow s$;
return $T_{\mathcal{B}}(b_1, b_2)$;

Future Work. While we have resolved the general question of unique assembly verification in the 2HAM, as well as addressed a natural restricted case of tree-bonded assemblies, there remain important directions for future research.

- Our hardness reduction utilizes a tile set that is roughly the size of the input assembly. All hardness results in the literature for the 2-handed self-assembly model have this property. Yet, the computational power of self-assembly allows for the assembly of large assemblies with small tile sets, as seen in the efficient self-assembly of squares [1, 24], or the implementation of "Busy Beaver" Turing machines [24]. How hard is UAV for large assemblies with substantially smaller tile sets. Does the hardness scale with assembly size or tile set size? Is there some form of fixed-parameter tractability for the UAV problem?
- We proved that UAV for the multiple tile (or q-tile) model is coNP-complete with polynomial-sized assemblies attaching. Is UAV polynomial in the multiple tile model and the 2HAM in the case where every producible assembly, except the one that grows into the terminal assembly, is bounded by a constant?
- A related question in the aTAM and the 2HAM is the number of two-handed operations actually required to make the problem hard. If all subassemblies can only grow by single tile attachments, how many two-handed operations to combine those subassemblies are needed for UAV to remain hard? The ability to more efficiently construct shapes by assembling parts separately has been studied in other models as well [25].
- Another direction initiated by our efficient tree assembly algorithm is the consideration of other natural restricted classes of the UAV problem. For example, how does UAV scale with respect to the genus of an assembly's connectivity graph? A related question involves verification for *fully connected* assemblies, a previously-studied concept [14] in which assemblies include positive bonds between all neighboring tiles.

References

1 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, STOC'01, pages 740–748, 2001. doi:10.1145/380752.380881.

2 Leonard M. Adleman, Qi Cheng, Ashish Goel, Ming-Deh A. Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proc. of the 34th ACM Sym. on Theory of Computing*, pages 23–32, 2002.

3 Pankaj K. Agarwal, Boris Aronov, Tzvika Geft, and Dan Halperin. On two-handed planar assembly partitioning with connectivity constraints. In *Proc. of the ACM-SIAM Syme on Discrete Algorithms*, SODA'21, pages 1740–1756, 2021. doi:10.1137/1.9781611976465.105.

4 Gagan Aggarwal, Qi Cheng, Michael H. Goldwasser, Ming-Yang Kao, Pablo Moisset de Espanes, and Robert T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005. doi:10.1137/S0097539704445202.

5 David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. The complexity of multiple handed self-assembly. In *Proceedings of the International Conference on Unconventional Computation and Natural Computation*, UCNC'21, pages 1–18, 2021.

6 David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Covert computation in staged self-assembly: Verification is pspace-complete. In *Proceedings of the 29th European Symposium on Algorithms*, ESA'21, pages 23:1–23:18, 2021.

7 David Caballero, Timothy Gomez, Robert Schweller, and Tim Wylie. Complexity of verification in self-assembly with prebuilt assemblies. In *Proceedings of the Symposium on Algorithmic Foundations of Dynamic Networks*, SAND'22, 2022.

8 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, David Furcy, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. On the effects of hierarchical self-assembly for reducing program-size complexity. *Theoretical Computer Science*, 894:50–78, 2021. doi:10.1016/j.tcs.2021.09.011.

9 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M Summers, and Andrew Winslow. Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM. In *30th Inter. Sym. on Theoretical Aspects of Computer Science*, volume 20 of *STACS'13*, pages 172–184, 2013.

10 Angel A. Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. Covert Computation in Self-Assembled Circuits. *Algorithmica*, 83:531–552, 2021. arXiv:1908.06068.

11 Cameron Chalk, Dominic Fernandez, Alejandro Huerta, Mario Maldonado, Robert Schweller, and Leslie Sweet. Strict self-assembly of fractals using multiple hands. *Algorithmica*, 76, July 2014. doi:10.1007/s00453-015-0022-x.

12 Ho-Lin Chen and David Doty. Parallelism and time in hierarchical self-assembly. *SIAM Journal on Computing*, 46(2):661–709, 2017. Preliminary version appeared in SODA 2012.

13 Erik D Demaine, Martin L Demaine, Sándor P Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T Schweller, and Diane L Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with o(1) glues. *Natural Computing*, 7(3):347–370, 2008.

14 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. *Theoretical Comp. Science*, 671:4–18, 2017.

15 David Doty. Theory of algorithmic self-assembly. *Comm. of the ACM*, 55(12):78–88, 2012.

16 David Doty. Producibility in hierarchical self-assembly. In *Proc. of the Inter. Conf. on Unconventional Computation and Natural Computation*, UCNC'14, pages 142–154, 2014.

17 Algorithmic Self-Assembly Research Group. VersaTile. github.com/asarg/VersaTile, 2014.

18 Jacob Hendricks and Joseph Opseth. Self-assembly of 4-sided fractals in the two-handed tile assembly model. *Natural Computing*, 18:75–92, 2018.

19 Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992. doi:10.1137/0405033.

20 Pierre-'Etienne Meunier, Damien Regnault, and Damien Woods. The program-size complexity of self-assembled paths. In *Proc. of the 52nd Annual ACM SIGACT Sym. on Theory of Computing*, STOC 2020, pages 727–737, 2020. doi:10.1145/3357713.3384263.

21 Pierre-'Etienne Meunier and Damien Woods. The non-cooperative tile assembly model is not intrinsically universal or capable of bounded turing machine simulation. In *Proc. of the 49th Annual ACM SIGACT Sym. on Theory of Computing*, STOC'17, pages 328–341, 2017.

22 Matt Patitz. Pytas. URL: http://self-assembly.net/wiki/index.php?title=PyTAS.

23 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, June 2014.

24 Paul WK Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proc. of the 32nd annual ACM Sym. on Theory of computing*, pages 459–468, 2000.

25 Arne Schmidt, Sheryl Manzoor, Li Huang, Aaron Becker, and Sándor P. Fekete. Efficient parallel self-assembly under uniform control inputs. *Robotics and Automation Letters*, 3:3521–3528, 2018.

26 Robert Schweller, Andrew Winslow, and Tim Wylie. Complexities for high-temperature two-handed tile self-assembly. In *Proc. of the 23rd Inter. Conf. on DNA Computing and Molecular Programming*, DNA'17, pages 98–109, 2017.

27 Robert Schweller, Andrew Winslow, and Tim Wylie. Verification in staged tile self-assembly. *Natural Computing*, 18(1):107–117, 2019.

28 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

29 Erik Winfree, Rebecca Schulman, and Constantine Evans. The xgrow simulator. URL: `https://www.dna.caltech.edu/Xgrow/`.

30 Damien Woods. Intrinsic universality and the computational power of self-assembly. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373(2046):20140214, 2015.

31 Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable dna self-assembly. *Nature*, 567(7748):366–372, 2019. `doi:10.1038/s41586-019-1014-9`.

Pairwise Reachability Oracles and Preservers Under Failures

Diptarka Chakraborty
National University of Singapore, Singapore

Kushagra Chatterjee
National University of Singapore, Singapore

Keerti Choudhary
Indian Institute of Technology Delhi, India

Abstract

In this paper, we consider reachability oracles and reachability preservers for directed graphs/networks prone to edge/node failures. Let $G = (V, E)$ be a directed graph on n-nodes, and $\mathcal{P} \subseteq V \times V$ be a set of vertex pairs in G. We present the first non-trivial constructions of single and dual fault-tolerant pairwise reachability oracle with constant query time. Furthermore, we provide extremal bounds for sparse fault-tolerant reachability preservers, resilient to two or more failures. Prior to this work, such oracles and reachability preservers were widely studied for the special scenario of single-source and all-pairs settings. However, for the scenario of arbitrary pairs, no prior (non-trivial) results were known for dual (or more) failures, except those implied from the single-source setting. One of the main questions is whether it is possible to beat the $O(n|\mathcal{P}|)$ size bound (derived from the single-source setting) for reachability oracle and preserver for dual failures (or $O(2^k n|\mathcal{P}|)$ bound for k failures). We answer this question affirmatively. Below we summarize our contributions.

- For an n-vertex directed graph $G = (V, E)$ and $\mathcal{P} \subseteq V \times V$, we present a construction of $O(n\sqrt{|\mathcal{P}|})$ sized dual fault-tolerant pairwise reachability oracle with constant query time. We further provide a matching (up to the word size) lower bound of $\Omega(n\sqrt{|\mathcal{P}|})$ on the size (in bits) of the oracle for the dual fault setting, thereby proving that our oracle is (near-)optimal.
- Next, we provide a construction of $O(n + \min\{|\mathcal{P}|\sqrt{n},\ n\sqrt{|\mathcal{P}|}\})$ sized oracle with $O(1)$ query time, resilient to single node/edge failure. In particular, for $|\mathcal{P}|$ bounded by $O(\sqrt{n})$ this yields an oracle of just $O(n)$ size. We complement the upper bound with a lower bound of $\Omega(n^{2/3}|\mathcal{P}|^{1/2})$ (in bits), refuting the possibility of a linear-sized oracle for $\mathcal{P}$ of size $\omega(n^{2/3})$.
- We also present a construction of $O(n^{4/3}|\mathcal{P}|^{1/3})$ sized pairwise reachability preservers resilient to dual edge/vertex failures. Previously, such preservers were known to exist only under single failure and had $O(n + \min\{|\mathcal{P}|\sqrt{n},\ n\sqrt{|\mathcal{P}|}\})$ size [Chakraborty and Choudhary, ICALP'20]. We also show a lower bound of $\Omega(n\sqrt{|\mathcal{P}|})$ edges on the size of dual fault-tolerant reachability preservers, thereby providing a sharp gap between single and dual fault-tolerant reachability preservers for $|\mathcal{P}| = o(n)$.
- Finally, we provide a generic pairwise reachability preserver construction that provides a $o(2^k n|\mathcal{P}|)$ sized subgraph resilient to k failures, for any $k \geq 1$. Before this work, we only knew of an $O(2^k n|\mathcal{P}|)$ bound implied from the single-source setting [Baswana, Choudhary, and Roditty, STOC'16].

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Sparsification and spanners

Keywords and phrases Fault-tolerant, Reachability Oracle, Reachability Preservers, Graph sparsification, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.35

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2110.11613

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 35; pp. 35:1–35:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Diptarka Chakraborty*: The author is supported in part by NUS ODPRT Grant, WBS No. A-0008078-00-00.
Kushagra Chatterjee: The author is supported in part by NUS ODPRT Grant, WBS No. A-0008078-00-00.
Keerti Choudhary: The author is supported in part by Google India Algorithms Research grant 2021.

Acknowledgements The authors would like to thank anonymous reviewers for many helpful comments on a preliminary version of this work, especially for pointing out [9].

1 Introduction

Networks in most real-life applications are prone to failures. These failures, though unpredictable, are transient due to some simultaneous repair process that is undertaken in the application. This motivates the research on designing fault-tolerant structures for various graph problems. In the past few years, a lot of work has been done in designing fault-tolerant structures for various graph problems like connectivity [34, 32, 4, 26, 11], finding shortest paths [20], graph-structures preserving approximate distances [30, 19, 15, 22, 5, 6, 10, 3] etc. Reachability is one of the fundamental graph properties which is as ubiquitous as graphs themselves. In this paper, we study pairwise reachability structures under edge/node failures. In particular, given any set $\mathcal{P}$ of node-pairs, we provide design of graph sparsification structures, and sensitivity oracles for the reachability problem. We present our results in terms of edge failures. However, all our upper bound results also hold for node failures.[1]

1.1 Sensitivity Oracle

In the Sensitivity oracle, the goal is to design a data structure for a network prone to edge/vertex failures to efficiently answer queries pertaining to the graph structure (e.g., connectivity, reachability, distance, etc.). We first formally define the notion of Fault-Tolerant Reachability Oracle (FTRO).

▶ **Definition 1** (FTRO). *Let $\mathcal{P} \in V \times V$ be any set of pairs of vertices. For a graph G, a data structure $DS(G)$ is said to be a* k-Fault-Tolerant Reachability Oracle *of G for $\mathcal{P}$, denoted as k-FTRO$(G, \mathcal{P})$, if given a query with any pair $(s, t) \in \mathcal{P}$ and any subset $F \subseteq E$ of at most k edges, $DS(G)$ efficiently decides whether or not t is reachable from s in $G \setminus F$.*

To date, no non-trivial bounds were known for FT-pairwise reachability oracle. The only known results are for single-source setting (i.e., $\mathcal{P} = \{s\} \times V$) and all-pairs setting $\mathcal{P} = V \times V$.

For single-source setting, i.e., when $\mathcal{P} = \{s\} \times V$ for some source vertex $s \in V$, under (single and) dual failure, we have an $O(n)$ size oracle with $O(1)$ query time due to [29, 17]. As an immediate corollary, for arbitrary $\mathcal{P}$ pairs, we get an $O(n|\mathcal{P}|)$-sized single/dual failure pairwise FTRO with constant query time. The bound is extremely bad for a large-sized set $\mathcal{P}$. By storing a subgraph that preserves pairwise reachability (to be discussed in detail in Section 1.2) under single failure due to [11], we get an 1-FTRO of size $O(n + \min\{\sqrt{n}|\mathcal{P}|, n\sqrt{|\mathcal{P}|}\})$ but with $O(n)$ query time. The $O(n)$ query time is due to the fresh reachability computation over

[1] In the input graph, each vertex v can be replaced by an edge (v_{in}, v_{out}), where all the incoming and outgoing edges of v are directed into v_{in} and directed out of v_{out} respectively. Thus the failure of vertex v is equivalent to failure of edge (v_{in}, v_{out}).

the stored subgraph on each query, which is entirely undesirable in terms of the efficiency of a data structure. For the special setting of all-pairs, i.e., $\mathcal{P} = V \times V$, Brand and Saranurak [37] provided a $O(n^2)$ sized k-FTRO that has $O(k^\omega)$ query time, where ω is the constant of matrix-multiplication.

One of the main questions is the following: Does there exist a pairwise reachability oracle of size $o(n|\mathcal{P}|)$ and query time $o(n)$ even for a single failure? In this paper, we answer this question affirmatively. We provide an efficient construction of a $O(n\sqrt{|\mathcal{P}|})$ sized FTRO with constant query time that is resilient to dual failure (not just single failure).

▶ **Theorem 2** (Upper Bound on 2-FTRO). *A directed graph $G = (V, E)$ with n vertices can be processed in randomized polynomial time for a given set $\mathcal{P} \subseteq V \times V$ of vertex-pairs, to build a data structure of size $O(n\sqrt{|\mathcal{P}|})$, such that for any pair $(s, t) \in \mathcal{P}$ and any set F of (at most) two edge failure, it decides whether there is an s to t path in $G \setminus F$ in time $O(1)$.*

We further show that the above size bound cannot be improved further by providing a matching (up to the word size) lower bound for two failures. To date, no non-trivial (better than linear) size lower bound is known for any pairwise FTRO.

▶ **Theorem 3** (Lower Bound on 2-FTRO). *For any positive integers n, r ($r \leq n^2$), there exists an n-vertex directed graph with a vertex-pair set $\mathcal{P}$ of size r, such that any 2-FTRO$(G, \mathcal{P})$ must be of size $\Omega(n\sqrt{|\mathcal{P}|})$ (in bits).*

In case of source-wise 2-FTRO for a source set S (i.e, when $\mathcal{P} = S \times V$), our lower bound construction provides a lower bound of $\Omega(n|S|)$ (in bits). It is again a matching (up to the word size) lower bound because we know of an $O(n|S|)$-sized 2-FTRO for any source set S due to [17]. It is also worth noting that our lower bound holds irrespective of the query time and also for directed acyclic graphs.

The above lower bound does not hold for a single failure. So it is natural to ask whether we can design a smaller data structure, more specifically, $O(n)$-sized oracle that is resilient to a single failure. We provide a construction of $O(n + \min\{|\mathcal{P}|\sqrt{n}, n\sqrt{|\mathcal{P}|}\})$ sized 1-FTRO with constant $O(1)$ query time. In particular, we show that as long as the number of pairs is bounded by $O(\sqrt{n})$, we can achieve an oracle with $O(n)$ size and $O(1)$ query time. This result provides us a sharp separation in optimum size of a FTRO between single and dual failure. To the best of our knowledge, this is the first separation result between single and dual failure reachability oracle.

▶ **Theorem 4** (Upper Bound on 1-FTRO). *A directed graph $G = (V, E)$ with n vertices can be processed in polynomial time for a given set $\mathcal{P} \subseteq V \times V$ of vertex-pairs, to build a data structure of size $O(n + \min\{|\mathcal{P}|\sqrt{n}, n\sqrt{|\mathcal{P}|}\})$, such that for any pair $(s, t) \in \mathcal{P}$ and a failure edge f, it decides whether there is an s to t path in $G \setminus \{f\}$ in time $O(1)$.*

Note, the size bound of the above theorem matches the current best known bound for the pairwise reachability preserving subgraph for single failure [11].

The above upper bound gives $O(n)$ sized oracle only when the number of pairs is $O(\sqrt{n})$. Is it always possible to get a linear-sized pairwise 1-FTRO? More specifically, does any n-node graph G and a set $\mathcal{P}$ of node-pairs always possess a 1-FTRO$(G, \mathcal{P})$ of size $O(n + |\mathcal{P}|)$?[2] In this paper, we refute this possibility by showing the following.

[2] The presence of $|\mathcal{P}|$ term in the bound is justifiable by the fact that for non-failure case (i.e., the standard static setting), we can get a trivial $O(|\mathcal{P}|)$ sized oracle.

▶ **Theorem 5** (Lower Bound on 1-FTRO). *For any positive integers $n, d \geq 2$, any $p = p(n)$, there exists an n-vertex directed graph and a node-pair set $\mathcal{P}$ of size p, such that any 1-FTRO$(G, \mathcal{P})$ must be of size $\Omega\big(n^{2/(d+1)}p^{(d-1)/d}\big)$ (in bits).*

By setting $d = 2$ in the above theorem, we get a lower bound of $\Omega(n^{2/3}p^{1/2})$. This shows us that for $p = \omega(n^{2/3})$, there is an n-node graph G and a pair set $\mathcal{P}$ of size p, for which linear size 1-FTRO is not possible. Again, our lower bound holds irrespective of the query time and also for DAGs. We show the lower bound by establishing a connection between the optimal sized pairwise 1-FTRO and pairwise reachability preserving subgraph without any failure. In general, we show that the optimal size of any pairwise k-FTRO must be at least that of the reachability preserving subgraph with $k-1$ failures. Instead of just deciding the reachability between a pair of vertex, suppose the data structure is also asked to report a path between them (if exists). Then by a standard information-theoretic argument, the optimal size of any such data structure resilient to k failures must be of size at least that of reachability preserving subgraph with k failures. Unfortunately, such a direct argument does not work for a (Boolean) data structure that only decides the reachability. Ours is the first such connection. Readers may note that there is a gap between our upper and lower bound for pairwise 1-FTRO. We leave this as an interesting open question.

1.2 Reachability Preservers

In the context of graph sparsification, *reachability preserver* (or *reachability subgraph*) for a directed graph G and a set $\mathcal{P}$ of vertex-pairs is a sparse subgraph H with as few edges as possible so that for any pair $(s,t) \in \mathcal{P}$ there is a path from s to t in H if and only if there is such a path in G. In the standard static setting (with no failure), this object has been studied widely [18, 8, 1]. We study these objects in the presence of edge/node failures.

Let us formally define fault-tolerant reachability subgraph (FTRS) for a set of node-pairs.

▶ **Definition 6** (FTRS). *Let $\mathcal{P} \in V \times V$ be any set of pairs of vertices. A subgraph H of G is said to be a* k-Fault-Tolerant Reachability Subgraph *of G for $\mathcal{P}$, denoted as k-FTRS$(G, \mathcal{P})$, if for any pair $(s,t) \in \mathcal{P}$ and for any subset $F \subseteq E$ of at most k edges, t is reachable from s in $G \setminus F$ if and only if t is reachable from s in $H \setminus F$.*

For the particular case of single-source, i.e., $\mathcal{P} = \{s\} \times V$, Baswana, Choudhary, and Roditty [4] provided a polynomial-time algorithm that, given any n-node directed graph, constructs an $O(2^k n)$-sized k-FTRS. As a corollary, to preserve reachability between arbitrary $\mathcal{P}$ pairs, we get an $O(2^k n|\mathcal{P}|)$-sized k-fault-tolerant reachability preserver. For the general setting of arbitrary pairs, the only previously known non-trivial result was for single failure [11], wherein the authors gave an upper bound of $O(n + \min(|\mathcal{P}|\sqrt{n},\ n\sqrt{|\mathcal{P}|}))$ edges. It was left open whether for dual or more failures whether keeping fewer than $O(n|\mathcal{P}|)$ edges sufficient to preserve the pairwise reachability. In particular, does any n-node graph and a set $\mathcal{P}$ of node-pairs always admit a k-FTRS of size $o(2^k n|\mathcal{P}|)$?

In this work, we answer the above question affirmatively. For dual failures, we provide an upper bound of $O(n^{4/3}|\mathcal{P}|^{1/3})$ edges on the structure of 2-FTRS$(G, \mathcal{P})$.

▶ **Theorem 7** (Upper Bound on 2-FTRS). *For any directed graph $G = (V, E)$ with n vertices and a set $\mathcal{P} \subseteq V \times V$ of vertex-pairs, there exists a 2-FTRS$(G, \mathcal{P})$ having at most $O(n^{4/3}|\mathcal{P}|^{1/3})$ edges. Furthermore, we can find such a subgraph in polynomial time.*

Clearly, for $\mathcal{P}$ of size $\omega(\sqrt{n})$, the above result breaks below the $O(n|\mathcal{P}|)$ bound. We complement our upper bound result the following lower bound.

▶ **Theorem 8** (Lower Bound on 2-FTRS). *For every n, r ($r \leq n^2$), there exists an n-vertex directed graph G and a vertex-pair set $\mathcal{P}$ of size r such that any 2-FTRS$(G, \mathcal{P})$ requires $\Omega(n\sqrt{|\mathcal{P}|})$ edges.*

Again, we show a lower bound for source-wise 2-FTRS for a source set S (i.e, when $\mathcal{P} = S \times V$), of $\Omega(n|S|)$. This matches the $O(n|S|)$ upper bound [4] of 2-FTRS for any source set S. So for the source-wise preserver, we completely resolve the question regarding the size of an optimal preserver resilient to two or more failures. For general $k > 2$ failures, we provide a lower bound of $\Omega(2^{k/2}n\sqrt{|\mathcal{P}|})$ on the size of pairwise k-FTRS.

Previously, seemingly a much weaker lower bound was known [11], where the authors could only show a lower bound of $\Omega(n|\mathcal{P}|^{1/8})$, for $\mathcal{P}$ of size n^ϵ with $\epsilon \leq 2/3$. Our result provides a sharp separation between single and dual fault-tolerant reachability preservers for any $\mathcal{P}$ satisfying $\omega(1) \leq |\mathcal{P}| \leq o(n)$.

We also consider the question of beating $O(2^k n|\mathcal{P}|)$ bound for general k-FTRS. We show that for a certain regime of the size of $\mathcal{P}$, it is indeed possible to attain $o(2^k n|\mathcal{P}|)$ bound.

▶ **Theorem 9** (Upper Bound on k-FTRS). *For any $k \geq 1$, a directed graph $G = (V, E)$ with n vertices and a set $\mathcal{P} \subseteq V \times V$ of vertex-pairs satisfying $|\mathcal{P}| = \omega(kn^{1-\frac{1}{k}} \log n)$, there exists a k-FTRS$(G, \mathcal{P})$ having only $o(2^k n|\mathcal{P}|)$ edges.*

We summarize our results on single and dual failures in Table 1. Readers may note that there is a gap between the size of 2-FTRO and 2-FTRS in our results. We pose closing this embarrassing gap as an interesting open question.

Table 1 A comparison of size of FTRO and FTRS for single and dual failures.

Problem	Single Failure	Dual Failure
Reachability Oracle	$O(n + \min(\lvert\mathcal{P}\rvert\sqrt{n}, n\sqrt{\lvert\mathcal{P}\rvert}))$ $\Omega(n^{2/3}\lvert\mathcal{P}\rvert^{1/2})$ (in bits) (New)	$O(n\sqrt{\lvert\mathcal{P}\rvert})$ $\Omega(n\sqrt{\lvert\mathcal{P}\rvert})$ (in bits) (New)
Reachability Preserver	$O(n + \min(\lvert\mathcal{P}\rvert\sqrt{n}, n\sqrt{\lvert\mathcal{P}\rvert}))$ [11]	$O(n^{4/3}\lvert\mathcal{P}\rvert^{1/3})$ $\Omega(n\sqrt{\lvert\mathcal{P}\rvert})$ (New)

1.3 Related Work

A simple version of reachability preserver is when there is a single source vertex s, and we would like to preserve reachability from s to all other vertices. Baswana *et al.* [4] provided an efficient construction of a k-fault-tolerant single-source reachability preserver of size $O(2^k n)$. Further, they showed that this upper bound on the size of a preserver is tight up to some constant factor. As an immediate corollary, we get a k-FTRS of size $O(2^k n|\mathcal{P}|)$ (by applying the algorithm of [4] to find subgraph for each source vertex in pairs of $\mathcal{P}$ and then taking the union of all these subgraphs). We do not know whether this bound is tight for general k. However, for the standard static setting (with no faulty edges) much better bound is known. We know that even to preserve all the pairwise distances, not just reachability, there is a subgraph of size $O\big(n + \min(n^{2/3}|\mathcal{P}|, n\sqrt{|\mathcal{P}|})\big)$ [18, 8]. Later Abboud and Bodwin [1] showed that for any directed graph $G = (V, E)$ given a set S of

source vertices and a pair-set $\mathcal{P} \subseteq S \times V$ we can construct a pairwise reachability preserver of size $O\big(n + \min(\sqrt{n|\mathcal{P}||S|}, (n|\mathcal{P}|)^{2/3})\big)$. It is further shown that for any integer $d \geq 2$ there is an infinite family of n-node graphs and vertex-pair sets $\mathcal{P}$ for which any pairwise reachability preserver must be of size $\Omega\big(n^{2/(d+1)}|\mathcal{P}|^{(d-1)/d}\big)$. Note, for undirected graphs, storing spanning forests is sufficient to preserve pairwise reachability information, and thus we can always get a linear size reachability preserver for undirected graphs. We would like to emphasize that all our results in this paper hold for directed graphs.

By [4] we immediately get an oracle of size $O(2^k n)$ for k edge (or vertex) failures with query time $O(2^k n)$. For just dual failures, we have an $O(n)$ size oracle with $O(1)$ query time due to [17].

For undirected graphs, the optimal bound of $O(kn)$ edges for k-fault-tolerant connectivity preserver directly follows from k-edge (vertex) connectivity certificate constructions provided by Nagamochi and Ibaraki [31]. For connectivity oracle, Pătraşcu and Thorup [35] presented a data structure of $O(m)$ size that can handle any k edge failures in $O(k \log^2 n \log\log n)$ time to subsequently answer connectivity queries between any two vertices in $O(\log\log n)$ time. For small values of k, Duan and Pettie [24] improved the update time of [35] to $O(k^2 \log\log n)$ by presenting a data structure of $\widetilde{O}(m)$ size. For handling vertex failures, Duan and Pettie [25] provided a data structure of $O(mk \log n)$ size with $O(k^3 \log^3 n)$ update time and $O(k)$ query time.

Other closely related problems that have been studied in the fault-tolerant model include computing distance preservers [20, 33, 32, 12], depth-first-search tree [2], spanners [15, 22], approximate distance preservers [5, 34, 7], approximate distance oracles [23, 16], compact routing schemes [16, 14].

2 Preliminaries

For any integer n, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. Given a directed graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, the following notations will be used throughout the paper.

- $V(H)$: The set of vertices present in a graph H.
- $E(H)$: The set of edges present in a graph H.
- $H \setminus F$: For a set of edges F, the graph obtained by deleting the edges in F from graph H.
- $s - t$ path : A directed path from a vertex s to another vertex t.
- $P \circ Q$: The concatenation of two paths P and Q, i.e., a path that first follows P and then Q.
- $P[L]$: The subpath of the path P containing the first L vertices of P.
- $P[-L]$: The subpath of the path P containing the last L vertices of P.
- $P[u - v]$: The $u - v$ subpath of the path P.

Our algorithm for computing *pairwise-reachability* preservers (and oracles) in a fault tolerant environment employs the concept of a *single-source* FTRS which is a sparse subgraph that preserves reachability from a designated source vertex even after the failure of at most k edges in G. Baswana *et al.* [4] provide a construction of sparse k-FTRS for any general $k \geq 1$ when there is a designated source vertex.

▶ **Theorem 10** ([4]). *For any directed graph $G = (V, E)$, a designated source vertex $s \in V$, and an integer $k \geq 1$, there exists a (sparse) subgraph H of G which is a k-FTRS$(G, \{s\} \times V)$ and contains at most $2^k n$ edges. Moreover, such a subgraph is computable in $O(2^k mn)$ time, where n and m are respectively the number of vertices and edges in graph G.*

Note, in the FTRO definition we restrict ourselves to a data structure with constant query time. (The term *oracle* came from its ability to answer a query in constant time.) Unlike single-source k-FTRS, there is no non-trivial construction of single-source k-FTRO for $k \geq 3$. For $k = 2$, the following result by Choudhary [17] provides an $O(n)$ size oracle.

▶ **Theorem 11** ([17]). *There is a polynomial time algorithm that given any directed graph $G = (V, E)$, a designated source vertex $s \in V$, constructs a 2-FTRO$(G, \{s\} \times V)$ of size $O(n)$.*

Our constructions will require the knowledge of the vertices reachable from a vertex s as well as the vertices that can reach s. So we will use FTRS (and FTRO) defined with respect to a source vertex ($\{s\} \times V$ case), as well as FTRS (and FTRO) defined with respect to a destination vertex ($V \times \{s\}$ case).

In this paper, we consider fault-tolerant structures with respect to edge failures only. Vertex failures can be handled by simply splitting a vertex v into an edge (v_{in}, v_{out}), where the incoming and outgoing edges of v are respectively directed into v_{in} and directed out of v_{out}.

3 Technical Overview

Construction of a 2-FTRS for a single pair

Our starting point is a simple construction of a linear (in the number of vertices) sized 2-FTRS for a single pair. Recall, we already know of such a subgraph by [4]. However, this new alternate construction will shed more light on the specific structure of a 2-FTRS, which will play a pivotal role in our constructions of pairwise 2-FTRS and 2-FTRO. Given a directed graph G and a vertex-pair (s, t), we construct a subgraph $H_{(s,t)}$ as follows: First, consider two "maximally disjoint" $s-t$ paths $P^1_{(s,t)}$ and $P^2_{(s,t)}$ (that meet only at the cut-edges and cut-vertices). We refer to these two paths as *outer strands*. Next, we add several *coupling paths* between these two outer strands, which are edge-disjoint with the outer strands. For each vertex v on the outer strands, we check for the "earliest" vertex on the strand $P^1_{(s,t)}$ (and $P^2_{(s,t)}$), from which there is a path $Q^1_{(s,t),v}$ (and $Q^2_{(s,t),v}$) to v that is edge-disjoint with both the outer strands. We refer to these path $Q^i_{(s,t),v}$ as coupling paths. Roughly speaking, two outer strands together with the coupling paths constitute the subgraph $H_{(s,t)}$ (see Figure 1). The actual construction is slightly different. Let us first briefly discuss why the above subgraph is a 2-FTRS for the pair (s, t). Then we will comment on the issue with the above simple construction and how we overcome that.

Consider any two failure edges f_1, f_2. W.l.o.g. assume, they do not form an $s - t$ cut-set; otherwise, after the failure there won't be any $s - t$ path. Thus if both f_1, f_2 lie on one of the two outer strands (i.e., either on $P^1_{(s,t)}$ or $P^2_{(s,t)}$), then since these two strands are maximally disjoint, one of them will survive after the failures. So, let f_1, f_2 lie on the strand $P^1_{(s,t)}$, $P^2_{(s,t)}$ respectively. Then consider the subpaths of $P^1_{(s,t)}$, $P^2_{(s,t)}$ above f_1, f_2, and the subpaths of $P^1_{(s,t)}$, $P^2_{(s,t)}$ below f_1, f_2. Since by assumption f_1, f_2 does not form an $s - t$ cut-set, there must be a coupling path (edge-disjoint with $P^1_{(s,t)}$, $P^2_{(s,t)}$) from one of the top subpaths to one of the bottom subpaths in G. Since $H_{(s,t)}$ consists of all the coupling paths, we get a surviving path in $H_{(s,t)} \setminus \{f_1, f_2\}$. This shows that $H_{(s,t)}$ is a 2-FTRS$(G, (s, t))$. Moreover, one may observe from the above argument that, after failure of any two edges, one of the surviving paths in $H_{(s,t)}$ must be of the following form: It first follows one of the outer strand from s to some vertex u, then takes a coupling path till some vertex v on one of

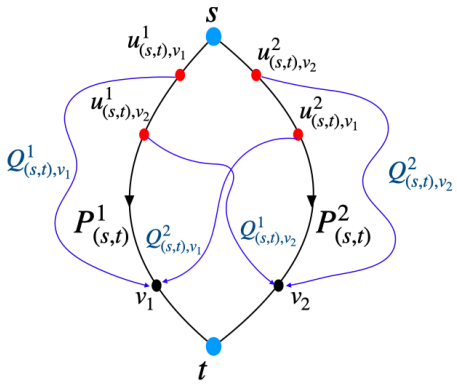


Figure 1 $H_{(s,t)}$=2-FTRS for a single pair (s,t). Two black paths are the outer strands and the purple paths are the coupling paths between them.

the outer strands, and finally follows the corresponding outer strand from v to t. We refer to such a path as *nice path*. The existence of such nice paths helps us in proving the correctness of our pairwise 2-FTRO and 2-FTRS construction in the subsequent sections.

As we mentioned earlier, our actual construction is slightly different. The main issue with the above simple construction is that the constituted subgraph could be of size $\omega(n)$ after adding all the coupling paths. To mitigate this issue, instead of adding all the coupling paths, we only add the "essential" coupling paths. (See the full version for the details.) It allows us to achieve $O(n)$ size bound without affecting the correctness of 2-FTRS. The guarantee of the existence of nice paths also remains unaffected. Of course, the correctness argument will become slightly more intricate.

Next, we use the above construction of a 2-FTRS of a single pair to study the pairwise dual fault-tolerant graph structures (reachability oracle and preserver). Our input is a directed graph $G = (V, E)$ with n nodes, and a node-pair set $\mathcal{P} \subseteq V \times V$.

Pairwise 2-FTRO: Upper bound

One of the main contributions of this work is a construction of a dual fault-tolerant pairwise reachability oracle (2-FTRO) of size $O(n\sqrt{|\mathcal{P}|})$. For simplicity, below, we briefly describe a construction that provides a slightly weaker bound, in particular, $O(n\sqrt{|\mathcal{P}|}\log n)$. Later we will comment on how to remove this extra $\log n$ factor.

We start with the 2-FTRS $H_{(s,t)}$, for each pair $(s,t) \in \mathcal{P}$. First, for all $(s,t) \in \mathcal{P}$, we consider the top and bottom $\Theta(n/\sqrt{|\mathcal{P}|})$ portion of the outer strands $(P^1_{(s,t)}, P^2_{(s,t)})$. We find a subset of vertices that intersects all these subpaths, i.e., acts as a "hitting set". Using a standard greedy algorithm we get a hitting set of size $O(\sqrt{|\mathcal{P}|}\log n)$. (Note, one may alternatively use random sampling to achieve the same bound for the hitting set with high probability.) Then we compute linear-sized single-source and single-destination 2-FTRO having query time $O(1)$, for each of the vertices in the hitting set using [17]. For each $(s,t) \in \mathcal{P}$, in a table $T_{(s,t)}$, we store one vertex from each of the top and bottom $\Theta(n/\sqrt{|\mathcal{P}|})$ length subpaths of $P^1_{(s,t)}$, $P^2_{(s,t)}$, that is also included in the hitting set. That constitutes the first part of our data structure.

Observe, for any two failure edges f_1, f_2 and a pair $(s,t) \in \mathcal{P}$, we know that there must be a surviving *nice* $s-t$ path in $H_{(s,t)} \setminus \{f_1, f_2\}$ (unless f_1, f_2 form an $s-t$ cut-set). Now, if that surviving path follows the bottom (or top) $\Theta(n/\sqrt{|\mathcal{P}|})$ length subpath of $P^j_{(s,t)}$ (for

some $j \in \{1, 2\}$), it must also pass through one of the stored vertices in $T_{(s,t)}$, say v (due to the hitting set property). In this scenario, we can easily check for presence of an $s - v$ and $v - t$ path after the failure of f_1, f_2 in G by trying with all the (at most four) stored vertices in $T_{(s,t)}$. For that, we only need $O(1)$ time during query.

Now, it only remains to consider the case when the surviving nice path in $H_{(s,t)} \setminus \{f_1, f_2\}$ passes though a coupling path $Q^i_{(s,t),v}$, for some vertex v lying on the bottom $\Theta(n/\sqrt{|\mathcal{P}|})$ length subpath of a outer strand, that starts from some vertex u lying on the top $\Theta(n/\sqrt{|\mathcal{P}|})$ length subpath of a outer strand. Informally, we only need to consider the scenario when both the outer strands are of length $O(n/\sqrt{|\mathcal{P}|})$. For simplicity, from now on we continue the description with this assumption. Intuitively, this enables us to look into a smaller graph (which need not be a subgraph of the original graph). We build an *auxiliary graph* $A_{(s,t)}$ (see Figure 2) from $H_{(s,t)}$. We define the auxiliary graph entirely on the vertex set of the outer strands ($P^1_{(s,t)}$, $P^2_{(s,t)}$). First, we add both the outer strands (i.e. all their edges) to the auxiliary graph. Next, we add *auxiliary edges* between the vertices if and only if there is a path between them in $H_{(s,t)}$ that is edge-disjoint with the outer strands. Then, We construct a 2-FTRO (having query time $O(1)$) for $A_{(s,t)}$ using [17]. We do this for all pair $(s, t) \in \mathcal{P}$. Since each auxiliary graph is defined over a set of $O(n/\sqrt{|\mathcal{P}|})$ sized vertex set, we need total $O(n\sqrt{|\mathcal{P}|})$ space. This finishes the description of our data structure. So, the final data structure consists of 2-FTROs of the vertices in the hitting set and 2-FTRO computed over $A_{(s,t)}$'s. Hence, the size of the whole data structure is $O(n\sqrt{|P|}\log n)$.

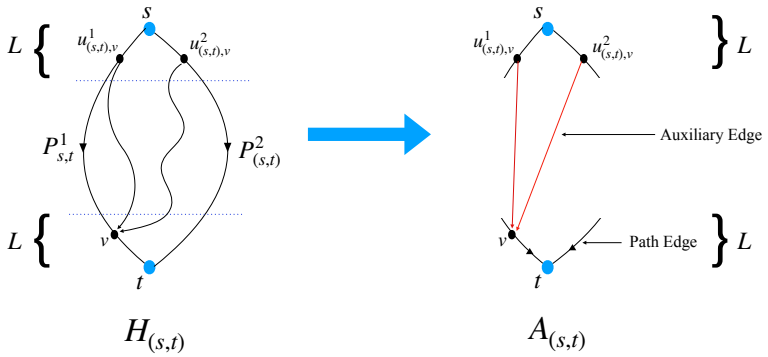


Figure 2 Auxiliary graph $A_{(s,t)}$ constructed from $H_{(s,t)}$.

It is not hard to see that any $s - t$ path of $H_{(s,t)}$ also leads to a valid $s - t$ path in $A_{(s,t)}$ and vice versa. However, it is not immediate that it will be the case even after the failure of f_1, f_2. The difficulty arises because many paths in $H_{(s,t)}$ now map to one path in $A_{(s,t)}$. We show that it is indeed the case that there is an $s - t$ path in $H_{(s,t)} \setminus \{f_1, f_2\}$ if and only if there is an $s - t$ path in the auxiliary graph $A_{(s,t)} \setminus \{f_1, f_2\}$ (given $P^1_{(s,t)}$, $P^2_{(s,t)}$ are of length at most $O(n/\sqrt{P})$). The actual description is slightly more involved because we cannot make any assumption on the length of $P^1_{(s,t)}$, $P^2_{(s,t)}$. Essentially, we need only to consider the top and bottom $O(n\sqrt{|\mathcal{P}|})$ portion of the outer strands and define the auxiliary graph over them. Then we prove the above claim without any assumption on the length of the outer strands. The guarantee on the existence of a nice $s - t$ path in $H_{(s,t)}$ after at most two failures comes handy in this case. Recall, in a nice path, there is at most one coupling sub-path. If both the endpoints of this coupling sub-path lie on the top and bottom $O(n\sqrt{|\mathcal{P}|})$ portion of the

outer strands, we get an auxiliary edge. As a result, we get an $s-t$ path in the auxiliary graph after the failures. We refer the readers to the full version for the details. Note, without the guarantee of a nice path, there could be many coupling sub-paths in a surviving $s-t$ path after failures. As a result, we may not get an auxiliary edge in our auxiliary graph.

During the query for a pair $(s,t) \in \mathcal{P}$, it suffices to either place $O(1)$ many reachability queries on the first part of our data structure or check the presence of an $s-t$ path in the auxiliary graph $A_{(s,t)}$. Hence, we get the overall query time to be only $O(1)$.

To remove the $O(\log n)$ factor from the size bound, we use the concept of *sparsifier with slack* [13, 27, 21, 9]. The extra $O(\log n)$ factor was coming due to the construction of the greedy hitting set. We show that if we prematurely terminate the same greedy hitting set algorithm, we get a *fractional hitting set* that hits a constant fraction of the input sets. As a consequence, we get a pairwise 2-FTRO with slack of size $O(n\sqrt{|\mathcal{P}|})$. Then we use an argument similar to that in [9] to get the same size bound for the (standard) pairwise 2-FTRO.

Optimality of pairwise dual fault-tolerant oracle

In this paper, we show that for pairwise 2-FTRO, our $O(n\sqrt{|\mathcal{P}|})$ bound is essentially tight up to the word size (Theorem 3). Actually, we show that for any source-wise 2-FTRO for a designated source set S, the trivial $O(n|S|)$ upper bound followed from [17], is tight. To do that, we first provide a $\Omega(n|S|)$ lower bound for source-wise 2-FTRS (leading to Theorem 8) by constructing a hard instance. Then we extend that lower bound to source-wise 2-FTRO using communication complexity.

We construct the hard instance for 2-FTRS as follows. Take any integer N, r. We consider two r-sized sets of vertex-disjoint (directed) paths each of length N. Then include a (directed) complete bipartite graph between the left set of r vertices and the right set of r vertices, for each level $k \in [N]$ (See Figure 3). The set S contains the start vertices of the paths in the left set and the terminal vertices of the paths in the right set. So, $|S| = 2r$. The number of vertices and the edges in this graph are $n = 2Nr$ and $\Theta(Nr^2)$ respectively. It is not difficult to observe that each edge in this graph must be present in any 2-FTRS for this graph with the pair set $\mathcal{P} = S \times S$ (See Figure 3). Consequently, we get a $\Omega(n|S|) = \Omega(n\sqrt{|\mathcal{P}|})$ lower bound.

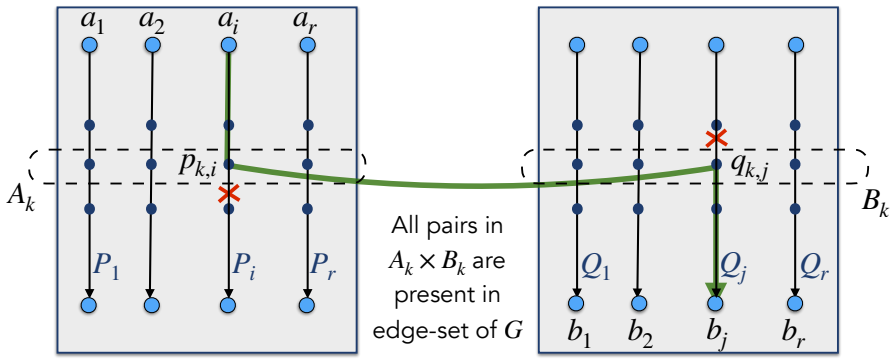


Figure 3 Dual fault-tolerant reachability preserver.

We then extend the above construction to 2-FTRO. Note, FTRO is a Boolean data structure that only decides reachability between a pair of vertices - does not report a path (if exists). If the data structure also reports a path (if exists), then using a standard information-

theoretic argument, it is possible to show that the data structure must be of a size equal to that of a FTRS. Such an argument does not work in general for FTRO. This is a general hurdle that we need to overcome to extend the lower of FTRS to FTRO. Fortunately, the hard instance we constructed for FTRS allows us to provide a reduction from (a close variant of) the *Index* problem. Then we use the randomized one-way communication complexity lower bound for the Index problem [28] to get a lower bound of $\Omega(n|S|)$ on the size (in bits) of any 2-FTRO. It is worth mentioning that our lower bound holds irrespective of the query time of 2-FTRO.

Pairwise 1-FTRO: Upper and lower bound

The above lower bound only holds for dual failures. So it remains open whether for single failure we can attain better than $O(n\sqrt{|\mathcal{P}|})$ size bound with $o(n)$ (ideally, constant) query time. In this section, we provide a construction of 1-FTRO of size $O(n + |\mathcal{P}|\sqrt{n})$ and query time $O(1)$. We show the construction by first designing a reachability oracle of size $O(n + |\mathcal{P}|\sqrt{n})$ resilient to single vertex failure. Then we extend that vertex fault-tolerant data structure to an edge fault-tolerant data structure of the same size.

One of the key ingredients of our construction is a clever data structure that could answer all-pairs reachability query under single vertex failure as long as all three vertices involved in the query belong to a cut-set. Formally, for a pair (s, t), let C be any subset of $s - t$ cut-vertices in the graph G. We build an $O(|C|)$ size data structure that, for any three vertices $x, y, z \in C$, decides the reachability between y, z upon failure of vertex x, in constant time. This data structure is inspired by the loop-nesting-forest [36]. We consider the ordering σ among the cut-vertices in C. Next, we build a predecessor forest and a successor forest with respect to this ordering σ. In the predecessor forest, the parent of any node w is the immediate predecessor u such that u, w are strongly connected even after the removal of all the predecessors of u from G. We symmetrically build the successor forest. Using constantly many Lowest Common Ancestor (LCA) and Level Ancestor (LA) queries on these forests, we can now decide the reachability between y, z upon failure of the vertex x (for any $x, y, z \in C$). By deploying any standard linear space LCA/LA data structure, we attain $O(|C|)$ space-bound and $O(1)$ query time. We refer the readers to the full version for the details.

Next, we use the above data structure to construct a pairwise fault-tolerant reachability oracle for single vertex failure. Observe, to decide the reachability between a pair upon failure, we need to check whether the failure vertex is a cut-vertex for that pair or not. At the high level, we need to keep the information about the cut-vertex-set for each pair. However, we cannot store them explicitly using small space. We first consider the cut-vertex-set for all the pairs in $\mathcal{P}$. Then we identify a *core pair-set* as follows: Take any pair in $\mathcal{P}$ with at least $\sqrt{n}$ cut-vertices, and add it into the core pair-set. Then iteratively add pairs from $\mathcal{P}$ with at least $\sqrt{n}$ new/uncovered (not part of the cut-vertex-set of any previously added pair) cut-vertices. For each pair added in the core pair-set, it *owns* the corresponding cut-vertices that were uncovered till then. Once we get the core pair-set, we are left with pairs each having at most $\sqrt{n}$ uncovered cut-vertices. Further note, the size of the core pair-set is at most $O(\sqrt{n})$. Consider the union of all the cut-vertices of the core pair-set, and build the previously mentioned data structure on it. For the remaining pairs, we store the uncovered cut-vertices associated with them using any static dictionary data structure. Note, each pair might have cut-vertices that are owned by some core pair. To keep track of them, for each pair and each core pair, we store the first and the last cut-vertex shared by them. These all constitute the final data structure. It is not hard to see that the size is $O(n + |\mathcal{P}|\sqrt{n})$. We

show that it suffices to either perform a query on the previously mentioned data structure or check whether the failure vertex is a cut-vertex in the dictionary structure during the query. (See the full version.)

Now, we extend the vertex fault-tolerant oracle to edge fault-tolerant oracle. We consider the set $\mathcal{C}$ of all the cut-edges for all the pairs in $\mathcal{P}$. Then we find the subset $\mathcal{C}_0$ that only contains the edges whose endpoints are strongly connected in G, but not after the failure of that edge. Observe, all the edges in $\mathcal{C}$ must be part of any strong-connectivity certificate, and all the edges in $\mathcal{C}_0$ must be part of any reachability preserver without any failure. Thus $|\mathcal{C}| = O(n)$ and $|\mathcal{C}_0| = O(n + \min\{|\mathcal{P}|\sqrt{n}, (n|\mathcal{P}|)^{2/3}\})$ by [1]. Next, we construct a new graph with $n + |\mathcal{C}_0|$ vertices such that checking the reachability upon an edge failure in the original graph reduces to checking the reachability (perhaps between a different pair) upon vertex failure in the new graph. As a consequence, we get an $O(n + |\mathcal{P}|\sqrt{n})$ sized 1-FTRO with constant query time.

We complement our upper bound result with a lower bound of $\Omega(n^{2/d+1}|\mathcal{P}|^{(d-1)/d})$ size (in bits) for any $d \geq 2$. We prove our lower bound by establishing a connection between the optimal size of a pairwise k-FTRO and that of a pairwise $(k-1)$-FTRS. In particular, we show that the optimal size of FTRO for any n-node graph and p-sized pair-set is at least that of $(k-1)$-FTRS for any $n/2$-node graph and p-sized pair-set. Such a connection between FTRO and FTRS is entirely new. To prove this relation, we use information-theoretic encoding-decoding argument. (See the full version.) Then the lower bound of 1-FTRO follows from the lower bound of reachability preserver without any failure by [1].

Pairwise 2-FTRS: Upper bound

We have already shown a lower bound of $\Omega(n\sqrt{|\mathcal{P}|})$ on the size of a pairwise 2-FTRS. However, so far we only known of $O(n|\mathcal{P}|)$ size 2-FTRS for any n-node graph G and pair set $\mathcal{P}$. In this work, we provide a deterministic polynomial time construction of a pairwise 2-FTRS of size $O(n^{4/3}|\mathcal{P}|^{1/3})$. For simplicity, below, we briefly describe a construction that provides a slightly weaker bound, in particular, $O(n\sqrt{|\mathcal{P}|}\log n)$. In the actual construction, we get rid of the $\log n$ factor by constructing a preserver with slack and then using the result of [9] - an idea similar to that used in pairwise 2-FTRO described earlier. Before proceeding further, let us emphasize that the main underlying idea behind our construction could be generalized to k-FTRS, for any $k \geq 1$, albeit with the help of randomization. We describe that generic construction later.

To get a sparse 2-FTRS for a node-pair set $\mathcal{P}$, we perform two-step sparsification. First, we apply our alternate construction of 2-FTRS for each of the pairs of $\mathcal{P}$. Then take a union of all of these subgraphs to get a $O(n|\mathcal{P}|)$ size intermediate subgraph H_{inter}, which is clearly a 2-FTRS for $\mathcal{P}$. Next, we further sparsify this intermediate subgraph. Similar to the technique used in oracle construction, we consider the top and bottom $\Theta(n^{2/3}|P|^{-1/3})$ portions of the outer strands ($P^1_{(s,t)}$, $P^2_{(s,t)}$) of $H_{s,t}$, for each $(s,t) \in \mathcal{P}$. Next, we construct a greedy hitting set containing $O(n^{1/3}|\mathcal{P}|^{1/3}\log n)$ vertices that intersects all these subpaths. We compute linear-sized single-source and single-destination 2-FTRS for each of these vertices in the hitting set using [4]. Let H_1 be the union of all these single-source and single-destination 2-FTRS. So, H_1 is of size $O(n^{4/3}|\mathcal{P}|^{1/3}\log n)$.

Consider any two failure edges f_1, f_2 and a pair $(s,t) \in \mathcal{P}$. If there is a surviving path in $G \setminus \{f_1, f_2\}$, we know that there is a *nice* $s-t$ path in $H_{(s,t)} \setminus \{f_1, f_2\}$. Using an argument similar to that in the oracle construction, if that nice path follows the top or bottom $\Theta(n^{2/3}|P|^{-1/3})$ portion of a outer strand, then there is also an $s-t$ path in $H_1 \setminus \{f_1, f_2\}$. So now on, it suffices to look into the case when the surviving nice path in

$H_{(s,t)} \setminus \{f_1, f_2\}$ passes though a coupling path $Q^i_{(s,t),v}$, for some vertex v lying on the bottom $\Theta(n^{2/3}|P|^{-1/3})$ length subpath of a outer strand, that starts from some vertex u lying on the top $\Theta(n^{2/3}|P|^{-1/3})$ length subpath of a outer strand.

If we could include the top and bottom $\Theta(n^{2/3}|P|^{-1/3})$ portion of the outer strands, and all the "essential" coupling path $Q^i_{(s,t),v}$'s with endpoints lying on the top and bottom $\Theta(n^{2/3}|P|^{-1/3})$ portions of the outer strands, we will be done. We indeed consider a union of the top and bottom $\Theta(n^{2/3}|P|^{-1/3})$ portion of the outer strands (for all $(s,t) \in \mathcal{P}$), and let us denote that by H_2. So, H_2 is of size $O(n^{2/3}|P|^{2/3})$. Unfortunately, we do not have a guarantee on the length of the coupling paths. Thus, if we include all the required coupling paths, we cannot argue about the sparsity of the final subgraph. (This portion of the construction differs significantly from that of our oracle construction.) We consider the subgraph obtained by taking a union of all the "essential" coupling paths with endpoints lying on the top and bottom $\Theta(n^{2/3}|P|^{-1/3})$ portions of the outer strands. (Let us denote this union by B.) Then we sparsify this subgraph further. For that purpose, we first isolate all the "high frequency" vertices (iteratively) and remove all the coupling paths containing them. Since total number of coupling paths in this subgraph is only $\Theta(n^{2/3}|P|^{2/3})$, we end up with "a few" ($O(n^{1/3}|P|^{1/3})$) high frequency vertices. Now, observe, in the remaining subgraph (denoted as H_4), degree of each vertex is "small" (at most $O(n^{1/3}|P|^{1/3})$). Next, for each of the high-frequency vertices, compute linear-sized single-source and single-destination 2-FTRS, and take a union of them to form a subgraph H_3. The union of H_1, H_2, H_3 and H_4 constitute the final subgraph.

It is not difficult to see that a surviving nice $s-t$ path in $H_{(s,t)} \setminus \{f_1, f_2\}$ either passes through one of the high frequency vertices, in which case we get an $s-t$ path in H_3; or is included in $H_2 \cup H_4$. Thus So, the union of all H_1, H_2, H_3 and H_4 will be a 2-FTRS for $\mathcal{P}$. It is worth mentioning that the correctness proof works only because of a guarantee of the existence of the nice path. Note, a nice path follows at most one coupling path as a subpath. Thus either that nice path follows the top or bottom $\Theta(n^{2/3}|P|^{-1/3})$ portion of the outer strands, or the coupling sub-path is part of B, which we further sparsify to get H_3 and H_4. Without the guarantee of a nice path, there could be many coupling sub-paths in a surviving $s-t$ path after failures. Endpoints of these coupling sub-paths may not lie on the top or bottom $\Theta(n^{2/3}|P|^{-1/3})$ portion of the outer strands. As a result, we miss them in B. As a consequence, $H_3 \cup H_4$ could not capture those coupling sub-paths.

Observe, each of the H_i's is of size at most $O(n^{4/3}|\mathcal{P}|^{1/3}\log n)$ (the $\log n$ factor is only there for H_1). So the total size is also $O(n^{4/3}|P|^{1/3}\log n)$.

Pairwise k-FTRS: Beating the $O(2^k n|\mathcal{P}|)$ bound

Currently, for any $k \geq 1$, we only know there always exists a pairwise k-FTRS of size $O(2^k n|\mathcal{P}|)$ for any pair set $\mathcal{P}$ [4]. Previously, we beat the above bound for $k = 2$. We cannot directly extend that bound for k-FTRS. The main obstacle is that for pairwise 2-FTRS, we have used a particular structure of a 2-FTRS for a single pair (i.e., the subgraph $H_{(s,t)}$). It is pretty difficult get such kind of structure for k-FTRS for any $k > 2$. However, the main underlying idea behind our pairwise 2-FTRS construction is to handle the "long" and "short" surviving paths separately. Informally, in the case of 2-FTRS, the hitting set helps us to look into only the short paths in $H_{(s,t)}$. We do a similar thing for k-FTRS using randomization.

Take a parameter ℓ, whose value we will fix later. We sample a uniformly random subset W of vertices of size $\tilde{O}(kn/\ell)$. Then we build a single-source and single-destination k-FTRS from those vertices. Let us denote the union of all these k-FTRSs as H_1. The size of H_1 is $O(k2^k n^2/\ell)$ by [4]. Suppose, for any failure edge-set F of size at most k, the length of any

shortest $s-t$ surviving path in $G \setminus F$ is at least ℓ. Consider an (arbitrarily chosen) shortest surviving $s-t$ path in $G \setminus F$ (which is of length at least ℓ). Then it is not hard to argue that W intersects that path with high probability. Thus we get an $s-t$ path in $H_1 \setminus F$. This part is similar to what we get in 2-FTRS using greedy hitting set.

So now on, we only need to handle the case when a shortest surviving path is of length at least ℓ. To do this, roughly speaking, we enumerate over all possible failure-set of size at most k and add a shortest surviving path in the subgraph. Trivially, we get a bound of only $O(n^k)$ on the number of possible failure edge-sets of size at most k. However, observe, we only need to handle the case when a surviving shortest path is of length at most ℓ. So before the failure also the length of a shortest path was at most ℓ. The initial shortest path (before any failure) will get destroyed only if one or more failure edges lie on that shortest path. This observation reduces the number of possible failure-set to $O(\ell^k)$. So, for each pair, we add $O(\ell^k)$ paths, each of length at most ℓ. For all the pairs, the total number of added edges is at most $O(\ell^{k+1}|\mathcal{P}|)$. We can further improve this bound to $O(\ell^k|\mathcal{P}|)$ by enumerating the failure-set of size up to $k-1$ and then adding two maximally edge-disjoint shortest paths. (See the full version for the details.) At the end, we get a subgraph of size $O(k2^k n^2/\ell + \ell^k|\mathcal{P}|)$. By optimizing the parameter ℓ, we get a size bound of $\widetilde{O}(k\ 2^k\ n^{\frac{2k}{k+1}}\ |\mathcal{P}|^{\frac{1}{k+1}})$. Note, this size bound beats the $O(2^k n|\mathcal{P}|)$ bound whenever $|\mathcal{P}| = \omega(kn^{\frac{k-1}{k}}\log n)$.

4 Conclusion

In this paper, we study compact oracle and sparse preservers for the problem of pairwise reachability under failures. For dual failures, we provide a construction of $O(n\sqrt{|\mathcal{P}|})$ sized reachability oracle that has constant query time, along with a matching (up to the word size) lower bound. It (almost) settles down the question on the optimal sized dual fault-tolerant pairwise reachability oracle. For single failure, we achieve a bound of $O(n + \min\{|\mathcal{P}|\sqrt{n}, n\sqrt{|\mathcal{P}|}\})$ on the size of oracle with constant query time. We complement our upper bound with a $\Omega(n^{2/3}|\mathcal{P}|^{1/2})$ size (in bits) lower bound, refuting the possibility of getting a linear-sized oracle for a single failure. We would like to pose the problem of closing the current gap between the upper and lower bound for a single fault-tolerant oracle as an open problem.

In the case of reachability preserver, we show an upper bound of $O(n^{4/3}|\mathcal{P}|^{1/3})$ edges for any n-node graph and a vertex-pair set $\mathcal{P}$, for dual failures setting. This improves the naive bound of $O(n|\mathcal{P}|)$ preserver (obtained from the result known for single-source setting). In addition, we obtain a lower bound of $\Omega(n\sqrt{|\mathcal{P}|})$ on the size of our reachability preserver under dual failures. Prior to our work it was known that for single failure, we can have preserver with $O(n + \min\{|\mathcal{P}|\sqrt{n},\ n\sqrt{|\mathcal{P}|}\})$ edges. Thus our lower bound provides a striking difference between the single and dual fault-tolerant setting as it gives a separation of $n^{1/4}$ factor for $|\mathcal{P}| = \Theta(\sqrt{n})$. One immediate open question after our work is whether a sparser pairwise reachability preserver exists for dual failures.

Next, we go beyond the dual failures and consider the question of getting sparse pairwise reachability preserver resilient up to k failures for any $k \geq 1$. Can there always exist a pairwise preserver of size $o(2^k n|\mathcal{P}|)$ for any k failures? We answer this question affirmatively by providing a randomized polynomial-time construction of a $o(2^k n|\mathcal{P}|)$ size preserver. We leave the question of finding an optimal k fault-tolerant pairwise preserver as an important open problem.

References

1. Amir Abboud and Greg Bodwin. Reachability preservers: New extremal bounds and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1865–1883, 2018.
2. Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in undirected graphs: Breaking the o(m) barrier. *SIAM J. Comput.*, 48(4):1335–1363, 2019.
3. Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single source fault tolerant shortest path. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1901–1915, 2018.
4. Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 509–518, 2016.
5. Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013.
6. Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Proceedings*, pages 167–178, 2015.
7. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, pages 18:1–18:14, 2016.
8. Greg Bodwin. Linear size distance preservers. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 600–615, 2017.
9. Greg Bodwin. A note on distance-preserving graph sparsification. *arXiv preprint arXiv:2001.07741*, 2020.
10. Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 73:1–73:14, 2017.
11. Diptarka Chakraborty and Keerti Choudhary. New extremal bounds for reachability and strong-connectivity preservers under failures. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 25:1–25:20, 2020.
12. Diptarka Chakraborty and Debarati Das. Sparse Weight Tolerant Subgraph for Single Source Shortest Path. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
13. T-H Hubert Chan, Michael Dinitz, and Anupam Gupta. Spanners with slack. In *European Symposium on Algorithms*, pages 196–207. Springer, 2006.
14. Shiri Chechik. Fault-tolerant compact routing schemes for general graphs. *Inf. Comput.*, 222:36–44, 2013.
15. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 435–444, 2009.
16. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. *f*-sensitivity distance oracles and routing schemes. In *18th Annual European Symposium on Algorithms - ESA (1)*, pages 84–96, 2010.

17 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 130:1–130:13, 2016.

18 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM J. Discrete Math.*, 20(2):463–501, 2006.

19 Artur Czumaj and Hairong Zhao. Fault-tolerant geometric spanners. *Discrete & Computational Geometry*, 32(2):207–230, 2004.

20 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

21 Michael Dinitz. Compact routing with slack. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 81–88, 2007.

22 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011*, pages 169–178, 2011.

23 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 506–515, 2009.

24 Ran Duan and Seth Pettie. Connectivity oracles for failure prone graphs. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 465–474, 2010.

25 Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 490–509, 2017.

26 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 127:1–127:15, 2017.

27 Goran Konjevod, Andrea Werneck Richa, Donglin Xia, and Hai Yu. Compact routing with slack in low doubling dimension. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 71–80, 2007.

28 Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Comput. Complex.*, 8(1):21–49, 1999.

29 Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.

30 Tamás Lukovszki. New results of fault tolerant geometric spanners. In *Algorithms and Data Structures, 6th International Workshop, WADS '99, Proceedings*, pages 193–204, 1999.

31 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

32 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015*, pages 481–490, 2015.

33 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Proceedings*, pages 779–790, 2013.

34 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1073–1092, 2014.

35 Mihai Patrascu and Mikkel Thorup. Planning for fast connectivity updates. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 263–271, 2007.

36 Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6:171–185, 1976. doi:10.1007/BF00268499.

37 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435, 2019.

Separations Between Combinatorial Measures for Transitive Functions

Sourav Chakraborty
Indian Statistical Institute, Kolkata, India

Chandrima Kayal
Indian Statistical Institute, Kolkata, India

Manaswi Paraashar
Aarhus University, Denmark

Abstract

The role of symmetry in Boolean functions $f : \{0,1\}^n \to \{0,1\}$ has been extensively studied in complexity theory. For example, symmetric functions, that is, functions that are invariant under the action of S_n, is an important class of functions in the study of Boolean functions. A function $f : \{0,1\}^n \to \{0,1\}$ is called transitive (or weakly-symmetric) if there exists a transitive group G of S_n such that f is invariant under the action of G. In other words, the value of the function remains unchanged even after the input bits of f are moved around according to some permutation $\sigma \in \mathsf{G}$. Understanding various complexity measures of transitive functions has been a rich area of research for the past few decades.

This work studies transitive functions in light of several combinatorial measures. The question that we try to address in this paper is what are the maximum separations between various pairs of combinatorial measures for transitive functions. Such study for general Boolean functions has been going on for many years. Aaronson et al. (STOC, 2021) have nicely compiled the current best-known results for general Boolean functions. But before this paper, no such systematic study had been done on the case of transitive functions.

Separations between a pair of combinatorial measures are shown by constructing interesting functions that demonstrate the separation. Over the past three decades, various interesting classes of functions have been designed for this purpose. In this context, one of the celebrated classes of functions is the "pointer functions". Ambainis et al. (JACM, 2017) constructed several functions, which are modifications of the pointer function in Göös et al. (SICOMP, 2018 / FOCS, 2015), to demonstrate the separation between various pairs of measures. In the last few years, pointer functions have been used to show separation between various other pairs of measures (Eg: Mukhopadhyay et al. (FSTTCS, 2015), Ben-David et al. (ITCS, 2017), Göös et al. (ToCT, 2018 / ICALP, 2017)).

However, the pointer functions themselves are not transitive. Based on the various kinds of pointer functions, we construct new transitive functions, which we use to demonstrate similar separations between various pairs of combinatorial measures as demonstrated by the original pointer functions. Our construction of transitive functions depends crucially on the construction of particular classes of transitive groups whose actions, though involved, help to preserve certain structural features of the input strings. The transitive groups we construct may be of independent interest in other areas of mathematics and theoretical computer science.

We summarize the current knowledge of relations between various combinatorial measures of transitive functions in a table similar to the table compiled by Aaronson et al. (STOC, 2021) for general functions.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Transitive functions, Combinatorial complexity of Boolean functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.36

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2103.12355

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 36; pp. 36:1–36:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ what is the relationship between its various combinatorial measures, like deterministic query complexity ($\mathsf{D}(f)$), bounded-error randomized and quantum query complexity ($\mathsf{R}(f)$ and $\mathsf{Q}(f)$ respectively), zero -randomized query complexity ($\mathsf{R}_0(f)$), exact quantum query complexity ($\mathsf{Q}_\mathsf{E}(f)$), sensitivity ($\mathsf{s}(f)$), block sensitivity ($\mathsf{bs}(f)$), certificate complexity ($\mathsf{C}(f)$), randomized certificate complexity ($\widetilde{\mathsf{RC}}(f)$), unambiguous certificate complexity ($\mathsf{UC}(f)$), degree ($\deg(f)$), approximate degree ($\widetilde{\deg}(f)$) and spectral sensitivity ($\lambda(f)$)[1]? For over three decades, understanding the relationships between these measures has been an active area of research in computational complexity theory. These combinatorial measures have applications in many other areas of theoretical computer science, and thus the above question takes a central position.

In the last couple of years, some of the more celebrated conjectures have been answered - like the quadratic relation between sensitivity and degree of Boolean functions [21]. We refer the reader to the survey [12] for an introduction to this area.

Understanding the relationship between various combinatorial measures involves two parts:

- Relationships – proving that one measure is upper bounded by a function of another measure. For example, for any Boolean function f, $\deg(f) \leq \mathsf{s}(f)^2$ and $\mathsf{D}(f) \leq \mathsf{R}(f)^2$.
- Separations – constructing functions that demonstrates separation between two measures. For example, there exists a Boolean function f with $\deg(f) \geq \mathsf{s}(f)^2$. Also there exists another Boolean function g with $\mathsf{D}(g) \geq \mathsf{R}(g)^2$.

Obtaining tight bounds between pairs of combinatorial measures - that is, when the relationship and the separation results match - is the holy grail of this area of research. The current best-known results for different pairs of functions have been nicely compiled in [2].

For special classes of Boolean functions the relationships and the separations might be different than that of general Boolean functions. For example, while it is known that there exists f such that $\mathsf{bs}(f) = \Theta(\mathsf{s}(f)^2)$ [26], for a symmetric function a more tighter result is known, $\mathsf{bs}(f) = \Theta(\mathsf{s}(f))$. The best-known relationship of $\mathsf{bs}(f)$ for a general Boolean functions is $\mathsf{s}(f)^4$ [21]. How the various measures behave for different classes of functions has been studied since the dawn of this area of research.

Transitive Functions. One of the most well-studied classes of Boolean functions is that of the transitive functions. A function $f : \{0,1\}^n \to \{0,1\}$ is transitive if there is a transitive group $G \leq \mathsf{S}_n$ such that the function value remains unchanged even after the indices of the input is acted upon by a permutation from G. Note that, when $G = \mathsf{S}_n$ then the function is symmetric. Transitive functions (also called "weakly symmetric" functions) has been studied extensively in the context of various complexity measure. This is because symmetry is a natural measure of the complexity of a Boolean function. It is expected that functions with more symmetry must have less variation among the various combinatorial measures. A recent work [7] has studied the functions under various types of symmetry in terms of quantum speedup. So, studying functions in terms of symmetry is important in various aspects.

For example, for symmetric functions, where the transitive group is S_n, most of the combinatorial measures become the same up to a constant [2]. Another example of transitive functions is the graph properties. The input is the adjacency matrix, and the transitive group

[1] For formal definitions of the various measures used in this paper please refer to the full version of this paper [15].

[2] There are still open problems on the tightness of the constants.

is the graph isomorphism group acting on the bits of the adjacency matrix. [31, 29, 23, 17] tried to obtain tight bounds on the relationship between sensitivity and block sensitivity for graph properties. They also tried to answer how low can sensitivity and block sensitivity go for graph properties?

In papers like [30, 14, 28, 16] it has been studied how low can the combinatorial measures go for transitive functions. The behavior of transitive functions can be very different from general Boolean functions. For example, while it is known that there are Boolean functions for which the sensitivity is as low as $\Theta(\log n)$ where n is the number of effective variables[3], it is known (from [28] and [21]) that if f is a transitive function on n effective variables then its sensitivity $s(f)$ is at least $\Omega(n^{1/12})$[4]. Similar behavior can be observed in other measures too. For example, it is easy to see that for a transitive function, the certificate complexity is $\Omega(\sqrt{n})$, while the certificate complexity for a general Boolean function can be as low as $O(\log n)$. Please see the full version of this paper [15] for a more detailed study.

A natural related question is:

What are tight relationships between various pairs of combinatorial measures for transitive functions?

By definition, the known relationship results for general functions hold for transitive functions. But tighter relationships may be obtained for transitive functions. On the other hand, the existing separations don't extend easily since the example used to demonstrate separation between certain pairs of measures may not be transitive. Some of the most celebrated examples are not transitive. For example some of the celebrated function construction like the pointer function in [4], used for demonstrating tight separations between various pairs like $\mathsf{D}(f)$ and $\mathsf{R}_0(f)$, are not transitive. Similarly, the functions constructed using the cheat sheet techniques [1] used for separation between quantum query complexity and degree, or approximate degree, are not transitive. Constructing transitive functions which demonstrate tight separations between various pairs of combinatorial measures is very challenging.

Our Results. We try to answer the above question for various pairs of measures. More precisely, our main contribution is to construct transitive functions that have similar complexity measures as the *pointer functions*. Hence for those pairs of measures where pointer functions can demonstrate separation for general functions, we prove that transitive functions can also demonstrate similar separation.

Our results and the current known relations between various pairs of complexity measures of transitive functions are compiled in Table 1. This table is along the lines of the table in [2] where the best-known relations between various complexity measures of general Boolean functions were presented.

Deterministic query complexity and zero-error randomized query complexity are two of the most basic measures and yet the tight relation between these measures was not known until recently. In [27] they showed that for the "balanced NAND-tree" function, $\widetilde{\wedge}$-tree, $\mathsf{D}(\widetilde{\wedge}\text{-tree}) \geq \mathsf{R}_0(\widetilde{\wedge}\text{-tree})^{1.33}$. Although the function $\widetilde{\wedge}$-tree is transitive, the best-known relationship was quadratic, that is for all Boolean function f, $\mathsf{D}(f) = O(\mathsf{R}_0(f)^2)$. In [4] a new function, A1, was constructed for which deterministic query complexity and zero-error randomized query complexity can have a quadratic separation between them, and this matched the known relationship results. The function in [4] was a variant of the pointer

[3] A variable is effective if the function is dependent on it.

[4] It is conjectured that the sensitivity of a transitive function is $\Omega(n^{1/3})$.

functions - a class of functions introduced by [20] that has found extensive usage in showing separations between various complexity measures of Boolean functions. The function, A1, also gave (the current best-known) separations between deterministic query complexity and other measures like quantum query complexity and degree. But the function A1 is not transitive. Using the A1 function we construct a transitive function that demonstrates a similar gap between deterministic query complexity and zero-error randomized query complexity, quantum query complexity, and degree.

▶ **Theorem 1.** *There exists a transitive function F_1 such that*

$$\mathsf{D}(F_1) = \widetilde{\Omega}(\mathsf{Q}(F_1)^4), \qquad \mathsf{D}(F_1) = \widetilde{\Omega}(\mathsf{R_0}(F_1)^2), \qquad \mathsf{D}(F_1) = \widetilde{\Omega}(\deg(F_1)^2).$$

The proof of Theorem 1 is presented in Section 4. In [4, 9] various variants of the pointer function have been used to show separation between other pairs of measures like $\mathsf{R_0}$ with R, $\mathsf{Q_E}$, deg, and Q, R with $\widetilde{\deg}$, deg, $\mathsf{Q_E}$ and sensitivity. Inspired by these functions, we construct transitive versions that demonstrate similar separation for transitive functions as general functions.

▶ **Theorem 2.** *There exists a transitive function F_2 such that*

$$\mathsf{R_0}(F_2) = \widetilde{\Omega}(\mathsf{R}(F_2)^2), \qquad \mathsf{R_0}(F_2) = \widetilde{\Omega}(\mathsf{Q_E}(F_2)^2), \qquad \mathsf{R_0}(F_2) = \widetilde{\Omega}(\deg(F_2)^2).$$

▶ **Theorem 3.** *There exists a transitive function F_3 such that*

$$\mathsf{R}(F_3) = \widetilde{\Omega}(\widetilde{\deg}(F_3)^4), \qquad \mathsf{R}(F_3) = \widetilde{\Omega}(\deg(F_3)^2).$$

The construction of these functions, though more complicated and involved, are similar in flavor to that of F_1. Due to lack of space, we skip the proofs of Theorem 2 and 3 in this conference version of this paper. The proofs are available in the full version of this paper [15]. Using standard techniques, we can also obtain the following theorems as corollaries to Theorem 3.

▶ **Theorem 4.** *There exists a transitive function F_4 such that $\mathsf{R_0}(F_4) = \widetilde{\Omega}(\mathsf{Q}(F_4)^3)$.*

▶ **Theorem 5.** *There exists a transitive function F_5 such that $\mathsf{R}(F_5) = \widetilde{\Omega}(\mathsf{Q_E}(F_5)^{1.5})$.*

▶ **Theorem 6.** *There exists transitive functions F_6 such that $\mathsf{R}(F_6) = \widetilde{\Omega}(\mathsf{s}(F_6)^3)$.*

Our proof techniques also help us make transitive versions of other functions like that used in[1] to demonstrate the gap between Q and certificate complexity.

▶ **Theorem 7.** *There exists a transitive function F_7 such that $\mathsf{Q}(F_7) = \widetilde{\Omega}(\mathsf{C}(F_7)^2)$.*

All our results are compiled (and marked in green) in Table 1.

One would naturally ask what stops us from constructing transitive functions analogous to the other functions, like cheat sheet-based functions. In fact, one could ask why to use ad-hoc techniques to construct transitive functions (as we have done in most of our proofs) and instead why not design a unifying technique for converting any function into a transitive function that would display similar properties in terms of combinatorial measures [5]. If one could do so, all the separation results for general functions (in terms of separation between pairs of measures) would translate to separation for transitive functions. In Section 5 we have discussed why such a task is challenging. We argue the challenges of making transitive versions of the cheat-sheet functions.

[5] In [7] they have demonstrated a technique that can be used for constructing a transitive partial function that demonstrates gaps (between certain combinatorial measures) similar to a given partial function that need not be transitive. But their construction need not construct a total function even when the given function is total.

Table 1 best-known separations between combinatorial measures for transitive functions.

	D	R_0	R	C	RC	bs	s	λ	Q_E	deg	Q	$\widetilde{deg}$
D		2 ; 2 T:1	2 ; 3 T:1	2 ; 2 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	3 ; 6 T:6	4 ; 6 T:3	2 ; 3 T:2	2 ; 3 T:1	4 ; 4 T:1	4 ; 4 T:3
R_0	1,1 $\oplus$		2 ; 2 T:2	2 ; 2 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	3 ; 6 T:6	4 ; 6 T:3	2 ; 3 T:2	2 ; 3 T:2	3 ; 4 T:4	4 ; 4 T:3
R	1 ; 1 $\oplus$	1 ; 1 $\oplus$		2 ; 2 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	3 ; 6 T:6	4 ; 6 T:3	1.5 ; 3 T:5	2 ; 3 T:3	2 ; 4 $\wedge$	4 ; 4 T:3
C	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 2 $\oplus$		2 ; 2 [18]	2 ; 2 [18]	2 ; 5 [26]	2 ; 6 $\wedge$	1.15 ; 3 [3]	1.63 ; 3 [25]	2 ; 4 $\wedge$	2 ; 4 $\wedge$
RC	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		1.5 ; 2 [18]	2 ; 4 [26]	2 ; 4 $\wedge$	1.15 ; 2 [3]	1.63 ; 2 [25]	2 ; 2 $\wedge$	2 ; 2 $\wedge$
bs	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		2 ; 4 [26]	2 ; 4 $\wedge$	1.15 ; 2 [3]	1.63 ; 2 [25]	2, 2 $\wedge$	2 ; 2 $\wedge$
s	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		2 ; 2 $\wedge$	1.15 ; 2 [3]	1.63 ; 2 [25]	2, 2 $\wedge$	2 ; 2 $\wedge$
λ	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$
Q_E	1 ; 1 $\oplus$	1.33 ; 2 $\widetilde{\wedge}$-tree	1.33 ; 3 $\widetilde{\wedge}$-tree	2 ; 2 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	2 ; 3 $\wedge\circ\vee$	2 ; 6 T:7	2 ; 6 T:7		1 ; 3 $\oplus$	2 ; 4 $\wedge$	1 ; 4 $\oplus$
deg	1 ; 1 $\oplus$	1.33 ; 2 $\widetilde{\wedge}$-tree	1.33 ; 2 $\widetilde{\wedge}$-tree	2 ; 2 $\wedge\circ\vee$	2 ; 2 $\wedge\circ\vee$	2 ; 2 $\wedge\circ\vee$	2 ; 2 $\wedge\circ\vee$	2 ; 2 $\wedge$	1 ; 1 $\oplus$		2 ; 2 $\wedge$	2 ; 2 $\wedge$
Q	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	2 ; 2 T:7	2 ; 3 T:7	2 ; 3 T:7	2 ; 6 T:7	2 ; 6 T:7	1, 1 $\oplus$	1 ; 3 $\oplus$		1 ; 4 $\oplus$
$\widetilde{deg}$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	

(1) Entry $a; b$ in row A and column B represents: for any *transitive* function f, $A(f) = O(B(f))^{b+o(1)}$, and there exists a *transitive* function g such that $A(g) = \Omega(B(g))^a$.
(2) Cells with a green background are those for which we constructed new *transitive functions* to demonstrate separations that match the best-known separations for general functions. The previously known functions that gave the strongest separations were not transitive. The second row (in each cell) gives the reference to the Theorems where the separation result is proved. Although for these green cells, the bounds match that of the general functions, for some cells (with a light green color), there is a gap between the known relationships and best-known separations.
(3) In the cells with a white background, the best-known examples for the corresponding separation were already *transitive functions*. For these cells, the second row either contains the function that demonstrates the separation or a reference to the paper where the separation was proved. So for these cells, the separations for transitive functions matched the current best-known separations for general functions. Note that for some of these cells, the bounds are not tight for general functions.
(4) Cells with a yellow background are those where the best-known separations for transitive functions do not match the best-known separations for general functions.

2 Notations and Background

2.1 Notations and basic definitions

We use $[n]$ to denote the set $\{1, \ldots, n\}$. $\{0,1\}^n$ denotes the set of all n-bit binary strings. For any $X \in \{0,1\}^n$ the Hamming Weight of X (denoted $|X|$) will refer to the number of 1 in X. 0^n and 1^n denotes all 0's string of n-bit and all 1's string of n-bit, respectively.

We denote by S_n the set of all permutations on $[n]$. Given an element $\sigma \in \mathsf{S}_n$ and a n-bit string $x_1, \ldots, x_n \in \{0,1\}^n$ we denote by $\sigma[x_1, \ldots, x_n]$ the string obtained by permuting the indices according to σ. That is $\sigma[x_1, \ldots, x_n] = x_{\sigma(1)}, \ldots, x_{\sigma(n)}$. This is also called the action of σ on the $x_1, \ldots, x_n$.

Following are a couple of interesting elements of S_n that will be used in this paper.

▶ **Definition 8.** *For any $n = 2k$ the* flip *swaps $(2i-1)$ and $2i$ for all $1 \le i \le k$. The permutation* $\mathsf{Swap}_{\frac{1}{2}}$ *swaps i with $(k+i)$, for all $1 \le i \le k$. That is,*

$$\mathsf{flip} = (1,2)(3,4)\ldots(n-1,n) \qquad \& \qquad \mathsf{Swap}_{\frac{1}{2}}[x_1, \ldots, x_{2k}] = x_{k+1}, \ldots, x_{2k}, x_1 \ldots, x_k.$$

Every integer $\ell \in [n]$ has the canonical $\log n$ bit string representation. However the number of 1's and 0's in such a representation is not same for all $\ell \in [n]$. The following representation of $\ell \in [n]$ ensures that for all $\ell \in [n]$ the encoding has same Hamming weight.

▶ **Definition 9** (Balanced binary representation)**.** *For any $\ell \in [n]$, let $\ell_1, \ldots, \ell_{\log n}$ be the binary representation of the number ℓ where $\ell_i \in \{0,1\}$ for all i. Replacing* 1 *by* 10 *and* 0 *by* 01 *in the binary representation of ℓ, we get a $2\log n$-bit unique representation, which we call* Balanced binary representation *of ℓ and denote as $bb(\ell)$.*

In this paper all the functions considered are of form $F : \{0,1\}^n \to \{0,1\}^k$. By Boolean functions we would mean a Boolean valued function that is of the form $f : \{0,1\}^n \to \{0,1\}$.

An input to a function $F : \{0,1\}^n \to \{0,1\}^k$ is a n-bit string but also the input can be thought of as different objects. For example, if the $n = NM$ then the input may be thought of as a $(N \times M)$-matrix with Boolean values. It may also be thought of as a $(M \times N)$-matrix.

If $\Sigma = \{0,1\}^k$ then $\Sigma^{(n\times m)}$ denotes an $(n \times m)$-matrix with an element of Σ (that is, a k-bit string) stored in each cell of the matrix. Note that $\Sigma^{(n\times m)}$ is actually $\{0,1\}^{mnk}$. Thus, a function $F : \Sigma^{(n\times m)} \to \{0,1\}$ is actually a Boolean function from a $\{0,1\}^{nmk}$ to $\{0,1\}$, where we think of the input as an $(n \times m)$-matrix over the alphabet Σ.

One particular nomenclature that we use in this paper is that of 1-cell certificate.

▶ **Definition 10** (1-cell certificate)**.** *Given a function $f : \Sigma^{(n\times m)} \to \{0,1\}$ (where $\Sigma = \{0,1\}^k$) the 1-cell certificate is a partial assignments to the cells which forces the value of the function to* 1*. So a 1-cell certificate is of the form $(\Sigma \cup \{*\})^{(n\times m)}$. Note the here we assume that the contents in any cell is either empty or a proper element of Σ (and not a partial k-bit string).*

Another notation that is often used is the following:

▶ **Notation 11.** *If $A \leq \mathsf{S}_n$ and $B \leq \mathsf{S}_m$ are groups on $[n]$ and $[m]$ then the group $A \times B$ act on the cells on the matrix. Thus for any $(\sigma, \sigma') \in A \times B$ and a $M \in \Sigma^{(n\times m)}$ by $(\sigma, \sigma')[M]$ we would mean the permutation on the cell of M according to (σ, σ') and move the contains in the cells accordingly. Note that the relative position of bits within the contents in each cell is not touched.*

Next, we define the composition of two Boolean functions.

▶ **Definition 12** (Composition of functions)**.** *Let $f : \{0,1\}^{nk} \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}^k$ be two functions. The* composition of f and g*, denoted by $f \circ g : \{0,1\}^{nm} \to \{0,1\}$, is defined to be a function on nm bits such that on input $x = (x_1, \ldots, x_n) \in \{0,1\}^{nm}$, where each $x_i \in \{0,1\}^m$, $f \circ g(x_1, \ldots, x_n) = f(g(x_1), \ldots, g(x_n))$. We will refer f as* outer function *and g as* inner function*.*

2.2 Transitive Groups and Transitive Functions

The central objects in this paper are transitive Boolean function. We first define transitive groups.

▶ **Definition 13.** *A group $G \leq \mathsf{S}_n$ is transitive if for all $i, j \in [n]$ there exists a $\sigma \in G$ such that $\sigma(i) = j$.*

▶ **Definition 14.** *For $f : A^n \to \{0,1\}$ and $G \leq \mathsf{S}_n$ we say f is invariant under the action of G, if for all $\alpha_1, \ldots, \alpha_n \in A$.*

$$f(\alpha_1, \ldots, \alpha_n) = f(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(n)}).$$

▶ **Observation 15.** *If $A \leq \mathsf{S}_n$ and $B \leq \mathsf{S}_m$ are transitive groups groups on $[n]$ and $[m]$ then the group $A \times B$ is a transitive group acting on the cells on the matrix.*

There are many interesting transitive groups. The symmetric group is indeed transitive. The graph isomorphism group (that acts on the adjacency matrix - minus the diagonal - of a graph by changing the ordering on the vertices) is transitive. The cyclic permutation over all the points in the set is a transitive group. The following is another non-trivial transitive group on $[k]$ that we will use extensively in this paper.

▶ **Definition 16.** *For any k that is a power of 2, the Binary-tree-transitive group Bt_k is a subgroup of S_k. To describe its generating set we think of group Bt_k acting on the elements $\{1, \ldots, k\}$ and the elements are placed in the leaves of a balanced binary tree of depth $\log k$ - one element in each leaf. Each internal node (including the root) corresponds to an element in the generating set of Bt_k. The element corresponding to an internal node in the binary tree swaps the left and right sub-tree of the node. The permutation element corresponding to the root node is called the Root-swap as it swaps the left and right sub-tree to the root of the binary tree.*

We now state two claims whose proofs we skip in this version of the paper but are available in the full version of the paper [15].

▷ Claim 17. The group Bt_k is a transitive group.

The following claim describes how the group Bt_k acts on various encoding of integers. Recall the balance-binary representation (Definition 9).

▷ Claim 18. For all $\widehat{\gamma} \in \mathsf{Bt}_{2\log n}$ there is a $\gamma \in \mathsf{S}_n$ such that for all $i, j \in [n]$, $\widehat{\gamma}[bb(i)] = bb(j)$ iff $\gamma(i) = j$.

Now let us consider another encoding that we will using for the set of rows and columns of a matrix.

▶ **Definition 19.** *Given a set R of n rows $r_1, \ldots, r_n$ and a set C of n columns $c_1, \ldots, c_n$ we define the balanced-pointer-encoding function $\mathcal{E} : (R \times \{0\}) \cup (\{0\} \times C) \to \{0,1\}^{4\log n}$, as follows:*

$$\mathcal{E}(r_i, 0) = bb(i) \cdot 0^{2\log n}, \text{ and, } \mathcal{E}(0, c_j) = 0^{2\log n} \cdot bb(j).$$

The following is a claim that is easy to verify.

▷ Claim 20. Let R be a set of n rows $r_1, \ldots, r_n$ and C be a set of n columns $c_1, \ldots, c_n$ and consider the *balanced-pointer-encoding function* $\mathcal{E} : (R \times \{0\}) \cup (\{0\} \times C) \to \{0,1\}^{4\log n}$. For any elementary permutation $\widehat{\sigma}$ in $\mathsf{Bt}_{4\log n}$ (other than the Root-swap) there is a $\sigma \in \mathsf{S}_n$ such that for any $(r_i, c_j) \in (R \times \{0\}) \cup (\{0\} \times C)$

$$\widehat{\sigma}[\mathcal{E}(r_i, c_j)] = \mathcal{E}(r_{\sigma(i)}, c_{\sigma(j)}),$$

where we assume $r_0 = c_0 = 0$ and any permutation of in S_n sends 0 to 0.

If $\widehat{\sigma}$ is the root-swap then for any $(r_i, c_j) \in (R \times \{0\}) \cup (\{0\} \times C)$

$$\widehat{\sigma}[\mathcal{E}(r_i, c_j)] = \mathsf{Swap}_{\frac{1}{2}}(\mathcal{E}(r_i, c_j)) = \mathcal{E}(c_j, r_i).$$

2.3 Pointer function

For the sake of completeness first we will describe the "pointer function" introduced in [4] that achieves separation between several complexity complexity measures like *Deterministic query complexity*, *Randomized query complexity*, *Quantum query complexity* etc. This function was originally motivated from a function in [20]. There are three three variants of the pointer function that have some special kind of non-Boolean domain, which we call *Pointer matrix*. Our function is a special "encoding" of that non-Boolean domain such that the resulting function becomes transitive and achieves the separation between complexity measures that matches the known separation between the general functions. Here we will define only the first variant of the pointer function.

▶ **Definition 21** (Pointer matrix over Σ)**.** *For $m, n \in \mathbb{N}$, let M be a $(m \times n)$ matrix with m rows and n columns. We refer to each of the $m \times n$ entries of M as* cells*. Each cell of the matrix is from a alphabet set Σ where $\Sigma = \{0,1\} \times \widetilde{P} \times \widetilde{P} \times \widetilde{P}$ and $\widetilde{P} = \{(i,j) | i \in [m], j \in [n]\} \cup \{\perp\}$. We call $\widetilde{P}$ as set of pointers where, pointers of the form $\{(i,j) | i \in [m], j \in [n]\}$ pointing to the cell (i,j) and $\perp$ is the null pointer. Hence,each entry $x_{(i,j)}$ of the matrix M is a 4-tuple from Σ. The elements of the 4-tuple we refer as* value*,* left pointer*,* right pointer *and* back pointer *respectively and denote by Value$(x_{(i,j)})$, LPointer$(x_{(i,j)})$, RPointer$(x_{(i,j)})$ and BPointer$(x_{(i,j)})$ respectively where Value $\in \{0,1\}$, LPointer, RPointer, BPointer $\in \widetilde{P}$. We call this type of matrix as* pointer matrix *and denote by $\Sigma^{n \times n}$.*

A special case of the pointer-matrix, which we call Type_1 pointer matrix over Σ*, is when for each cell of M, BPointer $\in \{[n] \cup \perp\}$ that is backpointers are pointing to the columns of the matrix.*

Also, in general when, BPointer $\in \{(i,j) | i \in [m], j \in [n]\} \cup \{\perp\}$, we call it a Type_2 pointer matrix over Σ.

Now we will define some additional properties of the domain that we need to define the pointer function.

▶ **Definition 22** (Pointer matrix with marked column)**.** *Let M be an $m \times n$* pointer-matrix *over Σ. A column $j \in [n]$ of M is defined to be a* marked column *if there exists exactly one cell (i,j), $i \in [m]$, in that column with entry $x_{(i,j)}$ such that $x_{(i,j)} \neq (1, \perp, \perp, \perp)$ and every other cell in that column is of the form $(1, \perp, \perp, \perp)$. The cell (i,j) is defined to be the* special element *of the* marked column *j.*

Let n be a power of 2. Let T be a rooted, directed and balanced binary tree with n-leaves and $(n-1)$ internal vertices. We will use the following notations that will be used in defining some functions formally.

▶ **Notation 23.** *Let n be a power of 2. Let T be a rooted, directed and balanced binary tree with n-leaves and $(n-1)$ internal vertices. Labels the edges of T as follows: the outgoing edges from each node are labeled by either $left$ or $right$. The leaves of the tree are labeled by the elements of $[n]$ from left to right, with each label used exactly once. For each leaf $j \in [n]$ of the tree, the path from the root to the leaf j defines a sequence of $left$ and $right$ of length $O(\log n)$, which we denote by $T(j)$.*

When n is not a power of 2, choose the largest $k \in \mathbb{N}$ such that $2^k \leq n$, consider a complete balanced tree with 2^k leaves and add a pair of child node to to each $n - 2^k$ leaves starting from left. Define $T(j)$ as before.

Now we are ready to describe the *Variant 1* of the pointer function.

▶ **Definition 24** (Variant 1 [4]). *Let* $\Sigma^{m\times n}$ *be a* Type_1 pointer matrix *where BPointer is a pointer of the form* $\{j|j \in [n]\}$ *that points to other column and LPointer, RPointer are as usual points to other cell. Define* $\mathsf{A1}_{(m,n)} : \Sigma^{m\times n} \to \{0,1\}$ *on a* Type_1 pointer matrix *such that for all* $x = (x_{i,j}) \in \Sigma^{m\times n}$*, the function* $\mathsf{A1}_{(m,n)}(x_{i,j})$ *evaluates to* 1 *if and only if it has a* 1- cell certificate *of the following form:*

1. *there exists exactly one* marked column $j^\star$ *in* M*,*
2. *There is a special cell, say* $(i^\star, j^\star)$ *which we call the special element in the the* marked column $j^\star$ *and there is a balanced binary tree* T *rooted at the special cell,*
3. *for each non-marked column* $j \in [n] \setminus \{j^\star\}$ *there exist a* cell l_j *such that* $Value(l_j) = 0$ *and* $BPointer(l_j) = j^\star$ *where* l_j *is the end of the path that starts at the* special element *and follows the pointers LPointer and RPointer as specified by the sequence* $T(j)$*.* l_j *exists for all* $j \in [n] \setminus \{j^\star\}$ *i.e. no pointer on the path is* $\perp$*. We refer* l_j *as the leaves of the tree.*

The above function achieves the separation between D vs. R_0 and D vs. Q for $m = 2n$. Here we will restate some of the results from [4] which we will use to prove the results for our function:

▶ **Theorem 25** ([4]). *The function* $\mathsf{A1}_{(m.n)}$ *in Definition 24 satisfies*

$\mathsf{D} = \Omega(n^2)$ *for* $m = 2n$ *where* $m, n \in \mathbb{N}$,

$\mathsf{R}_0 = \widetilde{O}(m+n)$ *for any* $m, n \in \mathbb{N}$,

$\mathsf{Q} = \widetilde{O}(\sqrt{m}+\sqrt{n})$ *for any* $m, n \in \mathbb{N}$.

Though [4] gives the deterministic lower bound for the function $\mathsf{A1}$ precisely for $2m \times m$ matrices following the same line of argument it can be proved that $\mathsf{D}(\Omega(n^2))$ holds for $n \times n$ matrices also. For sake of completeness we give a proof for $n \times n$ matrices in the full version of this paper [15].

▶ **Theorem 26.** $\mathsf{D}(\mathsf{A1}_{(n,n)}) = \Omega(n^2)$.

Also [20]'s function realises quadratic separation between D and deg and the proof goes via UC_{min} upper bound. But $\mathsf{A1}_{(n,n)}$ exhibits the same properties corresponding to UC_{min}. So, from the following observation it follows that $\mathsf{A1}_{(n,n)}$ also achieves quadratic separation between D and deg.

▶ **Observation 27.** $\deg(\mathsf{A1}_{(n,n)}) = O(n)$ *for any* $n \in \mathbb{N}$.

Another important observation that we need is the following:

▶ **Observation 28** ([4]). *For any input* $\Sigma^{n\times n}$ *to the function* $\mathsf{A1}_{(n,n)}$ *(in Definition 24) if we permute the rows of the matrix using a permutation* σ_r *and permute the columns of the matrix using a permutation* σ_c *and we update the pointers in each of the cells of the matrix accordingly then the function value does not change.*

3 High level description of our techniques

Pointer functions are defined over a special domain called *pointer matrix*, which is a $m \times n$ grid matrix. Each cell of the matrix contains some labels and some pointers that point either to some other cell or to a row or column [6]. As described in [20], the high level idea of pointer

[6] We naturally think of a pointer pointing to a cell as two pointers - one pointing to the row and the other to the column.

functions is the usage of pointers to make certificates unambiguous without increasing the input size significantly. This technique turns out to be very useful to give separations between various complexity measures as we see in [24], [19] and [4].

Now we want to produce a new function that possesses all the properties of pointer functions, along with the additional property of being transitive. To do so, first, we will encode the labels so that we can permute the bits (by a suitable transitive group) while keeping the structure of unambiguous certificates intact so that the function value remains invariant. One such natural technique would be to encode the contents of each cell in such a way that allows us to permute the bits of the contents of each cell using a transitive group and permute the cells among each other using another transitive group, and doing all of these while ensuring the unambiguous certificates remains intact [7]. This approach has a significant challenge: namely how to encode the pointers.

The information stored in each cell (other than the pointers) can be encoded using fixed logarithmic length strings of different Hamming weights - so that even if the strings are permuted and/or the bits in each string are permuted, the content can be "decoded". Unfortunately, this can only be done when the cell's contents have a constant amount of information - which is the case for pointer functions (except for the pointers). Since the pointers in the cell are strings of size $O(\log n)$ (as they are pointers to other columns or rows), if we want to use the similar Hamming weight trick, the size of the encoding string would need to be polynomial in $O(n)$. That would increase the size of the input compared to the unambiguous certificate. This would not give us tight separation results.

Also, there are three more issues concerning the encodings of pointers:

- As we permute the cells of the matrix according to some transitive group, the pointers within each cell need to be appropriately changed. In other words, when we move some cell's content to some other cell, the pointers pointing to the previous cell should point to the current cell now.
- If a pointer is encoded using a certain t-bit string, different permutations of bits of the encoded pointer can only generate a subset of all t-bit strings.
  For example: if we encode a pointer using a string of Hamming weight 10 then however we permute the bits of the string, the pointer can at most be modified to point to cells (or rows or columns) the encoding of whose pointers also have Hamming weight 10. (The issue is that permuting the bits of a string cannot change the Hamming weight of a string).
  The encoding of all the pointers should have the same Hamming weight.
- The encoding of the pointers has to be transitive. That is, we should be able to permute the bits of the encodings of the pointer using a transitive group in such a way that either the pointer value does not change or as soon as the pointer values changes, the cells gets permuted accordingly - kind of like an "entanglement".

The above three problems are somewhat connected. Our first innovative idea is to use *binary balance representation* (Definition 9) to represent the pointers. This way, we take care of the second issue. For the first and third issues, we define the transitive group – both the group acting on the contents of the cells (and hence on the encoding of the pointers) and the group acting on the cells itself – in a "entangled" manner. For this we induce a group action

[7] Here, we use the word "encode" since we can view the function defined only over codewords, and when the input is not a codeword, then it evaluates to 0. In our setting, since we are trying to preserve the one-certificates, the codewords are those strings where the unambiguous certificate is encoded correctly. At the same time, we must point out that the encoding of an unambiguous certificate is not necessarily unique.

acting on the nodes of a *balanced binary tree* and generate a transitive subgroup in S_n and $S_{2\log n}$ with the same action which will serve our purpose (Definition 16, Claim 18). This helps us to permute the rows (or columns) using a permutation while updating the encoding of the pointers accordingly.

By Claim 18, for every allowed permutation σ acting on the rows (or columns), there is a unique $\widehat{\sigma}$ acting on the encodings of the pointers in each of the cells such that the pointers are updated according to σ. This still has a delicate problem. Namely, each pointer is either pointing to a row or column. But the permutation $\widehat{\sigma}$ has no way to understand whether the encoding on which it is being applied points to a row or column. To tackle this problem, we think of the set of rows and columns as a single set. All of them are encoded by a string of size (say) $2t$, where for the rows, the second half of the encoding is all 0 while the columns have the first t bits all 0. This is the encoding described in Definition 19 using binary balanced representation. However, this adds another delicate issue about permuting between the first t bits of the encoding and the second t bits.

To tackle this problem, we modify the original function appropriately. We define a slightly modified version of existing pointer functions called $\mathsf{ModA1}$. This finally helps us obtain our "transitive pointer function," which has almost the same complexities as the original pointer function.

We have so far only described the high-level technique to make the 1st variation of pointer functions (Definition 24) transitive where there is the same number of rows and columns. The further variations need more delicate handling of the encoding and the transitive groups - though the central idea is similar.

4 Proof of Theorem 1

4.1 Transitive Pointer Function F_1 for Theorem 1

Our function $F_1 : \Gamma^{n\times n} \to \{0,1\}$ is a composition of two functions - an outer function $\mathsf{ModA1}_{(n,n)} : \bar{\Sigma}^{n\times n} \to \{0,1\}$ and an inner function $\mathsf{Dec} : \Gamma \to \bar{\Sigma}$. We will set Γ to be $\{0,1\}^{96\log n}$.

The outer function is a modified version of the $\mathsf{A1}_{(n,n)}$ - pointer function described in [4] (see Definition 24 for a description). The function $\mathsf{A1}_{(n,n)}$ takes as input a $(n\times n)$-matrix whose entries are from a set Σ and the function evaluates to 1 if a certain kind of 1-cell-certificate exists. Let us define a slightly modified function $\mathsf{ModA1}_{(n,n)} : \bar{\Sigma}^{n\times n} \to \{0,1\}$ where $\bar{\Sigma} = \Sigma \times \{\vdash, \dashv\}$. We can think of an input $A \in \bar{\Sigma}^{n\times n}$ as a pair of matrices $B \in \Sigma^{n\times n}$ and $C \in \{\vdash, \dashv\}^{n\times n}$. The function $\mathsf{ModA1}_{(n,n)}$ is defined as

$$\mathsf{ModA1}_{(n,n)}(A) = 1 \text{ iff } \begin{cases} \text{Either,} & \text{(i) } \mathsf{A1}_{(n,n)}(B) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate have } \vdash \text{ in the corresponding cells in } C \\ \text{Or,} & \text{(ii) } \mathsf{A1}_{(n,n)}(B^T) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate have } \dashv \text{ in the corresponding cells in } C^T \end{cases}$$

Note that both the two conditions (i) and (ii) cannot be satisfied simultaneously. From this it is easy to verify that the function $\mathsf{ModA1}_{(n,n)}$ has all the properties as $\mathsf{A1}_{(n,n)}$ as described in Theorem 25.

The inner function Dec (we call it a decoding function) is function from Γ to $\bar{\Sigma}$, where $\Gamma = 96\log n$. Thus our final function is

$$F_1 := \left(\mathsf{ModA1}_{(n,n)} \circ \mathsf{Dec}\right) : \Gamma^{n\times n} \to \{0,1\}.$$

4.1.1 Inner Function Dec

The input to $\mathsf{A1}_{(n,n)}$ is a Type_1 pointer matrix $\Sigma^{n\times n}$. Each cell of a Type_1 pointer matrix contains a 4-tuple of the form (Value, LPointer, RPointer, BPointer) where Value is either 0 or 1 and LPointer, RPointer are pointers to the other cells of the matrix and BPointer is a pointer to a column of the matrix (or can be a null pointer also). Hence, $\Sigma = \{0,1\} \times [n]^2 \times [n]^2 \times [n]$. For the function $\mathsf{A1}_{(n,n)}$ it was assumed (in [4]) that the elements of Σ is encoded as a k-length[8] binary string in a canonical way.

The main insight for our function $F_1 := \left(\mathsf{ModA1}_{(n,n)} \circ \mathsf{Dec}\right)$ is that we want to maintain the basic structure of the function $\mathsf{A1}_{(n,n)}$ (or rather of $\mathsf{ModA1}_{(n,n)}$) but at the same time we want to encode the $\bar{\Sigma} = \Sigma \times \{\vdash, \dashv\}$ in such a way that the resulting function becomes transitive. To achieve this, instead of having a unique way of encoding an element in $\bar{\Sigma}$ we produce a number of possible encodings[9] for any element in $\bar{\Sigma}$. The inner function Dec is therefore a decoding algorithm that given any proper encoding of an element in $\bar{\Sigma}$ will be able to decode it back.

For the ease of understanding we start by describing the possible "encodings" of $\bar{\Sigma}$, that is by describing the pre-images of any element of $\bar{\Sigma}$ in the function Dec.

"Encodings" of the content of a cell in $\bar{\Sigma}^{n\times n}$. We will encode any element of $\bar{\Sigma}$ using a string of size $96\log n$ bits. Recall that, an element in $\bar{\Sigma}$ is of the form $(V, (r_L, c_L), (r_R, c_R), (c_B), T)$, where V is the Boolean value, (r_L, c_L), (r_R, c_R) and c_B are the left pointer, right pointers and bottom pointer respectively and T take the value $\vdash$ or $\dashv$. The overall summary of the encoding is as follows:

- **Parts:** We will think of the tuple as 7 objects, namely V, r_L, c_L, r_R, c_R, c_B and T. We will use $16\log n$ bits to encode each of the first 6 objects. The value of T will be encoded in a clever way. So the encoding of any element of $\bar{\Sigma}$ contains 6 *parts* - each a binary string of length $16\log n$.
- **Blocks:** Each of 6 *parts* will be further broken into 4 *blocks* of equal length of $4\log n$. One of the blocks will be a special block called the "encoding block".

Now we explain, for a tuple $(V, (r_L, c_L), (r_R, c_R), (c_B), T)$ what is the 4 blocks in each part. We will start by describing a "standard-form" encoding of a tuple $(V, (r_L, c_L), (r_R, c_R), (c_B), T)$ when $T = \vdash$. Then, we will extend it to describe the standard for encoding the tuple when $T = \dashv$. And finally we will explain all other valid encoding of the tuple by describing all the allowed permutations on the bits of the encoding.

Standard-form encoding of $(V, (r_L, c_L), (r_R, c_R), (c_B), T)$ where $T = \vdash$. For the standard form of encoding we will assume that the information of $V, r_L, c_L, r_R, c_R, c_B$ are stored in parts $P1, P2, P3, P4, P5$ and $P6$ respectively. For all $i \in [6]$, the part P_i with have blocks B_1, B_2, B_3 and B_4, of which the block B_1 will be the encoding-block. The encoding will ensure that every parts within a cell will have distinct Hamming weight. The description is also compiled in the Table 2.

- For part $P1$ (that is the encoding of V) the encoding block B_1 will store $\ell_1 \cdot \ell_2$ where ℓ_1 be the $2\log n$ bit binary string with Hamming weight $2\log n$ and ℓ_2 is any $2\log n$ bit binary string with Hamming weight $2\log n - 1 - V$. The blocks B_2, B_3 and B_4 will store a $4\log n$ bit string that has Hamming weight $4\log n, 2\log n + 1$ and $2\log n + 2$ respectively. Any

[8] For the canonical encoding $k = (1 + 5\log n)$ was sufficient

[9] We use the term "encoding" a bit loosely in this context as technically an encoding means a unique encoding. What we actually mean is the pre-images of the function Dec.

Table 2 Standard form of encoding of element $(V,(r_L,c_L),(r_R,c_R),c_B,\vdash)$ by a $96\log n$ bit string that is broken into 6 parts $P_1,\ldots,P_6$ of equal size and each Part is further broken into 4 Blocks B_1,B_2,B_3 and B_4. So all total there are 24 blocks each containing a $4\log n$-bit string. For the standard form of encoding of element $(V,(r_L,c_L),(r_R,c_R),c_B,\dashv)$ we encode $(V,(r_L,c_L),(r_R,c_R),c_B,\vdash)$ in the standard form as described in the table and then apply the $\mathsf{Swap}_{\frac{1}{2}}$ on each block. The last column of the table indicates the Hamming weight of each Part.

...	B_1 "encoding"-block	B_2	B_3	B_4	Hamming weight
$P1$	$\ell_1\ell_2$, where $\lvert\ell_1\rvert = 2\log n$, and $\lvert\ell_2\rvert = 2\log n - 1 - V$	$4\log n$	$2\log n+1$	$2\log n+2$	$12\log n+2-V$
$P2$	$\mathcal{E}(r_L,0)$	$2\log n+3$	$2\log n+1$	$2\log n+2$	$7\log n+6$
$P3$	$\mathcal{E}(0,c_L)$	$2\log n+4$	$2\log n+1$	$2\log n+2$	$7\log n+7$
$P4$	$\mathcal{E}(r_R,0)$	$2\log n+5$	$2\log n+1$	$2\log n+2$	$7\log n+8$
$P5$	$\mathcal{E}(0,c_R)$	$2\log n+6$	$2\log n+1$	$2\log n+2$	$7\log n+9$
$P6$	$\mathcal{E}(0,c_B)$	$2\log n+7$	$2\log n+1$	$2\log n+2$	$7\log n+10$

fixed string with the correct Hamming weight will do. We are not fixing any particular string for the blocks B_2, B_3 and B_4 to emphasise the fact that we will be only interested in the Hamming weights of these strings.

- The encoding block $B1$ for parts $P2,P3,P4,P5$ and $P6$ will store the string $\mathcal{E}(r_L,0)$, $\mathcal{E}(0,c_L)$, $\mathcal{E}(r_R,0)$, $\mathcal{E}(0,c_r)$ and $\mathcal{E}(0,C_B)$ respectively, where $\mathcal{E}$ is the Balanced-pointer-encoding function (Definition 19). For part P_i (with $2\le i\le 6$) block B_2,B_3 and B_4 will store any $4\log n$ bit string with Hamming weight $2\log n+1+i$, $2\log n+1$ and $2\log n+2$ respectively.

Standard form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\dashv$. For obtaining a standard-form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\dashv$, first we encode $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\vdash$ using the standard-form encoding. Let $(P1,P2,\ldots,P6)$ be the standard-form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\vdash$. Now for each of the block apply the $\mathsf{Swap}_{\frac{1}{2}}$ operator.

Valid permutation of the standard form. Now we will give a set of valid permutations to the bits of the encoding of any element of $\bar{\Sigma}$. The set of valid permutations are classified into into 3 categories:

1. Part-permutation: The 6 parts can be permuted using any permutation from S_6
2. Block-permutation: In each of the part, the 4 blocks (say B_1,B_2,B_3,B_4) can be permuted is two ways. (B_1,B_2,B_3,B_4) can be send to one of the following:

   (a) Simple Block Swap: (B_3,B_4,B_1,B_2) (b) Block Flip: $(B_2,B_1,\mathsf{flip}(B_3),\mathsf{flip}(B_4))$

The "decoding" function $\mathbf{Dec}:\{0,1\}^{96\log n}\to\bar{\Sigma}$.

- Identify the parts containing the encoding of V, r_L, c_L, r_R, c_R and c_B. This is possible because every part has a unique Hamming weight.
- For each part identify the blocks. This is also possible as in any part all the blocks have distinct Hamming weight. Recall, the valid Block-permutations, namely Simple Block Swap and Block Flip. By seeing the positions of the blocks one can understand if flip was applied and to what and using that one can revert the blocks back to the standard-form (recall Definition 9).

- In the part containing the encoding of V consider the encoding-block. If the block is of the form $\{(\ell_1\ell_2)$ such that $|\ell_1| = 2\log n, |\ell_2| \leq 2\log n - 1\}$ then $T = \{\vdash\}$. If the block is of the form $\{(\ell_2\ell_1)$ such that $|\ell_1| = 2\log n, |\ell_2| \leq 2\log n - 1\}$ then $T = \{\dashv\}$.
- By seeing the encoding block we can decipher the original values and the pointers.
- If the $96\log n$ bit string doesn't have the form of a valid encoding, then decode it as $(0, \perp, \perp, \perp)$.

4.2 Proof of Transitivity of the function

We start with describing the transitive group for which F_1 is transitive.

The Transitive Group. We start with describing a transitive group $\mathcal{T}$ acting on the cells of the matrix A. The matrix has rows $r_1, \ldots, r_n$ and columns $c_1, \ldots, c_n$. And we use the encoding function $\mathcal{E}$ to encode the rows and columns. So the index of the rows and columns are encoded using a $4\log n$ bit string. A permutation from $\mathsf{Bt}_{4\log n}$ (see Definition 16) on the indices of a $4\log n$ bit string will therefore induce a permutation on the set of rows and columns which will give us a permutation on the cells of the matrix. We will now describe the group $\mathcal{T}$ acting on the cells of the matrix by describing the permutation group $\widehat{\mathcal{T}}$ acting on the indices of a $4\log n$ bit string. The group $\widehat{\mathcal{T}}$ will be the group $\mathsf{Bt}_{4\log n}$ acting on the set $[4\log n]$. We will assume that $\log n$ is a power of 2. The group $\mathcal{T}$ with be the resulting group of permutations on the cells of the matrix induced by the group $\widehat{\mathcal{T}}$ acting on the indices on the balanced-pointer-encoding. Note that $\mathcal{T}$ is acting on the domain of $\mathcal{E}$ and $\widehat{\mathcal{T}}$ is acting on the image of $\mathcal{E}$. Also $\widehat{\mathcal{T}}$ is a transitive subgroup of $\mathsf{S}_{4\log n}$ from Claim 17.

▶ **Observation 29.** *For any $1 \leq i \leq 2\log n$ consider the permutation "ith-bit-flip" in $\widehat{\mathcal{T}}$ that applies the transposition $(2i-1, 2i)$ to the indices of the balanced-pointer-encoding. Since the $\mathcal{E}$-encoding of the row $(r_k, 0)$ uses the balanced binary representation of k in the first half and all zero sting in the second half, the jth bit in the binary representation of k is stored in the $2j-1$ and $2j$-th bit in the $\mathcal{E}$-encoding of r_i. So the j-th-bit-flip acts on the sets of rows by swapping all the rows with 1 in the j-th bit of their index with the corresponding rows with 0 in the j-th bit of their index. Also, if $i > \log n$ then there is no effect of the i-th-bit-flip operation on the set of rows. Similarly for the columns.*

Using Observation 29 we have the following claim.

▷ Claim 30. The group $\mathcal{T}$ acting on the cells of of the matrix is a transitive group. That is, for all $1 \leq i_1, j_1, i_2, j_2 \leq n$ there is a permutation $\widehat{\sigma} \in \widehat{\mathcal{T}}$ such that $\widehat{\sigma}[\mathcal{E}(i_1, 0)] = \mathcal{E}(i_2, 0)$ and $\widehat{\sigma}[\mathcal{E}(0, j_1)] = (0, j_2)$. Or in other words, there is a $\sigma \in \mathcal{T}$ acting on the cell of the matrix that would take the cell corresponding to row r_{i_1} and column c_{j_1} to the cell corresponding to row r_{i_2} and column c_{j_2}.

From the Claim 30 we see the group $\mathcal{T}$ acting on the cells of of the matrix is a transitive. But it does not touch the contents within the cells of the matrix. But the input to the function F_1 contains element of $\Gamma = \{0,1\}^{96\log n}$ in each cell. So we now need to extend the group $\mathcal{T}$ to a group G that acts on all the indices of the bits of the input to the function F_1.

Recall that the input to the function F_1 is a $(n \times n)$-matrix with each cell of matrix containing a binary string of length $96\log n$ which has 6 parts of size $16\log n$ each and each part has 4 blocks of size $4\log n$ each. We classify the generating elements of the group G into 4 categories:

1. Part-permutation: In each of the cells the 6 parts can be permuted using any permutation from S_6

2. Block-permutation: In each of the Parts the 4 blocks can be permuted in the following ways. (B_1, B_2, B_3, B_4) can be send to one of the following
   a. Simple Block Swap: (B_3, B_4, B_1, B_2)
   b. Block Flip (#1): $(B_2, B_1, \mathsf{flip}(B_3), \mathsf{flip}(B_4))$
   c. Block Flip (#2)[10]: $(\mathsf{flip}(B_1), \mathsf{flip}(B_2), B_4, B_3)$
3. Cell-permutation: for any $\sigma \in \mathcal{T}$ the following two action has to be done simultaneously:
   a. (Matrix-update) Permute the cells in the matrix according to the permutation σ. This keeps the contents within each cells untouched - it just changes the location of the cells.
   b. (Pointer-update) For each of blocks in each of the parts in each of the cells permute the indices of the $4 \log n$-bit strings according to σ, that is apply $\widehat{\sigma} \in \widehat{\mathcal{T}}$ corresponding to σ.

We now have the following theorems that would prove that the function F_1 is transitive.

▶ **Theorem 31.** *G is a transitive group and the function F_1 is invariant under the action of the G.*

Proof of Theorem 31. To prove that the group G is transitive we show that for any indices $p, q \in [96n^2 \log n]$ there is a permutation $\sigma \in \mathsf{G}$ that would take p to q. Recall that the string $\{0,1\}^{96n^2 \log n}$ is a matrix $\Gamma^{(n \times n)}$ with $\Gamma = \{0,1\}^{96 \log n}$ and every element in Γ is broken into 6 parts and each part being broken into 4 block of size $4 \log n$ each. So we can think of the index p as sitting in k_pth position $(1 \leq k_p \leq 4 \log n)$ in the block B_p of the part P_p in the (r_p, c_p)-th cell of the matrix. Similarly, we can think of q as sitting in k_qth position $(1 \leq k_q \leq 4 \log n)$ in the block B_q of the part P_q in the (r_q, c_q)-th cell of the matrix.

We will give a step by step technique in which permutations from G can be applied to move p to q.

- **Step 1: Get the positions in the block correct:** If $k_p \neq k_q$ then take a permutation $\widehat{\sigma}$ from $\widehat{\mathcal{T}}$ that takes k_p to k_q. Since $\widehat{\mathcal{T}}$ is a transitive so such a permutation exists. Apply the cell-permutation $\sigma \in \mathcal{T}$ corresponding to $\widehat{\sigma}$. As a result the index p can be moved to a different cell in the matrix but, by the choice of $\widehat{\sigma}$ its position in the block in which it is will be k_q. Without loss of generality, we assume the the cell location does not change.
- **Step 2: Get the cell correct:** Using a cell-permutation that corresponds to a series of "bit-flip" operations change r_p to r_q and c_p to c_q. Since one bit-flip operations basically changes one bit in the binary representation of the index of the row or column such a series of operations can be made.
  Since each bit-flip operation is executed by applying the bit-flips in each of the blocks so this might have once again changed the position of the index p in the block. But, even if the position in the block changes it must be a flip operation away. Or in other word, since in the beginning of this step $k_p = k_q$, so if k_q is even (or odd) then after the series bit-flip operations the position of p in the block is either k_q or $(k_q - 1)$ (or $(k_q + 1)$).
- **Step 3: Align the Part:** Apply a suitable permutation to ensure that the part P_p moves to part P_q. Note this does not change the cell or the block within the part or the position in the block.

[10] Actually this Block flip can be generated by a combination of Simple Block Swap and Block Flip (#1)

- **Step 4: Align the Block:** Using a suitable combination of Simple Block Swap and Block Flip ensures the Block number gets matched, that is B_p goes to B_q. In this case the cell or the Part does not change. But depending on whether the Block Flip operation is applied the position in the block can again change. But, the current position in the block k_p is at most one flip away from k_q.
- **Step 5: Apply the final flip:** It might so happen that already we a done after the last step. If not we know that the current position in the block k_p is at most one flip away from k_q. So we apply the suitable Block-flip operation. Thus will not change the cell position, Part number, Block number and the position in the block will match.

Hence we have proved that the group G is transitive. Now we show that the the function F_1 is invariant under the action of G, i.e., for any elementary operations π from the group G and for any input $\Gamma^{(n\times n)}$ the function value does not change even if after the input is acted upon by the permutation π.

Case 1: π is a Part-permutation: It is easy to see that the decoding algorithm Dec is invariant under Part-permutation. This was observed in description of the decoding algorithm Dec in Section 4.1.1. So clearly that the function F_1 is invariant under any Part-permutation.

Case 2: π is a Block-permutation: Here also it is easy to see that the decoding algorithm Dec is invariant under Block-permutation. This was observed in description of the decoding algorithm Dec in Section 4.1.1. Thus F_1 is also invariant under any Block-permutation.

Case 3: π is a Cell-permutation From Observation 28 it is enough to prove that when we permute the cells of the matrix we update the points in the cells accordingly.

Let $\pi \in \mathcal{T}$ be a permutation that permutes only the rows of the matrix. By Claim 20, we see that the contents of the cells will be updated accordingly. Similarly if π only permute the columns of the matrix we will be fine.

Finally, if π swaps the row set and the column set (that is if π makes a transpose of the matrix) then for all i row i is swapped with column i and it is easy to see that $\widehat{\pi}[\mathcal{E}(i,0)] = \mathcal{E}(0,i)$. In that case the encoding block of the value part in a cell also gets swapped. This will thus be encoding the T value as $\dashv$. And so the function value will not be affected as the $T = \dashv$ will ensure that one should apply the π that swaps the row set and the column set to the input before evaluating the function. ◀

4.3 Properties of the Function

▷ Claim 32. Deterministic query complexity of F_1 is $\Omega(n^2)$.

Proof. The function $\mathsf{ModA1}_{(n,n)}$ is a "harder" function than $\mathsf{A1}_{(n,n)}$. So $\mathsf{D}(\mathsf{ModA1}_{(n,n)})$ is at least that of $\mathsf{D}(\mathsf{A1}_{(n,n)})$. Now since, F_1 is $\left(\mathsf{ModA1}_{(n,n)} \circ \mathsf{Dec}\right)$ so clearly the $\mathsf{D}(F_1)$ is at least $\mathsf{D}(\mathsf{A1}_{(n,n)})$. Theorem 26 proves that $\mathsf{D}(\mathsf{A1}_{(n,n)})$ is $\Omega(n^2)$. Hence $\mathsf{D}(F_1) = \Omega(n^2)$. ◁

The following Claim 33 follows from the definition of the function $\mathsf{ModA1}_{(n,n)}$.

▷ Claim 33. The following are some properties of the function $\mathsf{ModA1}_{(n,n)}$

1. $\mathsf{R}_0(\mathsf{ModA1}_{(n,n)}) \leq 2\mathsf{R}_0(\mathsf{A1}_{(n,n)}) + O(n \log n)$
2. $\mathsf{Q}(\mathsf{ModA1}_{(n,n)}) \leq 2\mathsf{Q}(\mathsf{A1}_{(n,n)}) + O(n \log n)$
3. $\deg(\mathsf{ModA1}_{(n,n)}) \leq 2\deg(\mathsf{A1}_{(n,n)}) + O(n \log n)$

Finally, from "composition theorem" (formal proof of which is presented in the full version of the paper [15]) we see that the $\mathsf{R}_0(F_1)$, $\mathsf{Q}(F_1)$ and $\deg(F_1)$ are at most $O(\mathsf{R}_0(\mathsf{ModA1}_{(n,n)} \cdot \log n)$, $O(\mathsf{Q}(\mathsf{ModA1}_{(n,n)} \cdot \log n)$ and $O(\deg(\mathsf{ModA1}_{(n,n)} \cdot \log n)$, respectively. So combining this fact with Claim 32, Claim 33 and Theorem 25 (from [4]) we have Theorem 1.

5 Challenges in transitive versions of "cheat sheet" based functions

In this section we show that it is not possible to give a quadratic separation between degree and quantum query complexity for transitive functions by modifying the cheat sheet function using the techniques in [1] which go via unambiguous certificate complexity.

Let us start by recalling the cheat sheet framework from [1]. Let $f : \{0,1\}^n \to \{0,1\}$ be a total Boolean function. Let $C(f)$ be its certificate complexity and $Q(f)$ be its bounded-error quantum query complexity. We consider the following cheat sheet function, which we denote by $f_{CS,t} : \{0,1\}^{n\times\log t+t\times\log t\times C(f)\times\log n} \to \{0,1\}$:

- There are $\log t$ copies of f on disjoint sets on inputs denoted by $f_1, \ldots, f_{\log t}$.
- There are t cheat sheets: each cheat sheet is a block of $(\log t \times C(f) \times \log n)$ many bits
- Let $x_1, \ldots, x_{\log t} \in \{0,1\}^n$ denote the input to the $\log t$ copies of f and let $Y_1, \ldots, Y_t$ denote the t cheat sheets.
- Let $\ell = (f(x_1), \ldots, f(x_{\log t}))$. $f_{CS,t}$ evaluates to 1 if and only if Y_ℓ is a valid cheat sheet.

Separations between various complexity measures was shown in [1] using the cheat sheet framework. In [1], the separations that lower bound bounded-error quantum query complexity in terms of other complexity measures, for example degree, are obtained as follows:

1. Start with a total function $f : \{0,1\}^n \to \{0,1\}$ that has quadratic separation between quantum query complexity and certificate complexity: $Q(f) = \widetilde{\Omega}(n)$ and $C(f) = \widetilde{O}(\sqrt{n})$. Consider the cheat sheet version of this function $f_{CS,t}$, with $t = n^{10}$.
2. Lower bound $Q(f_{CS,t})$, for $t = n^{10}$, by $Q(f)$. This uses the hybrid method ([10]) and strong direct product theorem ([22]).
3. Upper bound degree of $f_{CS,t}$ by using the upper bound on the unambiguous certificate complexity of $f_{CS,t}$.

Instead of degree, one might use approximate degree in the third step above for a suitable choice of f (see [1] for details).

A natural approach to obtain a transitive function with gap between a pair of complexity measures is to modify the cheat sheet framework to make it transitive. One possible modification is to allow a poly-logarithmic blowup in the input size of the resulting transitive function while preserving complexity measures of the cheat sheet function that are of interest (upto poly-logarithmic factors).

We show, however, that it is not possible to obtain a quadratic separation between degree and quantum query complexity for transitive functions by modifying the cheat sheet function using the techniques in [1] which go via unambiguous certificate complexity. The reason for this is that the unambiguous certificate complexity of a transitive cheat sheet function on N-bits is $\Omega(\sqrt{N})$ (see Observation 34) whereas we show (see Lemma 35) that the quantum query complexity of such a function is $o(N)$.

Note that this does not mean that cheat sheet framework can not be made transitive to show such a quadratic gap. If the cheat sheet version of a function that is being made transitive has a better degree upper bound than that given by unambiguous certificate complexity then a better gap might be possible.

To formalize the above discussion we first need the following observation that lower bounds the certificate complexity of any transitive function.

▶ **Observation 34** ([30]). *Let $f : \{0,1\}^N \to \{0,1\}$ be a transitive function, then $C(f) \geq \sqrt{N}$.*

Next, we upper bound on quantum query complexity of cheat sheet function using quantum amplitude amplification ([11]). The details of proof of the following lemma can be found in the full version of this paper [15].

▶ **Lemma 35.** *The quantum query complexity of $f_{CS,t}$ is $O(\sqrt{t} \times \log t \times \sqrt{n} \times \log n)$.*

The cheat sheet version of f, $f_{CS,t}$, is a function on $\widetilde{\Theta}(n + C(f)t)$ many variables, where t is polynomial in n. From the cheat sheet property the unambiguous certificate complexity of $f_{CS,t}$, denoted by $\mathsf{UC}(f_{CS,t})$, is $\widetilde{\Theta}(C(f))$.

Let $\widetilde{f_{CS,t}}$ be a modified transitive version of $f_{CS,t}$ that preserves the quantum query complexity and certificate complexity of $f_{CS,t}$ upto poly-logarithmic factors, respectively. From Observation 34 it follows that $\mathsf{UC}(\widetilde{f_{CS,t}}) = \widetilde{\Omega}(\sqrt{n + C(f)t})$. On the other hand, since $\widetilde{f_{CS,t}}$ preserves the certificate complexity upto poly-logarithmic factors, $\mathsf{UC}(\widetilde{f_{CS,t}}) = \widetilde{O}(C(f))$. This implies that $t = \widetilde{O}(C(f))$. Lemma 35 that $Q(f_{CS,t})$ is at most $\widetilde{O}(C(f)\sqrt{t})$. Thus in order to achieve quadratic separation between UC and Q, t has to be $\widetilde{\Omega}(C(f)^2)$.

We end this section by giving a concrete approach towards showing separation between degree and quantum query complexity for a transitive functions using the cheat sheet method. We believe the it is possible to start with $f_{CS,t}$, for transitive function f and $t = \sqrt{n}$ and convert it to a transitive function that preserves the unambiguous certificate complexity and quantum query complexity upto poly-logarithmic factors, while incurring a poly-logarithmic blowup in the input size. However, we do not know how to prove quantum query complexity lower bound matching our upper bound from Lemma 35 for $t = \sqrt{n}$. We make the following conjecture towards this end, which, if true, implies that for a transitive function f, $Q(f) = \widetilde{\Omega}(\deg(f)^{4/3})$.

▶ **Conjecture 36.** *There exists a transitive function $f : \{0,1\}^n \to \{0,1\}$ with $C(f) = \widetilde{O}(\sqrt{n})$ and $Q(f) = \widetilde{\Omega}(n)$. Let $f_{CS,\sqrt{n}}$ be the cheat sheet version of f with $\sqrt{n}$ cheat sheets. Then $Q(f_{CS,\sqrt{n}}) = \Omega(n^{3/4})$.*

It was showed in [1] that the quantum query complexity of the cheat function $f_{CS,t}$, i.e. $Q(f_{CS,t})$, is lower bounded by $Q(f)$, when $t = n^{10}$. Their proof goes via he hybrid method ([10]) and strong direct product theorem ([22]). Is is interesting to find the the constant smallest c such that $Q(f_{CS,n^c}) = \Omega(Q(f))$. We know that such a c must be at least than 1 (from Lemma 35) and is at most 10 (from [1]). We state this formally below:

▶ **Question 37.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a non-constant Boolean function and let f_{CS,n^c} be its cheat sheet version with n^c cheat sheets. What is the smallest c such that the following is true $Q(f_{CS,n^c}) = \Omega(Q(f))$.*

6 Conclusion

As far as we know, this is the first paper that presents a thorough investigation on the relationships between various pairs of complexity measures for transitive function.

The current best-known relationships and best-known separations between various pairs of measures for transitive functions are summarized in the Table 1. Unfortunately, a number of cells in the table are not tight. In this context, we would like to point out some important directions:

- For some of these cells, the separation results for transitive functions are weaker than that of the general functions. A natural question is the following: why can't we design a transitive version of the general functions that achieve the same separation? For some cases, like the cheat sheet-based functions, we discuss the difficulties and possible directions in Section 5. Thus following is a natural question.

  ▶ **Open Problem 38.** *For a pair of complexity measures for Boolean functions whose best-known separations are achieved via cheat sheets, obtain similar separations for transitive Boolean functions.*

- A total function was constructed in [13] that demonstrates quadratic separations between approximate degree with sensitivity and several other complexity measures. It is thus natural to investigate the following open problem.

  ▶ **Open Problem 39.** *Come up with transitive functions that achieve similar separations for those pair of measures whose best-known separations are shown by [13].*

- Recently [8], [5] and [6] came up with new classes of Boolean functions, starting with the HEX (see [8]) and EAH (see [6]) functions, that exhibit improved separations between certificate complexity and other complexity measures using the.
  In light of these recent developments is important to ask whether similar separations can be shown for transitive functions. Following open problem is a natural starting point.

  ▶ **Open Problem 40.** *Can the HEX and EAH functions be modified to a transitive functions, while preserving its desired complexity measures upto poly-logarithmic factors?*

- While we have been concerned only with lower bounds in this paper, it is an exciting research direction to bridge the gap between complexity measures of transitive Boolean functions by providing improved upper bounds.

  ▶ **Open Problem 41.** *Bridge the gaps in Table 1 by coming up with better upper bounds on complexity measures for transitive functions.*

In the full version of this paper, [15], we summarize the results on how low can individual complexity measures go for transitive function. Even with the recent results of [21] and [2], there are significant gaps between the best-known lower and upper bounds in this case which gives another set of open problems to investigate in the study of combinatorial measures of transitive Boolean functions.

References

1 Scott Aaronson, Shalev Ben-David, and Robin Kothari. Separations in query complexity using cheat sheets. In *STOC*, pages 863–876, 2016. doi:10.1145/2897518.2897644.

2 Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of Huang's sensitivity theorem. In *STOC*, pages 1330–1342, 2021. doi:10.1145/3406325.3451047.

3 Andris Ambainis. Superlinear advantage for exact quantum algorithms. *SIAM Journal on Computing*, pages 617–631, 2016. doi:10.1137/130939043.

4 Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. *Journal of the ACM*, 64(5):32:1–32:24, 2017. doi:10.1145/3106234.

5 Kaspars Balodis. Several separations based on a partial Boolean function. *CoRR*, 2021. arXiv:2103.05593.

6 Kaspars Balodis, Shalev Ben-David, Mika Göös, Siddhartha Jain, and Robin Kothari. Unambiguous DNFs and Alon-Saks-Seymour. In *FOCS*, pages 116–124, 2022. doi:10.1109/FOCS52979.2021.00020.

7 Shalev Ben-David, Andrew M. Childs, András Gilyén, William Kretschmer, Supartha Podder, and Daochen Wang. Symmetries, graph properties, and quantum speedups. In *FOCS*, pages 649–660, 2020. doi:10.1109/FOCS46700.2020.00066.

8 Shalev Ben-David, Mika Göös, Siddhartha Jain, and Robin Kothari. Unambiguous DNFs from Hex. *Electronic Colloquium on Computational Complexity*, 28:16, 2021.

9 Shalev Ben-David, Pooya Hatami, and Avishay Tal. Low-sensitivity functions from unambiguous certificates. In *ITCS*, volume 67, pages 28:1–28:23, 2017. doi:10.4230/LIPIcs.ITCS.2017.28.

10 Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997. doi:10.1137/S0097539796300933.

11 Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

12 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

13 Mark Bun and Justin Thaler. A nearly optimal lower bound on the approximate degree of AC^0. *SIAM Journal on Computing*, 49(4), 2020. doi:10.1137/17M1161737.

14 Sourav Chakraborty. On the sensitivity of cyclically-invariant Boolean functions. *Discrete Mathematics and Theoretical Computer Science*, 13(4):51–60, 2011. doi:10.46298/dmtcs.552.

15 Sourav Chakraborty, Chandrima Kayal, and Manaswi Paraashar. Separations between combinatorial measures for transitive functions. *ArXiv*, 2021. arXiv:2103.12355.

16 Andrew Drucker. Block sensitivity of minterm-transitive functions. *Theoretical Computer Science*, 412(41):5796–5801, 2011. doi:10.1016/j.tcs.2011.06.025.

17 Yihan Gao, Jieming Mao, Xiaoming Sun, and Song Zuo. On the sensitivity complexity of bipartite graph properties. *Theoretical Computer Science*, 468:83–91, 2013. doi:10.1016/j.tcs.2012.11.006.

18 Justin Gilmer, Michael E. Saks, and Srikanth Srinivasan. Composition limits and separating examples for some Boolean function complexity measures. *Combinatorica*, 36(3):265–311, 2016. doi:10.1007/s00493-014-3189-x.

19 Mika Göös, T. S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Transactions on Computation Theory*, 10(1):4:1–4:20, 2018. doi:10.1145/3170711.

20 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM Journal on Computing*, 47(6):2435–2450, 2018. doi:10.1137/16M1059369.

21 Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, 190(3):949–955, 2019. doi:10.4007/annals.2019.190.3.6.

22 Troy Lee and Jérémie Roland. A strong direct product theorem for quantum query complexity. *Computational Complexity*, 22(2):429–462, 2013. doi:10.1109/CCC.2012.17.

23 Qian Li and Xiaoming Sun. On the sensitivity complexity of k-uniform hypergraph properties. In *STACS*, volume 66, pages 51:1–51:12, 2017. doi:10.4230/LIPIcs.STACS.2017.51.

24 Sagnik Mukhopadhyay and Swagato Sanyal. Towards better separation between deterministic and randomized query complexity. In *FSTTCS*, volume 45, pages 206–220, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.206.

25 Noam Nisan and Avi Wigderson. On rank vs. communication complexity. *Combinatorica*, 15(4):557–565, 1995. doi:10.1007/BF01192527.

26 David Rubinstein. Sensitivity vs. block sensitivity of Boolean functions. *Combinatorica*, 15(2):297–299, 1995. doi:10.1007/BF01200762.

27 Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985. doi:10.1016/0304-3975(85)90210-5.

28 Xiaoming Sun. Block sensitivity of weakly symmetric functions. *Theoretical Computer Science*, 384(1):87–91, 2007. doi:10.1016/j.tcs.2007.05.020.

29 Xiaoming Sun. An improved lower bound on the sensitivity complexity of graph properties. *Theoretical Computer Science*, 412(29):3524–3529, 2011. doi:10.1016/j.tcs.2011.02.042.

30 Xiaoming Sun, Andrew Chi-Chih Yao, and Shengyu Zhang. Graph properties and circular functions: How low can quantum query complexity go? In *CCC*, pages 286–293, 2004. doi:10.1109/CCC.2004.1313851.

31 György Turán. The critical complexity of graph properties. *Information Processing Letters*, 18(3):151–153, 1984. doi:10.1016/0020-0190(84)90019-X.

Approximating k-Edge-Connected Spanning Subgraphs via a Near-Linear Time LP Solver

Parinya Chalermsook
Aalto University, Espoo, Finland

Chien-Chung Huang
École Normale Supérieure, Paris, France

Danupon Nanongkai
University of Copenhagen, Denmark
KTH Royal Institute of Technology, Stockholm, Sweden

Thatchaphol Saranurak
University of Michigan, Ann Arbor, MI, USA

Pattara Sukprasert
Northwestern University, Evanston, IL, USA

Sorrachai Yingchareonthawornchai
Aalto University, Espoo, Finland

Abstract

In the k-edge-connected spanning subgraph (kECSS) problem, our goal is to compute a minimum-cost sub-network that is resilient against up to k link failures: Given an n-node m-edge graph with a cost function on the edges, our goal is to compute a minimum-cost k-edge-connected spanning subgraph. This NP-hard problem generalizes the minimum spanning tree problem and is the "uniform case" of a much broader class of survival network design problems (SNDP). A factor of two has remained the best approximation ratio for polynomial-time algorithms for the whole class of SNDP, even for a special case of 2ECSS. The fastest 2-approximation algorithm is however rather slow, taking $O(mnk)$ time [Khuller, Vishkin, STOC'92]. A faster time complexity of $O(n^2)$ can be obtained, but with a higher approximation guarantee of $(2k-1)$ [Gabow, Goemans, Williamson, IPCO'93].

Our main contribution is an algorithm that $(1+\varepsilon)$-approximates the optimal fractional solution in $\tilde{O}(m/\varepsilon^2)$ time (independent of k), which can be turned into a $(2+\varepsilon)$ approximation algorithm that runs in time $\tilde{O}\left(\frac{m}{\varepsilon^2} + \frac{k^2 n^{1.5}}{\varepsilon^2}\right)$ for (integral) kECSS; this improves the running time of the aforementioned results while keeping the approximation ratio arbitrarily close to a factor of two.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Approximation Algorithms, Data Structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.37

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2205.14978

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 759557 & 715672. Nanongkai and Saranurak were also partially supported by the Swedish Research Council (Reg. No. 2015-04659.). Chalermsook was also supported by the Academy Research Fellowship (grant number 310415).

Acknowledgements We thank the 2021 Hausdorff Research Institute for Mathematics Program Discrete Optimization during which part of this work was developed. Parinya Chalermsook thanks Chandra Chekuri for clarifications of his FOCS 2017 paper and for giving some pointers. We thank Corinna Coupette for bringing [24] to our attention.

1 Introduction

In the k-Edge-Connected Spanning Subgraph problem (kECSS), we are given an undirected n-node m-edge graph $G = (V, E)$ together with edge costs, and want to find a minimum-cost k-edge connected spanning subgraph.[1] For $k = 1$, this is simply the minimum spanning tree problem, and thus can be solved in $O(m)$ time [28]. For $k \geq 2$, the problem is a classical NP-hard problem whose first approximation algorithm was given almost four decades ago, where Frederickson and Jaja [19] gave a 3-approximation algorithm that runs in $O(n^2)$ time for the case of $k = 2$. The approximation ratio was later improved to 2 by an $\tilde{O}(mnk)$-time algorithm of Khuller and Vishkin [32].[2] This approximation factor of 2 has remained the best for more than 30 years, even for a special case of 2ECSS called the weighted tree augmentation problem. When the running time is of the main concern, the fastest known algorithm takes $O(n^2)$ time at the cost of a significantly higher $(2k - 1)$-approximation guarantee, due to Gabow, Goemans, and Williamson [22].

This above state-of-the-arts leave a big gap between algorithms achieving the best approximation ratio and the best time complexity. This gap exists even for $k = 2$. In this paper, we improve the running time of both aforementioned algorithms of [32, 22] while keeping the approximation ratio arbitrarily close to two. Our main contribution is a near-linear time algorithm that $(1 + \varepsilon)$-approximates the optimal *fractional* solution.

▶ **Theorem 1.** *For any $\varepsilon > 0$, there is a randomized $\widetilde{O}(m/\varepsilon^2)$-time algorithm that outputs a $(1 + \varepsilon)$-approximate* fractional *solution for kECSS.*

Following, in the high-level, the arguments of Chekuri and Quanrud [7] (i.e. solving the minimum-weight k disjoint arborescences in the style of [32] on the support of the sparsified fractional solution), the above fractional solution can be turned into a fast $(2 + \varepsilon)$-approximation algorithm for the integral version of kECSS.

▶ **Corollary 2.** *For any $\varepsilon > 0$, there exist*

- *a randomized $\widetilde{O}(m/\varepsilon^2)$-time algorithm that estimates the* value *of the optimal solution for kECSS to within a factor $(2 + \varepsilon)$, and*
- *a randomized $\tilde{O}\left(\frac{m}{\varepsilon^2} + \frac{k^2 n^{1.5}}{\varepsilon^2}\right)$-time algorithm that produces a feasible kECSS solution of cost at most $(2 + \varepsilon)$ times the optimal value.*

We remark that the term $\tilde{O}(k^2 n^{1.5})$ is in fact "tight" up to the state-of-the-art algorithm for finding minimum-weight k disjoint arborescences.[3]

Prior to our results, a sub-quadratic time algorithm was not known even for special cases of kECSS, called *k-Edge-Connected Augmentation* (kECA). In this problem, we are given a $(k - 1)$-edge-connected subgraph H of a graph G, and we want to minimize the total cost of adding edges in G to H so that H becomes k-edge connected. It is not hard to see that if we can α-approximates kECSS, then we can α-approximates kECA by assigning cost 0 to all

[1] Note that this problem should not be confused with a variant that allows to pick the same edge multiple time, which is sometimes also called kECSS (e.g., [6]). We follow the convention in [13] and call the latter variant minimum-cost k-edge connected spanning sub-multigraph (kECSSM) problem. (See also the work by Pritchard [39].)

[2] $\tilde{O}$ hides polylog(n) factor.

[3] More formally, if a minimum-weight union of k edge-disjoint arborescences can be found in time $T(k, m, n)$, then our algorithm would run in time $T(k, kn, n)$. The term $O(k^2 n^{1.5})$ came from Gabow's algorithm [20] that runs in time $O(km\sqrt{n}\log(nc_{\max}))$.

edges in H. This problem previously admits a $O(kn^2)$-time 2-approximation algorithm for any even integer k [31][4]. The approximation ratio of 2 remains the best even for 2ECA. Our result in Corollary 2 improves the previously best time complexity by a $\tilde{\Theta}(\sqrt{n})$ factor.

Perspective. The gap between algorithms with best approximation ratio and best time complexity in fact reflects a general lack of understanding on *fast approximation algorithms*. While polynomial-time algorithms were perceived by many as efficient, it is not a reality in the current era of large data, where it is nearly impossible to take $O(n^3)$ time to process a graph with millions or billions of nodes. Research along this line includes algorithms for sparsest cut [30, 29, 42, 35], multi-commodity flow [23, 17, 36], and travelling salesman problem [6, 7]. Some of these algorithms have led to exciting applications such as fast algorithms for max-flow [43], dynamic connectivity [38, 8, 40, 45, 37], vertex connectivity [34] and maximum matching [44].

The kECSS problem belongs to the class of survivable network design problems (SNDPs), where the goal is to find a subgraph ensuring that every pair of nodes (u, v) are $\kappa(u, v)$-edge-connected for a given function κ. (kECSS is the *uniform* version of SNDP where $\kappa(u, v) = k$ for every pair (u, v).) These problems typically focus on building a network that is resilient against device failures (e.g. links or nodes), and are arguably among the most fundamental problems in combinatorial optimization. Research in this area has generated a large number of beautiful algorithmic techniques during the 1990s, culminating in the result of Jain [26] which gives a 2-approximation algorithm for the whole class of SNDPs. Thus, achieving a *fast* 2-approximation algorithm for SNDPs is a very natural goal.

Towards this goal and towards developing fast approximation algorithms in general, there are two common difficulties:

1. Many approximation algorithms inherently rely on solving a *linear program (LP)* to find a fractional solution, before performing rounding steps. However, the state-of-the-art general-purpose linear program solvers are still quite slow, especially for kECSS and SNDP where the corresponding LPs are *implicit*.

   In the context of SNDP, the state-of-the-art (approximate) LP solvers still require at least quadratic time: Fleischer [18] designs an $\tilde{O}(mnk)$ for solving kECSS LP, and more generally for SNDP and its generalization [18, 14] with at least $\Theta(m \min\{n, k_{\max}\})$ iterations of minimum cost flow's computation are the best known running time where $k_{\max}$ is the maximum connectivity requirements.
2. Most existing techniques that round fractional solutions to integral ones are not "friendly" for the design of fast algorithms. For instance, Jain's celebrated iterative rounding [26] requires solving the LP $\Omega(m)$ times. Moreover, most LP-based network design algorithms are fine-tuned to optimize approximation factors, while designing near-linear time LP rounding algorithms requires limiting ourselves to a relatively small set of tools, about which we currently have very limited understanding.

This paper completely resolves the first challenge for kECSS and manages to identify a fundamental bottleneck of the second challenge.

[4] In Khuller and Vishkin [31], the kECA problem aims at augmenting the connectivity from k to $(k + 1)$ (but for us it is from $(k - 1)$ to k.)

Challenges for LP Solvers. Our main challenge is handling the so-called *box constraints* in the LPs. To be concrete, below is the LP relaxation of kECSS on graph $G = (V, E)$.

$$\min\{\sum_{e\in E} c_e x_e : \sum_{e\in\delta_G(S)} x_e \geq k \ (\forall S \subseteq V), x \in [0,1]^E\} \tag{1}$$

where $\delta_G(S)$ is the set of edges between nodes in S and $V \setminus S$. The box constraints refer to the constraints $x \in [0,1]^E$. Without these constraints, we can select the same edge multiple times in the solution; this problem is called kECSSM in [13] (see Footnote 1). Removing the box constraints often make the problem significantly easier. For example, the min-cost st-flow problem without the box constraints become computing the shortest st-path, which admits a much faster algorithm.

For kECSS, it can be shown that solving (1) without the box constraints can be reduced to solving (1) *with* $k = 1$ and multiplying all x_e with k. In other words, without the box constraints, fractional kECSS is equivalent to fractional 1ECSS.This *fractional* 1ECSS can be $(1+\varepsilon)$-approximated in near-linear time by plugging in the dynamic minimum cut data structure of Chekuri and Quanrud [6] to the multiplicative weight update framework (MWU).

However, with the presence of box constraints, to use the MWU framework we would need a dynamic data structure for a much more complicated cut problem, that we call, the *minimum normalized free cut* problem (roughly, this is a certain normalization of the minimum cut problem where the costs of up to k heaviest edges in the cut are ignored.) For our problem, the best algorithm in the static setting we are aware of (prior to this work) is to use Zenklusen's $\tilde{O}(mn^4)$-time algorithm [47] for the *connectivity interdiction* problem.[5] This results in an $\tilde{O}(kmn^4)$-time static algorithm. Speeding up and dynamizing this algorithm seems very challenging. Our main technical contribution is an efficient dynamic data structure (in the MWU framework) for the $(1+\varepsilon)$-approximate minimum normalized free cut problem. We explain the high-level overview of our techniques in Section 2.

Further Related Works. The kECSS and its special cases have been studied extensively. For all $k \geq 2$, the kECSS problem is known to be APX-hard [15] even on bounded-degree graphs [9] and when the edge costs are 0 or 1 [39]. Although a factor 2 approximation for kECSS has not been improved for almost 3 decades, various special cases of kECSS admit better approximation ratios (see for instance [25, 16, 1]). For instance, the unit-cost kECSS ($c_e = 1$ for all $e \in E$) behaves very differently, admitting a $(1 + O(1/k))$ approximation algorithm [21, 33]. For the 2ECA problem, one can get a better than 2 approximation when the edge costs are bounded [1, 16]. Otherwise, for general edge costs, the factor of 2 has remained the best known approximation ratio even for the 2ECA problem.

The kECSS problem in special graph classes have also received a lot of attention. In Euclidean setting, a series of papers by Czumaj and Lingas led to a near-linear time approximation schemes for constant k [12, 11]. The problem is solvable in near-linear time when k and treewidth are constant [3, 5]. In planar graphs, 2ECSS, 2ECSSM and 3ECSSM admit a PTAS [10, 4].

Organization. We provide a high-level overview of our proofs in Section 2. In Section 3, we explain the background on Multiplicative Weight Updates (MWU) for completeness (although this paper is written in a way that one can treat MWU as a black box). In Section 4, we

[5] In the connectivity interdiction problem, we are given $G = (V, E)$ and $k \in \mathbb{N}$, our goal is to compute $F \subseteq E$ to delete from G in order to minimize the minimum cut in the resulting graph.

prove our main technical component. In Section 5, we present our LP solver. In Section 6, we show how to round the fractional solution obtained from the LP solver. Due to space limitations, many proofs are deferred to Appendix.

2 Overview of Techniques

In this section, we give a high-level overview of our techniques in connection to the known results. Our work follows the standard Multiplicative Weight Update (MWU) framework together with the Knapsack Covering (KC) inequalities (see Section 3 for more background). Roughly, in this framework, in order to obtain a near-linear time LP solver for kECSS, it suffices to provide a fast dynamic algorithm for a certain optimization problem (often called the oracle problem in the MWU literature):

▶ **Definition 3** (Minimum Normalized Free Cuts). *We are given a graph $G = (V, E)$, weight function $\boldsymbol{w} : E \to \mathbb{R}_{\geq 0}$, integer k, and our goal is to compute a cut $S \subseteq V$ together with edges $F \subseteq \delta_G(S) : |F| \leq k - 1$ that minimizes the following objective[6]:*

$$\min_{S \subsetneq V, F \subseteq \delta_G(S) : |F| \leq (k-1)} \frac{\boldsymbol{w}(\delta_G(S) \setminus F)}{k - |F|},$$

where $\delta_G(S)$ denotes the set of edges that has exactly one end point in S. We call the minimizer (S, F) the minimum normalized free cut.

This is similar to the minimum cut problem, except that we are allowed to "remove" up to $(k-1)$ edges (called *free edges*) from each candidate cut $S \subseteq V$, and the cost would be "normalized" by a factor of $(k - |F|)$.[7] Notice that there are (apparently) two sources of complexity for this problem. First, we need to find the cut S and second, given S, to compute the optimal set $F \subseteq \delta_G(S)$ of free edges. To our best knowledge, a previously fastest algorithm for this problem takes $\tilde{O}(mn^4)$ time by reducing to the connectivity interdiction problem [47], while we require near-linear time. This is our first technical challenge.

Our second challenge is as follows. To actually speed up the whole MWU framework, in addition to solving the oracle problem statically efficiently, we further need to implement a dynamic version of the oracle with polylog(n) update time. In our case, the goal is to maintain a dynamic data structure on graph $G = (V, E)$, weight function $\mathbf{w}$, cost function c, that supports the following operation:

▶ **Definition 4.** *The* PunishMin *operation computes a $(1 + O(\varepsilon))$-approximate normalized free cut and multiply the weight of each edge $e \in \delta_G(S) \setminus F$ by a factor of at most e^{ε}.*[8]

We remark that invoking the PunishMin operation does not return the cut (S, F), and the only change is the weight function $\mathbf{w}$ being maintained by the data structure.

▶ **Proposition 5** (Informal). *Assume that we are given a dynamic algorithm that supports* PunishMin *with amortized* polylog(n) *cost per operations, then the kECSS LP can be solved in time $\tilde{O}(m)$.*

---

[6] For any function f, for any subset S of its domain, we define $f(S) = \sum_{s \in S} f(s)$.

[7] This is in fact a special case of a similar objective considered by Feldmann, Könemann, Pashkovich and Sanità [14], who considered applying the MWU framework for the generalized SNDP

[8] The actual weight $w(e)$ is updated for all $e \in \delta_G(S) \setminus F$: $w(e) \leftarrow w(e) \cdot \exp(\frac{\varepsilon c_{\min}}{c_e})$ where $c_{\min}$ is the minimum edge capacity in $\delta_G(S) \setminus F$.

Let us call such a dynamic algorithm a fast **dynamic punisher**. The fact that a fast dynamic punisher implies a fast LP solver is an almost direct consequence of MWU [23].

Therefore, we focus on designing a fast dynamic algorithm for solving (and punishing) the minimum normalized free cut problem. Our key idea is an efficient and dynamic implementation of the **weight truncation** idea.

> **Weight truncation**: Let $G = (V, E)$ and $\rho \in \mathbb{R}_{\geq 0}$ be a threshold. For any weight function $\mathbf{w}$ of G, denote by $\mathbf{w}_\rho$ the truncated weight defined by $\mathbf{w}_\rho(e) = \min\{\mathbf{w}(e), \rho\}$ for each $e \in E$. Call an edge e with $\mathbf{w}(e) \geq \rho$ a ρ-heavy edge.

Our main contribution is to show that, when allowing $(1+\varepsilon)$-approximation, we can use the weight truncation to reduce the minimum normalized free cut to minimum cut with $O(\text{polylog}(n))$ extra factors in the running time. Moreover, this reduction can be implemented efficiently in the dynamic setting. We present the ideas in two steps, addressing our two technical challenges mentioned above respectively. First, we show how to solve the static version of minimum normalized free cut in near-linear time. Second, we sketch the key ideas to implement them efficiently in the dynamic setting, which can be used in the MWU framework.

We remark that weight truncation technique has been used in different context. For instance, Zenklusen [47] used it for reducing the connectivity interdiction problem to $O(|E|)$ instances of the minimum budgeted cut problem.

2.1 Step 1: Static Algorithm

We show that the minimum normalized free cut problem can be solved efficiently in the static setting. For convenience, we often use the term cut to refer to a set of edges instead of a set of vertices.

Define the objective function of our problem as, for any cut C,

$$\mathsf{val}_{\mathbf{w}}(C) = \min_{F \subseteq C : |F| \leq k-1} \frac{\mathbf{w}(C \setminus F)}{k - |F|}.$$

For any weight function $\mathbf{w}$, denote by $\mathsf{OPT}_{\mathbf{w}} = \min_C \mathsf{val}_{\mathbf{w}}(C)$. In this paper, the graph G is always fixed, while $\mathbf{w}$ is updated dynamically by the algorithm (so we omit the dependence on G from the notation val and OPT). When $\mathbf{w}$ is clear from context, we sometimes omit the subscript $\mathbf{w}$.

We show that the truncation technique can be used to establish a connection between our problem and minimum cut.

▶ **Lemma 6.** *We are given a graph $G = (V, E)$, weight function $\boldsymbol{w}$, integer k, and $\varepsilon > 0$. For any threshold $\rho \in (\mathsf{OPT}_{\boldsymbol{w}}, (1+\varepsilon)\mathsf{OPT}_{\boldsymbol{w}}]$,*

- *any optimal normalized free cut in $(G, \boldsymbol{w})$ is a $(1+\varepsilon)$-approximate minimum cut in $(G, \boldsymbol{w}_\rho)$, and*
- *any minimum cut C^* in $(G, \boldsymbol{w}_\rho)$ is a $(1+\varepsilon)$-approximation for the minimum normalized free cut.*

Proof. First, consider any cut C with $\mathsf{val}(C) = \mathsf{OPT}$. Let $F \subseteq C$ be an optimal set of free edges for C, so we have $\mathbf{w}_\rho(C \setminus F) \leq \mathbf{w}(C \setminus F) = (k - |F|)\mathsf{OPT}$. Moreover, $\mathbf{w}_\rho(F) \leq |F|\rho$. This implies that

$$\mathbf{w}_\rho(C) = \mathbf{w}_\rho(C \setminus F) + \mathbf{w}_\rho(F) < k\rho \tag{2}$$

Next, we prove that any cut in $(G, \mathbf{w}_\rho)$ is of value at least $k\mathsf{OPT}$ (so the cut C is a $(1+\varepsilon)$ approximate minimum cut). Assume for contradiction that there is a cut C' such that $\mathbf{w}_\rho(C') < k\mathsf{OPT}$. Let $F' \subseteq C'$ be the set of ρ-heavy edges. Observe that $|F'| \leq k-1$ since otherwise the total weight $\mathbf{w}_\rho(C')$ would have already exceeded $k\mathsf{OPT}$. This implies that $\mathbf{w}(C' \setminus F') = \mathbf{w}_\rho(C' \setminus F') < (k - |F'|)\mathsf{OPT}$ and that

$$\mathsf{val}(C') \leq \frac{\mathbf{w}(C' \setminus F')}{(k - |F'|)} < \mathsf{OPT}$$

which is a contradiction. Altogether, we have proved the first part of the lemma.

To prove the second part of the lemma, consider a minimum cut C^* in $(G, \mathbf{w}_\rho)$, we have that $\mathbf{w}_\rho(C^*) < \mathbf{w}_\rho(C) < k\rho$ (from Equation (2)). Again, the set of heavy edges $F^* \subseteq C^*$ can contain at most $k-1$ edges, so we must have $\mathbf{w}(C^* \setminus F^*) < (k-|F^*|)\rho \leq (k-|F^*|)(1+\varepsilon)\mathsf{OPT}$, implying that $\mathsf{val}(C^*) < (1+\varepsilon)\mathsf{OPT}$. ◀

We remark that this reduction from the minimum normalized free cut problem to the minimum cut problem does not give an exact correspondence, in the sense that a minimum cut in $(G, \mathbf{w}_\rho)$ cannot be turned into a minimum normalized free cut in $(G, \mathbf{w})$. In other words, the approximation factor of $(1+\varepsilon)$ is unavoidable.

▶ **Theorem 7.** *Given a graph $G = (V, E)$ with weight function $\boldsymbol{w}$ and integer k, the minimum normalized free cut problem can be $(1+\varepsilon)$ approximated by using $O(\frac{1}{\varepsilon} \cdot \log n)$ calls to the exact minimum cut algorithm.*

Proof. We assume that the minimum normalized free cut of G is upper bounded by some value M which is polynomial in $n = |V(G)|$ (we show how to remove this assumption in the full version of the paper). For each i such that $(1+\varepsilon)^i \leq M$, we compute the minimum cut C_i in $(G, \mathbf{w}_{\rho_i})$ where $\rho_i = (1+\varepsilon)^i$ and return one with minimum value $\mathsf{val}(C_i)$. Notice that there must be some i^* such that $\rho_{i^*} \in (\mathsf{OPT}_{\mathbf{w}}, (1+\varepsilon)\mathsf{OPT}_{\mathbf{w}}]$ and by the lemma, we must have that C_{i^*} is a $(1+\varepsilon)$-approximate solution for the normalized free cut problem. ◀

By using any near-linear time minimum cut algorithm e.g., [27], the collorary follows.

▶ **Corollary 8.** *There exists a $(1+\varepsilon)$ approximation algorithm for the minimum normalized free cut problem that runs in time $\tilde{O}(|E|/\varepsilon)$.*

2.2 Step 2: Dynamic Algorithm

The next idea we use is from Chekuri and Quanrud [6]. One of the key concepts there is that it is sufficient to solve a "range punishing" problem in near-linear time; for completeness we prove this sufficiency in Appendix. In particular, the following proposition is a consequence of their work:

▶ **Definition 9.** *A **range punisher**[9] is an algorithm that, on any input graph G, initial weight function $\boldsymbol{w} = \boldsymbol{w}^{\text{init}}$, real numbers ε, and $\lambda \leq \mathsf{OPT}_{\boldsymbol{w}^{\text{init}}}$, iteratively applies* PunishMin *on $(G, \boldsymbol{w})$ until the optimal becomes at least $\mathsf{OPT}_{\boldsymbol{w}} \geq (1+\varepsilon)\lambda$.*

The following proposition connects a fast range punisher to a fast LP solver.

[9] Our range punisher corresponds to an algorithm of Chekuri and Quanrud [7] in one epoch.

▶ **Proposition 10.** *If there exists a range punisher running in time*

$$\widetilde{O}\left(|E| + K + \sum_{e\in E} \log(\frac{\boldsymbol{w}(e)}{\boldsymbol{w}^{\text{init}}(e)})\right)$$

where K is the number of cuts punished, then, there exists a fast dynamic punisher, and consequently the kECSS LP can be solved in near-linear time.

This proposition applies generally in the MWU framework independent of problems. That is, for our purpose of solving kECSS LP, we need a fast range punisher for the minimum normalized free cut problem. For Chekuri and Quanrud [6], they need such algorithm for the minimum cut problem (therefore a fast LP solver for the Held-Karp bound).

▶ **Theorem 11** ([6], informal)**.** *There exists a fast range punisher for the minimum cut problem.*

Our key technical tool in this paper is a more robust reduction from the range punishing of normalized free cuts to the one for minimum cuts. This reduction works for all edge weights and is suitable for the dynamic setting. That is, it is a strengthened version of Lemma 6 and is summarized below (see its proof in Section 4).

▶ **Theorem 12** (Range Mapping Theorem)**.** *Let $(G = (V, E), \boldsymbol{w})$ be a weighted graph. Let $\lambda > 0$ and $\rho = (1 + \gamma)\lambda$.*

1. *If the value of optimal normalized free cut is in $[\lambda, (1 + \gamma)\lambda)$, then the value of minimum cut in $(G, \boldsymbol{w}_\rho)$ lies in $[k\rho/(1 + \gamma), k\rho)$.*
2. *For any cut C where $\boldsymbol{w}_\rho(C) < k\rho$, then $\frac{\boldsymbol{w}(C\setminus F)}{k-|F|} < (1 + \gamma)\lambda$ where F contains all ρ-heavy edges in C. In particular, $\mathsf{val}(C) < (1 + \gamma)\lambda$.*

Given the above reduction, we can implement range punisher fast. We present its full proof in Section 5 and sketch the argument below.

▶ **Theorem 13.** *There exists a fast range punisher for the minimum normalized free cut problem.*

Proof (sketch). We are given λ and weighted graph $(G, \mathbf{w}) : \mathbf{w} = \mathbf{w}^{\text{init}}$ such that $\mathsf{OPT}_{\mathbf{w}^{\text{init}}} \geq \lambda$. Our goal is to punish the normalized free cuts until the optimal value in $(G, \mathbf{w})$ becomes at least $(1+\varepsilon)\lambda$. We first invoke Theorem 7 to get a $(1+\varepsilon)$-approximate cut, and if the solution is already greater than $(1 + \varepsilon)^2\lambda$, we are immediately done (this means $\mathsf{OPT} > (1 + \varepsilon)\lambda$).

Now, we know that $\mathsf{OPT} \leq (1 + \varepsilon)^2\lambda \leq (1 + 3\varepsilon)\lambda$. We invoke Lemma 12(1) with $\gamma = 3\varepsilon$. The minimum cut in $(G, \mathbf{w}_\rho)$ has size in the range $[k\rho/(1+3\varepsilon), k\rho)$. We invoke (one iteration of) Theorem 11 with $\lambda' = k\rho(1 + 3/\varepsilon)$ to obtain a cut C whose size is less than $k\rho$ and therefore, by Lemma 12(1), $\mathsf{val}(C) < (1 + 3\varepsilon)\lambda$. This is a cut that our algorithm can punish (we ignore the detail of how we actually punish it – we would need to do that implicitly since the cut itself may contain up to m edges). We repeat this process until all cuts whose values are relevant have been punished, that is, we continue this process until the returned cut C has size at least $k\rho$.

The running time of this algorithm is

$$\widetilde{O}\left(|E| + K + \sum_{e\in E} \log(\frac{\mathbf{w}_\rho(e)}{\mathbf{w}_\rho^{\text{init}}(e)})\right) \leq \widetilde{O}\left(|E| + K + \sum_{e\in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\text{init}}(e)})\right)$$

Notice that we rely crucially on the property of our reduction using truncated weights. ◀

We remark that in the actual proof of Theorem 13, there are quite a few technical complications (e.g., how to find optimal free edges for a returned cut C?), and we cannot invoke Theorem 11 in a blackbox manner. We refer to Section 5 for the details.

2.3 LP Rounding for kECSS

Most known techniques for kECSS (e.g. [22, 33]) rely on iterative LP rounding, which is computationally expensive. We achieve fast running time by making use of the 2-approximation algorithm of Khuller and Vishkin [31].

Roughly speaking, this algorithm creates a directed graph H from the original graph G and then compute on H the minimum-weight k disjoint arboresences. The latter can be found by Gabow's algorithms, in either $\tilde{O}(|E||V|k)$ or $\tilde{O}(k|E|\sqrt{|V|}\log c_{\max})$ time.

To use their algorithm, we will construct H based on the support of the fractional solution x computed by the LP solver. By the integrality of the arborescence polytope [41], an integral solution is as good as the fractional solution. However, the support of x can be potentially large, which causes Gabow's algorithm to take longer time. Here our idea is a sparsification of the support, by extending the celebrated sparsification theorem of Benzcur and Karger [2] to handle our problem, i.e., we prove the following (see Section 6 for the proofs):

▶ **Theorem 14.** *Let G be a graph and c_G its capacities. There exists a capacitated graph (H, c_H) on the same set of vertices that can be computed in $\tilde{O}(m)$ such that (i) $|E(H)| = \tilde{O}(nk)$, and (ii) for every cut S and $F \subseteq S : |F| \leq (k-1)$, we have $c_G(S \setminus F) = (1 \pm \varepsilon)\, c_H(S \setminus F)$.*

Benzcur and Karger's theorem corresponds to this theorem when $k = 1$. We believe that this theorem might have further applications, e.g., for providing a fast algorithm for the connectivity interdiction problem. Our result implies the following (see Section 6 for the proof):

▶ **Theorem 15.** *Assume that there exists an algorithm that finds a minimum-weight k-arborescences in an m-edge n-node graph in time $T_k(m, n)$. Then there exists a $(2+\varepsilon)$ approximation algorithm for kECSS running in time $\tilde{O}(m/\varepsilon^2 + T_k(kn/\varepsilon^2, n))$*

Applying Theorem 15 with the Gabow's algorithm (see Theorem 35 in Section 6), we obtain Corollary 2.

3 Preliminaries

In this section, we review the multiplicative-weight update (MWU) framework for solving a (covering) LP relaxation of the form $\min\{c \cdot x : Ax \geq 1, x \geq 0\}$, where A is an m-by-n matrix with non-negative entries and $c \in \mathbb{R}^n_{\geq 0}$. Our presentation abstracts away the detail of MWU, so readers should feel free to skip this section.

Let $A_1, \ldots, A_m$ be the rows of matrix A. Here is a concrete example:

- **Held-Karp Bound:** The Held-Karp bound on input (G, c) aims at solving the LP:[10]

$$\min\{ \sum_{e \in E(G)} c_e x_e : \sum_{e \in S} x_e \geq 2 \text{ for any cut } S \subseteq E\}$$

[10] We refer the readers to [6] for more discussion about this LP and Held-Karp bound.

Matrix $A = A_G$ is a cut-edge incidence matrix of graph G where each row A_j corresponds to a cut $F_j \subseteq E(G)$, so there are exponentially many rows. Each column corresponds to an edge $e \in E(G)$. There are exactly $|E(G)|$ columns. The matrix is implicitly given as an input graph G.

We explain the MWU framework in terms of matrices. Some readers may find it more illustrative to work with concrete problems in mind.

MWU Framework for Covering LPs

In the MWU framework for solving covering linear programs, we are given as input an m-by-n matrix A and cost vectors c associated with the columns.[11] Let $\varepsilon > 0$ be a parameter; that is, we aim at computing a solution x that is $(1+\varepsilon)$ approximation of the optimal LP solution. Denote by $\textsc{MinRow}(A, w)$ the value $\min_{j\in[m]} A_j w$. We start with an initial weight vector $\mathbf{w}_i^{(0)} = 1/c_i$ for $i \in [n]$. On each day $t = 1, \ldots, T$, we compute an approximately "cheapest" row j^* such that $A_{j^*}\mathbf{w}^{(t-1)} \leq (1+\varepsilon)\textsc{MinRow}(A, \mathbf{w}^{(t-1)})$, and update the weight $\mathbf{w}_i^{(t)} \leftarrow \mathbf{w}_i^{(t-1)} \exp\left(\frac{\varepsilon A_{j^*,i} c_{\min}}{c_i}\right)$ where $c_{\min} = \min_{i\in[n]} \frac{c_i}{A_{j^*,i}}$.[12] After $T = O(n \log n/\varepsilon^2)$ many days, the solution can be found by taking the best scaled vectors; in particular, observe that, for any day t, the scaled vector $\bar{\mathbf{w}}^{(t)} = \mathbf{w}^{(t)}/\left(\min_{j\in[m]} A_j \mathbf{w}^{(t)}\right)$ is always feasible for the LP. The algorithm returns $\bar{\mathbf{w}}^{(t)}$ which has minimum cost. The following theorem shows that at least one such solution is near-optimal.

▶ **Theorem 16.** *For $T = O(\frac{n \log n}{\varepsilon^2})$, one of the solutions $\bar{\boldsymbol{w}}^{(t)}$ for $t \in [T]$ is a $(1 + O(\varepsilon))$ approximation of the optimal solution* $\min\{c \cdot x : Ax \geq 1, x \geq 0\}$.

Since we use slightly different language than the existing proofs in the literature, we provide a proof in the appendix.

KC Inequalities

Our LP is hard to work with mainly because of the mixed packing/covering constraints $x \in [0,1]^n$. There is a relatively standard way to get rid of the mixed packing/covering constraints by adding Knapsack covering (KC) inequalities into the LP. In particular, for each row (or constraint) $j \in [m]$, we introduce constraints:

$$(\forall F \subseteq \mathsf{supp}(A_j), |F| \leq (k-1)) : \sum_{i\in[n]\setminus F} A_{j,i}x_i \geq k - |F|, \text{ or } \sum_{i\in[n]\setminus F} \frac{A_{j,i}}{(k-|F|)} x_i \geq 1$$

Let A^{kc} be the new matrix after adding KC inequalities, that is, imagine the row indices of A^{kc} as (j, F) where $j \in [m]$ and $F \subseteq \mathsf{supp}(A_j)$; we define $A^{\mathsf{kc}}_{(j,F),i} = A_{j,i}/(k - |F|)$. The actual number of rows in A^{kc} can be as high as $m \cdot n^{O(k)}$, but our algorithm will not be working with this matrix explicitly.

The following lemma shows that we can now remove the packing constraints. We defer the proof to Appendix.

▶ **Lemma 17.** *Any solution to $\{x \in \mathbb{R}^n : A^{kc}x \geq 1, x \geq 0\}$ is feasible for $\{x \in \mathbb{R}^n : Ax \geq k, x \in [0,1]\}$. Conversely, for any point z in the latter polytope, there exists a point z' in the former such that $z' \leq z$.*

[11] There are several ways to explain such a framework. Chekuri and Quanrud [6] follow the continuous setting of Young [46]. We instead follow the combinatorial interpretation of Garg and Könemann [23].
[12] In the MWU literature, this is often referred to as an oracle problem.

▶ **Corollary 18.** *For any cost vector* $c \in \mathbb{R}^n_{\geq 0}$,

$$\min\{c^T x : A^{kc} x \geq 1, x \geq 0\} = \min\{c^T x : Ax \geq k, x \in [0,1]\}$$

4 Range Mapping Theorem

The goal of this section is to prove Theorem 12, a cornerstone of this paper. We emphasize that it works for *any* weight function $\mathbf{w}$. First, we introduce more notations for convenience. For any cut $C \in \mathcal{C}$, and any subset of edges $F \subseteq E$, we define $\mathsf{val}_{\mathbf{w}}(C,F) = \frac{\mathbf{w}(C \setminus F)}{k-|F|}$ if $F \subseteq C$ and $|F| < k$; otherwise, $\mathsf{val}_{\mathbf{w}}(C,F) = \infty$. Also, denote $\mathsf{val}_{\mathbf{w}}(C) = \min_{F \subseteq E} \mathsf{val}_{\mathbf{w}}(C,F)$. By definition, we have $\mathsf{val}_{\mathbf{w}}(C) = \min_{i \leq k-1} \mathsf{val}_{\mathbf{w}}(C, F_i)$ where F_i is the set of heaviest i edges in C with respect to weight function $\mathbf{w}$. We let $\mathsf{mincut}_{\mathbf{w}_\rho}$ be the value of a minimum cut with respect with weight $\mathbf{w}_\rho$. When it is clear from context, we sometimes omit the subscript $\mathbf{w}$. For any positive number ρ, let $H_{\mathbf{w},\rho} = \{e \in E \colon \mathbf{w}(e) \geq \rho\}$ be the set of ρ-heavy edges.[13] Define the weight truncation $\mathbf{w}_\rho(e) = \min\{\mathbf{w}(e), \rho\}$.

▶ **Theorem 19** (Restatement of Theorem 12)**.** *We are given a weighted graph* $(G, \boldsymbol{w})$, $\lambda > 0$ *be a parameter and* $\rho = (1+\gamma)\lambda$. *Then we have the following:*

1. *If* $\mathsf{OPT}_{\boldsymbol{w}} \in [\lambda, (1+\gamma)\lambda)$, *then* $\mathsf{mincut}_{\boldsymbol{w}_\rho} \in [k\rho/(1+\gamma), k\rho)$, *and*
2. *if a cut* C *satisfies* $\boldsymbol{w}_\rho(C) < k\rho$, *then* $\mathsf{val}_{\boldsymbol{w}}(C, H_{\boldsymbol{w},\rho} \cap C) < (1+\gamma)\lambda$.

Notice that the above theorem not only gives a mapping between solutions of the two problems but also that the heavy edges can be used as a set of free edges. We say that a cut C is *interesting* if it contains at most $k-1$ heavy edges, i.e., $|H_{\mathbf{w},\rho} \cap C| < k$.

▶ **Proposition 20.** *If cut* $C \subseteq E$ *is not interesting (i.e.,* $|H_{\boldsymbol{w},\rho} \cap C| \geq k$*), then* $\mathsf{val}_{\boldsymbol{w}}(C) \geq \rho$ *and* $\boldsymbol{w}_\rho(C) \geq k\rho$.

Proof. The fact that $\mathbf{w}_\rho(C) \geq k\rho$ follows immediately from the definition of heavy edges. Let F_i be the set heaviest i edges in C with respect to $\mathbf{w}$. Since C contains at least k heavy edges, we have that for all $i < k$, $C \setminus F_i$ contains at least $k-i$ heavy edges. Therefore, we have $\mathsf{val}_{\mathbf{w}}(C) = \min_{i \leq k-1} \frac{\mathbf{w}(C \setminus F_i)}{k-i} \geq \min_{i \leq k-1} \frac{(k-i)\rho}{k-i} = \rho$. ◀

Proposition 20 says that if a cut is not interesting it must be expensive as a normalized free cut (i.e., high $\mathsf{val}_{\mathbf{w}}(C)$) and as a graph cut (i.e., high $w_\rho(C)$). We next give a characterization that relates $\mathsf{val}_{\mathbf{w}}$ and the sizes of the cuts for interesting cuts.

▶ **Lemma 21.** *Let* C *be an interesting cut. Then* $\mathsf{val}_{\boldsymbol{w}}(C) \leq \mathsf{val}_{\boldsymbol{w}}(C, H_{\boldsymbol{w},\rho} \cap C) < \rho$ *if and only if* $\boldsymbol{w}_\rho(C) < k\rho$.

Proof. ($\rightarrow$) By definition of $\mathbf{w}_\rho$, we have

$$\mathbf{w}_\rho(C) = \mathbf{w}(C \setminus (H_{\mathbf{w},\rho} \cap C)) + \rho |H_{\mathbf{w},\rho} \cap C|. \tag{3}$$

If $\mathsf{val}_{\mathbf{w}}(C, H_{\mathbf{w},\rho} \cap C) < \rho$, then $\mathbf{w}(C \setminus H_{\mathbf{w},\rho} \cap C) < \rho(k - |H_{\mathbf{w},\rho} \cap C|)$. By Equation (3), we have $\mathbf{w}_\rho(C) < k\rho$.

($\leftarrow$) Denote $F = H_{\mathbf{w},\rho} \cap C$. By definition of val, we have

$$\mathsf{val}_{\mathbf{w}}(C) \leq \mathsf{val}_{\mathbf{w}}(C,F) = \frac{\mathbf{w}(C \setminus F)}{k-|F|} \overset{(3)}{=} \frac{\mathbf{w}_\rho(C) - \rho|F|}{k-|F|} < \frac{k\rho - \rho|F|}{k-|F|} = \rho.$$ ◀

[13] When it is clear from the context, for brevity, we might say that e is a *heavy edge* instead of *ρ-heavy edge*.

Proof of Theorem 19. For the first part, we begin by proving that $\mathsf{mincut}_{\mathbf{w}_\rho} < k\rho$. Let C^* be a cut such that $\mathsf{val}_{\mathbf{w}}(C^*) = \mathsf{OPT}_{\mathbf{w}}$. By Proposition 20, C^* must be interesting. Since $\mathsf{val}_{\mathbf{w}}(C^*) = \mathsf{OPT}_{\mathbf{w}} < (1+\gamma)\lambda = \rho$, Lemma 21 implies that we have $\mathbf{w}_\rho(C^*) < k\rho$. Therefore, $\mathsf{mincut}_{\mathbf{w}_\rho} < k\rho$.

Next, we prove that $\mathsf{mincut}_{\mathbf{w}_\rho} \geq k\rho/(1+\gamma)$. Let C be a cut, and denote $F = H_{\mathbf{w},\rho} \cap C$. If C is not interesting, then Proposition 20 implies that $\mathbf{w}_\rho(C) \geq k\rho \geq k\rho/(1+\gamma)$. If C is interesting, by definition of $\mathbf{w}_\rho$, we have

$$\mathbf{w}_\rho(C) = \mathbf{w}(C \setminus F) + \rho|F| \geq \mathsf{OPT}_{\mathbf{w}}(k - |F|) + \frac{\rho}{1+\gamma}|F| \geq \frac{\rho k}{1+\gamma}.$$

The last inequality follows since by assumption $\mathsf{OPT}_{\mathbf{w}} \geq \rho/(1+\gamma)$.

For the second part of the theorem, as $\mathbf{w}_\rho(C) < k\rho$, Proposition 20 implies that C is interesting. By Lemma 21, $\mathsf{val}_{\mathbf{w}}(C, H_{\mathbf{w},\rho} \cap C) < \rho = (1+\gamma)\lambda$. ◀

5 Fast Approximate LP Solver

In this section, we construct the fast range punisher for the normalized free cut problem. Our algorithm cannot afford to maintain the actual MWU weights, so it will instead keep track of *lazy weights*. From now on, we will use $\mathbf{w}^{\text{mwu}}$ to denote the actual MWU weights and $\mathbf{w}$ the weights that our data structure maintains.

▶ **Theorem 22** (Fast Range Punisher)**.** *Given graph G initial weight function $\boldsymbol{w}^{\text{init}}$ and two real values $\lambda, \varepsilon > 0$ such that $\lambda \leq \mathsf{OPT}_{\boldsymbol{w}^{\text{init}}}$, there is a randomized algorithm that iteratively applies* PunishMin *until the optimal with respect to the final weight function $\boldsymbol{w}^{\text{mwu}}$ becomes at least $\mathsf{OPT}_{\boldsymbol{w}^{\text{mwu}}} \geq (1+\varepsilon)\lambda$, in time $\widetilde{O}(|E| + K + \frac{1}{\varepsilon}\sum_{e\in E}\log(\cdot\frac{\boldsymbol{w}^{\text{mwu}}(e)}{\boldsymbol{w}^{\text{init}}(e)}))$, where K is the number of cuts punished.*

The following theorem is almost standard: the fast range punisher, together with a fast algorithm for approximating OPT_w for any weight $\mathbf{w}$, implies a fast approximate LP solver (e.g., see [6, 18]). For completeness, we provide the proof in the Appendix.

▶ **Theorem 23** (Fast LP Solver)**.** *Given a fast range punisher as described in Theorem 22, and a near-linear time algorithm for approximating OPT_w for any weight function w, there is an algorithm that output $(1+O(\varepsilon))$-approximate solution to kECSS LP in $\tilde{O}(m/\varepsilon^2)$ time.*

Notice that the above theorem implies our main result, Theorem 1. The rest of this section is devoted to proving Theorem 22. Following the high-level idea of [6], our data structure has two main components:

- **Range cut-listing data structure**: This data structure maintains dynamic (truncated) weighted graph $(G, \mathbf{w}_\rho)$ and is able to find a (short description of) $(1+O(\varepsilon))$-approximate cut whenever one exists, that is, it returns a cut of size between λ and $(1+O(\varepsilon))\lambda$ for some parameter λ. Since our weight function $\mathbf{w}$ changes over time, the data structure also has an interface that allows such changes to be implemented. The data structure can be taken and used directly in a blackbox manner, thanks to [6].
- **Lazy weight data structures**: Notice that a fast range punisher can only afford the running time of $\widetilde{O}\left(\sum_e \log \frac{\mathbf{w}^{\text{mwu}}(e)}{\mathbf{w}^{\text{init}}(e)}\right)$ for updating weights, while in the MWU framework, some edges would have to be updated much more often. We follow the idea of [6] to maintain approximate (lazy) weights that do not get updated too often but are still sufficiently close to the real weights. We remark that $\mathbf{w}^{\text{mwu}}$ only depends on the sequence of cuts, that PunishMin actually punishes. This lazy weight data structure is responsible for maintaining $\mathbf{w}$ that satisfies the following invariant:

▶ **Invariant 24.** *We have* $(1-\varepsilon)\boldsymbol{w}^{\mathrm{mwu}} \leq \boldsymbol{w} \leq \boldsymbol{w}^{\mathrm{mwu}}$.
That is, we allow $\mathbf{w}$ to underestimate weights, but they cannot deviate more than by a factor of $(1-\varepsilon)$. In this way, our data structure only needs to update the weight implicitly and output necessary increments to the cut listing data structure whenever the invariant is violated.

In sum, our range punisher data structures deal with three weight functions $\mathbf{w}$ (lazy weights), $\mathbf{w}_\rho$ (truncated lazy weights, used by the range cut listing data structure) and $\mathbf{w}^{\mathrm{mwu}}$ (actual MWU weights, maintained implicitly).

The rest of this section is organized as follows. In Section 5.1–Section 5.3, we explain the components that will be used in our data structure, and in Section 5.4, we prove Theorem 22 using these components.

5.1 Compact representation of cuts

This part serves as a "communication language" for various components in our data structure. Since a cut can have up to $\Omega(m)$ edges, the data structure cannot afford to describe it explicitly. We will use a compact representation of cuts [6], which allows us to describe any $(1+\varepsilon)$-approximate solution in a given weighted graph using $\tilde{O}(1)$ bits; notice that, in the MWU framework, we only care about (punishing) near-optimal solutions, so it is sufficient for us that we are able to concisely describe such cuts.

Formally, we say that a family $\mathcal{F}$ of subsets of edges is ε-canonical for $(G, \mathbf{w})$ if (i) $|\mathcal{F}| \leq \widetilde{O}(|E|)$, (ii) any $(1+\varepsilon)$-approximate minimum cut of $(G, \mathbf{w})$ is a disjoint union of at most $\tilde{O}(1)$ sets in $\mathcal{F}$, (iii) any set $S \in \mathcal{F}$ can be described concisely by $\tilde{O}(1)$ bits, and (iv) every edge in the graph belongs to $\tilde{O}(1)$ sets in $\mathcal{F}$. It follows that any $(1+\varepsilon)$-approximate cut admits a short description. Denote by $[[S]]$ a short description of cut $S \in \mathcal{F}$, and for each $(1+\varepsilon)$ approximate cut C, $[[C]]$ a short description of C.

▶ **Lemma 25** (implicit in [6]). *There exists a randomized data structure that, on input* $(G, \boldsymbol{w})$, *can be initialized in near-linear time, (w.h.p) constructs an* ε*-canonical family* $\mathcal{F} \subseteq 2^{E(G)}$, *and handles the following queries:*

- *Given a description* $[[C]]$ *of a* $(1+\varepsilon)$*-approximate cut, output a list of* $\tilde{O}(1)$ *subsets in* $\mathcal{F}$ *such that* C *is a disjoint union of those subsets in* $\tilde{O}(1)$ *time.*
- *Given a description of* $[[S]]$, $S \in \mathcal{F}$, *output a list of edges in* S *in* $\widetilde{O}(|S|)$ *time.*

5.2 Range Cut-listing Data Structure

The cut listing data structure is encapsulated in the following theorem.

▶ **Theorem 26** (Range Cut-listing Data Structure [6]). *The cut-listing data structure, denoted by* $\mathcal{D}$, *maintains dynamically changing weighted graph* $(G, \widehat{\boldsymbol{w}})$ *and supports the following operations.*

- $\mathcal{D}.\textsc{Init}(G, \boldsymbol{w}^{\mathrm{init}}, \lambda, \varepsilon)$ *where* G *is a graph,* $\widehat{\boldsymbol{w}}$ *is an initial weight function, and* $\mathsf{mincut}_{\widehat{\boldsymbol{w}}} \geq \lambda$*: initialize the data structure and the weight* $\widehat{\boldsymbol{w}} \leftarrow \boldsymbol{w}^{\mathrm{init}}$ *in* $\tilde{O}(m)$ *time.*
- $\mathcal{D}.\textsc{FindCut}()$: *output either a short description of a* $(1+O(\varepsilon))$*-approximate mincut* $[[C]]$ *or* $\emptyset$ *(when* $\mathsf{mincut}_{\widehat{\boldsymbol{w}}} > (1+\varepsilon)\lambda$*). The operation takes amortized* $\tilde{O}(1)$ *time.*
- $\mathcal{D}.\textsc{Increment}(\Delta)$ *where* $\Delta = \{(e, \delta_e)\}$ *is the set of increments (defined by a pair of an edge* $e \in E$ *and a value* $\delta_e \in \mathbb{R}_{\geq 0}$*): For each* $(e, \delta_e) \in \Delta$, $\widehat{\boldsymbol{w}}(e) \leftarrow \widehat{\boldsymbol{w}}(e) + \delta_e$. *The operation takes* $\tilde{O}(|\Delta|)$ *time (note that* $|\Delta|$ *corresponds to the number of increments).*

As outlined earlier, the cut listing data structure will be invoked with $\widehat{\mathbf{w}} = \mathbf{w}_\rho$.

5.3 Truncated Lazy MWU Increment

The data structure is formally summarized by the definition below.

▶ **Definition 27** (Truncated Lazy MWU Increment). *A truncated lazy MWU increment denoted by $\mathcal{L}$ maintains the approximate weight function $\boldsymbol{w}$ explicitly, and exact weight $\boldsymbol{w}^{\text{mwu}}$ implicitly and supports the following operations:*[14]

- $\mathcal{L}.\textsc{Init}(G, \boldsymbol{w}^{\text{init}}, \rho)$ *where G is a graph, $\boldsymbol{w}^{\text{init}}$ is the initial weight function, $\rho \in \mathbb{R}_{>0}$: Intialize the data structure, and set $\boldsymbol{w} \leftarrow \boldsymbol{w}^{\text{init}}$.*
- $\mathcal{L}.\textsc{Punish}([[C]])$ *where C is a cut: Internally punish the free cut (C, F) for some F (to be made precise later) and output a list of increment $\Delta = \{(e, \delta_e)\}$ so that for each $e \in E$, $\boldsymbol{w}^{\text{init}}(e)$ plus the total increment over e is $\boldsymbol{w}_\rho(e)$.*
- $\mathcal{L}.\textsc{Flush}()$*: Return the exact weight $\boldsymbol{w}^{\text{mwu}}$.*

Remark that the output list of increments returned by Punish is mainly for the purpose of syncing with the cut listing data structure (so it aims at maintaining $\mathbf{w}_\rho$ instead of $\mathbf{w}$). Also, in the Punish operation, the data structure must compute the set $F \subseteq C$ of free edges efficiently (these are the edges whose weights would not be increased). This is one of the reasons for which we cannot use the lazy update data structure in [6] as a blackbox.

▶ **Theorem 28.** *There exists a lazy MWU increment with the following time complexity: (i) init operation takes $\tilde{O}(m)$ time, (ii) Punish takes $\tilde{O}(K) + \widetilde{O}\left(\sum_e \log \frac{\boldsymbol{w}^{\text{mwu}}(e)}{\boldsymbol{w}^{\text{init}}(e)}\right)$ time in total where K is the number of calls to Punish and outputs at most $\widetilde{O}\left(\sum_e \log \frac{\boldsymbol{w}^{\text{mwu}}(e)}{\boldsymbol{w}^{\text{init}}(e)}\right)$ increments, and (iii) flush takes $\tilde{O}(m)$ time. Moreover, the Invariant 24 is maintained throughout the execution.*

For space reason, we prove Theorem 28 in the full version of the paper.

5.4 A Fast Range Punisher for Normalized Free Cut Problem

Now we have all necessary ingredients to prove Theorem 22. The algorithm is very simple and described in Algorithm 1. We initialize the cut-listing data structure $\mathcal{D}$ so that it maintains the truncated weight $\mathbf{w}_\rho$ and the lazy weight data structure $\mathcal{L}$. We iteratively use $\mathcal{D}$ to find a cheap cut in $(G, \mathbf{w}_\rho)$ until no such cut exists. Due to our mapping theorem, such a cut found can be used for our problem, and the data structure $\mathcal{L}$ is responsible for punishing the weights (Line 8) and returns the list of edges to be updated (this is for the cut-listing $\mathcal{D}$ to maintain its weight function $\mathbf{w}_\rho$).

[14] This is implicit in the sense that w is divided into parts and they are internally stored in different memory segments. Whenever needed, the real weight can be constructed from the memory content in near-linear time.

Algorithm

Algorithm 1 FastRangePunisher$(G, \mathbf{w}, \lambda)$.

Input: $G, \mathbf{w}^{\text{init}}, \lambda, \varepsilon$ such that $\mathsf{OPT}_{\mathbf{w}^{\text{init}}} \geq \lambda$.
Output: a correct weight function $\mathbf{w} = \mathbf{w}^{\text{mwu}}$ such that $\mathsf{OPT}_{\mathbf{w}} \geq (1+\varepsilon)\lambda$.

1 $\mathbf{w} \leftarrow \mathbf{w}^{\text{init}}$ and $\rho \leftarrow (1+\varepsilon)\lambda$
2 Let $\mathbf{w}_\rho$ be the truncated weight function of $\mathbf{w}$.
3 **if** $\mathsf{mincut}_{\mathbf{w}_\rho} \geq k\rho$ **then return** $\boldsymbol{w}$.
4 Let $\mathcal{D}$ and $\mathcal{L}$ be cut listing data structure, and truncated lazy MWU increment.
5 $\mathcal{D}$.Init$(G, \mathbf{w}_\rho, k\rho/(1+\varepsilon), \varepsilon)$
6 $\mathcal{L}$.Init$(G, \mathbf{w}, \rho, \varepsilon)$
7 **while** $\mathcal{D}$.FindCut() returns $[[C]]$ **do**
8 $\quad \Delta \leftarrow \mathcal{L}$.Punish$([[C]])$
9 $\quad \mathcal{D}$.Increment(Δ)
10 $\mathbf{w} \leftarrow \mathcal{L}$.Flush()
11 **return** $\boldsymbol{w}$.

Analysis

By input assumption, we have $\mathsf{OPT}_w \geq \lambda$. If $\mathbf{w}$ is returned at line 3, then $\mathsf{mincut}_{\mathbf{w}_\rho} \geq k\rho$. By Theorem 19(1), $\mathsf{OPT}_{\mathbf{w}^{\text{mwu}}} \geq \mathsf{OPT}_{\mathbf{w}} \geq \rho = (1+\varepsilon)\lambda$, and we are done (since minimum cut can be computed in near-linear time). Now, we assume that $\mathbf{w}$ is returned at the last line. The following three claims imply Theorem 22.

▷ Claim 29. For every cut $[[C]]$ returned by the range cut listing data structure during the execution of Algorithm 1, we have that $(C, H_{\mathbf{w},\rho} \cap C)$ is a $(1+O(\varepsilon))$-approximation to $\mathsf{OPT}_{\mathbf{w}^{\text{mwu}}}$ at the time $[[C]]$ is returned.

We remark that it is important that our cut punished must be approximately optimal w.r.t. the actual MWU weight.

Proof. By definition of $\mathcal{L}$.Flush() operation, we always have that the exact weight function and approximate weight function are identical at the beginning of the loop. By definition of $\mathcal{L}$.Punish$([[C]])$, the total increment plus the initial weight at the beginning of the loop for every edge e is $\mathbf{w}_\rho(e)$ and Invariant 24 holds. Therefore, by definition of $\mathcal{D}$.Increment(Δ), the range cut-listing data structure maintains the weight function $\mathbf{w}_\rho$ internally. We now bound the approximation of each cut $[[C]]$ that $\mathcal{D}$.FindCut() returned. Let $F = H_{\mathbf{w},\rho} \cap C$. By definition of FindCut(), we have that $\mathbf{w}_\rho(C) < k\rho$. By Theorem 19(2), $\mathsf{val}_{\mathbf{w}}(C, F) < (1+\varepsilon)\lambda$. By Invariant 24, we have that $\mathsf{val}_{\mathbf{w}^{\text{mwu}}}(C, \tilde{F}) < (1+O(\varepsilon))\lambda$. Since $\mathsf{OPT}_{\mathbf{w}^{\text{mwu}}} \geq \mathsf{OPT}_{w^{\text{init}}} \geq \lambda$, we have (C, F) is a $(1+O(\varepsilon))$-approximation to OPT_w. ◁

▷ Claim 30. At the end of Algorithm 1, we have $\mathsf{OPT}_{\mathbf{w}^{\text{mwu}}} \geq (1+\varepsilon)\lambda$.

Proof. Consider the time when $\mathcal{D}$.FindCut() outputs $\emptyset$. The fact that this procedure terminates means that $\mathsf{mincut}_{\mathbf{w}_\rho} \geq k\rho$. Therefore, Theorem 19(1) implies that $\mathsf{OPT}_{\mathbf{w}} \geq (1+\varepsilon)\lambda$. Let (C^*, F^*) be an optimal normalized free cut with respect to $\mathbf{w}^{\text{mwu}}$. We have

$$\begin{aligned}\mathsf{OPT}_{\mathbf{w}^{\text{mwu}}} &= \mathsf{val}_{\mathbf{w}^{\text{mwu}}}(C^*, F^*) \\ &\geq \mathsf{val}_{\mathbf{w}}(C^*, F^*) \\ &\geq \mathsf{OPT}_{\mathbf{w}} \\ &\geq (1+\varepsilon)\lambda\end{aligned}$$

where the first inequality follows from Invariant 24. ◁

▷ Claim 31. Algorithm 1 terminates in $\tilde{O}(m + K + \frac{1}{\varepsilon} \cdot \sum_{e \in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\text{init}}(e)}))$ time where K is the number of PUNISH operations.

Proof. We first bound the running time due to truncated lazy MWU increment. By Theorem 28, the total running time due to $\mathcal{L}$ (i.e., $\mathcal{L}$.INIT, $\mathcal{L}$.PUNISH, $\mathcal{L}$.FLUSH) is $\tilde{O}(m + K + \frac{1}{\varepsilon} \cdot \sum_{e \in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\text{init}}(e)}))$ time where K is the number of PUNISH operations. We bound the running time due to cut-listing data structure. Observe that the number of cuts listed equals the number of calls of PUNISH operations, and the total number of edge increments in $\mathcal{D}$ is $\widetilde{O}\left(\frac{1}{\varepsilon} \cdot \sum_{e \in E} \log(\frac{\mathbf{w}(e)}{\mathbf{w}^{\text{init}}(e)})\right)$. By Theorem 26, the total running time due to $\mathcal{D}$ (i.e, $\mathcal{D}$.INIT, $\mathcal{D}$.FINDCUT(), $\mathcal{D}$.INCREMENT(Δ)) is as desired. ◁

6 LP Rounding for kECSS (Proof of Theorem 15)

In this section, we show how to round the LP solution x found by invoking Theorem 1. The main idea is use a sampling technique to sparsify the support of x. On the subgraph $G' \subseteq G$ based on this sparsified support, we apply the 2-approximation algorithm of Khuller and Vishkin [32] to obtain a $(2+\varepsilon)$-approximation solution.

Let G be a graph with capacities c (we omit capacities whenever it is clear from the context). Our algorithm performs the following steps.

Step 1: Sparsification

We will be dealing with the following LP relaxation for kECSS.

$$\min\{\sum_{e \in E(G)} c(e)x_e : \sum_{e \in C \setminus S} x_e \geq k - |S|, \quad \forall C \in \mathcal{C}\ \forall S \in \{F : |F| \leq k-1 \wedge F \subseteq C\}, x \geq 0\}$$

Denote by $\mathsf{LP}_{kECSS}(G)$ the optimal LP value on input G. We prove the following lemma in the full version of the paper that will allow us to sparsify our graph without changing the optimal fractional value by too much:

▶ **Lemma 32.** *Given an instance (G, c), and in $\tilde{O}(m/\varepsilon^2)$ time, we can compute a subgraph G' having at most $\widetilde{O}(nk/\varepsilon^2)$ edges such that $\mathsf{LP}_{kECSS}(G') = (1 \pm O(\varepsilon))\mathsf{LP}_{kECSS}(G)$.*

The first step is simply to apply this lemma to obtain G' from G.

Step 2: Reduction to k-arborescences

Next, we reduce the kECSS problem to the minimum-cost k-arborescence problem which, on capacitated directed graph (H, c_H), can be described as the following IP:

$$\min\{\sum_{e \in E(H)} c_H(e)z_e : \sum_{e \in \delta^+(C)} z_e \geq k \text{ for } C \in \mathcal{C}; z \in \{0,1\}^{E(H)}\}$$

where $\mathcal{C}$ is the set of all cuts C such that $\{r\} \subseteq C \subsetneq V(G)$. Denote by $\mathsf{OPT}_{ar}(H)$ and $\mathsf{LP}_{ar}(H)$ the optimal integral and fractional values[15] of the minimum-cost k-arborescence problem respectively. We use the following integrality of its polytope:

▶ **Theorem 33** ([41], Corollary 53.6a)**.** *The minimum-cost k-arborescence's polytope is integral, so we have that $\mathsf{OPT}_{ar}(H) = \mathsf{LP}_{ar}(H)$ for every capacitated input graph H.*

[15] The relaxation is simply $\min\{\sum_{e \in E(H)} c_H(e)z_e : \sum_{e \in \delta^+(C)} z_e \geq k \text{ for } C \in \mathcal{C}; z \in [0,1]^{E(H)}\}$.

For any undirected graph G, denote by $D[G]$ the directed graph obtained by creating, for each (undirected) edge uv in G, two edges $(u \to v)$ and $(v \to u)$ in $D[G]$ whose capacities are just $c(uv)$. We will use the following theorem by Khuller and Vishkin (slightly modified) that relates the optimal values of the two optimization problems.

▶ **Theorem 34.** *For any graph (H, c), the following properties hold:*

- $\mathsf{LP}_{ar}(D[H]) \leq 2\mathsf{LP}_{kECSS}(H)$, *and*
- *Any feasible solution for k-arborescences in $D[H]$ induces a feasible kECSS solution in H of at most the same cost.*

Note that Theorem 33 and the algorithm by Khuller and Vishkin imply that the integrality gap of the kECSS LP is at most 2. While this result is immediate, it seems to be a folklore. To the best of our knowledge, it was not explicitly stated anywhere in the literature. This integrality gap allows us to obtain the first part of Corollary 2. We defer the proof of Theorem 34 to the full version of the paper. Our final tool to obtain Theorem 15 (and the second part of Corollary 2) is Gabow's algorithm:

▶ **Theorem 35** ([20])**.** *Given a graph $G = (V, E, c)$ with positive cost function c, a fixed root $r \in V$, and let $c_{\max}$ be the maximum cost on edges, there exists an algorithm that in $\tilde{O}(km\sqrt{n}\log(nc_{\max}))$ time outputs the integral minimum-cost k-arborescence.*

Algorithm of Theorem 15

Now, using the graph G' created in the first step, we create $D[G']$, and invoke Gabow's algorithm to compute an optimal k-arborescence in $D[G']$. Let $S \subseteq E(G)$ be the induced kECSS solution.

The cost of S is at most:

$$\begin{aligned}\mathsf{OPT}_{ar}(D[G']) &\leq \mathsf{LP}_{ar}(D[G']) \\ &\leq 2\mathsf{LP}_{kECSS}(G') \\ &\leq 2(1+O(\varepsilon))\mathsf{LP}_{kECSS}(G) \\ &\leq 2(1+O(\varepsilon))\mathsf{OPT}_{kECSS}(G)\end{aligned}$$

The first inequality is due to Theorem 33. The second one is due to Theorem 34 (first bullet). The third one is due to Lemma 32.

Analysis

Step 1 takes $\tilde{O}(m/\varepsilon^2)$ time, by Lemma 32. As the sparsified G' has $m' = \tilde{O}(\frac{nk}{\varepsilon^2})$ edges, for Step 2, by Theorem 35, we can compute the arborescence in $O(km'\sqrt{n}\log(nc_{\max})) = \tilde{O}(\frac{k^2 n^{1.5}}{\varepsilon^2}\log c_{\max})$ time. We show in the full version how to remove the term $\log c_{\max}$ in our case. In summary, the total running time is $\tilde{O}\left(\frac{m}{\varepsilon^2} + \frac{k^2 n^{1.5}}{\varepsilon^2}\right)$. Notice that the running time can be $\tilde{O}(\frac{m}{\varepsilon^2} + T_k(kn/\varepsilon^2, n))$ if we let the running time of Theorem 35 be $T_k(m, n)$, this complete the proof for Theorem 15.

References

1 David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. *ACM Transactions on Algorithms (TALG)*, 15(2):1–26, 2018.

2 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015.

3 André Berger and Michelangelo Grigni. Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 90–101. Springer, 2007.

4 Glencora Borradaile, Erik D Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.

5 Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2018.

6 Chandra Chekuri and Kent Quanrud. Approximating the held-karp bound for metric TSP in nearly-linear time. *CoRR*, abs/1702.04307, 2017. `arXiv:1702.04307`.

7 Chandra Chekuri and Kent Quanrud. Fast approximations for metric-tsp via linear programming. *CoRR*, abs/1802.01242, 2018. `arXiv:1802.01242`.

8 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019. `arXiv:1910.08025`.

9 Béla Csaba, Marek Karpinski, and Piotr Krysta. Approximability of dense and sparse instances of minimum 2-connectivity, tsp and path problems. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 74–83. Society for Industrial and Applied Mathematics, 2002.

10 Artur Czumaj, Michelangelo Grigni, Papa Sissokho, and Hairong Zhao. Approximation schemes for minimum 2-edge-connected and biconnected subgraphs in planar graphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2004.

11 Artur Czumaj and Andrzej Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 281–290. Citeseer, 1999.

12 Artur Czumaj and Andrzej Lingas. Fast approximation schemes for euclidean multi-connectivity problems. In *International Colloquium on Automata, Languages, and Programming*, pages 856–868. Springer, 2000.

13 Artur Czumaj and Andrzej Lingas. Approximation schemes for minimum-cost k-connectivity problems in geometric graphs. In *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.

14 Andreas Emil Feldmann, Jochen Könemann, Kanstantsin Pashkovich, and Laura Sanità. Fast approximation algorithms for the generalized survivable network design problem. In *ISAAC*, volume 64 of *LIPIcs*, pages 33:1–33:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

15 Cristina G Fernandes. A better approximation ratio for the minimum sizek-edge-connected spanning subgraph problem. *Journal of Algorithms*, 28(1):105–124, 1998.

16 Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via chvátal-gomory cuts. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 817–831. SIAM, 2018.

17 Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

18 Lisa Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1001–1010, 2004.

19 Greg N. Frederickson and Joseph JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981.

20 Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.

21 Harold N Gabow, Michel X Goemans, Éva Tardos, and David P Williamson. Approximating the smallest k-edge connected spanning subgraph by lp-rounding. *Networks: An International Journal*, 53(4):345–357, 2009.

22 Harold N. Gabow, Michel X. Goemans, and David P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Math. Program.*, 82:13–40, 1998. announced at IPCO'93.

23 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007.

24 Michel X. Goemans, Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved approximation algorithms for network design problems. In *SODA*, pages 223–232. ACM/SIAM, 1994.

25 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 632–645, 2018.

26 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/s004930170004.

27 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000. announced at STOC'96.

28 David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.

29 Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. On a cut-matching game for the sparsest cut problem. *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 2007.

30 Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4):19:1–19:15, 2009. doi:10.1145/1538902.1538903.

31 Samir Khuller and Ramakrishna Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.

32 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994. announced at STOC'92.

33 Bundit Laekhanukit, Shayan Oveis Gharan, and Mohit Singh. A rounding by sampling approach to the minimum size k-arc connected subgraph problem. In *International Colloquium on Automata, Languages, and Programming*, pages 606–616. Springer, 2012.

34 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *STOC*, pages 317–329. ACM, 2021.

35 Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *FOCS*, pages 245–254. IEEE Computer Society, 2010.

36 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 121–130, 2010. doi:10.1145/1806689.1806708.

37 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o (n1/2-ε)-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1122–1129, 2017.

38 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, pages 950–961. IEEE Computer Society, 2017.

39 David Pritchard. k-edge-connectivity: Approximation and LP relaxation. In *WAOA*, volume 6534 of *Lecture Notes in Computer Science*, pages 225–236. Springer, 2010.

40 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *SODA*, pages 2616–2635. SIAM, 2019.

41 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

42 Jonah Sherman. Breaking the multicommodity flow barrier for o(vlog n)-approximations to sparsest cut. In *FOCS*, pages 363–372. IEEE Computer Society, 2009.

43 Jonah Sherman. Nearly maximum flows in nearly linear time. In *FOCS*, pages 263–269. IEEE Computer Society, 2013.

44 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *FOCS*, pages 919–930. IEEE, 2020.

45 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143, 2017. doi:10.1145/3055399.3055415.

46 Neal E. Young. Nearly linear-time approximation schemes for mixed packing/covering and facility-location linear programs. *CoRR*, abs/1407.3015, 2014. arXiv:1407.3015.

47 Rico Zenklusen. Connectivity interdiction. *Oper. Res. Lett.*, 42(6-7):450–454, 2014.

Polylogarithmic Sketches for Clustering

Moses Charikar
Stanford University, CA, USA

Erik Waingarten
Stanford University, CA, USA

Abstract

Given n points in ℓ_p^d, we consider the problem of partitioning points into k clusters with associated centers. The cost of a clustering is the sum of p^{th} powers of distances of points to their cluster centers. For $p \in [1, 2]$, we design sketches of size $\text{poly}(\log(nd), k, 1/\epsilon)$ such that the cost of the optimal clustering can be estimated to within factor $1 + \epsilon$, despite the fact that the compressed representation does not contain enough information to recover the cluster centers or the partition into clusters. This leads to a streaming algorithm for estimating the clustering cost with space $\text{poly}(\log(nd), k, 1/\epsilon)$. We also obtain a distributed memory algorithm, where the n points are arbitrarily partitioned amongst m machines, each of which sends information to a central party who then computes an approximation of the clustering cost. Prior to this work, no such streaming or distributed-memory algorithm was known with sublinear dependence on d for $p \in [1, 2)$.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases sketching, clustering

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.38

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2204.12358`

Funding *Moses Charikar*: Supported by a Simons Investigator Award.
Erik Waingarten: Supported by the National Science Foundation under Award no. 2002201.

1 Introduction

Given a large number of high-dimensional points, is it possible to compress the raw representation into a very compact sketch so that we can understand how clusterable the data is from just this highly compressed representation? Given n points in d dimensions, we consider the problem of approximating the cost of clustering them into k clusters from a compressed representation whose size is polylogarithmic in both n and d.

For $n, d \in \mathbb{N}$, let $P = \{x_1, \ldots, x_n\} \in \mathbb{R}^d$ be any set of points with polynomially bounded entries (i.e., all coordinates may be represented with $O(\log(nd))$ bits). The (k, z)-clustering problem in ℓ_p^d, asks to partition P into at most k clusters $C_1, \ldots, C_k$ so as to minimize

$$\sum_{\ell=1}^{k} \min_{c_\ell \in \mathbb{R}^d} \sum_{x \in C_\ell} \|x - c_\ell\|_p^z. \tag{1}$$

The problem is a generalization of the k-means and k-median problem; in particular, in Euclidean space ($p = 2$), $z = 2$ corresponds to k-means, and $z = 1$ to k-median.

We note that the raw representation of the dataset uses $O(nd \log(nd))$ bits, and that any algorithm which outputs optimal cluster centers $c_1, \ldots, c_k \in \mathbb{R}^d$, or the optimal clustering $C_1, \ldots, C_k$ must utilize $\Omega(kd)$, or $\Omega(n \log k)$ bits of space, respectively. Hence, such algorithms cannot decrease the dependency on both n and d simultaneously. However, this does not rule out an exponential compression, from $O(nd \log(nd))$ bits to $\text{polylog}(nd)$ bits (for constant k and z), for algorithms which approximate the optimal clustering *cost*, which only needs $O(\log(nd))$ bits. In this work, we show that it is indeed possible to design sketches of size

poly$(\log(nd), 1/\epsilon)$ bits which ϵ-approximate the optimal clustering *cost*, despite the fact that we do not have enough information to compute the clusters nor the cluster centers which achieve such cost.

Our results fit into a line of prior work on approximating the cost of optimization problems without necessarily computing an optimal solution. These have been investigated before for various problems and in various contexts, including estimating minimum spanning tree [40, 31, 33], minimum cost matchings and Earth Mover's Distance [40, 5, 3, 13], minimum vertex cover and maximum matching [59, 56, 62, 57, 26, 47, 15] and model-fit [49, 48, 17]. Specifically for clustering problems, the value of the clustering cost is an important statistic; used, for example, in the "elbow method" for determining the number of clusters needed. We will show these sketches may be efficiently maintained on a stream as well as for distributed-memory models, implying polylog(nd)-bit algorithms for these models of computation.

We start by reviewing a set of techniques in the literature to either reduce the dependence on the data set size or the dependence on the dimension.

Coresets

The coreset technique is a "dataset compression" mechanism, aiming to reduce the dependency on n. From the n points $P \subset \mathbb{R}^d$, an algorithm computes a much smaller (weighted) set of points $S \subset \mathbb{R}^d$, $w\colon S \to \mathbb{R}_{\geq 0}$, such that the cost of clustering the weighted points S, w approximates that of P. Following a long line of work [11, 36, 1, 24, 52, 34, 35, 20, 61, 39, 29], the best coreset constructions for (k, z)-clustering in ℓ_p achieve sizes $\tilde{O}(k/\epsilon^4) \cdot \min\{1/\epsilon^{z-2}, k\}$ for a $(1 \pm \epsilon)$-approximation. The ensuing streaming and distributed-memory algorithms maintain a coreset of the input; these algorithms find (approximately) optimal centers $c_1, \ldots, c_k \in \mathbb{R}^d$ and use space complexity $d \cdot \tilde{O}(k/\epsilon^4) \cdot \min\{1/\epsilon^{z-2}, k\} \cdot \text{polylog}(n)$.[1]

Dimension Reduction and Sketching

In addition to constructing coresets, an algorithm may seek to optimize the dependence on d. There is a large body of work studying (oblivious) dimensionality reduction and sketching, where strong compression results are known for computing distances [2, 51, 60, 12, 23, 30, 42, 41, 44, 46, 7, 8, 18]. For example, for $p \in [1, 2]$ there exists a (randomized) sketch $\mathbf{sk}\colon \mathbb{R}^d \to \mathbb{R}^t$ with t much smaller than d such that, for any two vectors $x, y \in \mathbb{R}^d$, an algorithm can approximate $\|x - y\|_p$ from $\mathbf{sk}(x)$ and $\mathbf{sk}(y)$ with high probability. While these results are encouraging, leveraging such sketches for distance computation in order to compress entire optimization problems like (1) is highly nontrivial. The challenge is that (1) implicitly considers distances among infinitely many vectors, and we need to rule out the possibility of spurious low cost solutions in the "sketched" space which do not have an analog in the original space. In particular, prior to this work, no streaming or distributed-memory algorithm was known which could reduce the dependence on d for $p \in [1, 2)$.

There is one setting, of Euclidean space ($p = 2$), where one can sketch vectors while preserving (1). A sequence of works [19, 27, 14, 55] show that applying a Johnson-Lindenstrauss [45] map $\mathbf{\Pi}\colon \mathbb{R}^d \to \mathbb{R}^t$ with $t = O(z^4 \log(k/\epsilon)/\epsilon^2)$, sketches Euclidean vectors to $O(t \log(nd))$ bits and preserves (1) up to $1 \pm \epsilon$. We emphasize that Euclidean space $p = 2$ is special in this regard, because the Johnson-Lindenstrauss map achieves *dimension reduction*, a property known not to hold in ℓ_1 [22, 54, 4]. In particular, d-dimensional vectors $x \in \mathbb{R}^d$ in Euclidean space are sketched to vectors $\mathbf{\Pi}(x) \in \mathbb{R}^t$ in Euclidean space, i.e., one estimates

[1] The $\log n$-factors arise from utilizing the "merge-and-reduce" framework for maintaining coresets on a stream [16, 1], and the fact the coreset constructions are randomized.

$\|x-y\|_2$ by $\|\mathbf{\Pi}(x)-\mathbf{\Pi}(y)\|_2$. Thus the optimization problem (1) for d-dimensional Euclidean space reduces to the same optimization problem for a much smaller dimensional Euclidean space. This can therefore be composed with known coreset constructions. Importantly, the "sketched" space inherits all geometric properties of Euclidean spaces, a key aspect of prior works, and the reason they do not extend beyond Euclidean space. The technical challenge in applying sketches for ℓ_p when $p \neq 2$ is that the "sketched" space is non-geometric.[2]

1.1 Our results

We give a streaming and distributed-memory algorithm for (k,p)-clustering in ℓ_p with space complexity $\mathrm{poly}(\log(nd), k, 1/\epsilon)$ bits.

▶ **Theorem 1** (Streaming (k,p)-Clustering in ℓ_p). *For $p \in [1,2]$, there exists an insertion-only streaming algorithm which processes a set of n points $x_1,\ldots,x_n \in \mathbb{R}^d$ utilizing $\mathrm{poly}(\log(nd), k, 1/\epsilon)$ bits which outputs a parameter $\boldsymbol{\eta} \in \mathbb{R}$ satisfying*

$$(1-\epsilon)\min_{\substack{C_1,\ldots,C_k\\ \text{partition } [n]}}\sum_{\ell=1}^{k}\min_{c_\ell\in\mathbb{R}^d}\sum_{i\in C_\ell}\|x_i-c_\ell\|_p^p \le \boldsymbol{\eta} \le (1+\epsilon)\min_{\substack{C_1,\ldots,C_k\\ \text{partition } [n]}}\sum_{\ell=1}^{k}\min_{c_\ell\in\mathbb{R}^d}\sum_{i\in C_\ell}\|x_i-c_\ell\|_p^p$$

with probability at least 0.9.

▶ **Theorem 2** (Distributed-Memory (k,p)-Clustering in ℓ_p). *For $p \in [1,2]$, there exists a public-coin protocol where m machines receive an arbitrary partition of n points $x_1,\ldots,x_n \in \mathbb{R}^d$, each communicates $\mathrm{poly}(\log(md), k, 1/\epsilon)$ bits to a central authority who outputs a parameter $\boldsymbol{\eta} \in \mathbb{R}$ satisfying*

$$(1-\epsilon)\min_{\substack{C_1,\ldots,C_k\\ \text{partition } [n]}}\sum_{\ell=1}^{k}\min_{c_\ell\in\mathbb{R}^d}\sum_{i\in C_\ell}\|x_i-c_\ell\|_p^p \le \boldsymbol{\eta} \le (1+\epsilon)\min_{\substack{C_1,\ldots,C_k\\ \text{partition } [n]}}\sum_{\ell=1}^{k}\min_{c_\ell\in\mathbb{R}^d}\sum_{i\in C_\ell}\|x_i-c_\ell\|_p^p$$

with probability at least 0.9.

Both algorithms will follow from applying a coreset and compressing the representation of the coreset points into sketches to recover single-cluster cost. Specifically, the bottleneck for our algorithm will be estimating the cost of (k,p)-clustering in ℓ_p for $k=1$. We give a linear sketch such that given a set of points $x_1,\ldots,x_n \in \mathbb{R}^d$, one may approximate the ℓ_p^p-median cost:

$$\min_{y\in\mathbb{R}^d}\sum_{i=1}^{n}\|x_i-y\|_p^p.$$

Most of the technical work will be devoted to sketching this "ℓ_p^p-median cost" objective. Then, the streaming and distributed-memory algorithm will evaluate the sum of ℓ_p^p-median costs for all possible partitions of the coreset points into k parts. The following theorem gives a linear sketch for approximating the ℓ_p^p-median cost.[3]

[2] For example, the sketched space for ℓ_p with $p \neq 2$ does not satisfy the triangle inequality: it is not the case that for any $x,y,z \in \mathbb{R}^d$, the estimate of $(\mathbf{sk}(x),\mathbf{sk}(y))$ plus the estimate of $(\mathbf{sk}(y),\mathbf{sk}(z))$ is less than the estimate of $(\mathbf{sk}(x),\mathbf{sk}(z))$. On the other hand, for $p=2$, the estimates of $(\mathbf{sk}(x),\mathbf{sk}(y))$, $(\mathbf{sk}(y),\mathbf{sk}(z))$, and $(\mathbf{sk}(x),\mathbf{sk}(z))$ are $\|\mathbf{sk}(x)-\mathbf{sk}(y)\|_2$, $\|\mathbf{sk}(y)-\mathbf{sk}(z)\|_2$, and $\|\mathbf{sk}(x)-\mathbf{sk}(z)\|_2$, so the triangle inequality does hold in the sketched space.

[3] A related although different work is that of approximating the ℓ_p^p-median (for instance, see Appendix F of [10]). An ℓ_p^p-median is a vector in $\mathbb{R}^d$ which means the sketch outputs d numbers; however, we will sketch the ℓ_p^p-median cost, which is a real number. Hence, our sketch will use $\mathrm{poly}(\log(nd), 1/\epsilon)$ space, as opposed to $\Omega(d)$ space needed to describe an ℓ_p^p-median.

▶ **Theorem 3** (ℓ_p^p-Median Sketch)**.** *For $p \in [1,2]$, there exists a linear sketch which processes a set of n points $x_1, \ldots, x_n \in \mathbb{R}^d$ into a vector $\mathbb{R}^t$ with $t = \text{poly}(\log(nd), 1/\epsilon)$ and outputs a parameter $\boldsymbol{\eta} \in \mathbb{R}$ satsifying*

$$(1-\epsilon) \min_{y \in \mathbb{R}^d} \sum_{i=1}^{n} \|x_i - y\|_p^p \leq \boldsymbol{\eta} \leq (1+\epsilon) \min_{y \in \mathbb{R}^d} \sum_{i=1}^{n} \|x_i - y\|_p^p$$

with probability at least 0.9*.*

There are a few important remarks to make:

- The requirement that $p \leq 2$ is necessary for the exponential compression we desire. For $p > 2$, there are strong lower bounds for sketching distances which show that such sketches require $\Omega(d^{1-2/p})$ space [12]. For $p < 1$, we are not aware of small coresets.
- The focus of this work is on optimizing the space complexity of the sketch, and while we do not explicitly specify the running time of the sketching and streaming algorithms, a naive implementation runs in time $(k \log(nd)/\epsilon)^{(k \log n/\epsilon)^{O(1)}}$. The exponential factor is due to the fact that we evaluate the cost of all possible partitions of the $(k \log(n)/\epsilon)^{O(1)}$-coreset points into k clusters. One could alleviate the exponential dependence to $(k/\epsilon)^{O(1)}$ (as opposed to $(k \log n/\epsilon)^{O(1)}$) by running more sophisticated approximation algorithms [11, 50] on the sketched representation of the coreset.[4] We note that a super-polynomial dependence on k should is unavoidable, because $(1 \pm \epsilon)$-approximations for (k, z)-clustering problems, for *non-constant* k, are NP-hard [9, 53, 28].
- It would be interesting to generalize Theorem 1 to dynamic streams. The reason our algorithm works in the insertion-only model is that we utilize the coreset of [39] with the merge-and-reduce framework [16, 1] which do not support deletions. While there exist dynamic coreset constructions for the streaming model [21, 38], our use of coresets is not entirely black-box. Other dynamic coresets, like [37], focus on update time and do not optimize the space complexity. We must ensure that the algorithm for constructing the coreset does not utilize the d-dimensional representation of the dataset points. The coreset construction of [39] only consider distances between the dataset points, so it suffices for us to only maintain a sketch of the dataset points.
- The fact that $z = p$ in our theorems above is a consequence of our techniques. It is unclear to us whether this assumption is necessary, although our approach hinges on the fact ℓ_p^p is additive over the d coordinates. We leave this as a problem for future work.

A similar, yet importantly different notion of (k, z)-clustering considers *medoid* cost, where the centers of the k clusters $c_1, \ldots, c_k$ are restricted to be dataset points. While seemingly similar to the (k, z)-clustering objective where centers are unrestricted, these two are qualitatively very different from a sketching perspective. In the full version, we show that while a two-pass sketching algorithm may ϵ-approximate the medoid cost, ϵ-approximations for one-pass sketching algorithm require polynomial space.

1.2 Technical Overview

We give an overview of Theorem 3. Once that is established, combining the ℓ_p^p-median sketch with coresets, thereby establishing Theorems 1 and 2 is (relatively) straight-forward (for more details, we refer the reader to the full version). Recall that for $p \in [1,2]$, we will process n points $x_1, \ldots, x_n \in \mathbb{R}^d$, and aim to return an approximation to the ℓ_p^p-median cost:

[4] The one subtlety is that the algorithm should be implemented without explicitly considering the d-dimensional representation of the points. Instead, it should only use the sketches of Theorem 3.

$$\min_{c\in\mathbb{R}^d}\sum_{i=1}^{n}\|x_i-c\|_p^p. \tag{2}$$

It will be useful to assume that the points are centered, i.e., $\sum_{i=1}^n x_i = 0 \in \mathbb{R}^d$ (we can enforce this because our sketches will be linear). The approach will come from the fact that the above optimization problem decomposes into a sum of d independent optimizations, one for each coordinate, and (2) seeks to evaluate the sum. Specifically, we may write

$$\min_{c\in\mathbb{R}^d}\sum_{i=1}^{n}\|x_i-c\|_p^p = \sum_{j=1}^{d}\min_{c_j\in\mathbb{R}}\sum_{i=1}^{n}|x_{ij}-c_j|^p.$$

Furthermore, for any fixed $j\in[d]$, estimating

$$\min_{c_j\in\mathbb{R}}\sum_{i=1}^{n}|x_{ij}-c_j|^p \tag{3}$$

is much more amenable to ℓ_p-sketching. Specifically, we let $x_{\cdot,j}\in\mathbb{R}^n$ be the vector containing the j-th coordinates of all n points, and $\mathbf{1}\in\mathbb{R}^n$ be the all-1's vector. Then, the quantity $\sum_{i=1}^n |x_{ij}-c_j|^p = \|x_{\cdot,j}-c_j\mathbf{1}\|_p^p$, and since the ℓ_p-sketches are linear, an algorithm may maintain $\mathbf{sk}(x_{\cdot,j})\in\mathbb{R}^t$ (for $t=\mathrm{poly}(\log(nd))$) and after processing, could iterate through various values of $c_j\in\mathbb{R}$ to evaluate

$$\left(\sum_{i=1}^{n}|x_{ij}-c_j|^p\right)^{1/p} = \|x_{\cdot,j}-c_j\mathbf{1}\|_p \approx_{1\pm\epsilon} \text{ estimate of } (\mathbf{sk}(x_{\cdot,j}),\mathbf{sk}(c_j\mathbf{1})),$$

and output the smallest value of $c_j\in\mathbb{R}$ found. In order to guarantee an $(1\pm\epsilon)$-approximation of (3), only $\mathrm{poly}(1/\epsilon)$ values of c_j need to be tried (since first evaluating $c_j=0$ will specify the range where the optimal c_j may lie). A simple union bound implies that for any fixed $j\in[d]$, we can prepare a small sketch $\mathbf{sk}(x_{\cdot,j})$ from which we can approximate (3).

In summary, we want to estimate the sum of d minimization problems. Even though each of the d problems could be solved independently with a linear sketch, we do not want to process d linear sketches (as this increases space). In addition, we do not know which of the d minimizations will significantly affect the sum; hence, if we only (uniformly) sampled few $\boldsymbol{j}_1,\ldots,\boldsymbol{j}_t\sim[d]$ and only processed $t\ll d$ sketches along the sampled dimensions, the variance of the estimator may be too large, making it completely useless. The technique we will use was recently developed in [25], building on [6, 43], under the name "ℓ_p-sampling with meta-data." In this paper, we further develop the ideas, and apply them to sketches for clustering in a simple and modular way. We refer the reader to Remark 4 (following this technical overview), where we expand on the comparison to [25].

The goal is to approximate the sum of the d minimization problems by *importance sampling* (see Chapter 9 of [58]). While importance sampling is a well-known technique, it's use in (one-pass) linear sketching algorithms is counter-intuitive, and we are not aware of any linear sketches which use importance sampling in the literature, expect for this and the recent work of [25, 32]. Importance sampling will aim to estimate (2) by sampling with respect to an alternate distribution $\mathcal{D}$. In particular, (2) may be re-written as

$$d\cdot\mathop{\mathbf{E}}_{\boldsymbol{j}\sim[d]}\left[\min_{c_{\boldsymbol{j}}\in\mathbb{R}}\sum_{i=1}^{n}|x_{i\boldsymbol{j}}-c_{\boldsymbol{j}}|^p\right] = d\mathop{\mathbf{E}}_{\boldsymbol{j}\sim\mathcal{D}}[\mathbf{Y}_{\boldsymbol{j}}] \qquad \text{where}$$

$$\mathbf{Y}_{\boldsymbol{j}} \stackrel{\text{def}}{=} \min_{c_{\boldsymbol{j}}\in\mathbb{R}}\sum_{i=1}^{n}|x_{i\boldsymbol{j}}-c_{\boldsymbol{j}}|^p\cdot\frac{1}{\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}]}, \tag{4}$$

where $\mathcal{D}$ is a distribution chosen so the variance of the random variable $\mathbf{Y}_{\boldsymbol{j}}$ for $\boldsymbol{j} \sim \mathcal{D}$ is bounded. Once the variance of the random variable is bounded, only a few samples are needed to estimate its expectation in (4). In general, the alternate distribution $\mathcal{D}$ depends on the data in order to decrease the variance; for instance, coordinates $j \in [d]$ whose value of (3) is higher should be sampled more often. Hence, importance sampling inherently interacts with the data in a two-stage process: 1) first, it samples $\boldsymbol{j} \sim \mathcal{D}$ (where the distribution is data-dependent), and 2) second, it evaluates $\mathbf{Y}_{\boldsymbol{j}}$ by using (3) and $\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}]$ for the value $\boldsymbol{j} \in [d]$ specified in the first step.

In a two-pass algorithm, the two steps may be implemented sequentially. A *sampling* sketch, like that of [43], is used to sample $\boldsymbol{j} \sim \mathcal{D}$ in the first pass. In the second pass, the algorithm knows the value of the sampled $\boldsymbol{j}$, so it maintains a sketch $\mathbf{sk}(x_{\cdot,\boldsymbol{j}})$ of size t and a sketch $\mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}])$ of size t' (to estimate $\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}]$) from which it can evaluate the random variable $\mathbf{Y}_{\boldsymbol{j}}$. The counter-intuitive aspect is that, in this case, we will perform both steps in one-pass:

- We will use an ℓ_p-sampling sketch of [43] to sample from an importance sampling distribution $\mathcal{D}$, and
- Concurrently, we prepare $2d$ linear sketches: d sketches $\mathbf{sk}(x_{\cdot,j})$ to evaluate (3), one for each $j \in [d]$, and d sketches $\mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[j])$ to evaluate $\mathbf{Pr}_{\mathcal{D}}[j]$, one for each $j \in [d]$. The non-trivial part is to *sketch the sketches*. by compressing the $2d$ linear sketches into a $O(\text{polylog}(nd))$-bit Count-Min data structure [30].

The guarantee will be that the ℓ_p-sampling sketch of [43] generates a sample $\boldsymbol{j} \sim \mathcal{D}$, and the Count-Min data structure can recover an approximation

$$\widehat{\mathbf{sk}_1} \approx \mathbf{sk}(x_{\cdot,\boldsymbol{j}}) \qquad \text{and} \qquad \widehat{\mathbf{sk}_2} \approx \mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}]).$$

Furthermore, the sketch evaluation algorithm, which executes on the approximation $\widehat{\mathbf{sk}_1}$ and $\widehat{\mathbf{sk}_2}$, should be able to recover $(1 \pm \epsilon)$-approximations to (3) and $\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}]$, so that the ratio of the two is a $(1 \pm 2\epsilon)$-approximation to $\mathbf{Y}_{\boldsymbol{j}}$.

While the above plan provides a general recipe for importance sampling, the idea of "sketching the sketches" may not be applied in a black-box manner. First, the alternate distribution $\mathcal{D}$ should admit a sampling sketch. Second, the sketch evaluation algorithm for $\mathbf{sk}(x_{\cdot,j})$ and $\mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[j])$ should be robust to the errors introduced by the Count-Min compression. Bounding the errors introduced by the Count-Min data structure, and ensuring that the approximate sketches $\widehat{\mathbf{sk}_1}$ and $\widehat{\mathbf{sk}_2}$ constitutes the bulk of the technical work. Specifically for us, the plan is executed as follows: when $\sum_{i=1}^{n} x_i = 0 \in \mathbb{R}^d$, every j satisfies (we refer the reader to the full version for a thorough justification)

$$\frac{\min_{c_j \in \mathbb{R}} \sum_{i=1}^{n} |x_{ij} - c_j|^p}{\|x_{\cdot,j}\|_p^p} \in [2^{-p}, 1]. \tag{5}$$

Hence, we will let $\mathcal{D}$ be the distribution supported on $[d]$ given by setting, for each $j \in [d]$,

$$\Pr_{\boldsymbol{j} \sim \mathcal{D}}[\boldsymbol{j} = j] = \frac{\|x_{\cdot,j}\|_p^p}{Z} \qquad \text{where} \qquad Z = \sum_{j=1}^{d} \|x_{\cdot,j}\|_p^p = \sum_{i=1}^{n} \sum_{j=1}^{d} |x_{ij}|^p.$$

Note that (5) implies the variance of $\mathbf{Y}_{\boldsymbol{j}}$ for $\boldsymbol{j} \sim \mathcal{D}$ is appropriately bounded. Furthermore, since $\mathcal{D}$ is an ℓ_p-sampling distribution, the ℓ_p-sampling sketches of [43] are useful for sampling $\boldsymbol{j} \sim \mathcal{D}$. Finally, the approach of [43] is particularly suited for bounding the errors incurred by Count-Min on $\widehat{\mathbf{sk}_1}$ and $\widehat{\mathbf{sk}_2}$, which we overview below.

At a high level, the ℓ_p-sampling sketch of [43] generates a sample $\boldsymbol{j}$ from $[d]$ by identifying a *heavy hitter* in a random scaling of the vector specifying the sampling probabilities. In particular, the algorithm generates $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d \sim \text{Exp}(1)$ and identifies an entry $j \in [d]$ in the vector

$$\left(\frac{\|x_{\cdot,1}\|_p}{\boldsymbol{u}_1^{1/p}}, \frac{\|x_{\cdot,2}\|_p}{\boldsymbol{u}_2^{1/p}}, \ldots, \frac{\|x_{\cdot,d-1}\|_p}{\boldsymbol{u}_{d-1}^{1/p}}, \frac{\|x_{\cdot,d}\|_p}{\boldsymbol{u}_d^{1/p}} \right) \in \mathbb{R}^d,$$

whose value satisfies

$$\frac{\|x_{\cdot,j}\|_p}{\boldsymbol{u}_j^{1/p}} \gtrsim \left(\sum_{j'=1}^{d} \frac{\|x_{\cdot,j'}\|_p^p}{\boldsymbol{u}_{j'}} \right)^{1/p}, \tag{6}$$

and is the largest among those heavy hitters. For the coordinate $\boldsymbol{j} \in [d]$ recovered by the ℓ_p-sampling sketch [43], the inequality (6) gives a lower bound on how large $1/\boldsymbol{u}_{\boldsymbol{j}}^{1/p}$ will be. In particular, by applying the same transformation to the vector of *sketches*,

$$\left(\frac{\mathbf{sk}(x_{\cdot,1})}{\boldsymbol{u}_1^{1/p}}, \ldots, \frac{\mathbf{sk}(x_{\cdot,d})}{\boldsymbol{u}_d^{1/p}} \right) \in (\mathbb{R}^t)^d \qquad \text{and}$$
$$\left(\frac{\mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[1])}{\boldsymbol{u}_1^{1/p}}, \ldots, \frac{\mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[d])}{\boldsymbol{u}_d^{1/p}} \right) \in (\mathbb{R}^{t'})^d, \tag{7}$$

the t and t' coordinates corresponding to the sketches $\mathbf{sk}(x_{\cdot,\boldsymbol{j}}) \in \mathbb{R}^t$ and $\mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}]) \in \mathbb{R}^{t'}$ will be heavy hitters of those vectors as well. Namely, with only $\text{poly}(\log(nd), 1/\epsilon)$-bits, the Count-Min data structure will recover the entries of $\mathbf{sk}(x_{\cdot,\boldsymbol{j}})$ and $\mathbf{sk}'(\mathbf{Pr}_{\mathcal{D}}[\boldsymbol{j}])$ up to a small additive error, proportional to the ℓ_1-norm of (7). We know the distribution of sketched vectors (7) (since these are simply ℓ_p-sketches [41]), so we will be able to bound the additive error and show that the sketch evaluation algorithms of $\widehat{\mathbf{sk}_1}$ and $\widehat{\mathbf{sk}_2}$ return the desired $(1 \pm \epsilon)$-approximations.

▶ Remark 4 (Comparison to [25]). The technique, "ℓ_p-sampling with meta-data", arises in [25] in the following context. They seek a linear sketch $\mathbf{sk}\colon \mathbb{R}^d \to \mathbb{R}^t$ which can process a vector $y \in \mathbb{R}^d$ and evaluate a weighted ℓ_1-norm, $\sum_{i=1}^d w_i(y) \cdot |y_i|$, where the weights $w_1(y), \ldots, w_d(y) \in \mathbb{R}_{\geq 0}$ are themselves dependent on the vector y. This arises as an algorithmic step in streaming algorithms for geometric minimum spanning tree and the earth-mover's distance. Mapping the above formulation to our setting, we want to evaluate a weighted ℓ_1-norm as well, where the i-th weight corresponds to $\mathbf{Pr}_{\boldsymbol{j} \sim \mathcal{D}}[\boldsymbol{j} = i]$, and the i-th value seek to sum is $\mathbf{Y}_i$ (as in (4)). The perspective of this technique as importance sampling (as presented in this work) is new. Indeed, the appropriate setting of weights is only apparent once one multiplies and divides the contribution of the j-th coordinate by $\|x_{\cdot,j}\|_p^p$ to define $\mathcal{D}$.

2 Sketching Median Costs

2.1 Statement of Main Lemma

▶ **Theorem 5.** *Fix $n, d \in \mathbb{N}$, as well as $p \in [1, 2]$ and $\epsilon, \delta \in (0, 1)$. There exists a linear sketch using $\text{poly}(\log d, 1/\epsilon, \log(1/\delta))$ space which processes a set of n points $x_1, \ldots, x_n \in \mathbb{R}^d$, and outputs a parameter $\boldsymbol{\eta} \in \mathbb{R}$ which satisfies*

$$\min_{y \in \mathbb{R}^d} \sum_{i=1}^{n} \|y - x_i\|_p^p \leq \boldsymbol{\eta} \leq (1+\epsilon) \min_{y \in \mathbb{R}^d} \sum_{i=1}^{n} \|y - x_i\|_p^p$$

with probability at least $1 - \delta$.

We work with the following representation of a linear sketch. The processed set of n points in $\mathbb{R}^d$ are stacked to form a vector $x \in \mathbb{R}^{nd}$. A linear sketch using space s is a distribution $\mathcal{M}$ supported on $s \times (nd)$ matrices. The theorem states that for any fixed $x_1, \ldots, x_n \in \mathbb{R}^d$, with probability $1 - \delta$ over the draw of $\mathbf{S} \sim \mathcal{M}$, an algorithm with access to the vector $\mathbf{S}x \in \mathbb{R}^s$ and $\mathbf{S}$ can output $\boldsymbol{\eta}$ satisfying the above guarantees.

Linear sketches of the above form imply efficient streaming algorithms, albeit with some subtleties. It is useful to first assume that the streaming algorithm can store its randomness for free (we will address this in the full version) so that it knows the matrix $\mathbf{S}$. In particular, since $\mathbf{S} \in \mathbb{R}^{s \times nd}$ acts on the vector $x \in \mathbb{R}^{nd}$ which vertically stacks $x_1, \ldots, x_n \in \mathbb{R}^d$, the columns of $\mathbf{S}$ may be broken up into n groups of size d, so

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{S}_2 & \ldots & \mathbf{S}_n \end{bmatrix}, \qquad \text{and} \qquad \mathbf{S}x = \sum_{i=1}^{n} \mathbf{S}_i x_i.$$

In the *insertion-only* model, an algorithm would process the points one-at-a-time, and at time-step j, maintain $\sum_{i=1}^{j} \mathbf{S}_i x_i \in \mathbb{R}^s$. In the *turnstile* model of streaming, there is a subtlety in the implementation; namely, as the algorithm receives insertions and deletions of points in $\mathbb{R}^d$, it must know which index $i \in [n]$ it is considering. The reason is that the algorithm should know which of the sub-matrix $\mathbf{S}_i$ to update the point with.

For our application of the ℓ_p^p-median sketch to (k, p)-clustering in ℓ_p, we consider a weighted ℓ_p^p-median. Namely, for points $x_1, \ldots, x_n \in \mathbb{R}^d$ and weights $\lambda_1, \ldots, \lambda_n \in [0, 1]$ with $\sum_{i=1}^{n} \lambda_i = 1$, the ℓ_p^p-median cost with respect to weights $\lambda_1, \ldots, \lambda_n$ is

$$\min_{y \in \mathbb{R}^d} \sum_{i=1}^{n} \lambda_i \|y - x_i\|_p^p.$$

It is useful to first consider of $\lambda_1 = \cdots = \lambda_n = 1/n$. For general weights, the sketch will receive as input $\mathbf{S} = [\mathbf{S}_1, \ldots, \mathbf{S}_n] \in \mathbb{R}^{s \times (nd)}$, the vector $\sum_{i=1}^{n} \lambda_i^{1/p} \mathbf{S}_i x_i \in \mathbb{R}^s$, and the weights $\lambda_1, \ldots, \lambda_n$.

Centering Points

There is a straight-forward way to process the points so as to assume they are centered. Specifically, the average point may be subtracted from every point by applying a linear map, and since our sketch is linear, subtracting the average point may be incorporated into the sketch. For weights $\lambda_1, \ldots, \lambda_n \in [0, 1]$ satisfying $\sum_{i=1}^{n} \lambda_i = 1$, we consider the linear map

$$(x_1, \ldots, x_n) \mapsto \left(x_1 - \sum_{i=1}^{n} \lambda_i x_i, \ldots, x_n - \sum_{i=1}^{n} \lambda_i x_i \right) \in \mathbb{R}^{nd}.$$

Hence, we assume, without loss of generality, that the points $x_1, \ldots, x_n \in \mathbb{R}^d$ satisfy

$$\sum_{i=1}^{n} \lambda_i x_i = 0 \in \mathbb{R}^d. \tag{8}$$

The centering is useful for deriving the following set of inequalities, which will be useful for our sketching procedures. Suppose we denote $y \in \mathbb{R}^d$ as the point which minimizes $\sum_{i=1}^{n} \lambda_i \|y - x_i\|_p^p$. Then, for every $j \in [d]$,

$$\sum_{i=1}^{n} \lambda_i |y_j - x_{ij}|^p \leq \sum_{i=1}^{n} \lambda_i |x_{ij}|^p \leq 2^p \sum_{i=1}^{n} \lambda_i |y_j - x_{ij}|^p.$$

Importantly for us, every $j \in [d]$ satisfies

$$2^{-p} \leq \frac{\min_{y_j \in \mathbb{R}} \sum_{i=1}^n \lambda_i |y_j - x_{ij}|^p}{\sum_{i=1}^n \lambda_i |x_{ij}|^p} \leq 1. \tag{9}$$

We let $\mathcal{D}$ be the distribution supported on $[d]$ given by setting, for each $j \in [d]$,

$$\mathop{\mathbf{Pr}}_{\boldsymbol{j} \sim \mathcal{D}}[\boldsymbol{j} = j] = \frac{1}{Z} \sum_{i=1}^n \lambda_i |x_{ij}|^p \qquad \text{where} \qquad Z = \sum_{j'=1}^d \sum_{i=1}^n \lambda_i |x_{ij'}|^p = \sum_{i=1}^n \lambda_i \|x_i\|_p^p.$$

Then, the quantity we want to estimate may be equivalently re-written as:

$$\sum_{j=1}^d \min_{y_j \in \mathbb{R}} \sum_{i=1}^n \lambda_i |y_j - x_{ij}|^p = Z \cdot \mathop{\mathbf{E}}_{\boldsymbol{j} \sim \mathcal{D}} \left[\frac{\min_{y_{\boldsymbol{j}} \in \mathbb{R}} \sum_{i=1}^n \lambda_i |y_{\boldsymbol{j}} - x_{i\boldsymbol{j}}|^p}{\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}}|^p} \right], \tag{10}$$

where the value within the expectation is bounded between 2^{-p} and 1. Furthermore, the quantity Z will be sketched with an ℓ_p-sketch, and a sample $\boldsymbol{j} \sim \mathcal{D}$ will be drawn with an ℓ_p-sampling sketch. Hence, the plan is to produce $t = O(1/\epsilon^2)$ samples of $\boldsymbol{j}_1, \ldots, \boldsymbol{j}_t \sim \mathcal{D}$, and produce a sketch to evaluate the numerator inside the expectation, and the denominator inside the expectation. Taking an empirical average of the samples to estimate the expectation, and multiplying it by the estimate of Z will give the desired estimator.

▶ **Lemma 6** (Main Lemma). *For any $n, d \in \mathbb{N}$, $p \in [1, 2]$ and $\epsilon, \delta \in (0, 1)$, let $s = \mathrm{poly}(\log d, 1/\epsilon, 1/\delta)$.*[5] *There exists a distribution $\mathcal{S}$ over $s \times (nd)$ matrices, and an algorithm such that for any n vectors $x_1, \ldots, x_n \in \mathbb{R}^d$ and any $\lambda_1, \ldots, \lambda_n \in [0, 1]$ with $\sum_{i=1}^n \lambda_i = 1$ and $\sum_{i=1}^n \lambda_i x_i = 0$, the following occurs:*

- *We sample $\mathbf{S} = [\mathbf{S}_1, \ldots, \mathbf{S}_n] \sim \mathcal{S}$, and we give the algorithm as input $\mathbf{S}$, $\sum_{i=1}^n \mathbf{S}_i(\lambda_i^{1/p} x_i)$, and $\lambda_1, \ldots, \lambda_n$.*
- *The algorithm outputs a tuple of three numbers $(\boldsymbol{j}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \in [d] \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$. With probability at least $1 - \delta$ over the draw of $\mathbf{S} \sim \mathcal{S}$, we have the following two inequalities:*

$$(1-\epsilon) \left(\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}}|^p \right)^{1/p} \leq \boldsymbol{\alpha} \leq (1+\epsilon) \left(\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}}|^p \right)^{1/p},$$

$$(1-\epsilon) \min_{z \in \mathbb{R}} \left(\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}} - z|^p \right)^{1/p} \leq \boldsymbol{\beta} \leq (1+\epsilon) \min_{z \in \mathbb{R}} \left(\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}} - z|^p \right)^{1/p}.$$

- *Furthermore, the distribution of the random variable $\boldsymbol{j}$ is $\epsilon 2^{-p}$-close in total variation distance to $\mathcal{D}$.*

Proof of Theorem 5 assuming Lemma 6. Given Lemma 6, the proof of Theorem 5 is straight-forward. We fix $\lambda_1 = \cdots = \lambda_n = 1/n$, and we first handle the centering. We will utilize Lemma 6 which requires vectors $x_1, \ldots, x_n$ to satisfy $\sum_{i=1}^n \lambda_i x_i = 0$; hence, we sketch the vectors $x'_1, \ldots, x'_n$ given by $x'_i = x_i - \sum_{h=1}^n \lambda_h x_h$, which are now centered. By linearity, this is equivalent to maintaining the vector

$$\sum_{i=1}^n \lambda_i^{1/p} \mathbf{S}_i (x_i - \sum_{h=1}^n \lambda_h x_h) = \sum_{i=1}^n \left(\lambda_i^{1/p} \mathbf{S}_i - \lambda_i \sum_{h=1}^n \lambda_h^{1/p} \mathbf{S}_h \right) x_i \in \mathbb{R}^s.$$

[5] See the full version for the specific polynomial bounds.

We take $t = \omega(1/\epsilon^2)$ independent sketches from Lemma 6 with accuracy parameter $\epsilon/2$ and error probability $\delta = o(1/t)$. This, in turn, gives us t independent samples $(\boldsymbol{j}_1, \boldsymbol{\alpha}_1, \boldsymbol{\beta}_1)$, $\ldots$, $(\boldsymbol{j}_t, \boldsymbol{\alpha}_1, \boldsymbol{\beta}_t)$. By taking a union bound over the t executions of Lemma 6, with high probability, every $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_t$ satisfy

$$\boldsymbol{\alpha}_\ell^p \approx_{(1+\epsilon p/2)} \sum_{i=1}^n \lambda_i |x_{i\boldsymbol{k}}|^p \qquad \text{and} \qquad \boldsymbol{\beta}_\ell^p \approx_{(1+\epsilon p/2)} \min_{z\in\mathbb{R}} \sum_{i=1}^n \lambda_i |x_{i\boldsymbol{k}} - z|^p,$$

and $\boldsymbol{j}_1, \ldots, \boldsymbol{j}_t$ are independent draws from a distribution $\mathcal{D}'$ which is $\epsilon 2^{-p}$-close to $\mathcal{D}$. For estimating Z, we use an ℓ_p-sketch to accuracy $\epsilon/2$ and failure probability $\delta = o(1)$. For example, the sketch for Z may proceed by applying an ℓ_p-sketch [41] to the stacked vector $x' \in \mathbb{R}^{nd}$ where

$$x'_{ij} = \lambda_i^{1/p} \cdot x_{ij},$$

so that the ℓ_p norm of x' is exactly $Z^{1/p}$. Let $\widehat{\mathbf{Z}}$ be the estimate for the Z. For our estimate $\boldsymbol{\eta}$ that we will output, we set

$$\boldsymbol{\eta} = \widehat{\mathbf{Z}} \cdot \frac{1}{t} \sum_{\ell=1}^t \operatorname{minmax}\left\{2^{-p}, \left(\frac{\boldsymbol{\beta}_\ell}{\boldsymbol{\alpha}_\ell}\right)^p, 1\right\},$$

where $\operatorname{minmax}(l, x, u)$ is l if $x \le l$, u if $u \ge x$, and x otherwise. To see why our estimator approximates (10), we have $\widehat{\mathbf{Z}}$ is a $(1 \pm \epsilon/2)$-approximation of Z. The latter quantity is the empirical average of t i.i.d random variables, each of which is bounded by 2^{-p} and 1. In particular, we have that with probability at least $1 - o(1)$, Chebyshev's inequality, and the conditions of $\boldsymbol{\beta}_\ell$ and $\boldsymbol{\alpha}_\ell$,

$$\mathop{\mathbf{E}}_{\boldsymbol{j}\sim\mathcal{D}'}\left[\frac{\min_{z\in\mathbb{R}} \sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}} - z|^p}{\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}}|^p}\right] \approx_{(1+2\epsilon p)} \frac{1}{t}\sum_{\ell=1}^t \operatorname{minmax}\left\{2^{-p}, \left(\frac{\boldsymbol{\beta}_\ell}{\boldsymbol{\alpha}_\ell}\right)^p, 1\right\}.$$

It remains to show that

$$\mathop{\mathbf{E}}_{\boldsymbol{j}\sim\mathcal{D}'}\left[\frac{\min_{z\in\mathbb{R}} \sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}} - z|^p}{\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}}|^p}\right] \approx_{(1\pm\epsilon)} \mathop{\mathbf{E}}_{\boldsymbol{j}\sim\mathcal{D}}\left[\frac{\min_{z\in\mathbb{R}} \sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}} - z|^p}{\sum_{i=1}^n \lambda_i |x_{i\boldsymbol{j}}|^p}\right].$$

This follows from two facts: (1) $\mathcal{D}'$ and $\mathcal{D}$ are $\epsilon 2^{-p}$ close, since the random variable is at most 1, the expectations are off by at most an additive $\epsilon 2^{-p}$-factor, and (2) both quantities above are the average of random variables which are at least 2^{-p}, so an additive $\epsilon 2^{-p}$ error is less than a multiplicative $(1 \pm \epsilon)$-error.

The above gives an estimate which is a $1 \pm \epsilon$-approximation with probability $1 - o(1)$, in order to boost the probability of success to $1 - \delta$, we simply repeat $O(\log(1/\delta))$ times and output the median estimate. ◀

The remainder of the section is organized as follows. We give in the (next) Subsection 2.2, the necessary sketches for obtaining $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ for a fixed coordinate j. Then, in the following Subsection 2.3, we show how we combine various sketches from Subsection 2.2 for different $j \in [d]$ to obtain $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ up to some additive error. Finally, the proof of Lemma 6 appears in the full version, where we apply a randomized transformation to the input so that the additive error from Subsection 2.3 is a multiplicative error for the specific sampled $\boldsymbol{j}$.

2.2 Sketch for Optimizing a Single Coordinate

In this subsection, we give linear sketches which are useful for optimizing over a single coordinate. Specifically, given the n vectors $x_1, \ldots, x_n \in \mathbb{R}^d$ and $j \in [d]$, we consider the k-th coordinate of the n vectors $x_{1j}, x_{2j}, \ldots, x_{nj} \in \mathbb{R}$. Hence, the linear sketches in this section will act on vectors in $\mathbb{R}^n$, corresponding to the j-th coordinates of the points, and will give approximations to

$$\sum_{i=1}^{n} \lambda_j |x_{ij}|^p \quad \text{(Corollary 8)} \quad \text{and} \quad \min_{y_j \in \mathbb{R}} \sum_{i=1}^{n} \lambda_j |y_j - x_{ij}|^p. \quad \text{(Lemma 9)}$$

The lemma statements also consider an additive error term, $\text{err} \in \mathbb{R}_{\geq 0}$, which will be necessary when combining these sketches in Subsection 2.3; however, it may be helpful to consider $\text{err} = 0$ on first reading.

▶ **Lemma 7.** *For any $n \in \mathbb{N}$, $p \in [1,2]$ and $\epsilon, \delta \in (0,1)$, let $s = O(\log(1/\delta)/\epsilon^2)$. There exists a distribution $\mathcal{M}$ over $s \times n$ matrices, and an algorithm such that for any $x \in \mathbb{R}^n$, $\lambda_1, \ldots, \lambda_n \in [0,1]$, and $y \in \mathbb{R}$, the following occurs:*

- *We sample $\mathbf{S} \sim \mathcal{M}$ as well as a random vector $\boldsymbol{\chi} = (\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_s) \in \mathbb{R}^s$ where each is an i.i.d p-stable random variable. For any $\text{err} \in \mathbb{R}_{\geq 0}$, we give the algorithm as input $\mathbf{S}$, $\mathbf{S}(\lambda^{1/p} \circ x) + \text{err} \cdot \boldsymbol{\chi}$, the parameters $\lambda_1, \ldots, \lambda_n$, and y.*[6]
- *The algorithm outputs a parameter $\widehat{\boldsymbol{\eta}} \in \mathbb{R}_{\geq 0}$, which depends on $\mathbf{S}, \mathbf{S}(\lambda^{1/p} \circ x) + \text{err} \cdot \boldsymbol{\chi}$, the parameters $\lambda_1, \ldots, \lambda_n$, and y which satisfies with probability at least $1 - \delta$ over $\mathbf{S}$ and $\boldsymbol{\chi}$,*

$$(1-\epsilon)\left(\sum_{i=1}^{n} \lambda_i |x_i - y|^p + \text{err}^p\right)^{1/p} \leq \widehat{\boldsymbol{\eta}} \leq (1+\epsilon)\left(\sum_{i=1}^{n} \lambda_i |x_i - y|^p + \text{err}^p\right)^{1/p}. \quad (11)$$

Furthermore, for every $j \in [s]$, the random variables $(\mathbf{S}(\lambda^{1/p} \circ x))_j \in \mathbb{R}$ are distributed as $\|\lambda^{1/p} \circ x\|_p \cdot \boldsymbol{\chi}_j$, where $\boldsymbol{\chi}_j$ are independent, p-stable random variables.

Proof. We notice that this simply corresponds to an ℓ_p-sketch of the vector $z \in \mathbb{R}^n$, which is given by letting each $z_i = \lambda^{1/p} \circ (x_i - y)$, so that the ℓ_p-sketch of [41] would accomplish this task. Since the algorithm receives $\mathbf{S}$, $\mathbf{S}(\lambda^{1/p} \circ x) + \text{err} \cdot \chi$ and $y \in \mathbb{R}$, the algorithm may compute $\mathbf{S}(\lambda^{1/p} \circ y \cdot \mathbf{1})$, where $\mathbf{1} \in \mathbb{R}^n$ is an all-1's vector, and evaluate the sketch

$$\mathbf{S}(\lambda^{1/p} \circ x) + \text{err} \cdot \boldsymbol{\chi} - \mathbf{S}(\lambda^{1/p} \circ y \cdot \mathbf{1}) = \mathbf{S}(\lambda^{1/p} \circ (x - y \cdot \mathbf{1})) + \text{err} \cdot \boldsymbol{\chi}$$

by linearity. Furthermore, note that the error simply corresponds to an ℓ_p-sketch of the vector $z' \in \mathbb{R}^{n+1}$ which sets $z'_i = z_i$ for $i \neq n+1$ and $z'_{n+1} = \text{err}$. ◀

▶ **Corollary 8.** *For any $n \in \mathbb{N}$, $p \in [1,2]$ and $\epsilon, \delta \in (0,1)$, let $s = O(\log(1/\delta)/\epsilon^2)$. There exists a distribution $\mathcal{M}$ over $s \times n$ matrices, and an algorithm such that for any $x \in \mathbb{R}^n$, and any $\lambda_1, \ldots, \lambda_n \in [0,1]$, the following occurs:*

- *We sample $\mathbf{S} \sim \mathcal{M}$ and a random vector $\boldsymbol{\chi} = (\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_s) \in \mathbb{R}^s$ of i.i.d p-stable random variables. For any $\text{err} \in \mathbb{R}_{\geq 0}$. We give the algorithm as input $\mathbf{S}$ and $\mathbf{S}(\lambda^{1/p} \circ x) + \text{err} \cdot \boldsymbol{\chi}$.*

[6] The notation $\lambda^{1/p} \circ x \in \mathbb{R}^n$ denotes the Hadamard product, where $(\lambda^{1/p} \circ x)_i = \lambda_i^{1/p} \cdot x_i$.

- *With probability at least* $1-\delta$ *over* $\mathbf{S}$ *and* $\boldsymbol{\chi}$*, the algorithm outputs a parameter* $\widehat{\boldsymbol{\gamma}} \in \mathbb{R}_{\geq 0}$*, which depends on* $\mathbf{S}$*, and* $\mathbf{S}(\lambda^{1/p} \circ x)$ *which satisfies*

$$(1-\epsilon)\left(\sum_{i=1}^{n} \lambda_i |x_i|^p + \mathrm{err}^p\right)^{1/p} \leq \widehat{\boldsymbol{\gamma}} \leq (1+\epsilon)\left(\sum_{i=1}^{n} \lambda_i |x_i|^p + \mathrm{err}^p\right)^{1/p}.$$

Furthermore, for every $j \in [s]$*, the random variables* $(\mathbf{S}(\lambda^{1/p} \circ x))_j \in \mathbb{R}$ *are independent and distributed as* $\|\lambda^{1/p} \circ x\|_p \cdot \boldsymbol{\chi}_j$*, where* $\boldsymbol{\chi}_j$ *are independent, p-stable random variables.*

Proof. We apply Lemma 7 to the vector $x \in \mathbb{R}^n$ with $y = 0$. ◀

▶ **Lemma 9.** *For any* $n \in \mathbb{N}$*,* $p \in [1,2]$ *and* $\epsilon, \delta \in (0,1)$*, let* $s = O(\log(1/(\epsilon\delta))/\epsilon^2)$*. There exists a distribution* $\mathcal{M}$ *over* $s \times n$ *matrices, and an algorithm such that for any* $x \in \mathbb{R}^n$ *and any* $\lambda_1, \ldots, \lambda_n \in [0,1]$ *with* $\sum_{i=1}^n \lambda_i = 1$*, whenever* $\sum_{i=1}^n \lambda_i x_i = 0$*, the following occurs:*

- *We sample* $\mathbf{S} \sim \mathcal{M}$ *and a random vector* $\boldsymbol{\chi} = (\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_s)$ *of i.i.d p-stable random variables. For any* $\mathrm{err} \in \mathbb{R}_{\geq 0}$*. We give the algorithm as input* $\mathbf{S}$*,* $\mathbf{S}(\lambda^{1/p} \circ x) + \mathrm{err} \cdot \boldsymbol{\chi}$*, the parameters* $\lambda_1, \ldots, \lambda_n$*, and a parameter* $\gamma \in \mathbb{R}_{\geq 0}$ *satisfying*

$$(1-\epsilon)\left(\sum_{i=1}^{n} \lambda_i |x_i|^p\right)^{1/p} \leq \gamma \leq (1+\epsilon)\left(\sum_{i=1}^{n} \lambda_i |x_i|^p\right)^{1/p}.$$

- *The algorithm outputs a parameter* $\widehat{\boldsymbol{\beta}} \in \mathbb{R}_{\geq 0}$ *which satisfies*

$$(1-\epsilon)\min_{z\in\mathbb{R}}\left(\sum_{i=1}^{n} \lambda_i |x_i - z|^p + \mathrm{err}^p\right)^{1/p} \leq \widehat{\boldsymbol{\beta}} \leq (1+\epsilon)\min_{z\in\mathbb{R}}\left(\sum_{i=1}^{n} \lambda_i |x_i - z|^p + \mathrm{err}^p\right)^{1/p}. \tag{12}$$

Furthermore, for every $j \in [s]$*, the random variables* $(\mathbf{S}(\lambda^{1/p} \circ x))_j$ *are independent and distributed as* $\|\lambda^{1/p} \circ x\|_p \cdot \boldsymbol{\chi}_j$*, where* $\boldsymbol{\chi}_j$ *are independent, p-stable random variables.*

Proof. We will utilize the sketch from Lemma 7, while varying the y's to find the minimum. Specifically, let $t = 16 \cdot 2^p/\epsilon$, and let the distribution $\mathcal{M}$ be the same as that of Lemma 7 instantiated with error probability $1 - t\delta$ and accuracy parameter $\epsilon/2$. We discretize the interval $[-4\gamma, 4\gamma]$ into t, evenly-spaced out points $y_1, \ldots, y_t \subset [-4\gamma, 4\gamma]$ such that $y_{\ell+1} - y_\ell = 8\gamma/t$. We utilize the algorithm in Lemma 7 to obtain estimates $\widehat{\boldsymbol{\eta}}_1, \ldots, \widehat{\boldsymbol{\eta}}_t$ satisfying (11) with $y_1, \ldots, y_t$, respectively. Then, we output

$$\widehat{\boldsymbol{\beta}} = \min_{\ell\in[t]} \widehat{\boldsymbol{\eta}}_\ell.$$

Since we amplified the error probability to less than $t\delta$, we may assume, by a union bound, that all estimates $\{\boldsymbol{\eta}_\ell\}_{\ell\in[t]}$ satisfy (11) with y_ℓ with probability at least $1-\delta$. First, for any $\ell \in [t]$,

$$\min_{z\in\mathbb{R}}\left(\sum_{i=1}^{n} \lambda_i |x_i - z|^p\right)^{1/p} \leq \left(\sum_{i=1}^{n} \lambda_i |x_i - y_\ell|^p\right)^{1/p},$$

and therefore, the lower bound in (12) is implied by (11). To prove the upper bound in (12), denote $z \in \mathbb{R}$ as the true minimizer of $(\sum_{i=1}^n \lambda_i |x_i - z|^p)^{1/p}$. By the fact $\sum_{i=1}^n \lambda_i = 1$ and the triangle inequality, we have

$$|z| = \left(\sum_{i=1}^n \lambda_i |z|^p\right)^{1/p} \leq \left(\sum_{i=1}^n \lambda_i |x_i - z|^p\right)^{1/p} + \left(\sum_{i=1}^n \lambda_i |x_i|^p\right)^{1/p} \leq 2\left(\sum_{i=1}^n \lambda_i |x_i|^p\right)^{1/p},$$

so $|z| \leq 2(1+\epsilon)\gamma \leq 4\gamma$, and thus $z \in [-4\gamma, 4\gamma]$. Let $\ell \in [t]$ be such that $|y_t - z| \leq 4\gamma/t$. Then, again by the triangle inequality and the fact $\sum_{i=1}^n \lambda_i x_i = 0$,

$$\begin{aligned}\left(\sum_{i=1}^n \lambda_i |x_i - y_t|^p\right)^{1/p} &\leq \left(\sum_{i=1}^n \lambda_i |x_i - z|^p\right)^{1/p} + 4\gamma/t \\ &\leq (1 + 4(1+\epsilon)2^p/t)\left(\sum_{i=1}^n \lambda_i |x_i - z|^p\right)^{1/p}.\end{aligned}$$

By the setting of t, $(\sum_{i=1}^n \lambda_i |x_i - y_t|^p)^{1/p} \leq (1+\epsilon/2)(\sum_{i=1}^n \lambda_i |x_i - z|^p)^{1/p}$, and by (11), we obtain the desired upper bound. ◀

2.3 Grouping Single Coordinate Sketches

In this subsection, we show how to compress d linear sketches (one for each coordinate) from Subsection 2.2. In the lemma that follows, the parameter $m \in \mathbb{N}$ should be considered the sketch size of the sketches in Subsection 2.2, and the linear sketch will take the d sketches from Subsection 2.2 (represented as a vector $\mathbb{R}^{dm}$). Each of the d linear sketches have each coordinate of $\mathbb{R}^m$ distributed as an i.i.d scaled p-stable random variable (specified by the last sentence in Corollary 8 and Lemma 9). Thus, we write the d sketches as $\boldsymbol{\Psi}_1 v_1, \ldots, \boldsymbol{\Psi}_d v_d \in \mathbb{R}^m$, where $v_j \in \mathbb{R}$ is a scaling, and $\boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_d \in \mathbb{R}^{m \times n}$ are i.i.d p-stable matrices.

▶ **Lemma 10** (p-stable Sketch Compression via Count-Min)**.** *Let $d, m \in \mathbb{N}$, $\epsilon, \delta \in (0,1)$, and let $t = O(\log(d/\delta))$. There exists a distribution $\mathcal{C}$ over $(10tm/\epsilon^p) \times (dm)$ matrices, and an algorithm such that for any $v \in \mathbb{R}^d$, the following occurs:*

- *We sample $\mathbf{C} \sim \mathcal{C}$ and a $(dm) \times d$ matrix $\boldsymbol{\Psi}$, where $\boldsymbol{\Psi} = \mathrm{diag}(\boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_d)$, and each $\boldsymbol{\Psi}_j = (\boldsymbol{\chi}_{j1}, \ldots, \boldsymbol{\chi}_{jm}) \in \mathbb{R}^m$ are independent p-stable random vectors.[7] The algorithm receives as input $\mathbf{C}$ and $\mathbf{C}\boldsymbol{\Psi}v$.*
- *The algorithm outputs, for each $j \in [d]$, a sequence of t vectors $\widehat{\boldsymbol{z}}_j^{(1)}, \ldots, \widehat{\boldsymbol{z}}_j^{(t)} \in \mathbb{R}^m$ which satisfy, for each $t' \in [t]$,*

$$\widehat{\boldsymbol{z}}_j^{(t')} = v_j \boldsymbol{\Psi}_j + \mathbf{err}_j^{(t')} \cdot \boldsymbol{\chi}_j^{(t')}. \tag{13}$$

where $\boldsymbol{\chi}_j^{(t')} \in \mathbb{R}^m$ is a vector of independent p-stable random variables, and $\mathbf{err}_j^{(t')} \in \mathbb{R}_{\geq 0}$ only depends on $\mathbf{C}$. With probability at least $1 - \delta$ over $\mathbf{C}$, for every $j \in [d]$

$$\left|\left\{t' \in [t] : \mathbf{err}_j^{(t')} \leq \epsilon \|v\|_p\right\}\right| \geq t/2. \tag{14}$$

[7] Hence, the vector $\boldsymbol{\Psi}v \in \mathbb{R}^{dm}$ is given by vertically stacking d vectors of the form $v_j \boldsymbol{\Psi}_j \in \mathbb{R}^m$.

Proof. The matrix $\mathbf{C}$ is a Count-Min matrix which given a vector $u \in \mathbb{R}^{dm}$ given by vertically stacking d vectors $u_1, \ldots, u_d \in \mathbb{R}^m$ repeats the following process: for each $t' \in [t]$, we sample a hash function $\boldsymbol{h}_{t'} : [d] \to [10/\epsilon^p]$, and for each $\ell \in [10/\epsilon^p]$ store the vector

$$\boldsymbol{b}_{t',\ell} \stackrel{\text{def}}{=} \sum_{j \in [d]} \mathbf{1}\{\boldsymbol{h}_{t'}(j) = \ell\} \cdot u_j \in \mathbb{R}^m.$$

In particular, the output $\mathbf{C}u$ consists of stacking $t \cdot 10/\epsilon^p$ vectors $(\boldsymbol{b}_{t',\ell} \in \mathbb{R}^m : t' \in [t], \ell \in [10/\epsilon^p])$, which gives the desired bound of $10mt/\epsilon^p$ on the output dimension of $\mathbf{C}$. For each $j \in [d]$ and $t' \in [t]$, the algorithm lets $\boldsymbol{\ell} = \boldsymbol{h}_{t'}(j)$ and sets

$$\widehat{\boldsymbol{z}}_j^{(t')} = \boldsymbol{b}_{t',\ell} = v_j \boldsymbol{\Psi}_j + \sum_{j' \in [d] \setminus \{j\}} \mathbf{1}\{\boldsymbol{h}_{t'}(j') = \boldsymbol{\ell}\} \cdot v_{j'} \cdot \boldsymbol{\Psi}_{j'}.$$

We now apply the p-stability property to the right-most summand, to notice that

$$\mathbf{err}_j^{(t')} = \left(\sum_{j' \in [d] \setminus \{j\}} \mathbf{1}\{\boldsymbol{h}_{t'}(j') = \boldsymbol{\ell}\} \cdot v_{j'}^p \right)^{1/p},$$

which only depends on $\mathbf{C}$. Furthermore, the inner most summand is at most $\epsilon^p/10 \sum_{j' \in [d] \setminus \{j\}} v_{j'}^p$ in expectation. By Markov's inequality, each $\mathbf{err}_j^{(t')} \leq \epsilon \|v\|_p$ with probability at least $9/10$. Since $t = O(\log(d/\delta))$, the probability that (14) is not satisfied for each $j \in [d]$ is at most δ/d by a Chernoff bound, so that a union bound gives the desired guarantees. ◀

The above lemma allows us to compress d many p-stable sketches into $O(\log(d/\delta)/\epsilon^p)$ many p-stable sketches, albeit with some error. Since the p-stable sketches that we will use (from Corollary 8 and Lemma 9) are exactly of the form $\boldsymbol{\Psi}v$ for some vector v, Lemma 10 will allow us to compress them. Namely, we will consider d sketches from Corollary 8 and Lemma 9 and utilize Lemma 10; for each $j \in [d]$, we will be able to recover t noisy versions of the sketch of Corollary 8 and Lemma 9 for coordinate j. Importantly, the noise is of the form an error times a p-stable random variable, and these are the kinds of errors that Corollary 8 and Lemma 9 can easily handle.

▶ **Lemma 11** (p-stable Sketch Recovery for Sample). *For $n, d \in \mathbb{N}$, $p \in [1, 2]$ and $\epsilon, \delta \in (0, 1)$, let $s = O(\log^2(d/\delta)/\epsilon^{2+p})$. There exists a distribution $\mathcal{R}$ over $s \times (nd)$ matrices, and an algorithm such that for any vectors $y_1, \ldots, y_n \in \mathbb{R}^d$ and weights $\lambda_1, \ldots, \lambda_n \in [0, 1]$ with $\sum_{i=1}^n \lambda_i = 1$, the following occurs with probability at least $1 - \delta$:*

- *We sample $\mathbf{S} = [\mathbf{S}_1, \ldots, \mathbf{S}_n] \sim \mathcal{R}$ and we give the algorithm as input $\mathbf{S}$, and the vector $\sum_{i=1}^n \mathbf{S}_i(\lambda_i^{1/p} y_i) \in \mathbb{R}^s$.*
- *The algorithm outputs d numbers $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_d \in \mathbb{R}_{\geq 0}$ such that each $j \in [d]$ satisfies*

$$(1-\epsilon)\left(\sum_{i=1}^n \lambda_i |y_{ij}|^p\right)^{1/p} - \text{err} \leq \boldsymbol{\alpha}_j \leq (1+\epsilon)\left(\sum_{i=1}^n \lambda_i |y_{ij}|^p\right)^{1/p} + \text{err},$$

where $\text{err} \in \mathbb{R}_{\geq 0}$ is an additive error satisfying

$$\text{err} \leq \epsilon \left(\sum_{j=1}^d \sum_{i=1}^n \lambda_i |y_{ij}|^p\right)^{1/p}.$$

Proof. We combine Corollary 8 and Lemma 10. We describe the distribution $\mathcal{R}$ over $s \times (nd)$ matrices by giving a procedure for sampling $\mathbf{S} \sim \mathcal{R}$. $\mathbf{S}$ can be naturally expressed as a concatenation of matrices $\mathbf{S} = [\mathbf{S}_1, \ldots, \mathbf{S}_n]$.

- We let $\mathcal{M}$ be the distribution over $s_0 \times n$ matrices of Corollary 8 with error probability at most $\delta/(2dt)$ and accuracy ϵ (so that we may union bound over d sketches later) so that $s_0 = O(\log(dt/\delta)/\epsilon^2)$. We take d independent samples $\mathbf{S}'_1, \ldots, \mathbf{S}'_d \sim \mathcal{M}$.
- For each $j \in [d]$, we let P_j be the $n \times (nd)$ matrix where given the vector $y' \in \mathbb{R}^{nd}$ given by vertically stacking $\lambda_1^{1/p} y_1, \ldots, \lambda_n^{1/p} y_n \in \mathbb{R}^d$, sets $y'_{\cdot,j} = P_k j'$, where $y'_{\cdot,j} = \lambda^{1/p} \circ (y_{i,j})_{i \in [n]} \in \mathbb{R}^n$. Let P be the $(nd) \times (nd)$ matrix which stacks these matrices vertically.
- We sample $\mathbf{C} \sim \mathcal{C}$ as in Lemma 10 with $m = s_0$, where we set the accuracy parameter $\epsilon/2$ and the failure probability $\delta/2$. We let

$$\mathbf{S} = \mathbf{C} \cdot \mathrm{diag}(\mathbf{S}'_1, \ldots, \mathbf{S}'_d) \cdot P.$$

Intuitively, we will apply our sketch $\mathbf{S}$ on the vector the matrix $\mathbf{S}$ may be interpreted as first applying d sketches of Corollary 8 to the vectors $(\lambda^{1/p} \circ y_{\cdot,1}), \ldots, (\lambda^{1/p} \circ y_{\cdot,d}) \in \mathbb{R}^n$, and then applying $\mathbf{C}$ from Lemma 10. The algorithm for producing the estimates $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_d$ proceeds by applying the algorithm of Lemma 10 to obtain, for each $j \in [d]$ a sequence of t vectors $\widehat{\boldsymbol{z}}_j^{(1)}, \ldots, \widehat{\boldsymbol{z}}_j^{(t)} \in \mathbb{R}^{s_0}$. We apply the algorithm of Corollary 8 to each of the t vectors to obtain estimates $\boldsymbol{\alpha}_j^{(1)}, \ldots, \boldsymbol{\alpha}_j^{(t)} \in \mathbb{R}_{\geq 0}$, and we let $\boldsymbol{\alpha}_j = \mathrm{median}\{\boldsymbol{\alpha}_j^{(t')} : t' \in [t]\}$.

To see why this works, consider the collection of d vectors

$$\boldsymbol{z}_j = \mathbf{S}'_j(\lambda^{1/p} \circ y_{\cdot,j}) \in \mathbb{R}^{s_0},$$

and notice that by Corollary 8, every $j \in [d]$ and $\ell \in [s]$, $\boldsymbol{z}_{j,\ell} \sim (\sum_{i=1}^n \lambda_i |y_{i,j}|^p)^{1/p} \cdot \boldsymbol{\chi}_{j,\ell}$, where $\boldsymbol{\chi}_{k,\ell}$ are independent, p-stable random variables. Indeed, if we write $v \in \mathbb{R}^d$ as the vector which sets

$$v_j = \left(\sum_{i=1}^n \lambda_i |y_{i,j}|^p \right)^{1/p},$$

then vertically stacking the vectors $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d \in \mathbb{R}^{s_0}$ gives a vector which is equivalently distributed as $\boldsymbol{\Psi} v$, where $\boldsymbol{\Psi}$ is the matrix from Lemma 10. In particular, with probability at least $1 - \delta/2$, the algorithm of Lemma 10 outputs dt vectors $(\widehat{\boldsymbol{z}}_j^{(t')} : j \in [d], t' \in [t])$ which satisfy

$$\widehat{\boldsymbol{z}}_j^{(t')} = \mathbf{S}'_k(\lambda^{1/p} \circ y_{\cdot,j}) + \mathbf{err}_j^{(t')} \cdot \boldsymbol{\chi}_{j,t'}. \tag{15}$$

Hence, with probability at least $1 - \delta/(2dt)$, the algorithm of Corollary 8 applied to $\widehat{\boldsymbol{z}}_j^{(t')}$ outputs an estimate $\boldsymbol{\alpha}_j^{(t')}$ satisfying $(1-\epsilon)(v_j^p + (\mathbf{err}_j^{(t')})^p)^{1/p} \leq \boldsymbol{\alpha}_j^{(t')} \leq (1+\epsilon)(v_j^p + (\mathbf{err}_j^{(t')})^p)^{1/p}$, and therefore, we have that each $\boldsymbol{\alpha}_j^{(t')}$ satisfies

$$(1-\epsilon)v_j - \mathbf{err}_j^{(t')} \leq \boldsymbol{\alpha}_j^{(t')} \leq (1+\epsilon)v_j + 2 \cdot \mathbf{err}_j^{(t')}.$$

Since at least $t/2$ of $t' \in [t]$ satisfies $\mathbf{err}_j^{(t')} \leq \epsilon/2 \cdot \|v\|_p$, the median $\boldsymbol{\alpha}_j^{(t')}$ satisfies the desired error guarantee. Applying a union bound over all dt applications of Corollary 8 and Lemma 10 gives the desired guarantees. ◀

▶ **Lemma 12** (p-stable Sketch Recovery for Optimizer)**.** *For $n, d \in \mathbb{N}$, $p \in [1,2]$ and $\epsilon, \delta \in (0,1)$, let $s = O(\log(d/\delta) \cdot \log(\log d/(\epsilon\delta))/\epsilon^{2+p})$. There exists a distribution $\mathcal{O}$ over $s \times (nd)$ matrices, and an algorithm such that for any vectors $y_1, \ldots, y_n \in \mathbb{R}^d$ and any set of weights $\lambda_1, \ldots, \lambda_n \in [0,1]$ where $\sum_{i=1}^n \lambda_i = 0$, whenever $\sum_{i=1}^n \lambda_i y_i = 0$, the following occurs with probability at least $1 - \delta$:*

- *We sample $\mathbf{S} = [\mathbf{S}_1, \ldots, \mathbf{S}_n] \sim \mathcal{O}$ and we give the algorithm as input $\mathbf{S}$, $\sum_{i=1}^n \mathbf{S}_i(\lambda_i^{1/p} y_i)$, the parameters $\lambda_1, \ldots, \lambda_n$, an index $j_0 \in [d]$, and a parameter $\gamma \in \mathbb{R}_{\geq 0}$ satisfying*

$$(1-\epsilon)\left(\sum_{i=1}^n \lambda_i |y_{ij_0}|^p\right)^{1/p} \leq \gamma \leq (1+\epsilon)\left(\sum_{i=1}^n \lambda_i |y_{ij_0}|^p\right)^{1/p}.$$

- *The algorithm outputs a parameter $\widehat{\boldsymbol{\beta}} \in \mathbb{R}_{\geq 0}$ which satisfies*

$$(1-\epsilon)\min_{z\in\mathbb{R}}\left(\sum_{i=1}^n \lambda_i |y_{ij_0} - z|^p\right)^{1/p} - \mathrm{err} \leq \widehat{\boldsymbol{\beta}} \leq (1+\epsilon)\min_{z\in\mathbb{R}}\left(\sum_{i=1}^n \lambda_i |y_{ij_0} - z|^p\right)^{1/p} + \mathrm{err},$$

where $\mathrm{err} \in \mathbb{R}_{\geq 0}$ is an additive error satisfying

$$\mathrm{err} \leq \epsilon\left(\sum_{j=1}^d \sum_{i=1}^n \lambda_i |y_{i,j}|^p\right)^{1/p}.$$

Proof. The proof follows similarly to that of Lemma 11; the only difference is that instead of using the sketch of Corollary 8, we use the sketch of Lemma 9. For completeness, we describe the distribution $\mathcal{O}$ over $s \times (nd)$ matrices by giving a procedure for sampling $\mathbf{S} \sim \mathcal{O}$:

- We let $\mathcal{M}$ be the distribution over $s_0 \times n$ matrices from Lemma 9 with accuracy ϵ and failure probability $\delta/(2t)$, where $s_0 = O(\log(t/(\epsilon\delta))/\epsilon^2)$. We take d independent samples $\mathbf{S}'_1, \ldots, \mathbf{S}'_d \sim \mathcal{M}$. Note that even though we take d independent samples, we will only require that the sketch t evaluations of the $\mathbf{S}_{j_0} \sim \mathcal{M}$ succeed (hence, we amplify the error probability to $\delta/(2t)$, as opposed to $\delta/(2td)$ as in Lemma 11).
- We sample $\mathbf{C} \sim \mathcal{C}$ as in Lemma 10 with $m = s_0$, where we set the accuracy parameter $\epsilon/2$ and failure probability $\delta/2$. Recalling the definition of P (see Item 2 in the proof of Lemma 11, we let

$$\mathbf{S} = \mathbf{C} \cdot \mathrm{diag}(\mathbf{S}'_1, \ldots, \mathbf{S}'_d) \cdot P.$$

Similarly to the proof of Lemma 11, $\mathbf{S}$ may be interpreted as applying the sketch of Lemma 10 to d vectors in $\mathbb{R}^{s_0}$, each $j \in [d]$ of which is an independent sketch $\mathbf{S}'_j(\lambda^{1/p} \circ y_{\cdot,j}) \in \mathbb{R}^{s_0}$, where $\mathbf{S}'_j \sim \mathcal{M}$ is the sketch of Lemma 9. Again, we consider the collection of d vectors $\boldsymbol{z}_j = \mathbf{S}'_j(\lambda^{1/p} \circ y_{\cdot,j}) \in \mathbb{R}^{s_0}$, for all $j \in [d]$, and by Lemma 9, every $j \in [d]$ has $\boldsymbol{z}_j \sim \|\lambda^{1/p} \circ y_{\cdot,j}\|_p \cdot \boldsymbol{\Psi}_j \in \mathbb{R}^{s_0}$, where $\boldsymbol{\Psi}_j$ is an independent, p-stable random vector. Writing $v \in \mathbb{R}^d$ by $v_j = \|\lambda^{1/p} \circ y_{\cdot,j}\|_p$, and we apply the algorithm of Lemma 10 and focus on the t vectors $\widehat{\boldsymbol{z}}_{j_0}^{(1)}, \ldots, \widehat{\boldsymbol{z}}_{j_0}^{(t)} \in \mathbb{R}^{s_0}$ which satisfy

$$\widehat{\boldsymbol{z}}_{j_0}^{(t')} = \mathbf{S}'_{j_0}(\lambda^{1/p} \circ y_{\cdot,j_0}) + \mathbf{err}_{j_0}^{(t')} \cdot \boldsymbol{\chi}_{j,t'}.$$

We apply the algorithm of Lemma 9 to each of the vectors $\widehat{\boldsymbol{z}}_{j_0} \in \mathbb{R}^s$, while giving as input the parameter γ to obtain the estimate $\widehat{\boldsymbol{\beta}}^{(1)}, \ldots, \widehat{\boldsymbol{\beta}}^{(t)}$. Then, we set $\widehat{\boldsymbol{\beta}} = \mathrm{median}\{\widehat{\boldsymbol{\beta}}^{(t')} : t' \in [t]\}$. Similarly to the proof of Lemma 11, $\widehat{\boldsymbol{\beta}}$ provides the desired approximation guarantees. ◀

References

1 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 2005.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

3 Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David Woodruff. Efficient sketches for earth-mover distance, with applications. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2009)*, 2009.

4 Alexandr Andoni, Moses Charikar, Ofer Neiman, and Huy L. Nguyen. Near linear lower bound for dimension reduction in l1. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '2011)*, 2011.

5 Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA '2008)*, pages 343–352, 2008.

6 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS '2010)*, 2010.

7 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms from precision sampling. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '2011)*, 2011.

8 Alexandr Andoni, Robert Krauthgamer, and Ilya Razenshteyn. Sketching and embedding are equivalent for norms. In *Proceedings of the 47th ACM Symposium on the Theory of Computing (STOC '2015)*, pages 479–488, 2015. Available as `arXiv:1411.2577`.

9 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of euclidean k-means. In *31st International Symposium on Computational Geometry (SoCG 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

10 Arturs Backurs, Piotr Indyk, Ilya Razenshteyn, and David P. Woodruff. Nearly-optimal bounds for sparse recovery in generic norms, with applications to k-median sketching. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA '2016)*, pages 318–337, 2016. Available as `arXiv:1504.01076`.

11 Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th ACM Symposium on the Theory of Computing (STOC '2002)*, 2002.

12 Ziv Bar-Yossef, T.S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

13 Arturs Bačkurs and Piotr Indyk. Better embeddings for planar earth-mover distance over sparse sets. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP '2014)*, 2014.

14 Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: Beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51th ACM Symposium on the Theory of Computing (STOC '2019)*, 2019.

15 Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '2021)*, 2021.

16 Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

17 Guy Blanc, Neha Gupta, Jane Lange, and Li-Yang Tan. Estimating decision tree learnability with polylogarithmic sample complexity. In *Proceedings of Advances in Neural Information Processing Systems 33 (NeurIPS '2020)*, 2020.

18 Jaroslaw Blasiok, Vladimir Braverman, Stephen R. Chestnut, and Robert Krauthgamerand Lin F. Yang. Streaming symmetric norms via measure concentration. In *Proceedings of the 50th ACM Symposium on the Theory of Computing (STOC '2017)*, 2017.

19 Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. Random projections for k-means clustering. In *Proceedings of Advances in Neural Information Processing Systems 23 (NeurIPS '2010)*, 2010.

20 Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint*, 2016. `arXiv:1612.00889`.

21 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning (ICML '2017)*, 2017.

22 Bo Brinkman and Moses Charikar. On the impossibility of dimension reduction in ℓ_1. *Journal of the ACM*, 52(5):766–788, 2005.

23 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

24 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

25 Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. New streaming algorithms for high dimensional emd and mst. In *Proceedings of the 54th ACM Symposium on the Theory of Computing (STOC '2022)*, 2022.

26 Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear algorithms and lower bounds for metric tsp cost estimation. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

27 Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Mădălina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the 47th ACM Symposium on the Theory of Computing (STOC '2015)*, 2015.

28 Vincent Cohen-Addad and CS Karthik. Inapproximability of clustering in lp metrics. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 519–539. IEEE, 2019.

29 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In *Proceedings of the 53rd ACM Symposium on the Theory of Computing (STOC '2021)*, 2021.

30 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

31 Artur Czumaj, Funda Engün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 2005.

32 Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. Streaming facility location in high dimension via new geometric hashing. *arXiv preprint*, 2022. `arXiv:2204.02095`.

33 Artur Czumaj and Chirstian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 2009.

34 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on the Theory of Computing (STOC '2011)*, 2011.

35 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA '2013)*, 2013.

36 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th ACM Symposium on the Theory of Computing (STOC '2004)*, 2004.

37 Monika Henzinger and Sagar Kale. Fully-dynamic coresets. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

38 Wei Hu, Zhao Song, Lin F. Yang, and Peilin Zhong. Nearly optimal dynamic k-means clustering for high-dimensional data. *arXiv preprint*, 2019. `arXiv:1802.00459`.

39 Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. In *Proceedings of the 52nd ACM Symposium on the Theory of Computing (STOC '2020)*, 2020.

40 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th ACM Symposium on the Theory of Computing (STOC '2004)*, 2004.

41 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM*, 53(3):307–323, 2006.

42 Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th ACM Symposium on the Theory of Computing (STOC '2005)*, pages 202–208, 2005.

43 Rajesh Jayaram and David Woodruff. Perfect lp sampling in a data stream. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2018)*, 2018.

44 Thathachar S. Jayram and David Woodruff. The data stream complexity of cascaded norms. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2009)*, 2009.

45 William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. AMS, 1984.

46 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA '2010)*, 2010.

47 Michael Kapralov, Slobodan Mitrović, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1753–1772. SIAM, 2020.

48 Weihao Kong, Emma Brunskill, and Gregory Valiant. Sublinear optimal policy value estimation in contextual bandits. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS '2020)*, 2020.

49 Weihao Kong and Gregory Valiant. Estimating learnability in the sublinear data regime. In *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS '2018)*, pages 5455–5464, 2018.

50 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. A simple linear time $(1+\epsilon)$-approximation algorithm for k-means clustering in any dimension. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2004)*, 2004.

51 Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

52 Michael Langberg and Leonard J. Schulman. Universal epsilon-approximators for integrals. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA '2010)*, 2010.

53 Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Information Processing Letters*, 120:40–43, 2017.

54 James R. Lee and Assaf Naor. Embedding the diamond graph in L_p and dimension reduction in L_1. *Geometric and Functional Analysis*, 14(4):745–747, 2004.

55 Konstantin Makarychev, Yuri Makarychev, and Ilya Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51th ACM Symposium on the Theory of Computing (STOC '2019)*, 2019.

56 Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336. IEEE, 2008.

57 Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1123–1131. SIAM, 2012.

58 Art B. Owen. Monte carlo theory, methods, and examples, 2013.

59 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

60 Michael Saks and Xiaodong Sun. Space lower bounds for distance approximation in the data stream model. In *Proceedings of the 34th ACM Symposium on the Theory of Computing (STOC '2002)*, 2002.

61 Christian Sohler and David Woodruff. Strong coresets for k-median and subspace approximation. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2018)*, 2018.

62 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 225–234, 2009.

Approximation Algorithms for Interdiction Problem with Packing Constraints

Lin Chen[1]
Department of Computer Science, Texas Tech University, Lubbock, TX, USA

Xiaoyu Wu
School of Mathematical Sciences, Zhejiang University, Hangzhou, China

Guochuan Zhang
School of Computer Science, Zhejiang University, Hangzhou, China

Abstract

We study a bilevel optimization problem which is a zero-sum Stackelberg game. In this problem, there are two players, a leader and a follower, who pick items from a common set. Both the leader and the follower have their own (multi-dimensional) budgets, respectively. Each item is associated with a profit, which is the same to the leader and the follower, and will consume the leader's (follower's) budget if it is selected by the leader (follower). The leader and the follower will select items in a sequential way: First, the leader selects items within the leader's budget. Then the follower selects items from the remaining items within the follower's budget. The goal of the leader is to minimize the maximum profit that the follower can obtain. Let s_A and s_B be the dimension of the leader's and follower's budget, respectively. A special case of our problem is the bilevel knapsack problem studied by Caprara et al. [SIAM Journal on Optimization, 2014], where $s_A = s_B = 1$. We consider the general problem and obtain an $(s_B + \epsilon)$-approximation algorithm when s_A and s_B are both constant. In particular, if $s_B = 1$, our algorithm implies a PTAS for the bilevel knapsack problem, which is the first $\mathcal{O}(1)$-approximation algorithm. We also complement our result by showing that there does not exist any $(4/3 - \epsilon)$-approximation algorithm even if $s_A = 1$ and $s_B = 2$. We also consider a variant of our problem with resource augmentation when s_A and s_B are both part of the input. We obtain an $\mathcal{O}(1)$-approximation algorithm with $\mathcal{O}(1)$-resource augmentation, that is, we give an algorithm that returns a solution which exceeds the given leader's budget by $\mathcal{O}(1)$ times, and the objective value achieved by the solution is $\mathcal{O}(1)$ times the optimal objective value that respects the leader's budget.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Bilevel Integer Programming, Interdiction Constraints, Knapsack

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.39

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2204.11106` [7]

Funding *Lin Chen*: NSF No. 2004096
Guochuan Zhang: NSFC Key Program No. 12131003

1 Introduction

In recent years, there is an increasing interest in adopting the *Stackelberg competition model* [16] to address the critical security concern that arises in protecting our ports, airports, transportation, and other critical national infrastructures (see, e.g., [1, 29, 33]). In these

[1] Optional footnote, e.g. to mark corresponding author

problems, the attacker's target is to maximize the illicit gain, while the defender tries to mitigate the attack by minimizing the attacker's objective through deploying defending resources.

In this paper, we consider an abstract model for general defending problems called *interdiction with packing constraints* (IPC). In IPC, given are a set of items, together with a *leader* and a *follower*. Both the leader and the follower have their own (multi-dimensional) budgets, respectively. Each item is associated with a profit, which is the same to the leader and the follower, and will consume the leader's (follower's) budget if it is selected by the leader (follower). The leader and the follower will select items in a sequential way: First, the leader selects items within the leader's budget. Then the follower selects items from the remaining items within the follower's budget. The goal of the leader is to minimize the maximum profit that the follower can obtain. IPC captures the general setting where the follower is the attacker who gets profit by attacking items, and the leader is the defender who tries to minimize the attacker's gain by protecting a subset of items.

IPC can be formulated as a bilevel integer program (IP) as follows. Denote by $I = \{1, 2, \cdots, n\}$ the set of items. Each item $j \in I$ is associated with a *profit* $p_j \in \mathbb{Q}_{>0}$, an s_A-dimensional *cost vector* $\mathbf{A}_j \in \mathbb{Q}_{\geq 0}^{s_A}$ to the leader and an s_B-dimensional *weight vector* $\mathbf{B}_j \in \mathbb{Q}_{\geq 0}^{s_B}$ to the follower. The leader and the follower have their own *budget vectors*, denoted by $\mathbf{a} \in \mathbb{Q}_{\geq 0}^{s_A}$ and $\mathbf{b} \in \mathbb{Q}_{\geq 0}^{s_B}$, respectively. We introduce 0-1 variables x_j and y_j for each $j \in I$ as the decision variables for the leader and the follower. More precisely, if the leader chooses item j, then $x_j = 1$, otherwise $x_j = 0$. Similarly, $y_j = 1$ if the follower chooses item j and $y_j = 0$ otherwise. Denote by $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ and $\mathbf{1} = (\underbrace{1, \ldots, 1}_{n})$. IPC can be formulated as a bilevel program $\mathbf{IPC}(I, \mathbf{a}, \mathbf{b})$ as follows:

$$\mathbf{IPC}(I, \mathbf{a}, \mathbf{b}) : \min_{\mathbf{x}} \ \mathbf{py} \tag{1a}$$

$$s.t.\ \mathbf{Ax} \leq \mathbf{a} \tag{1b}$$

$$\mathbf{x} \in \{0,1\}^n \tag{1c}$$

where $\mathbf{y}$ solves the following:

$$\max_{\mathbf{y}} \quad \mathbf{py} \tag{1d}$$

$$s.t. \quad \mathbf{By} \leq \mathbf{b} \tag{1e}$$

$$\mathbf{x} + \mathbf{y} \leq \mathbf{1} \tag{1f}$$

$$\mathbf{y} \in \{0,1\}^n \tag{1g}$$

where $\mathbf{A} = (\mathbf{A}_1, \cdots, \mathbf{A}_n)$ and $\mathbf{B} = (\mathbf{B}_1, \cdots, \mathbf{B}_n)$ are $s_A \times n$ and $s_B \times n$ non-negative rational matrices, respectively.

The most relevant prior work to our IPC model is the well-known knapsack interdiction problem introduced by DeNegre [14], which is the special case of IPC where $s_A = 1$ and $s_B = 1$. Very recently, Caprara et al. [4] proved that DeNegre's knapsack interdiction problem is $\sum_2^p$-complete and strongly NP-hard, which also implies the $\sum_2^p$-completeness and strongly NP-hardness for IPC. Caprara et al. showed a polynomial time approximation scheme (PTAS) for a special case of knapsack interdiction problem where the profit of an item is equal to its weight to the follower.

Except for the knapsack interdiction problem, we are not aware of any approximation algorithms for other special cases of IPC. However, if we relax the follower's problem by allowing $\mathbf{y}$ to take fractional value, then there are several research works in the literature. The most relevant work is the packing interdiction problem studied by Dinitz and Gupta [15],

where the follower's problem is given by $\max\{\sum_j p_j(1-x_j)y_j : B\mathbf{y} \leq \mathbf{b}, \mathbf{y} \geq \mathbf{0}\}$, while the leader's constraints are the same as Equation 1b and Equation 1c except that $s_A = 1$. Dinitz and Gupta provided an approximation algorithm whose ratio depends on the sparsity of the matrix B. Their techniques crucially rely on the fact that $\mathbf{y}$ can take fractional value, and therefore duality theory can be applied to the follower's problem, allowing the bilevel problem to be transformed to a single level problem. Besides the packing interdiction problem, quite a few graph interdiction problems have been studied in the literature, where the follower's problem is a standard graph optimization problem, and the leader can remove edges or vertices to minimize the follower's optimal objective value on the graph after edge-removal or vertex-removal. On planar graphs, polynomial time approximation schemes (PTASs) were obtained for network flow interdiction [32, 37] and matching interdiction [28]. On general graphs, approximation algorithms were also obtained for, e.g., connectivity interdiction [38], minimum spanning tree interdiction [26, 39], matching interdiction [15, 36], network flow interdiction [3, 9, 10], etc. All of these algorithms crucially rely on the follower's specific graph optimization problem and do not apply directly to IPC.

Our Contributions

The main contribution of this paper is an $(s_B + \epsilon)$-approximation polynomial time algorithm for IPC when s_A and s_B are both constant. In particular, when $s_B = 1$, our algorithm is a PTAS. Since the knapsack interdiction problem is a special case of IPC when $s_A = s_B = 1$, our result gives the first $\mathcal{O}(1)$-approximation algorithm for this problem. To complement our result, we also show that IPC does not admit any $(4/3 - \epsilon)$-approximation algorithm even if $s_B = 2$ and $s_A = 1$, assuming $P \neq NP$. This implies that the PTAS for $s_B = 1$ cannot be further extended to the case of $s_B \geq 2$.

We also consider a natural variant of IPC where the leader's budget can be violated. For this variant we obtain a $(\frac{\rho}{1-\alpha}, \frac{1}{\alpha})$-bicriteria approximation algorithm for any $\alpha \in (0, 1)$, which runs in polynomial time when s_A and s_B are arbitrary (not necessarily polynomial in the input size). More precisely, the algorithm takes as input two oracles, a ρ-approximation algorithm to the follower's optimization problem $\max\{\mathbf{py} : \mathbf{By} \leq \mathbf{b}, \mathbf{y} \in \{0,1\}^n\}$; and a separation oracle for the leader's problem that given any $\mathbf{x} = \mathbf{x}^0$, it either asserts that $\mathbf{Ax}^0 \leq \mathbf{a}$ or returns a violating constraint. Then in polynomial oracle time the algorithm returns a solution $\mathbf{x}^*$ for the leader such that $\mathbf{Ax}^* \leq \frac{1}{\alpha}\mathbf{a}$, and the objective value is at most $\frac{\rho T^*}{1-\alpha}$, where T^* is the optimal objective value with the leader's budget being $\mathbf{a}$. When we take, e.g., $\alpha = 1/2$, we achieve an objective of $2\rho T^*$ with the leader's budget augmented to $2\mathbf{a}$.

In terms of techniques, our main contribution is a general method for bilevel optimization problems where the leader's and follower's decision variables are both integral. Most prior works on bilevel optimization require follower's decision variables to take fractional values, which accommodates the application of LP duality to transform the bilevel optimization problem to a standard (single level) optimization, and are thus inapplicable when the follower's decision variables become integral. A common technique used in many single level optimization problems is to first classify items into large and small based on whether they can make a significant contribution to the objective value, then guess out large items via enumeration, and handle small items fractionally via LP (see, e.g., [5, 21, 22]. However, such a technique encounters a fundamental challenge in IPC: we can guess out all large items selected by the leader, however, the follower may still select arbitrarily from the remaining large items. In other word, the follower's choice on large items can never be guessed out, and therefore we cannot apply duality to the follower's problem. We overcome the challenge

based on the following two ideas: First, we show that given leader's choice on large items, there is a fixed number of "dominant choices" such that the follower's choice on remaining large items always belong to the dominant choices. Second, we show that there exists a subset of "critical items" such that the follower's choice on small items can be characterized through linear constraints given that these critical items are known. The two observations allow us to transform the bilevel program for IPC to an LP without utilizing duality.

The characterization of dominant choices and critical items become sophisticated in the general case when s_B is an arbitrary constant, but is much simpler in the special case $s_B = 1$. Hence, for ease of presentation, in the main part we present our algorithm for the special case to give an overview on the technical insights, and meanwhile provide a proof sketch towards generalizing the algorithm for the general case. Our techniques may be of separate interest to other bilevel optimization problems.

Related work

Our IPC problem lies generally in the area of *bilevel optimization*, which has received extensive research in the literature. Jeroslow [23] showed that in general, bilevel optimization problems are NP-hard even when the objectives and the constraints are linear. We refer readers to Colson et al. [11] for a comprehensive survey on bilevel optimization.

Within the area of bilevel optimization, *Mixed-Integer Bilevel Linear Problem* (MIBLP) is related to our IPC. MIBLP is a bilevel optimization problem where the objective functions and the constraints for the leader and follower are both linear. MIBLP has been studied extensively in the literature, see, e.g., [19, 20, 35]. We also refer the reader to [18, 25] for an overview on MIBLP solvers and related applications. Most of these algorithmic results are for finding exact solutions through, e.g., branch and bound based approach. For DeNegre's knapsack interdiction problem, an improved exact algorithm was derived by Federico Della Croce and Rosario Scatamacchia [12].

It is worth mentioning that besides DeNegre's knapsack interdiction problem (i.e., $s_A = s_B = 1$ in IPC), other variants of bilevel knapsack problems have also been studied in which the leader interferes the follower's program in a different way. One kind of bilevel knapsack problem was introduced by Dempe and Richter [13] where two players hold one knapsack, the leader determines the knapsack's capacity while the follower picks items into the knapsack to maximize his own total profit. The goal is to maximize the objective of the leader. Brotcorne et al. [2] gave a dynamic programming algorithm for both cases of this model. Chen and Zhang [8] proposed a bilevel knapsack variant where two players hold their own knapsacks and the leader can only influence the profit of the items. The follower is interested in his own revenue while the leader aims at maximizing the total profit of both players. The improved approximation results for this problem were derived by Xian Qiu and Walter Kern [34]. Another bilevel knapsack variant occurred in the work of Pferschy et al. [30] where the leader controls the weights of a subset of the follower's items and the follower aims at maximizing his own profit. The leader's payoff is the total weight of the items he controls and selected by the follower. Very recently, Pferschy et al. [31] tackled a "symmetrical" problem in which the leader can control the profits instead of item weights. In addition to these works, a matrix interdiction problem was studied by Kasiviswanathan and Pan [24].

It is also worth mentioning that the continuous version of DeNegre's knapsack interdiction problem, where the leader and the follower can both fractionally choose an item, has also been studied in recent years. Carvalho et al. [6] gave the first polynomial time optimal algorithm. Later on, a faster optimal algorithm was proposed by Woeginger and Fischer [17].

Some notations. We write column vectors in boldface, e.g. $\mathbf{x}, \mathbf{y}$, and their entries in normal font. For a vector $\mathbf{x}$, we either denote its entries by $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, or by $\mathbf{x} = (\mathbf{x}[1], \mathbf{x}[2], \cdots, \mathbf{x}[n])$. Given two vectors $\mathbf{x}$ and $\mathbf{y}$ with the same dimension, we use $\mathbf{xy}$ to represent their dot product, i.e., $\mathbf{xy} = \sum_j x_j y_j$.

2 Hardness results

▶ **Theorem 1.** *Assuming $P \neq NP$, for arbitrary small $\epsilon > 0$, there does not exist a $(4/3 - \epsilon)$-approximation polynomial time algorithm for IPC when $s_A \geq 1$ and $s_B \geq 2$.*

Towards the proof, we need the 3 hitting set (3HS) problem.

Problem: 3 Hitting Set

Instance: A ground set $U = \{u_1, u_2, \cdots, u_n\}$; a collection C of m subsets $S_1, S_2, \cdots, S_m$ whose union is U, where each subset S_h contains exactly 3 elements; a positive integer k.

Question: Is there a hitting subset $S \subseteq U$ such that $|S| \leq k$, and S contains at least one element from each subset in C?

Proof. Recall that 3HS problem is a natural generalization of the well-known Vertex Cover problem, and both are NP-complete [27]. Our reduction is from the 3HS problem. Given an instance of the 3HS, we construct an instance of the IPC where $s_A = 1$, $s_B = 2$ as follows. Let $E = 10 \cdot \sum_{i=1}^{n} 10^i$, and Q be any sufficiently large integer, say, $Q = 10E$. Let $\mathbf{a} = k$ and $\mathbf{b} = (E, 4Q - E)$. The profit of every item constructed below is 1. For every element u_i, we construct an *element-item* (item i) whose interdiction cost is 1, and whose weight vector is $(10^i, Q - 10^i)$. For every subset $S_h = \{u_i, u_j, u_k\}$, we construct a *set-item* (item $n + h$) whose interdiction cost is $k + 1$ (that is, the leader cannot interdict a set-item), and whose weight vector is $(E - 10^i - 10^j - 10^k, Q - E + 10^i + 10^j + 10^k)$. In total we construct $n + m$ items.

We first claim that the objective value of any feasible solution for the IPC instance is at most 4. Suppose on the contrary the claim is false, then the follower is able to select at least 5 items under the budget $\mathbf{b} = (E, 4Q - E)$. Notice that for any $1 \leq i \leq n$, $Q - 10^i > Q - E \geq 0.9Q$, and for any $1 \leq i, j, k \leq n$ we have $Q - E + 10^i + 10^j + 10^k > 0.9Q$, if we sum up the weight vectors of any 5 items, then the second coordinate is at least $4.5Q$, which exceeds the budget $4Q - E$, hence the claim is true.

Suppose the 3HS instance admits hitting set S of size at most k, we show that the optimal objective value of the IPC instance is at most 3. Let $S = \{u_{\ell_1}, u_{\ell_2}, \cdots, u_{\ell_k}\}$ (if S contains less than k elements, we simply add arbitrary elements to make it contain exactly k items), then we consider the solution $\mathbf{x}$ where $x_{\ell_i} = 1$ for $1 \leq i \leq k$, and $x_j = 0$ otherwise. We claim that for any $\mathbf{y}$ satisfying $\mathbf{x} + \mathbf{y} \leq \mathbf{1}$, $\mathbf{py} = \sum_j y_j \leq 3$. Suppose on the contrary that the claim is false, then the follower can select at least 4, and hence exactly 4 items (given our claim in the above paragraph that shows the objective value cannot exceed 4). Notice that for every item, if we add the first and second coordinate of its weight vector, then the sum is exactly Q. Hence, if we add up the weight vector of the 4 items, it must be $(z, 4Q - z)$ for some z, and meanwhile, we have $(z, 4Q - z) \leq (E, 4Q - E)$, that is $z \leq E$ and $4Q - z \leq 4Q - E$. Hence, $z = E$, which means the sum of the first coordinate of the weight vectors of the 4 items is exactly $E = 10 \sum_i 10^i$. We first observe that it is impossible for the 4 items to be all element-items, this is because the first coordinate of the weight vector for any element-item is at most $10^n < 0.1E$. We then observe that there cannot be two set-items among the 4 items, because the first coordinate of the weight vector for any set-item is at least $E - 0.1E = 0.9E$. Hence, among the 4 items, there must be exactly 1 set-item and 3 element-items. Let the 3 element-items be those corresponding to u_i, u_j, u_k and the set-item be the one corresponding

to $\{u_{i'}, u_{j'}, u_{k'}\}$, then it follows that $10^i + 10^j + 10^k + E - 10^{i'} - 10^{j'} - 10^{k'} = E$, implying that $\{i, j, k\} = \{i', j', k'\}$. However, this is not possible because the hitting set S contains at least one element from $\{u_{i'}, u_{j'}, u_{k'}\} = \{u_i, u_j, u_k\}$, which implies that $x_i + x_j + x_k \geq 1$, and whereas y_i, y_j, y_k cannot be 1 simultaneously. Thus, the optimal objective value of the IPC instance is at most 3.

Suppose the optimal objective value of the IPC instance is at most 3, we show that the 3HS problem admits a hitting set of size at most k. Let $\mathbf{x}^*$ be the optimal solution for IPC. Consider the set $S^* = \{u_i : x_i^* = 1\}$. Given that $\sum_i x_i \leq k$, we know $|S^*| \leq k$. We claim that S^* is a hitting set. Suppose on the contrary that the claim is false, then there exists some subset $\{u_i, u_j, u_k\}$ such that $S^* \cap \{u_i, u_j, u_k\} = \emptyset$. Then we consider the 3 element-items whose weight vectors are $(10^i, Q - 10^i)$, $(10^j, Q - 10^j)$, $(10^k, Q - 10^k)$, and the set-item whose weight vector is $(E - 10^i - 10^j - 10^k, Q - E + 10^i + 10^j + 10^k)$. It is easy to see that the follower can select all the 4 items, leading to an objective value of 4, contradicting the fact that the optimal objective value is at most 3.

Now suppose there exists a $(4/3 - \epsilon)$-approximation polynomial time algorithm for the IPC. We apply the algorithm to the IPC instance constructed from the 3HS instance. If the 3HS instance admits a hitting set of size at most k, then the approximation algorithm returns a solution with objective value at most $4 - \epsilon < 4$, which means it must return a solution with objective value at most 3. If the 3HS instance does not admit a hitting set of size at most k, then the approximation algorithm returns a solution with objective value at least 4. Hence the polynomial time approximation algorithm can be used to determine whether 3HS problem admits a feasible solution, contradicting the NP-hardness of 3HS problem. ◀

3 A PTAS for IPC where $s_B = 1$ and s_A is a fixed constant

The goal of this section is to prove the following Theorem 2. Theorem 2 is a special case of our main result, however, its proof shares similar key ideas as the general case (where s_A and s_B are arbitrary fixed constants). Therefore, we provide a full presentation to demonstrate the technical insights, and in the next section we will show how to extend the techniques when $s_B \geq 2$.

▶ **Theorem 2.** *When $s_B = 1$ and s_A is an arbitrary fixed constant, there exists a polynomial time approximation scheme for IPC.*

The rest of this section is dedicated to proving the following Lemma 3, which implies Theorem 2 directly by scaling item profits (here we write $\mathbf{IPC}(I, \mathbf{a}, b)$ instead of $\mathbf{IPC}(I, \mathbf{a}, \mathbf{b})$ as $\mathbf{b}$ becomes 1-dimensional given that $s_B = 1$).

▶ **Lemma 3.** *Let OPT be the optimal objective value of $\boldsymbol{IPC}(I, \mathbf{a}, b)$. If $OPT \leq 1$, then for an arbitrarily small number $\epsilon > 0$, there exists a polynomial time algorithm that returns a feasible solution to $\boldsymbol{IPC}(I, \mathbf{a}, b)$ with an objective value of at most $1 + \mathcal{O}(\epsilon)$.*

3.1 Preprocessing

From now on we assume $OPT \leq 1$. Without loss of generality, we further assume that $\max_j p_j \leq 1$.

Scaling. We scale the matrix $\mathbf{A}$ and $\mathbf{B}$ such that $\mathbf{a} = \mathbf{1}$ and $b = 1$. From now on we denote this IPC instance as $\mathbf{IPC}(I, \mathbf{1}, 1)$. Without loss of generality, we further assume that $\max_j B_j \leq 1$.

Rounding down the profits. We apply the standard geometric rounding. Let $\delta > 0$ be some small parameter to be fixed later (in particular, we can choose $\delta = \epsilon^2$). Consider each item profit p_j. If $p_j \leq \delta^2$, we keep it as it is; otherwise $p_j > \delta^2$, we round the profit down to the largest value of the form $\delta^2(1+\delta)^h$. For profits whose values are at least δ^2, simple calculation shows there are at most $\tilde{\mathcal{O}}(1/\delta)$ distinct rounded profits. This rounding scheme introduces an additive loss of at most $\mathcal{O}(\delta)$ times the objective value. For simplicity, we still denote the rounded profits by p_j's.

Item classification. Recall that each item j is associated with a profit p_j and a weight vector $\mathbf{B}_j$. Since $s_B = 1$, we write B_j as its weight.

Classifying Weights: We say an item j has a *large* weight if $B_j > \delta$; otherwise, it has a *small* weight.

Classifying Profits: We say an item j has a *large* profit if $p_j > \delta$; a *medium* profit if $\delta^2 < p_j \leq \delta$; and a *small* profit if $p_j \leq \delta^2$.

We say an item is *large* if it has a large-profit, or a large-weight. Otherwise, the item is *small*. Large items and small items will be handled separately.

Denote by S^* the items selected by the leader in an optimal solution of $\mathbf{IPC}(I, \mathbf{1}, 1)$.

3.2 Handling Large Items

3.2.1 Determining the leader's choice on large items

The goal of this subsection is to guess large items in S^* in polynomial time.

Large-profit small-weight items. Notice that if there are at least $1/\delta$ such items for the follower to select, then selecting any $1/\delta$ of them gives a solution with an objective value strictly larger than 1, contradicting to the assumption that $OPT \leq 1$. Thus S^* must include all except at most $1/\delta - 1$ such items, which can be guessed out via $n^{\mathcal{O}(1/\delta)}$ enumerations. Hence, we have the following observation.

▶ **Observation 4.** *With $n^{\mathcal{O}(1/\delta)}$ enumerations, we can guess out all large-profit small-weight items in S^*.*

Small-profit large-weight items. Notice that the follower can select at most $1/\delta$ items from this subgroup and their total profit is at most $\delta^2 * \frac{1}{\delta} = \delta$. Hence, even if the leader does not select any such item, the objective value can increase by at most δ, which leads to the following observation.

▶ **Observation 5.** *With $\mathcal{O}(\delta)$ additive error, we may assume that S^* does not contain small-profit large-weight items.*

Large/medium-profit large-weight items. Notice that the follower can select at most $1/\delta$ items from this group. Since we are considering the case of $s_B = 1$, if there are two items that are not selected by the leader, and they have the same profit, then the follower always prefers the one with a smaller weight. Hence, we have the following lemma.

▶ **Lemma 6.** *With $n^{\tilde{\mathcal{O}}(1/\delta^2)}$ enumerations, we can guess out all large/medium-profit large-weight items in S^*.*

Proof. Recall that there are at most $\tilde{\mathcal{O}}(1/\delta)$ distinct large/medium profits. Let S_h be the set of large-weight items whose profits are all $\delta^2(1+\delta)^h$. We observe two facts: (i). Among items in $S_h \setminus S^*$, the follower always selects the ones with the smallest weights; (ii). The follower can select at most $1/\delta$ items from $S_h \setminus S^*$. We claim that, $S^* \cap S_h$ can be determined through guessing out the following $1/\delta$ key items in S_h: among items in $S_h \setminus S^*$, which is the item that has the k-th smallest weight for $k = 1, 2, \cdots, 1/\delta$?[2] To see the claim, let w_h^{max} be the weight of the item in $S_h \setminus S^*$ that has the $1/\delta$-th smallest weight. Consider any item in S_h: if its weight is smaller than w_h^{max}, and is not one of the key items, then this item must belong to S^* (by the definition of key items); if its weight is larger than or equal to w_h^{max}, and is not one of the key items, then it is not in S^* (there is no need for the leader to select such an item since the follower will never select this item even if it is available). Thus via $n^{\mathcal{O}(1/\delta)}$ enumerations, we can guess out all large/medium-profit large-weight items in $S^* \cap S_h$. Moreover, via total $n^{\tilde{\mathcal{O}}(1/\delta^2)}$ enumerations, we can guess out all large/medium-profit large-weight items in S^*. ◀

To summarize, our above analysis leads to the following lemma:

▶ **Lemma 7.** *With $\mathcal{O}(\delta)$ additive error, we can guess out all the large items in the optimal solution S^*, i.e., all items that either have a large profit or a large weight, by $n^{\tilde{\mathcal{O}}(\frac{1}{\delta^2})}$ enumerations.*

Let $\bar{I} \subseteq I = \{1, 2, \cdots, n\}$ be the set of small items, i.e., items of medium/small-profit and small-weight. Then $I \setminus \bar{I}$ is the set of large items. Denote by $\mathbf{x}^*$ the optimal solution to $\mathbf{IPC}(I, \mathbf{1}, 1)$, which is corresponding to S^*. In the following we assume a correct guess on large items. Hence, the values of $\{x_j^* : j \in I \setminus \bar{I}\}$ are known. We let $\mathbf{a}'$ be the total cost of these guessed-out large items.

3.2.2 Finding the follower's dominant choices on large items

Consider all the large items. Even if the leader's choice on large items is fixed, the follower may still have exponentially many different choices on the remaining large items. The goal of this subsection is to show that, among these choices of the follower, it suffices to restrict our attention to a few "dominant" choices that always outperform other choices.

For simplicity, we re-index items such that $\bar{I} = \{1, 2, \cdots, \bar{n}\}$, where $\bar{n} \leq n$.

We further assume that items in $\bar{I}$ are sorted in decreasing order of the profit-weight ratios p_j/B_j. For any $\bar{\mathbf{a}} \leq \mathbf{1}$ and $\bar{b} \leq 1$, denote by $\mathbf{IPC}(\bar{I}, \bar{\mathbf{a}}, \bar{b})$ the "residual instance" where the item set is $\bar{I}$, the budget vector of the leader is $\bar{\mathbf{a}}$ and the budget of the follower is $\bar{b}$.

Denote by $I' \subseteq I \setminus \bar{I}$ the subset of large items which are *not* selected by the leader. Note that due to the assumption $OPT \leq 1$ and that we have guessed out correct large items in S^*, the follower cannot select items from I' with total profit larger than 1. Hence, for each integer $k \in [1, 1 + 1/\epsilon]$, we can define the following sub-problem: among items in I', find out a subset of items with minimal total weight such that their total profit is within $[(k-1)\epsilon, k\epsilon)$. Denote by $SP(k)$ this sub-problem and by $KP(k\epsilon)$ its optimal solution, if it exists. We claim that the follower can select at most $\mathcal{O}(1/\delta)$ items from I', thus via $n^{\mathcal{O}(1/\delta)}$ enumerations, we can return $KP(k\epsilon)$ or assert there does not exist a feasible solution to $SP(k)$. The claim is guaranteed by the following two facts: (i). The total profit the follower could obtain from I' is at most 1; (ii). Items in I' either have a large-profit, or a large-weight.

[2] If there are less than $1/\delta$ items in S_h, we can simply guess out all items in $S_h \setminus S^*$ via $n^{\mathcal{O}(\frac{1}{\delta})}$ enumerations.

Let $\Theta = \{KP(k\epsilon) : k \in \{1, 2, \cdots, 1 + \frac{1}{\epsilon}\}\}$, which contains the follower's $\mathcal{O}(1/\epsilon)$ possible choices on I'. For $\ell \in \{1, 2, \cdots, 1 + \frac{1}{\epsilon}\}$, we let b_ℓ and P_ℓ be the total weight and the total profit of the items selected by the follower, respectively[3]. Then the follower has a residual budget of $1 - b_\ell$ for items in $\bar{I}$. Recall that the leader has a residual budget vector of $\mathbf{1} - \mathbf{a}'$ for items in $\bar{I}$.

Define $\mathbf{y}[\bar{I}] = (y_1, y_2 \cdots, y_{\bar{n}})$. Recall that by guessing we already know the value of x_j^* for $j \in I \setminus \bar{I}$. Consider the following bilevel program:

$$\textbf{Bi-IP}(I, \mathbf{1}, 1) : \min_{\mathbf{x}} P_\ell + \sum_{j=1}^{\bar{n}} p_j y_j$$

$$s.t. \sum_{j=1}^{\bar{n}} \mathbf{A}_j x_j \le \mathbf{1} - \mathbf{a}' \tag{2a}$$

$$x_j = x_j^*, \ \ \forall j \in I \setminus \bar{I} \tag{2b}$$

$$x_j \in \{0, 1\}, \ \ \forall j \in \bar{I} \tag{2c}$$

where integer $\ell, \mathbf{y}[\bar{I}]$ solves the following:

$$\max_{1 \le \ell \le 1 + \frac{1}{\epsilon}} \max_{\mathbf{y}[\bar{I}]} \ \ P_\ell + \sum_{j=1}^{\bar{n}} p_j y_j \tag{2d}$$

$$s.t. \qquad \sum_{j=1}^{\bar{n}} B_j y_j \le 1 - b_\ell \tag{2e}$$

$$y_j \le 1 - x_j, \ \ \forall j \in \bar{I} \tag{2f}$$

$$y_j \in \{0, 1\}, \ \ \forall j \in \bar{I} \tag{2g}$$

What is the difference between Bi-IP$(I, \mathbf{1}, 1)$ and IPC$(I, \mathbf{1}, 1)$, assuming the correct guess of x_j^* for $j \in I \setminus \bar{I}$? In **Bi-IP**$(I, \mathbf{1}, 1)$, the follower's choices on remaining large items are restricted to the $\mathcal{O}(1/\epsilon)$ choices in Θ, while in **IPC**$(I, \mathbf{1}, 1)$, the follower can choose any remaining large items. However, we observe that Θ contains all the follower's "dominant choices of remaining large items" in the sense that the follower uses the smallest budget to achieve a profit within $[(k-1)\epsilon, k\epsilon)$. Consequently, the objective value of **Bi-IP**$(I, \mathbf{1}, 1)$ differs by at most ϵ to that of **IPC**$(I, \mathbf{1}, 1)$. A formal description is given below.

▶ **Lemma 8.** *Let $\bar{\mathbf{x}}$ be any feasible solution to **Bi-IP**$(I, \mathbf{1}, 1)$. Then $\bar{\mathbf{x}}$ is also feasible to **IPC**$(I, \mathbf{1}, 1)$. Let $Obj_{Bi}(\bar{\mathbf{x}})$ and $Obj(\bar{\mathbf{x}})$ be the objective values of **Bi-IP**$(I, \mathbf{1}, 1)$ and **IPC**$(I, \mathbf{1}, 1)$ for $\mathbf{x} = \bar{\mathbf{x}}$, respectively. We have*

$$Obj_{Bi}(\bar{\mathbf{x}}) \le Obj(\bar{\mathbf{x}}) \le Obj_{Bi}(\bar{\mathbf{x}}) + \epsilon.$$

*Furthermore, let OPT_{Bi} and OPT be the optimal objective values of **Bi-IP**$(I, \mathbf{1}, 1)$ and **IPC**$(I, \mathbf{1}, 1)$, respectively, then we have*

$$OPT_{Bi} \le OPT \le OPT_{Bi} + \epsilon.$$

Proof. Compare the follower's possible choices in **Bi-IP**$(I, \mathbf{1}, 1)$ and **IPC**$(I, \mathbf{1}, 1)$ when the leader's solution is fixed to $\bar{\mathbf{x}}$. It is easy to see that in **IPC**$(I, \mathbf{1}, 1)$, the follower's feasible choices on the remaining large items contain Θ, it thus follows that $Obj_{Bi}(\bar{\mathbf{x}}) \le Obj(\bar{\mathbf{x}})$. Particularly, since the optimal solution $\mathbf{x}^*$ of **IPC**$(I, \mathbf{1}, 1)$ is a feasible solution of **Bi-IP**$(I, \mathbf{1}, 1)$ and the optimal solution of **Bi-IP**$(I, \mathbf{1}, 1)$ may achieve an even smaller value, it follows that $OPT_{Bi} \le OPT$. It remains to prove that $Obj(\bar{\mathbf{x}}) \le Obj_{Bi}(\bar{\mathbf{x}}) + \epsilon$ and $OPT \le OPT_{Bi} + \epsilon$.

[3] If there is no feasible solution to $SP(\ell)$, we let $b_\ell = 1$ and $P_\ell = 0$.

Note that $Obj(\bar{\mathbf{x}})$ is exactly the optimal objective value of the following integer program:

$$\begin{aligned}\mathbf{IP}(\bar{\mathbf{x}}): \max_{\mathbf{y}} \quad & \mathbf{py} \\ s.t. \quad & \sum_{j=1}^{n} B_j y_j \le 1 \\ & \mathbf{y} \le \mathbf{1} - \bar{\mathbf{x}} \\ & \mathbf{y} \in \{0,1\}^n\end{aligned}$$

Let $\bar{\mathbf{y}}$ be an optimal solution of $\mathbf{IP}(\bar{\mathbf{x}})$, then $Obj(\bar{\mathbf{x}}) = \sum_{j\in I\setminus\bar{I}} p_j\bar{y}_j + \sum_{j\in\bar{I}} p_j\bar{y}_j$. Recall that $\mathbf{x}^*$ is an optimal solution of $\mathbf{IPC}(I,\mathbf{1},1)$, and $\bar{\mathbf{x}}_j = \mathbf{x}^*_j$ for $j \in I\setminus\bar{I}$ by (2b). Consequently, if we compare the follower in $\mathbf{IPC}(I,\mathbf{1},1)$ and the follower in $\mathbf{Bi\text{-}IP}(I,\mathbf{1},1)$, the subset of items in $I\setminus\bar{I}$ available for the two followers to select is the same, and we let this subset be $R = \{j : x^*_j = 0, j \in I\setminus\bar{I}\}$. Given that we assume $OPT \le 1$, the maximal profit the follower could obtain from R is at most 1, thus there exists some integer $\bar{\ell} \in [1, 1+1/\epsilon]$ such that $\sum_{j\in I\setminus\bar{I}} p_j\bar{y}_j \in [(\bar{\ell}-1)\epsilon, \bar{\ell}\epsilon)$. By the definitions of $P_{\bar{\ell}}$ and $b_{\bar{\ell}}$, we have $\sum_{j\in I\setminus\bar{I}} p_j\bar{y}_j \le P_{\bar{\ell}} + \epsilon$ and $b_{\bar{\ell}} \le \sum_{j\in I\setminus\bar{I}} B_j\bar{y}_j$. Define $\mathbf{y}' \in \{0,1\}^n$ such that the $\mathbf{y}'$ is a combination of two partial solutions: in $I\setminus\bar{I}$, $\mathbf{y}'$ is the same as $KP(\bar{\ell}\epsilon)$; and in $\bar{I}$, $\mathbf{y}'$ is the same as $\bar{\mathbf{y}}$. Then $\mathbf{y}'$ is a feasible solution of the following program:

$$\begin{aligned}\overline{\mathbf{IP}}(\bar{\mathbf{x}}): \max_{\ell} \max_{\mathbf{y}[\bar{I}]} \quad & P_\ell + \sum_{j=1}^{\bar{n}} p_j y_j \\ s.t. \quad & \sum_{j=1}^{\bar{n}} B_j y_j \le 1 - b_\ell \\ & y_j \le 1 - \bar{x}_j, \ \ \forall j \in \bar{I} \\ & y_j \in \{0,1\}, \ \ \forall j \in \bar{I}\end{aligned}$$

Notice that the optimal objective value of $\overline{\mathbf{IP}}(\bar{\mathbf{x}})$ is $Obj_{Bi}(\bar{\mathbf{x}})$, thus $\mathbf{py}' = P_{\bar{\ell}} + \sum_{j=1}^{\bar{n}} p_j\bar{y}_j \le Obj_{Bi}(\bar{\mathbf{x}})$. To conclude, we have

$$Obj(\bar{\mathbf{x}}) = \sum_{j\in I\setminus\bar{I}} p_j\bar{y}_j + \sum_{j=1}^{\bar{n}} p_j\bar{y}_j \le P_{\bar{\ell}} + \epsilon + \sum_{j=1}^{\bar{n}} p_j\bar{y}_j \le Obj_{Bi}(\bar{\mathbf{x}}) + \epsilon$$

Particularly, given an optimal solution $\bar{\mathbf{x}}^*$ of $\mathbf{Bi\text{-}IP}(I,\mathbf{1},1)$, we have $Obj(\bar{\mathbf{x}}^*) \le OPT_{Bi} + \epsilon$. Since the optimal solution of $\mathbf{IPC}(I,\mathbf{1},1)$ may achieve an even smaller objective value, it follows that $OPT \le OPT_{Bi} + \epsilon$. Hence Lemma 8 is proved. ◀

3.3 Handling Small Items

According to Lemma 8, to solve $\mathbf{IPC}(I,\mathbf{1},1)$, it suffices to solve $\mathbf{Bi\text{-}IP}(I,\mathbf{1},1)$, which is the goal of this subsection. Towards this, we first obtain a linear relaxation of $\mathbf{Bi\text{-}IP}(I,\mathbf{1},1)$ where both the leader and the follower can select items fractionally. Then we reformulate this bilevel linear relaxation as a single level linear program and find an extreme point optimal fractional solution. Finally we round this fractional solution to obtain a feasible solution to $\mathbf{Bi\text{-}IP}(I,\mathbf{1},1)$ with an objective value of at most $1+\mathcal{O}(\epsilon)$, which is thus also a feasible solution to $\mathbf{IPC}(I,\mathbf{1},1)$ with an objective value of at most $1+\mathcal{O}(\epsilon)$.

Replace (2c) and (2g) in **Bi-IP**$(I, \mathbf{1}, 1)$ with $x_j \in [0,1](\forall j \in \bar{I})$ and $y_j \in [0,1](\forall j \in \bar{I})$, respectively, we obtain a relaxation of **Bi-IP**$(I, \mathbf{1}, 1)$ as follows.

$$\textbf{Bi-IP}_r(I, \mathbf{1}, 1): \min_{\mathbf{x}} P_\ell + \sum_{j=1}^{\bar{n}} p_j y_j$$

$$s.t. \sum_{j=1}^{\bar{n}} \mathbf{A}_j x_j \le \mathbf{1} - \mathbf{a}' \tag{3a}$$

$$x_j = x_j^*, \ \ \forall j \in I \setminus \bar{I} \tag{3b}$$

$$x_j \in [0,1], \ \ \forall j \in \bar{I} \tag{3c}$$

where integer $\ell, \mathbf{y}[\bar{I}]$ solves the following:

$$\max_{1 \le \ell \le 1+\frac{1}{\epsilon}} \max_{\mathbf{y}[\bar{I}]} \quad P_\ell + \sum_{j=1}^{\bar{n}} p_j y_j \tag{3d}$$

$$s.t. \quad \sum_{j=1}^{\bar{n}} B_j y_j \le 1 - b_\ell \tag{3e}$$

$$y_j \le 1 - x_j, \ \ \forall j \in \bar{I} \tag{3f}$$

$$y_j \in [0,1], \ \ \forall j \in \bar{I} \tag{3g}$$

Denote by OPT^r_{Bi} the optimal objective value of **Bi-IP**$_r(I, \mathbf{1}, 1)$. Note that items in $\bar{I}$ are sorted in decreasing order of the profit-weight ratios p_j/B_j. Consider any fixed leader's solution $\mathbf{x} \in [0,1]^n$ and any fixed ℓ, the follower is solving a knapsack problem in the remaining (fractional) items. The maximal objective value of the follower, given $\mathbf{x} \in [0,1]^n$ and ℓ, is obtained by a simple greedy algorithm that selects remaining fractional items in $\bar{I}$ in the natural order of indices (recall that items are re-indexed in non-increasing order of ratios), until the budget $1 - b_\ell$ is exhausted. Note that the greedy algorithm will stop at some (fractional) item when the budget $1 - b_\ell$ is exhausted[4]. We say this item is *critical* and let its index be c_ℓ. Given any fixed $\mathbf{x} \in [0,1]^n$ and ℓ, the maximal objective value of program (3d)-(3g) for $\mathbf{x}$ is

$$P_\ell + \sum_{j=1}^{c_\ell - 1} p_j(1 - x_j) + p_{c_\ell} \frac{1 - b_\ell - \sum_{j=1}^{c_\ell - 1} B_j(1 - x_j)}{B_{c_\ell}}, \tag{4a}$$

where c_ℓ is the critical item given $\mathbf{x}$ and ℓ. The following two formulas are directly given by the definition of critical.

$$\sum_{j=1}^{c_\ell - 1} B_j(1 - x_j) \le 1 - b_\ell \tag{5a}$$

$$B_{c_\ell} + \sum_{j=1}^{c_\ell - 1} B_j(1 - x_j) \ge 1 - b_\ell \tag{5b}$$

We first show that the optimal objective value of the **Bi-IP**$_r(I, \mathbf{1}, 1)$ is at most $OPT_{Bi} + \delta$.

[4] The greedy algorithm may pack all remaining (fractional) items without using up the budget $1 - b_\ell$. To patch this case, we add a dummy item whose profit is 0, cost vector is $\mathbf{0}$ and weight is sufficiently large. We assume the last item $\bar{n}$ is the dummy item.

▶ **Lemma 9.** *Let OPT_{Bi} and OPT^r_{Bi} be the optimal objective values of **Bi-IP**$(I, \mathbf{1}, 1)$ and **Bi-IP**$_r(I, \mathbf{1}, 1)$, respectively, then $OPT^r_{Bi} \leq OPT_{Bi} + \delta$.*

Proof. It is not straightforward if we compare **Bi-IP**$(I, \mathbf{1}, 1)$ with **Bi-IP**$_r(I, \mathbf{1}, 1)$ directly, as both the follower and the leader become stronger in the relaxation (in the sense they can pack items fractionally). Towards this, we introduce an intermediate bilevel program **Bi-IP**$_{in}(I, \mathbf{1}, 1)$, which is obtained by replacing (3c) in **Bi-IP**$_r(I, \mathbf{1}, 1)$ with $x_j \in \{0, 1\}(\forall j \in \bar{I})$, that is, we only allow the follower to select items fractionally but not the leader. Denote by OPT_{in} the optimal objective value of **Bi-IP**$_{in}(I, \mathbf{1}, 1)$.

First, we compare **Bi-IP**$_{in}(I, \mathbf{1}, 1)$ with **Bi-IP**$_r(I, \mathbf{1}, 1)$. We see that in **Bi-IP**$_r(I, \mathbf{1}, 1)$ the follower is facing a stronger leader who is allowed to fractionally pack items, and it thus follows that $OPT^r_{Bi} \leq OPT_{in}$.

Next, we compare **Bi-IP**$_{in}(I, \mathbf{1}, 1)$ and **Bi-IP**$(I, \mathbf{1}, 1)$. Note that the leader's solution must be integral in both programs. Any feasible solution of **Bi-IP**$_{in}(I, \mathbf{1}, 1)$ is a feasible solution of **Bi-IP**$(I, \mathbf{1}, 1)$, and vice versa. Let $\bar{\mathbf{x}}^*$ be an optimal solution of **Bi-IP**$(I, \mathbf{1}, 1)$, then $\bar{\mathbf{x}}^*$ is also a feasible solution of **Bi-IP**$_{in}(I, \mathbf{1}, 1)$. Once the leader fixes his solution as $\bar{\mathbf{x}}^*$ in **Bi-IP**$_{in}(I, \mathbf{1}, 1)$, there exist ℓ and $c_\ell \in \bar{I}$, such that the objective value of **Bi-IP**$_{in}(I, \mathbf{1}, 1)$ is

$$Obj_{in} = P_\ell + \sum_{j=1}^{c_\ell - 1} p_j(1 - \bar{x}^*_j) + p_{c_\ell} \frac{1 - b_\ell - \sum_{j=1}^{c_\ell - 1} B_j(1 - \bar{x}^*_j)}{B_{c_\ell}},$$

where c_ℓ is the critical item corresponding to $\bar{\mathbf{x}}^*$ and ℓ. We have the following two observations:

- $OPT_{Bi} \geq Obj_{in} - p_{c_\ell} \geq Obj_{in} - \delta$. This is because the follower in **Bi-IP**$(I, \mathbf{1}, 1)$ can guarantee an objective value of $P_\ell + \sum_{j=1}^{c_\ell - 1} p_j(1 - \bar{x}^*_j)$, and $p_j \leq \delta$ for $j \in \bar{I}$;
- $OPT_{in} \leq Obj_{in}$. This is because $\bar{\mathbf{x}}^*$ is just a feasible solution of **Bi-IP**$_{in}(I, \mathbf{1}, 1)$, while the optimal solution of the leader may achieve an even smaller objective value.

To summarize, we know $OPT^r_{Bi} \leq OPT_{in} \leq Obj_{in} \leq OPT_{Bi} + \delta$. Lemma 9 is proved. ◀

Given Lemma 9, we are still facing two questions: how can we solve **Bi-IP**$_r(I, \mathbf{1}, 1)$; and even if we obtain a fractional solution to **Bi-IP**$_r(I, \mathbf{1}, 1)$, how can we transform it to an integral solution without incurring a huge loss. Towards this, consider the optimal solution $\mathbf{x}^r$ to **Bi-IP**$_r(I, \mathbf{1}, 1)$. Note that leader's choice on large items is guessed out in **Bi-IP**$_r(I, \mathbf{1}, 1)$. Consider the scenario when the follower adopts the ℓ-th dominant choice on the remaining large items, and recall the definition of critical items (see Equation 4a). Given solution $\mathbf{x}^r$, for any $\ell \in \{1, 2, \cdots, 1 + \frac{1}{\epsilon}\}$ there must exist a critical item. Therefore, there are $1 + \frac{1}{\epsilon}$ critical items corresponding to $\mathbf{x}^r$. The crucial fact is that, while we cannot guess out $\mathbf{x}^r$ directly, we can guess out all the critical items corresponding to $\mathbf{x}^r$. More precisely, with $\bar{n}^{\mathcal{O}(1/\epsilon)}$ enumerations, we can guess out the critical item c^r_ℓ for $\mathbf{x}^r$ and each ℓ. Suppose we have guessed out the correct c^r_ℓ's corresponding to the optimal solution $\mathbf{x}^r$, we consider the following LP:

$$
\begin{aligned}
\mathbf{LP}_{\text{Bi-IP}}: \quad \min_{\mathbf{x},M} \quad & M \\
& \sum_{j=1}^{\bar{n}} \mathbf{A}_j x_j \le \mathbf{1} - \mathbf{a}' \\
& P_\ell + \sum_{j=1}^{c^r_\ell - 1} p_j(1-x_j) + p_{c^r_\ell} \frac{1 - b_\ell - \sum_{j=1}^{c^r_\ell - 1} B_j(1-x_j)}{B_{c^r_\ell}} \le M, \quad \forall\, \ell \in \{1, 2, \cdots, 1+\frac{1}{\epsilon}\} \\
& \sum_{j=1}^{c^r_\ell - 1} B_j(1-x_j) \le 1 - b_\ell, \quad \forall \quad \ell \in \{1, 2, \cdots, 1+\frac{1}{\epsilon}\} \\
& B_{c^r_\ell} + \sum_{j=1}^{c^r_\ell - 1} B_j(1-x_j) \ge 1 - b_\ell, \quad \forall \quad \ell \in \{1, 2, \cdots, 1+\frac{1}{\epsilon}\} \\
& x_j = x^*_j, \quad j \in I \setminus \bar{I} \\
& x_j \in [0,1], \quad j \in \bar{I}
\end{aligned}
$$

We have the following simple observation.

▶ **Observation 10.** *Let M^* and OPT^r_{Bi} be the optimal objective values of $\boldsymbol{LP}_{Bi\text{-}IP}$ and $\boldsymbol{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$, respectively, then $M^* \le OPT^r_{Bi}$.*

Let $\mathbf{x}^r$ be an optimal solution to $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$. The observation follows directly as $\mathbf{x}^r$ together with OPT^r_{Bi} form a feasible solution to $\mathbf{LP}_{\text{Bi-IP}}$.

In the meantime, we also have the following observation.

▶ **Observation 11.** *Let $\{\mathbf{x}^{ex}, M^*\}$ be an extreme point optimal solution to $\boldsymbol{LP}_{Bi\text{-}IP}$, then $\mathbf{x}^{ex}$ is also a feasible solution to $\boldsymbol{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$ whose objective value is at most M^*.*

The observation follows since by the definition of critical, (4a) is the largest profit the follower can achieve. Hence, when $\mathbf{x} = \mathbf{x}^{ex}$ in $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$, the objective value is bounded by M^*. Given the two observations above, we know $\mathbf{x}^{ex}$ is an optimal solution to $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$ and we have $M^* = OPT^r_{Bi}$. Finally, a near-optimal solution to $\mathbf{IPC}(I, \mathbf{1}, 1)$ can be obtained through the optimal solution to $\mathbf{LP}_{\text{Bi-IP}}$, as implied by the following lemma.

▶ **Lemma 12.** *Let $\{\mathbf{x}^{ex}, M^*\}$ be an extreme point optimal solution to $\boldsymbol{LP}_{Bi\text{-}IP}$. Define $\tilde{\mathbf{x}}$ such that $\tilde{x}_j = 1$ if $x^{ex}_j = 1$, and $\tilde{x}_j = 0$ otherwise. Then $\tilde{\mathbf{x}}$ is a feasible solution of $\boldsymbol{IPC}(I, \mathbf{1}, 1)$ with an objective value of at most $OPT + \mathcal{O}(\epsilon)$, where OPT is the optimal objective value of $\boldsymbol{IPC}(I, \mathbf{1}, 1)$.*

Proof. The feasibility of $\tilde{\mathbf{x}}$ to $\mathbf{IPC}(I, \mathbf{1}, 1)$ is straightforward since

$$
\sum_{j=1}^{n} \mathbf{A}_j \tilde{x}_j = \sum_{j \in I \setminus \bar{I}} \mathbf{A}_j x^*_j + \sum_{j \in \bar{I}} \mathbf{A}_j \tilde{x}_j \le \mathbf{a}' + \sum_{j \in \bar{I}} \mathbf{A}_j x^{ex}_j \le \mathbf{a}' + \mathbf{1} - \mathbf{a}' \le \mathbf{1}.
$$

Notice that $\tilde{\mathbf{x}}$ is also feasible for $\mathbf{Bi\text{-}IP}(I, \mathbf{1}, 1)$ and $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$. Let $Obj(\tilde{\mathbf{x}})$, $Obj_{Bi}(\tilde{\mathbf{x}})$ and $Obj^r_{Bi}(\tilde{\mathbf{x}})$ be the objective values of $\mathbf{IPC}(I, \mathbf{1}, 1)$, $\mathbf{Bi\text{-}IP}(I, \mathbf{1}, 1)$ and $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$ by taking $\mathbf{x} = \tilde{\mathbf{x}}$, respectively. Since in $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$, the leader is facing a stronger follower who can select fractional items in $\bar{I}$, it follows that $Obj_{Bi}(\tilde{\mathbf{x}}) \le Obj^r_{Bi}(\tilde{\mathbf{x}})$.

Now we compare the objective values of two solutions to $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$, $\mathbf{x}^{ex}$ and $\tilde{\mathbf{x}}$. It is easy to see that in $\mathbf{x}^{ex}$, there are at most $(s_A + \frac{3(1+\epsilon)}{\epsilon})$ variables taking fractional values, and all these variables are in $\{x^{ex}_j : j \in \bar{I}\}$. So the leader in $\mathbf{x}^{ex}$ selects at most $(s_A + \frac{3(1+\epsilon)}{\epsilon})$

more items in $\bar{I}$, compared with the leader in $\tilde{\mathbf{x}}$. Consequently, the follower in $\mathbf{x}^{ex}$ may select at most $(s_A + \frac{3(1+\epsilon)}{\epsilon})$ less items in $\bar{I}$, compared with the follower in $\tilde{\mathbf{x}}$. Given that the objective value of $\mathbf{x}^{ex}$ is $OPT^r_{Bi} = M^*$, we have that

$$Obj^r_{Bi}(\tilde{\mathbf{x}}) \leq OPT^r_{Bi} + (s_A + \frac{3(1+\epsilon)}{\epsilon})\delta.$$

According to Lemma 8 and Lemma 9, we have $Obj(\tilde{\mathbf{x}}) \leq Obj_{Bi}(\tilde{\mathbf{x}}) + \epsilon$ and $OPT^r_{Bi} \leq OPT_{Bi} + \delta$. In conclusion, we have

$$Obj(\tilde{\mathbf{x}}) \leq OPT_{Bi} + (s_A + 1 + \frac{3(1+\epsilon)}{\epsilon})\delta + \epsilon.$$

Furthermore, $OPT_{Bi} \leq OPT$ by Lemma 8. By setting $\delta = \epsilon^2$, Lemma 12 is proved. ◀

Hence, if $OPT \leq 1$, then a feasible solution with objective value of at most $OPT + \mathcal{O}(\epsilon) \leq 1 + \mathcal{O}(\epsilon)$ is found, thus Lemma 3 is proved, and Theorem 2 follows.

4 Approximation algorithm for IPC where s_B and s_A are constant

In this section, we prove our main result – Theorem 13.

▶ **Theorem 13.** *When s_B and s_A are fixed constants, for an arbitrarily small number $\epsilon > 0$, there exists an $(s_B + \mathcal{O}(\epsilon))$-approximation polynomial time algorithm for IPC.*

Similar to the special case, by scaling item profits it suffices to show the following:

▶ **Lemma 14.** *Let OPT be the optimal objective value of $\boldsymbol{IPC}(I, \mathbf{a}, \mathbf{b})$. If $OPT \leq 1$, then for an arbitrarily small number $\epsilon > 0$, there exists a polynomial time algorithm that returns a feasible solution to $\boldsymbol{IPC}(I, \mathbf{a}, \mathbf{b})$ with an objective value of at most $s_B + \mathcal{O}(\epsilon)$.*

By further scaling the cost vectors and weight vectors, it suffices to find a near-optimal solution for $\mathbf{IPC}(I, \mathbf{1}, \mathbf{1})$.

Major technical challenge. Recall that the key to solving IPC for the special case of $s_B = 1$ is the establishment of $\mathbf{Bi\text{-}IP}(I, \mathbf{1}, 1)$, which is essentially equivalent to $\mathbf{IPC}(I, \mathbf{1}, 1)$. $\mathbf{Bi\text{-}IP}(I, \mathbf{1}, 1)$ is built upon the observation that the follower admits only $\mathcal{O}(\frac{1}{\epsilon})$ dominant choices on large items, where each dominant choice corresponds to the minimal budget needed by the follower to ensure a profit of $[(k-1)\epsilon, k\epsilon)$ where $k \in \{1, 2, \cdots, 1 + \frac{1}{\epsilon}\}$. Because the number of follower's choices on large items is small, its relaxation $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$ has a small number of constraints (see Equation 3e), and therefore we can further transform $\mathbf{Bi\text{-}IP}_r(I, \mathbf{1}, 1)$ to $\mathbf{LP}_{\text{Bi-IP}}$ with a small number of constraints whose extreme point solution promises a good rounding. We aim to follow a similar method, however, when $s_B \geq 2$, we can no longer bound the follower's choices on large items. This is because to achieve a profit of $[(k-1)\epsilon, k\epsilon)$ where $k \in \{1, 2, \cdots, 1 + \frac{1}{\epsilon}\}$, the follower may have a huge number of different choices utilizing different budgets, where the budgets are now vectors instead of numbers, and are thus incomparable. To handle the problem, we use the idea of rounding: let $\mathbf{b} = (\mathbf{b}[1], \mathbf{b}[2], \cdots, \mathbf{b}[s_B])$ and $\mathbf{B}_j = (\mathbf{B}_j[1], \mathbf{B}[2], \cdots, \mathbf{B}[s_B])$. We call the first dimension ($\mathbf{b}[1]$ and $\mathbf{B}_j[1]$'s) the principal dimension. The principal dimension will be treated the same as the special case and will *not* be rounded. The coordinates of other dimensions ($\mathbf{b}[h]$ and $\mathbf{B}_j[h]$'s for $2 \leq h \leq s_B$) will be rounded. Then, we will be able to compare follower's different choices on large items: if there are two choices both achieving profit within $[(k-1)\epsilon, k\epsilon)$ for the same $k \in \mathcal{O}(\frac{1}{\epsilon})$, and furthermore, the summation of their weight vectors share the

same rounded value in each dimension $h \in [2, s_B]$, then the choice with smaller value in the principal dimension of the summed weight vectors dominates the other choice. By doing so, our argument for the special case can be carried over to the principal dimension.

There is one problem with the idea of rounding above, that is, if we round up weight vectors and meanwhile enlarge the follower's budget vector in dimension $h \in [2, s_B]$ (to accommodate the rounding up), the optimal objective value of IPC may increase. However, we are able to show that the optimal objective value only increases by a factor of s_B (see Lemma 15). This explains our approximation ratio of $s_B + \epsilon$.

Below we give a very brief walk through and the reader is referred to the full version [7] for details.

Step 1. We pick a small parameter δ as the rounding precision, keep the coordinates of weight vectors on principal dimension intact, and round the coordinates on other dimensions. We also round the profits. By doing so we obtain a rounded instance $\tilde{I}_\delta$. Then we pick another small parameter τ and enlarge the weight budget on dimension $h \in [2, s_B]$ by a factor of $1 + \tau$. By doing so we obtain:

$$
\begin{aligned}
\mathbf{IPC}_\tau(\tilde{I}_\delta, \mathbf{1}, \mathbf{1}) : \min_{\mathbf{x}} \ & \tilde{\mathbf{p}}\mathbf{y} \\
s.t. \ & \mathbf{A}\mathbf{x} \le \mathbf{1} \\
& \mathbf{x} \in \{0,1\}^n \\
& \text{where } \mathbf{y} \text{ solves the following:} \\
& \max_{\mathbf{y}} \quad \tilde{\mathbf{p}}\mathbf{y} \\
& s.t. \quad \sum_{j=1}^{n} \tilde{\mathbf{B}}_j[1] y_j \le 1 \\
& \qquad \sum_{j=1}^{n} \tilde{\mathbf{B}}_j[i] y_j \le 1 + \tau, \ \ \forall 2 \le i \le s_B \\
& \qquad \mathbf{x} + \mathbf{y} \le \mathbf{1} \\
& \qquad \mathbf{y} \in \{0,1\}^n
\end{aligned}
$$

where $\tilde{\mathbf{B}}_j[1] = \mathbf{B}_j[1]$, $\tilde{\mathbf{B}}_j[h]$, $2 \le h \le s_B$ and $\tilde{\mathbf{p}}$ are rounded weights and profits. We are able to prove the following lemma which ensures that solving $\mathbf{IPC}_\tau(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ gives a good approximate solution to $\mathbf{IPC}(I, \mathbf{1}, \mathbf{1})$:

▶ **Lemma 15.** *Let $0 < \tau \le 1/2$. Let $\tilde{\mathbf{x}}$ be any feasible solution to $\boldsymbol{IPC}_\tau(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$. Then $\tilde{\mathbf{x}}$ is a feasible solution to $\boldsymbol{IPC}(I, \mathbf{1}, \mathbf{1})$. Let $\widetilde{Obj}_\tau(\tilde{\mathbf{x}})$ and $Obj(\tilde{\mathbf{x}})$ be the objective values of $\boldsymbol{IPC}_\tau(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ and $\boldsymbol{IPC}(I, \mathbf{1}, \mathbf{1})$ for $\mathbf{x} = \tilde{\mathbf{x}}$, respectively. If $2\delta \le \tau \le 1/2$, we have*

$$Obj(\tilde{\mathbf{x}}) \le (1+\delta)\widetilde{Obj}_\tau(\tilde{\mathbf{x}}) \le s_B(1+\delta)Obj(\tilde{\mathbf{x}}).$$

Furthermore, let $\widetilde{OPT}_\tau$ and OPT be the optimal objective values of $\boldsymbol{IPC}_\tau(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ and $\boldsymbol{IPC}(I, \mathbf{1}, \mathbf{1})$, respectively. We have

$$OPT \le (1+\delta)\widetilde{OPT}_\tau \le s_B(1+\delta)OPT.$$

Step 2. We handle large items. We first classify item profits into large, medium and small. We then classify item weights into large and small based on the largest coordinate in the weight vector, i.e., $\|\mathbf{B}_j\|_\infty$. We say an item is large if it has a large weight or a large profit, and small otherwise. Let S^* be the leader's optimal solution in $\mathbf{IPC}_\tau(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$. Using a similar argument as the special case, we can prove the following.

▶ **Lemma 16.** *With $\mathcal{O}(s_B\delta)$ additive error, we can guess out all items in S^* that have a large weight or a large profit by $n^{\tilde{\mathcal{O}}(s_B/\delta^{s_B+1})}$ enumerations.*

Utilizing the fact that coordinates in dimension $h \in [2, s_B]$ are all rounded, we can show that the follower only has a small number (i.e. $\tilde{\mathcal{O}}(s_B/\epsilon^{s_B})$) of dominant choices on large items, denoted as Θ. By restricting the follower's choices to Θ, we can obtain a new bilevel integer programming $\mathbf{MBi\text{-}IP}(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$. Similar to $\mathbf{Bi\text{-}IP}(I, \mathbf{1}, \mathbf{1})$ in the special case, $\mathbf{MBi\text{-}IP}(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ has a small number of constraints.

Step 3. We handle small items. Since $\mathbf{MBi\text{-}IP}(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ has a small number of constraints, we remove the integral constraint to obtain a relaxation $\mathbf{MBi\text{-}IP}_r(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$. Next, we transform this bilevel LP $\mathbf{MBi\text{-}IP}_r(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ to a standard (single level) LP, denoted as $\mathbf{cen\text{-}LP}_\lambda$. Note that here the transformation is much more complicated than that in the special case: in the special case we know that if the follower can choose items fractionally, then its optimal fractional solution is always obtained greedily with respect to the ratio (i.e., profit to weight), whereas it suffices to guess one single critical item. In the general case, if the follower can choose items fractionally, we can only guarantee that among all items whose rounded weight vector are the same except for the principal dimension (i.e., $\mathbf{B}_j[h]$'s have the same rounded value for every $2 \le h \le s_B$), the follower selects items greedily with respect to the principal ratio (i.e., profit to weight coordinate in the principal dimension). Therefore, we need to guess a subset of critical items, and the subscript λ in $\mathbf{cen\text{-}LP}_\lambda$ corresponds to a set of parameters characterizing the subset of critical items. The most technical part is to show that the optimal solution to $\mathbf{cen\text{-}LP}_\lambda$ gives a good approximation to $\mathbf{MBi\text{-}IP}_r(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ (see Lemma 31 in the full version [7]), where we need to create a sequence of "intermediate" LPs. Finally, we obtain an extreme point solution to $\mathbf{MBi\text{-}IP}_r(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ by solving $\mathbf{cen\text{-}LP}_\lambda$, and round it to an integral solution. The rounding error can be bounded due to that $\mathbf{MBi\text{-}IP}_r(\tilde{I}_\delta, \mathbf{1}, \mathbf{1})$ contains a small number of constraints.

5 Approximation algorithm for IPC where s_B and s_A are arbitrary

We consider the most general setting of IPC where s_A and s_B are arbitrary (not necessarily polynomial in the input size).

We define $\max\{\mathbf{py} : \mathbf{By} \le \mathbf{b}, \mathbf{y} \in \{0,1\}^n\}$ as the *follower's problem*. A separation oracle for the *leader's problem* is an oracle such that given any $\mathbf{x} = \mathbf{x}^0 \in [0,1]^n$, it either asserts that $\mathbf{x}^0 \in \{\mathbf{x} : \mathbf{Ax} \le \mathbf{a}, \mathbf{x} \in [0,1]^n\}$, or returns a violating constraint. The goal of this section is to prove the following theorem.

▶ **Theorem 17.** *Given a separation oracle O_L for the leader's problem, and an oracle O_F for the follower's problem that returns a ρ-approximation solution, there exists a $(\frac{\rho}{1-\alpha}, \frac{1}{\alpha})$-bicriteria approximation algorithm for any $\alpha \in (0,1)$ that returns a solution $\mathbf{x}^* \in \{0,1\}^n$ such that $\mathbf{Ax}^* \le \frac{1}{\alpha} \cdot \mathbf{a}$, and*

$$\max\{\mathbf{py} : \mathbf{By} \le \mathbf{b}, \mathbf{y} \le \mathbf{1} - \mathbf{x}^*, \mathbf{y} \in \{0,1\}^n\} \le \frac{\rho T^*}{1-\alpha},$$

where T^* is the optimal objective value of $\boldsymbol{IPC}(I, \mathbf{a}, \mathbf{b})$. Furthermore, the algorithm runs in polynomial oracle time.

We omit the proof of Theorem 17 here, and refer the reader to the full version [7].

6 Conclusions

In this paper, we consider a general two-player zero-sum Stackelberg game in which the leader interdicts some items to minimize the total profit that the follower could obtain from the remaining items. We obtain an $(s_B + \epsilon)$-approximation algorithm when s_A and s_B are both constant, and show that there does not exist any $(4/3 - \epsilon)$-approximation algorithm when $s_B \geq 2$. Our algorithm is the best possible when $s_B = 1$, however, it is not clear whether it is the best possible when s_B is larger than or equal to 2. In particular, it is not clear whether an approximation algorithm with a ratio independent of s_B can be obtained. Furthermore, can we hope for a PTAS if $s_B \geq 2$ but the constraints of the leader or the follower are not given by general inequalities but follow from common optimization problems? For example, what if the follower's optimization problem is a bin packing problem? It would be interesting to investigate the bilevel generalization of well-known optimization problems, e.g., scheduling and bin packing.

References

1. Bo An, Fernando Ordóñez, Milind Tambe, Eric Shieh, Rong Yang, Craig Baldwin, Joseph DiRenzo III, Kathryn Moretti, Ben Maule, and Garrett Meyer. A deployed quantal response-based patrol planning system for the us coast guard. *Interfaces*, 43(5):400–420, 2013.
2. Luce Brotcorne, Saïd Hanafi, and Raïd Mansi. One-level reformulation of the bilevel knapsack problem using dynamic programming. *Discrete Optimization*, 10(1):1–10, 2013.
3. Carl Burch, Robert Carr, Sven Krumke, Madhav Marathe, Cynthia Phillips, and Eric Sundberg. A decomposition-based pseudoapproximation algorithm for network flow inhibition. In *Network Interdiction and Stochastic Integer Programming*, chapter 3, pages 51–68. Springer, 2003.
4. Alberto Caprara, Margarida Carvalho, Andrea Lodi, and Gerhard J. Woeginger. A complexity and approximability study of the bilevel knapsack problem. In *Proceedings of the 16th International Conference on Integer Programming and Combinatorial Optimization*, pages 98–109, 2013.
5. Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345, 2000.
6. Margarida Carvalho, Andrea Lodi, and Patrice Marcotte. A polynomial algorithm for a continuous bilevel knapsack problem. *Operations Research Letters*, 46(2):185–188, 2018.
7. Lin Chen, Xiaoyu Wu, and Guochuan Zhang. Approximation algorithms for interdiction problem with packing constraints. *arXiv*, 2022.
8. Lin Chen and Guochuan Zhang. Approximation algorithms for a bi-level knapsack problem. *Theoretical Computer Science*, 497:1–12, 2013.
9. Stephen R Chestnut and Rico Zenklusen. Hardness and approximation for network flow interdiction. *Networks*, 69(4):378–387, 2017.
10. Stephen R. Chestnut and Rico Zenklusen. Interdicting structured combinatorial optimization problems with {0, 1}-objectives. *Mathematics of Operations Research*, 42(1):144–166, 2017.
11. Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.
12. Federico Della Croce and Rosario Scatamacchia. An exact approach for the bilevel knapsack problem with interdiction constraints and extensions. *Mathematical Programming*, 183(1):249–281, 2020.
13. Stephan Dempe and Klaus Richter. Bilevel programming with knapsack constraints. *Central European Journal of Operations Research*, 8(2):93–108, 2000.
14. Scott DeNegre. *Interdiction and discrete bilevel linear programming*. PhD thesis, Lehigh University, 2011.

15 Michael Dinitz and Anupam Gupta. Packing interdiction and partial covering problems. In *Proceedings of the 16th international conference on Integer Programming and Combinatorial Optimization*, pages 157–168, 2013.

16 Avinash Dixit. The role of investment in entry-deterrence. *The economic journal*, 90(357):95–106, 1980.

17 Dennis Fischer and Gerhard J. Woeginger. A faster algorithm for the continuous bilevel knapsack problem. *Operations Research Letters*, 48(6):784–786, 2020.

18 Matteo Fischetti, Ivana Ljubic, Michele Monaci, and Markus Sinnl. A new general-purpose algorithm for mixed-integer bilevel linear programs. *Operations Research*, 65(6):1615–1637, 2017.

19 Matteo Fischetti, Ivana Ljubic, Michele Monaci, and Markus Sinnl. Interdiction games and monotonicity, with application to knapsack problems. *INFORMS Journal on Computing*, 31(2):390–410, 2019.

20 Matteo Fischetti, Michele Monaci, and Markus Sinnl. A dynamic reformulation heuristic for generalized interdiction problems. *European Journal of Operational Research*, 267(1):40–51, 2018.

21 Klaus Jansen. An eptas for scheduling jobs on uniform processors: using an milp relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010.

22 Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 39(4):1392–1412, 2010.

23 Robert G. Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical Programming*, 32(2):146–164, 1985.

24 Shiva Prasad Kasiviswanathan and Feng Pan. Matrix interdiction problem. In *Proceedings of the 7th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 219–231, 2010.

25 Thomas Kleinert, Martine Labbé, Ivana Ljubic, and Martin Schmidt. A survey on mixed-integer programming techniques in bilevel optimization. *EURO Journal on Computational Optimization*, 2021.

26 André Linhares and Chaitanya Swamy. Improved algorithms for mst and metric-tsp interdiction. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming*, pages 32:1–32:14, 2017.

27 D. johnson) M. Garey. Computers and intractability: a guide to the theory of np-completeness. *Siam Review*, 24(1):90, 1982.

28 Feng Pan and Aaron Schild. Interdiction problems on planar graphs. *Discrete Applied Mathematics*, 198:215–231, 2016.

29 Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Efficient algorithms to solve bayesian stackelberg games for security applications. In *Proceedings of the 23th AAAI Conference on Artificial Intelligence*, pages 1559–1562, 2008.

30 Ulrich Pferschy, Gaia Nicosia, and Andrea Pacifici. A stackelberg knapsack game with weight control. *Theoretical Computer Science*, 799:149–159, 2019.

31 Ulrich Pferschy, Gaia Nicosia, Andrea Pacifici, and Joachim Schauer. On the stackelberg knapsack game. *European Journal of Operational Research*, 291(1):18–31, 2021.

32 Cynthia A. Phillips. The network inhibition problem. In *Proceedings of the 25th annual ACM symposium on Theory of computing*, pages 776–785, 1993.

33 James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 125–132, 2008.

34 Xian Qiu and Walter Kern. Improved approximation algorithms for a bilevel knapsack problem. *Theoretical Computer Science*, 595:120–129, 2015.

35 Yen Tang, Jean-Philippe P. Richard, and J. Cole Smith. A class of algorithms for mixed-integer bilevel min-max optimization. *Journal of Global Optimization*, 66(2):225–262, 2016.

36 Rico Zenklusen. Matching interdiction. *Discrete Applied Mathematics*, 158(15):1676–1690, 2010.

37 Rico Zenklusen. Network flow interdiction on planar graphs. *Discrete Applied Mathematics*, 158(13):1441–1455, 2010.

38 Rico Zenklusen. Connectivity interdiction. *Operations Research Letters*, 42(6-7):450–454, 2014.

39 Rico Zenklusen. An o(1)-approximation for minimum spanning tree interdiction. In *Proceedings of the 56th Annual Symposium on Foundations of Computer Science*, pages 709–728, 2015.

Online Weighted Cardinality Joint Replenishment Problem with Delay

Ryder Chen
The University of Sydney, Australia

Jahanvi Khatkar
The University of Sydney, Australia

Seeun William Umboh
The University of Sydney, Australia

Abstract

We study a generalization of the classic Online Joint Replenishment Problem (JRP) with Delays that we call the Online Weighted Cardinality JRP with Delays. The JRP is an extensively studied inventory management problem wherein requests for different item types arrive at various points in time. A request is served by ordering its corresponding item type. The cost of serving a set of requests depends on the item types ordered. Furthermore, each request incurs a delay penalty while it is left unserved. The objective is to minimise the total service and delay costs. In the Weighted Cardinality JRP, each item type has a positive weight and the cost of ordering is a non-decreasing, concave function of the total weight of the item types ordered. This problem was first considered in the offline setting by Cheung et al. (2015) but nothing is known in the online setting. Our main result is a deterministic, constant competitive algorithm for this problem.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online Algorithms, Delay, Joint Replenishment Problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.40

Category Track A: Algorithms, Complexity and Games

1 Introduction

The Joint Replenishment Problem (JRP) is a class of optimisation problems that are fundamental to inventory management theory. The problem involves a sequence of requests on items that arrive at various times. Serving a set of requests incurs a cost that is determined by a given cost function that depends on the set of items the requests are on. The goal of the problem is to serve all requests whilst minimising the total cost incurred. There are two variants under which this problem is often studied. In the deadline variant, each request has an associated deadline that is must be served before whilst in the more general delay variant, each request incurs a delay penalty which must be paid. The delay is a non-decreasing, continuous function of the time the request was left unserved. Under the delay model, the goal is to minimise the total service and delay costs. We will be considering the JRP under a *make-to-order* mechanism [18], where items must be made to serve some request and cannot be held in inventory. In this paper, we consider the online setting. Here, requests arrive over time together with their deadline or delay functions and at any point in time, the algorithm may choose to serve some set of requests.

In the Classic JRP, each item type i has a corresponding *item ordering cost* K_i and there is a fixed *joint ordering cost* K_0. Whenever a set of items is served, the cost incurred is K_0 plus K_i for each item type i served. We note that regardless of the number of units of an item type i that gets served, only a fixed K_i is paid. This problem captures the well-known TCP Acknowledgement Problem. Buchbinder et al. [15] gave a 3-competitive algorithm for

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 40; pp. 40:1–40:18


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Online Classic JRP with Delays and a lower bound of 2.64. Then, Bienkowski et al. [10] improved the lower bound to 2.754 and showed that the optimal competitive ratio for Online Classic JRP with Deadlines is 2.

With the competitive ratio of Online Classic JRP with Delays mostly settled, there has been a lot of interest in generalisations of the problem where different costs of serving requests are considered. One of these generalizations is Multi-Level Aggregation [9, 14] where we are given a rooted tree T with node costs and requests arrive at nodes of the tree. The cost of serving a set of requests is the cost of the subtree of T induced by the root and the nodes corresponding to the requests. Note that when T is depth 2, this captures JRP. The current best upper bounds for this problem depends on either the depth of T [14, 7], or are logarithmic in the number of vertices of T [8]. No super-constant lower bounds are known.

This problem was later generalized to Online Network Design with Delays [8] where we are given a graph G with edge/node costs and receive connectivity requests over time. The cost of serving a set of connectivity requests is the cost of the subgraph of G that satisfies the requests. This captures a wide class of problems such as Multi-Level Aggregation and Set Cover with Delay. Azar and Touitou [8] provided a framework to reduce a network design problem with delay or deadline to the classic offline variant without delay or deadline while incurring a logarithmic loss in the competitive ratio. Using this framework, they gave polylogarithmic-competitive algorithms for many network design problems with delays or deadlines. Recently, Touitou [30] recently showed that a sub-logarithmic competitive ratio is not possible for Online Network Design with Delays in its full generality. Thus, as powerful as this framework is, it cannot be used as is to improve the competitive ratios for problems such as Multi-Level Aggregation. For these problems, we will need to take advantage of the special structure of these problems to improve their competitive ratios.

In this paper, we introduce a natural generalisation of Online Classic JRP with Delays called the Online Weighted Cardinality JRP with Delays and show that the Azar-Touitou framework can be refined to give a constant competitive algorithm for the problem. In the Online Weighted Cardinality JRP with Delays, each item type i has an associated weight w_i and we are also given a non-negative, non-decreasing concave function f. The cost of serving a set of requests on a set λ of item types is $f(\sum_{i\in\lambda} w_i)$. The concave cost function captures a natural type of economies-of-scale in the real-world production of goods. It has also been considered in other optimization problems such as buy-at-bulk network design (see Section 1.2 for details). The special case of unit weights (called Cardinality JRP) was first studied by Cheung et al. [17] in the offline setting, and they gave a 5-approximation algorithm. We remark that this problem captures the Classic JRP by setting f to be the affine function $f(x) = K_0 + x$ and the weights $w_i = K_i$.

The main technical result of this paper is a constant-competitive algorithm for the case of unit weights.

▶ **Theorem 1.** *There is an $O(1)$-competitive, deterministic polynomial time algorithm for Online Cardinality JRP with Delay.*

For the weighted variant, we design a pseudo-polynomial time reduction from Weighted Cardinality JRP to Cardinality JRP to get the following result.

▶ **Theorem 2.** *There exists an $O(1)$-competitive, deterministic algorithm for Online Weighted Cardinality JRP with Delay.*

1.1 Our Techniques

We now outline the key ideas behind our algorithm for Online Cardinality JRP with Delay. First, as is common when dealing with concave functions, we focus on the special case where the function $f(x)$ is the minimum of n affine functions $g_i(x) = \sigma_i + \delta_i x$, and the σ_i's are geometrically increasing with i while the δ_i's are geometrically decreasing with i. Moreover, when we make a service, we specify which of these affine functions we use to pay for the service. When we make a service λ using g_i, we call it a *level* i service. The cost of the service is $g_l(\lambda) = \sigma_i + \delta_i|\lambda|$; we call σ_i the *shared cost* of the service and δ_i its *individual cost*. We call this the *Piecewise Cardinality JRP*. In the remainder of this section, we discuss our approach for Piecewise Cardinality JRP.

In Classic JRP, the main challenge is to balance the competing demands of aggregating requests into few orders to minimize the number of services and hence the total joint ordering costs incurred, and of aggregating requests on the same item types to minimize the total item ordering costs incurred. The additional challenge in Piecewise Cardinality JRP is deciding which level each service serves at. Since $2\sigma_i \leq \sigma_{i+1}$ and $\delta_i \geq 2\delta_{i+1}$ for all $i \in [1, n)$, a lower level service pays a lower shared cost but higher individual cost for each item type and is hence preferred for services with fewer items. On the other hand, a higher level service must pay a larger shared cost but can then serve items at a lower individual cost and should thus be used when serving more item types.

Our algorithm is inspired by the Azar-Touitou framework [8] and augmented with ideas from Gupta et al. [26] to decide which level we should make services at. In fact, we also show that applying the Azar-Touitou framework directly leads to a logarithmic competitive ratio at best (see full version). We first discuss how to handle the simpler deadline setting.

In the deadline setting, requests have levels which are initialised to 1 on arrival. When an unserved request of level j reaches a deadline, it triggers a level j service. We then upgrade its level to $j + 1$ if there are sufficient level j services made recently whose total individual cost can pay for the shared cost of the upgraded services. This makes sense because if there are too many level j services in a relatively small time period then the optimal solution could have aggregated the requests served by these requests into fewer higher-level services and incurred lower cost overall. While this seems to lead to a competitive ratio that depends on the number of levels, by making a careful choice of which level j services to charge to, we are able to achieve a constant competitive ratio. Once we have decided on the level of the service, say at level j', we set the budget of the service to its shared cost $\sigma_{j'}$ and serve unserved requests of level at most j' in ascending order of their deadline until the individual cost of the service reaches or exceeds the budget.

To generalise the algorithm to the delay setting, we adapt and extend the idea of investments used by [8]. Conceptually, instead of thinking of the delay penalty on requests being paid off continuously as time progresses, online algorithms with delay normally pay off the delay accumulated by requests when the request is served. However, with the notion of investments, services will pay off the delay accumulated by all requests, regardless of whether or not it serves it, and will also pay off and "invest" in the delay requests may accumulate in the future. This can be thought of as services investing in the delay requests accumulate and might accumulate in the future and incrementally paying it off as opposed to paying it all off in one go at service time. Our key innovation to extend upon the idea of investments used by [8] is to keep track of and utilise how much has been invested into each request. More specifically, we will invest in requests and once a sufficient amount has been invested into a set of requests of the same item type, we will serve the set of requests. To then generalise

the level updating condition used in the deadline variant, our algorithm will explicitly bound the amount invested in requests of a particular level by recent services and use this bound to determine when we update levels.

Our analysis uses similar ideas to [8] by defining service pointers to define different types of services and then breaking up our algorithm's cost into the costs of different types of services. Using our extended notion of investments, we introduce a novel charging argument. Previous works on online JRP and related problems often charge their algorithm's delay costs to the costs of its services which are then charged to the optimal solution. Our analysis differs by instead charging the algorithm's service costs to the delay costs we have invested in and then charging these costs to the optimal solution.

1.2 Related work

Multi-Level Aggregation was first studied by Bienkowski et al. [9] who gave an algorithm whose competitive ratio is exponential in the depth D of the tree T. This was later improved to $O(D)$ for the deadline setting by Buchbinder et al. [14] and then to $O(D^2)$ for the general delay model by Azar and Touitou [7]. The framework of Azar-Touitou [8] yields a competitive ratio that is logarithmic in the number of vertices of T.

Online Network Design with Delays was first studied by Azar and Touitou [8]. They proved polylogarithmic competitive ratios for many network design problems with delay and gave a $\Omega(\sqrt{log|V|})$ lower bound for the case of online node-weighted Steiner tree with delay and online directed Steiner tree with delay. This was recently improved to $\Omega(\log|V|/\log\log|V|)$ by Touitou [30].

A related problem is Set Cover with Delays where we are given a universe of elements and a collection of sets with costs, requests arrive on elements. The cost of serving a set of requests is the min-cost set cover for the corresponding set of elements. This problem was first studied by Carrasco et al. [16]. They gave a $O(\log N)$-competitive algorithm (where N is the number of requests) and proved a matching lower bound. Later, Azar et al. [3] gave an algorithm that is polylogarithmic in the number of sets and elements.

Two other online problems with delay that have received a lot of attention are matching [19, 1, 20, 12, 4, 13, 11, 6] and k-server [5, 13, 7, 24]. In Matching with Delays, requests arrive on points of a metric space and accumulate delay until they are matched. The objective is to minimize the length of the matching and the total delay cost. In the k-Server with Delays problem, we have k servers in a metric space and requests arrive on points of the metric space. A request is served by moving a server to its location. The goal is to minimize the total distance traveled by the servers and the delay incurred by the requests.

Concave cost functions have been widely-studied in the network design literature, both in the offline and online settings. The problem that is most closely relevant to our paper is Offline Single-Sink Buy-at-Bulk Network Design. We are given an undirected graph $G = (V, E)$ with edge lengths d_e, a concave cost function f, a sink t and a set of sources s_i. The cost of routing x_e units of flow on edge e is $f(x_e) \cdot d_e$. The total cost is the sum of the routing cost over all edges. The goal is to route one unit of flow between from each source s_i to the sink t with minimum total cost. The problem is known to be NP-hard and admits constant-factor approximation algorithms [28, 23, 21, 29, 25, 22, 27]. In the online setting, the sources arrive one-by-one. For the online problem, there is a tight deterministic $O(\log k)$-competitive algorithm [2, 31, 26].

2 Preliminaries

As mentioned in the Introduction, we will be mainly dealing with Piecewise Cardinality JRP with Delay. The Piecewise Cardinality JRP is a special case of Cardinality JRP where the cost function is a concave, piecewise function defined by taking the minimum of n affine functions, where n is arbitrary. More precisely, the cost of a service λ is $g(\lambda) = \min_i\{\sigma_i+\delta_i|\lambda| : i \in [1,n]\}$ where the cardinality $|\lambda|$ is the number of item types served in λ. The affine functions must also satisfy $2\sigma_i \leq \sigma_{i+1}$ and $\delta_i \geq 2\delta_{i+1}$ for all $i \in [1,n)$. We will also require that $\sigma_i \geq \delta_i$ for all i. Requests will then arrive over time and upon arrival, they have an associated deadline or delay function that is revealed.

When a service with cost $\sigma_i + \delta_i|\lambda|$ for some i is made, we will say that a *level i service* has been made and call the σ_i paid the *shared cost* of the service and the $\delta_i|\lambda|$ cost paid the *individual cost* of the service. Typically, a solution to Piecewise Cardinality JRP would specify when services are made and what requests are served by these services. The cost of this service would then be determined by taking the minimum of the piecewise affine functions. Equivalently, we can also require the solution to specify for each service not only the requests it serves but also the level it serves it in. This is a more useful formulation that we will be using in the sequel.

For the delay variant, which is the variant under which we are studying this problem, the delay penalty function $d_q(t)$ for a request q is a non-decreasing, continuous function of the time the request has been left unserved. We will also assume that the delay penalty for each request tends to infinity as the time tends to infinity which is a natural assumption also made by [8] to ensure that all requests must eventually be served.

Using standard techniques for dealing with concave functions we can reduce Cardinality JRP to Piecewise Cardinality JRP losing only a constant factor in the approximation ratio. We defer the details to the full version.

3 Piecewise Cardinality JRP with Delay

We now prove the following theorem. Omitted proofs can be found in the full version.

▶ **Theorem 3.** *There is a deterministic $O(1)$-competitive algorithm for Piecewise Cardinality JRP with Delay.*

3.1 Algorithm Intuition

We first introduce some terminology. We will say a request is *active* if it has arrived and is unserved. We will also assign each request a *request level* which is initially set to 1 upon arrival and is updated as the algorithm progresses. We say that an request is *eligible* for a level l service at time t if it is active and has level at most l at time t. Our algorithm allows services to incrementally pay off the delay that requests have accumulated in the past and may accumulate in the future; we call the latter an *investment cost*. In particular, we say that a service λ at time t *invests* an amount x into request q when λ pays off x amount of the delay cost that may be incurred by q after t. The *residual delay* of a request q at time t is defined as the amount of delay accumulated by the request up to time t that has not been paid for by some service.

When do we make a service? A level l service λ is *triggered* when the set of requests E eligible for a level l service accumulate a total residual delay of σ_l; we say that E are the eligible requests of λ. The intuition here is that our algorithm will make level l services whose total cost is $O(1) \cdot \sigma_l$ and hence waiting until σ_l unpaid delay accumulates means that our delay cost will be comparable to the service cost.

What level service should we make? When a level l service λ is triggered at time t, we first check if we can upgrade it to $l+1$. Roughly speaking, we upgrade to level $l+1$ if the amount invested by recent level l services so far is at least σ_{l+1}. This invariant allows us to bound our investment costs. We then proceed to the next step of deciding which requests to serve.

Which requests to serve? Next, the algorithm uses an investment process to decide which requests to serve. For each item type we add to our service, we incur an additional cost of δ_l. The investment process continually invests into the future delay of eligible requests. Once the total investment into requests of an item type i, from this service as well as prior services, reaches δ_l, we add i to the service, reset the counter to 0 and stop investing in requests of type i. The entire investment process stops once the total amount invested by the service reaches σ_l or the service contains the item types of all eligible requests, and we make the service. Intuitively, this investment process allows us to serve requests by order of urgency. Urgent requests can be interpreted as those that accumulate delay faster and hence will have investment counters that reach the required service threshold faster.

3.2 Algorithm Description

We now formally describe the algorithm. Our algorithm maintains the following information.

Service and request pointers. Each request and service will be assigned a pointer that points to some service. This will be specified in more detail later on. We classify services into the following types.

▶ **Definition 4** (Service types). *A service λ is:*

- *a* primary service *if it does not point to any service;*
- *an* upgrade service *if it is triggered initially at level l and the algorithm decided to upgrade it to level $l+1$;*
- *a* tail service *if it is neither a primary nor an upgrade service and no service points to λ;*
- *a* normal service *if it is not one of the above types, i.e. it is not a primary nor an upgrade service and there is a service pointing to it.*

Note that a tail service can become a normal service later on.

Investment counters. The algorithm maintains an investment counter $counter(l,i)$ for level l and each item type i. This counter keeps track of the amount invested into requests of type i since the last level l service containing type i or the beginning of time if there is no such service.

Investment intervals. To decide whether to upgrade service, the algorithm needs to keep track of the investments made by recent normal services. At the end of each service λ that is neither primary nor upgrade, for each eligible request q, the algorithm creates a *investment interval* $[t,\tau]$ on q with cost equal to the amount that λ invested in q. Here, t is the service time of λ. The interval also has a level, which is the level of the service. The details of the investment process and the definition of τ will be specified later.

Our algorithm consists of the following components:

Initialisation. Before any requests arrive, the investment counters $counter(l,i)$ are initialised to 0. When a request q arrives, we set its level to 1, its pointer to NULL.

Serving requests. A level l service λ is triggered when active requests of level at most l accumulate a total of σ_l residual delay. The algorithm then proceeds through the following four steps:

Step 1: Setting service pointer. The *triggering requests* for λ are the active level l requests with positive residual delay. The service pointer of λ will be determined by looking at its triggering requests. If all the triggering requests have a NULL pointer then the service pointer will be NULL. Otherwise, the service pointer will be set to be any of the non-NULL triggering request pointers. As will be shown later (Observation 8), all requests in the triggering requests set with a non-NULL pointer must in fact point to the same service so it does not matter which non-NULL triggering request pointer we choose.

Step 2: Determining the service level. When a level l service λ is triggered, if it is non-primary and not of the highest level n already then it is eligible for a level upgrade. To determine whether to upgrade the service, we use the following notion of witness sets.

▶ **Definition 5** (Witness sets). *Let λ be a level l service and a_q be the earliest arrival time among its eligible requests. The* witness set *of λ is the set W_λ of level l investment intervals that begin after a_q and were created by previous level l normal services. The cost of the witness set W_λ is the total cost of the investment intervals in it and is denoted by $c(W_\lambda)$.*

If $c(W_\lambda) \geq \sigma_{l+1}$, we upgrade λ's level to $l+1$; otherwise, it stays at level l. Note that upgrading the level of a service will not change its triggering requests or pointer but after upgrading, all requests of level $l+1$ will now also be eligible for λ and hence λ's set of eligible requests E_λ may increase. Algorithm 1 gives the pseudocode for Steps 1 and 2.

Step 3: Making the service. Our level l service λ at time t first pays off the residual delay that each eligible request $q \in E_\lambda$ has accumulated up to the service time t. Let $r_q(t_1, t_2)$ denote the residual delay accumulated by request q from times t_1 to t_2 then for each eligible request $q \in E_\lambda$, we pay off all residual delay since their arrival time, that is, $r_q(a_q, t)$.

Then the service begins the investment phase with an investment budget of σ_l where it invests in the future residual delay that the eligible requests accumulate from t. This begins with initializing the following variables: future time $\tau \leftarrow t$, the previous time $t' \leftarrow t$, the set of served requests $Q_\lambda \leftarrow \emptyset$ and the service investment counter to be 0. We then continuously increase τ. Each time τ increases, the residual delay incurred from time t' to τ by each request $q \in E_\lambda \setminus Q_\lambda$ is paid off and invested in. That is, we pay off $r_q(t', \tau)$ for each $q \in E_\lambda \setminus Q_\lambda$. This residual delay $r_q(t', \tau)$ invested in is added to the level l investment counter for q's corresponding item type as well as the service investment counter. We also add this amount to a variable $I_q(\lambda)$ which keeps track of how much λ has invested in the request q and will be used later to construct our investment intervals. If the investment counter for an item type i reaches δ_l then all eligible requests of item type i are added to the set Q_λ to be served and we stop investing in these requests. Finally, we set $t' \leftarrow \tau$ and iteratively continue the process. This process terminates if all eligible requests have been served and added to Q_λ or if the service investment counter equals σ_l which signifies a total of σ_l has been invested in future delay incurred by eligible requests. We note that at the end of this process, all eligible requests will have their delay paid off until time τ and hence can only accumulate residual delay beginning from time τ. To finish the service, we serve the requests in Q_λ, paying a fixed cost of σ_l as well as δ_l for each item type in Q_λ. This process is captured in Algorithm 2.

Note that it is possible for Q_λ to be empty.[1] In this case, we say that λ is an *empty service*. The delay costs paid off by the algorithm remains paid but it does not pay the shared or individual cost.

Step 4: Updating request pointers, levels, investment counters and investment intervals. At the end of the service, all eligible requests for λ left unserved have their pointers set to λ and levels set to l. If λ serves item type i then the level l investment counter for item i is reset to 0. If the service is not a primary or upgrade service then for each request that was eligible for λ, we construct the level l investment interval $[t, \tau]$ for this request with cost $I_q(\lambda)$ which is how much λ invested in the request. Note that all eligible requests for this service will have the same investment interval start and end times but the cost of each interval may differ. Pseudocode for these steps is given at the end of Algorithm 2.

Algorithm 1 Procedure to handle triggering and upgrading services.

Function *OnTrigger(level l)*
- Start a new service λ at current time t;
- `/* determine service pointer using triggering requests */`
- Let E_λ be all eligible requests;
- Let $Q_{trigger} \subseteq E_\lambda$ be those with positive residual delay and level l;
- $pointer(\lambda) \leftarrow pointer(q)$ for an arbitrary request $q \in Q_{trigger}$;
- `/* upgrade the service level if possible */`
- **if** $pointer(\lambda) \neq NULL$ *and* $l \neq n$ **then**
  - Let a_q be the earliest arrival time among requests in E_λ;
  - Let W_λ be the set of investment intervals created by level l normal services and begin after a_q;
  - **if** $\sum_{c \in W_\lambda} cost(c) \geq \sigma_{l+1}$ **then**
    - $l \leftarrow l + 1$;
    - Update E_λ to be the eligible requests for level $l + 1$;
- $level(\lambda) \leftarrow l$;
- **MakeService**(λ, t);

3.3 Analysis

We will bound the costs of the different types of services individually: primary services, normal services, upgrade services and tail services. By noting that every service must fall under one of these categories, this will enable us to bound the total cost of our algorithm. In the following, we abuse notation and use ALG to denote both the algorithm's solution and its cost, and OPT to denote both the optimal solution and its cost.

We first examine the structure and properties of our solution in Section 3.3.1. In particular, we will argue that the directed graph induced by the services and service pointers consist of node-disjoint directed paths and that all non-NULL pointers in a set of triggering requests must be the same.

Next, we look at the structure of our costs in Section 3.3.2. Most importantly, we will introduce the notion of *charged costs* which represent the costs of our services that need to be charged to OPT and show that the charged cost of a level l service is at most $3\sigma_l$.

[1] This can happen, for instance, if there are many eligible requests on many different item types so the service investment counter reaches σ_l before any of the $counter(l, i)$ reaches δ_l.

Algorithm 2 Procedure to handle serving requests and updating information post-service.

Function *MakeService(service λ, service time t)*
  `/* pay off all residual delay on eligible requests */`
  foreach $q \in E_\lambda$ **do**
    Pay off the residual delay $r_q(a_q, t)$;
    `/* set up counter to track how much` λ `invests in each request */`
    $I_q(\lambda) \leftarrow 0$;
  `/* Investment Phase */`
  $l \leftarrow level(\lambda)$; $\tau \leftarrow t$; $t' \leftarrow t$; $Q_\lambda \leftarrow \emptyset$; $invested \leftarrow 0$;
  Let $E_{\lambda,i}$ be the set of eligible requests of item type i;
  while $Q_\lambda \neq E_\lambda$ *and* $invested < \sigma_l$ **do**
    Continuously increment τ until either $invested + \sum_{q \in E_\lambda \setminus Q_\lambda} r_q(t', \tau) = \sigma_l$, or for some type i not in Q_λ, $counter(l, i) + \sum_{q \in E_{\lambda,i} \setminus Q_\lambda} r_q(t', \tau) = \delta_l$;
    `/* Update investment counters and` t' `*/`
    $invested \leftarrow invested + \sum_{q \in E_\lambda \setminus Q_\lambda} r_q(t', \tau)$;
    foreach *item type* i **do**
      $counter(l, i) \leftarrow counter(l, i) + \sum_{q \in E_{\lambda,i} \setminus Q_\lambda} r_q(t', \tau)$;
    foreach $q \in E_\lambda \setminus Q_\lambda$ **do**
      $I_q(\lambda) \leftarrow I_q(\lambda) + r_q(t', \tau)$;
      Pay off the residual delay $r_q(t', \tau)$;
    $t' \leftarrow \tau$;
    if *for some type* i *not in* Q_λ, $counter(l, i) = \delta_l$ **then**
      $Q_\lambda \leftarrow Q_\lambda \cup E_{\lambda,i}$;
      $counter(l, i) \leftarrow 0$;
  Serve Q_λ;
  `/* update pointers and levels of unserved eligible requests */`
  foreach $q \in E_\lambda \setminus Q_\lambda$ **do**
    $level(q) \leftarrow l$ and $pointer(q) \leftarrow \lambda$;
  `/* construct investment intervals */`
  if λ *is not a primary or upgrade service* **then**
    foreach $q \in E_\lambda$ **do**
      Construct a level l investment interval $[t, \tau]$ on q with cost $I_q(\lambda)$;

Section 3.3.3 will then begin our charging argument by showing that the charged cost of all services can be bounded by the charged costs of the primary and normal services. This is done by bounding the charged cost of the tail services by the charged cost of the primary and upgrade services and then charging the charged cost of the upgrade services to the charged cost of the normal services.

Finally, we finish our charging argument in Section 3.3.4 by charging the charged costs of the primary and normal services to OPT. This is the crux of our analysis. The charged cost of primary services is charged to OPT using a disjointness argument. The charged cost of the normal services is charged to the cost of our investment intervals. These investment intervals' costs are then charged to OPT by showing that for any service λ^* made by OPT and any request q served in λ^*, the total cost of q's investment intervals can be charged to the costs of λ^*.

To simplify our analysis, we assume that at most one service can be triggered at any given time. This is without loss of generality since the delay functions can be perturbed by an infinitesimal amount to ensure this holds.

3.3.1 Analysis: Solution Structure

We begin our analysis by proving properties of ALG's solution structure.

▶ **Observation 6.** ALG *serves all requests eventually. Moreover, when a request q is served at time t, the algorithm would have paid off its delay up till at least time t by the end of the service.*

Proof. We first argue that ALG serves all requests. This is because the delay on each request tends to infinity with time so each request will eventually accumulate enough delay to trigger a service and then have enough invested to trigger the service of the requests. This is because as the delay tends to infinity, ALG will never reach a scenario where it stops making services or stops investing in requests. The second part of the observation follows from the fact it was eligible for the service it was served in, and the service would have ensured that the delay of q up till time t has been paid off entirely. ◀

The following lemma implies that service pointers belonging to services of the same level are, in a sense, non-overlapping.

▶ **Lemma 7.** *Suppose a service λ' points to another service λ and their service times are $t' > t$, respectively. Let l be the level of λ. Then, there cannot be a service of level at least l made between t and t'.*

Proof. Suppose, towards a contradiction, that there is a level l service λ_0 made between t and t'. By definition of pointers, there exists a triggering request q for λ' that pointed to λ at the start of the service λ'; moreover, q is active and has level l between t and t'. Thus, q is eligible for λ_0. Since λ_0 did not serve q, it must have overwritten q's pointer and so q would not have pointed to λ at the start of λ', a contradiction. ◀

▶ **Observation 8.** *A service's set of triggering requests must be non-empty and all the triggering requests with a non-NULL pointer must have the same pointer.*

Proof. Consider any level l service λ. We first observe that the set of triggering requests must be non-empty. If this was not the case, then by definition there is no level l eligible request with positive residual delay when λ is triggered. Hence, all the requests whose residual delay contributed to triggering λ would have a level less than l and since $\sigma_{l-1} < \sigma_l$, a lower level service should have been triggered instead. Therefore, λ should never have been triggered in the first place.

Next we show that all triggering requests with a non-NULL pointer must point to the same service. Let q_1 be the triggering request whose pointer λ_1 was used as the service pointer for λ and let q_2 be another triggering request with pointer λ_2. Since q_1 and q_2 are level l requests, they must point to level l services. Moreover, the pointer of q_2 must have been set after λ_1 but before λ since otherwise it would have been eligible for λ_1 and been changed. Then, Lemma 7 implies that λ_2 must in fact be λ_1. ◀

From Observation 8, we know that a service pointer will always be uniquely defined and not dependent on which triggering request was chosen to determine the pointer as alluded to in the algorithm description. Next, we show that the directed graph induced by the services and service pointers consist of node-disjoint directed paths, which we call *service chains*.

▶ **Lemma 9.** *Every service can only point to at most one other service and be pointed at by at most one other service.*

Proof. By construction, a service pointer can only be one other service and hence every service can only point to at most one other service. Now we will argue that two services cannot point to the same service λ' of level l. Suppose there are two services λ_1 and λ_2 that point at λ'. By construction of service pointers, λ_1 and λ_2 must have been level l when they were triggered. By Lemma 7, we get that λ_1 and λ_2 must be the same service. Thus, each service can be pointed to by at most one other service. ◀

The following lemma implies that at any time, only the last service of a level can become normal in the future, all previous services of the level that are tail must remain tail. This ensures that our witness sets will not exclude any investment intervals that belong to tail services that later become normal services which is required later to prove Lemma 24.

▶ **Lemma 10.** *Let λ_1 and λ_2 be two level l services with service times $t_1 < t_2$, respectively. If λ_1 is tail at time t_2, then it will stay a tail service at all future times $t > t_2$.*

Proof. Suppose, towards a contradiction, that λ_1 became normal later on. Then, at some time $t_3 > t_2$, a level l service λ_3 was triggered and λ_3 pointed to λ_1. However, this contradicts Lemma 7 so λ_1 must stay a tail. ◀

3.3.2 Analysis: Cost Structure

We now analyse the properties of different costs incurred by our algorithm.

▶ **Definition 11** (Triggering, investment and charged costs). *For each level l service λ, we define the following three costs: (1) its* triggering delay cost *which is the residual delay on all eligible requests that is paid off at the beginning of the service; (2) its* investment cost *which are the delay costs invested in during the λ's investment phase; (3) its* charged cost *which is its shared cost plus its triggering delay cost plus its investment cost.*

Note that the algorithm may make an "empty service" in which no requests are served, in which case its charged cost is its triggering delay cost and its investment cost. We will first show that in order to bound the cost of ALG, it suffices to look at the charged costs.

▶ **Lemma 12.** *The total cost of* ALG *is at most twice the total charged cost of its services.*

Proof. The total cost of ALG is the total shared cost of its services plus the total individual cost of its services plus its delay cost. It suffices to bound the latter two costs in terms of total investment and triggering delay costs. A level l service only serves an item type i when the level l investment counter for i reaches δ_l and the counter is reset to 0 when it reaches δ_l. So, the total individual cost is at most the total investment cost. Observation 6 implies that the delay cost of ALG is at most the triggering delay cost plus investment cost. Putting the above bounds together gives the lemma. ◀

Next, we will bound the charged cost of any service which will later allow us to charge these costs to OPT.

▶ **Observation 13.** *The triggering delay costs incurred by a level l service is at most σ_l.*

Proof. Let λ be a level l service made by ALG. If λ is not an upgrade service, then its triggering delay cost is exactly σ_l, by construction. On the other hand, if λ is an upgrade service, then it was triggered because the active requests of level at most $l-1$ had a residual delay of σ_{l-1} but then its level was upgraded and its pool of eligible requests increased to include the level l requests. However, the fact that a level l service was not triggered at this time implies that the active requests of level at most l must have a residual delay less than σ_l. Therefore, in either case, the triggering delay is at most σ_l as required. ◀

▶ **Lemma 14.** *The charged cost of a level l service is at most $3\sigma_l$.*

Proof. The charged cost of a level l service as defined earlier is comprised of the triggering delay cost, the shared cost and the investment cost. By Observation 13, the triggering delay cost is at most σ_l. The shared cost is either σ_l or 0 in the case that no requests are served and hence the shared cost does not need to be paid. Lastly, by construction of the algorithm, the investment cost of the service is at most σ_l. Adding these three components up, the total charged cost is at most $3\sigma_l$ as required. ◀

The following lemma bounds the investment cost of normal services which will be used later to charge the costs of the normal services to OPT.

▶ **Lemma 15.** *Every normal service of level l has investment cost σ_l.*

Proof. The investment cost of a level l normal service λ is exactly σ_l. This is because by construction, the investment cost of a level l service is at most σ_l and can only be less if λ served all eligible requests. However, that would mean that λ is a tail service since there are no unserved requests pointing to it. ◀

3.3.3 Analysis: Bounding charged costs

We now begin our charging argument by showing that the charged costs of ALG can be bounded by the charged costs of the primary and normal services.

▶ **Lemma 16.** *The total charged costs of the tail services are at most 3 times the total charged costs of the primary and upgrade services.*

Proof. Consider any tail service λ of level l. By definition, it ends a chain that has reached level l and so the chain has an earlier service λ' that is a level l upgrade service or a primary service, in the case that $l = 1$. The service λ' has an investment cost of exactly σ_l as otherwise the service would have served all eligible requests and the chain would have ended already. Therefore, λ' has a charged cost of at least σ_l. By Lemma 14, λ's charged cost is at most 3 times that of λ'. Lastly, Lemma 9 implies that each primary or upgrade service is charged by at most one tail service. Summing up across all tail services, we get that their charged costs are at most 3 times the charged costs of the primary and upgrade services. ◀

We now bound the charged cost of the upgrade services by the charged cost of the normal services. In order to do so, we first show that the witness sets of our upgrade services must be pairwise disjoint.

▶ **Lemma 17.** *An investment interval cannot belong to the witness set of more than one upgrade service.*

Proof. Suppose towards a contradiction that there is a request q_c whose level l investment interval $[t, \tau]$ belongs to the witness sets of two upgrade services. Let the two level $l+1$ upgrade services be λ_1 and λ_2 with service times $t_1 < t_2$.

By construction, the interval $[t, \tau]$ was created by a service at time t. Since it belongs to the witness set of λ_1, it must have been created before t_1, and thus, $t < t_1$. Now consider λ_2. In order for $[t, \tau]$ to belong to its witness set, λ_2 must have an eligible request q_2 that arrived before t. The fact that q_2 is eligible for λ_2 at time t_2 implies that it is at most level l and active for all times prior to t_2. Moreover, the fact that q_2 arrives before t implies that it also arrives before $t_1 > t$. Therefore, at time t_1, q_2 has arrived, is active and of level at most l which makes it eligible for λ_1. After λ_1, q_2 would have its level set to $l+1$ and hence it would no longer be eligible for λ_2 when λ_2 is initially triggered (prior to upgrading). This contradicts the fact that q_2 is eligible for λ_2. See Figure 1 for an illustration of this proof. ◀

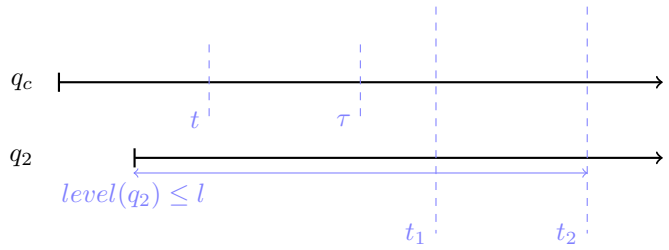


Figure 1 Illustration of the proof of Lemma 17.

▶ **Lemma 18.** *The total charged costs of the upgrade services is at most 3 times the total charged costs of the normal services.*

Proof. Let λ be a level l upgrade service at time t. By construction, the cost of its witness set W_λ is at least σ_l. By Lemma 14, λ has a charged cost of at most $3\sigma_l \leq 3cost(W_\lambda)$. Using Lemma 17, we get that the total charged costs of upgrade services is at most 3 times the total investment costs made by normal services. Since the investment costs of the normal services are at most the charged costs of the normal services, the lemma follows. ◀

3.3.4 Analysis: Charging to OPT

Having bounded the cost of ALG by the charged costs of the primary and normal services, it remains to charge these costs to OPT.

▶ **Lemma 19.** *The charged cost of the primary services is at most* 3 OPT.

Proof. Let Λ_1 be the set of primary services made by ALG. For each primary service $\lambda \in \Lambda_1$, let I_λ denote the interval $[a_\lambda, t_\lambda]$ where a_λ is the earliest arrival time among λ's triggering requests and t_λ is the service time of λ. By definition of primary services, the set of intervals $\mathcal{I} = \{I_\lambda\}_{\lambda \in \Lambda_1}$ are pairwise disjoint.

Consider an interval $I_\lambda \in \mathcal{I}$. If OPT made a service λ^* during I_λ, the shared cost of λ^* is at least σ_1. On the other hand, if OPT did not make a service during I_λ, then it must have incurred a total delay of at least σ_1 on the triggering requests of λ since they arrive no earlier than the start of I_λ. In both cases, OPT pays a cost of at least σ_1 during the interval I_λ.

Since the intervals $\mathcal{I}$ are disjoint, we have that OPT $\geq \sigma_1|\mathcal{I}|$. Thus, by Lemma 14, the charged cost of primary services is at most $3\sigma_1|\mathcal{I}| \leq 3$ OPT. ◀

▶ **Lemma 20.** *The charged cost of a normal service is at most 3 times its investment cost.*

Proof. This follows from Lemmas 15 and 14. ◀

Using Lemma 20, the charged cost of any normal service is at most 3 times its investment cost. Hence, to bound the charged costs of all normal services, it suffices to only look at the investment costs incurred by the normal services which, by definition is equal to the cost of the investment intervals made by normal services. Hence, we will show that the cost of every investment interval made by a normal service can be charged to some distinct cost incurred by OPT.

We will refer to investment intervals created by normal services as *normal investment intervals* and will now analyse the properties of these intervals.

▶ **Lemma 21.** *Let $[t, \tau]$ be a level l normal investment interval created by the normal service λ at time t. Then there cannot be a level $l' \geq l$ service triggered between times t and τ (inclusive) other than λ.*

Proof. Since λ is a normal service, there exists a later service λ_s at time t_s pointing to λ. Thus, there is a request q that is eligible for λ but is left unserved and later became a triggering request for λ_s. By definition of pointers and triggering requests, at time t_s, q points to λ and has positive residual delay. Since λ paid off the residual delay on eligible requests (which includes triggering requests) until time τ, λ_s occurred at time $t_s > \tau$.

Now we show that after t and before t_s, q is level l and there is no service for which q is eligible. Since q points to λ at time t_s, there cannot be a service between t (the service time of λ) and t_s for which q is eligible; otherwise, that service would have changed q's pointer and q would no longer be pointing to λ at time t_s. Thus, the level of q between times t and $\tau < t_s$ is l, the level of λ, and so there is no service of level at least l between t and τ. ◀

▶ **Lemma 22.** *For any request, its normal investment intervals are all disjoint.*

Proof. Assume towards a contradiction there is a request q with two non-disjoint normal investment intervals $[t_1, \tau_1]$ of level l_1 and $[t_2, \tau_2]$ of level l_2 where wlog $t_1 < t_2 < \tau_1$. We first notice that $l_2 \geq l_1$ since the level of a request can never decrease so after the service at time t_1, q is of level at least l_1. By Lemma 21, there is no service of level at least l_1 between times t_1 and τ_1 which contradicts our assumption. ◀

▶ **Lemma 23.** *Consider any two level l normal investment intervals that intersect. They must be created by the same level l normal service.*

Proof. Suppose this was not the case and that we have two intersecting level l investment intervals $[t_1, \tau_1]$, $[t_2, \tau_2]$ created by distinct normal services where wlog $t_1 \leq t_2 \leq \tau_1$. This would imply that we have a level l normal service at time t_1 and another normal level l service at time $t_2 \geq t_1$. However, Lemma 21 implies that there cannot be another level l service between times t_1 and τ_1 inclusive and hence the service at time t_2 should not have occurred, a contradiction. ◀

We now charge the cost of our normal investment intervals to OPT which will be the crux of our analysis. For this lemma, we will refer to "normal investment intervals" simply as "investment intervals" unless otherwise specified. The full proof of the lemma can be found in the Appendix.

▶ **Lemma 24.** *The total cost of normal investment intervals is at most* $6\,\mathrm{OPT}$.

Proof sketch. Consider a service λ^* made by OPT at level l^* and time t^* and let the set of requests served by λ^* be Q_{λ^*}.

For any request $q \in Q_{\lambda^*}$ we will first consider its investment intervals ending before t^*. These investment intervals ending before time t^* correspond to delay costs incurred prior to time t^* that had been invested in by a normal service and is hence at most the total delay accrued by the request up till time t^*. Since OPT pays the delay for the request q up till time t^*, the cost of the investment intervals ending before time t^* can be charged to the delay costs paid by OPT.

Next we consider the investment intervals belonging to requests in Q_{λ^*} that contain time t^*. There are two cases to consider: the intervals of level $l < l^*$ and the intervals of level $l \geq l^*$. For the intervals of a fixed level $l < l^*$ that intersect with time t^* we know by Lemma 23 that they must have all been created by the same normal service. Since a normal service can only invest a maximum amount of σ_l, the total cost of these intervals of level l must be at most σ_l. Summing up across all levels $l < l^*$ and noting that our σ values form a geometric series, we conclude that the cost of all the intervals with level $l < l^*$ is at most σ_{l^*} which is the shared cost that OPT must pay in serving λ^*. For the intervals of a fixed level $l \geq l^*$ we will consider a fixed item type i. Once again, by Lemma 23, these intervals must have been created by the same normal service and since a normal service can only invest at most δ_l in a particular item type before serving it, the total cost of the intervals at level l and for item i requests is at most δ_l. Summing up across all levels $l \geq l^*$ and noting that our δ values form a geometric series, we conclude that the cost of all item i intervals of level $l \geq l^*$ is at most $2\delta^*$ which is 2 times the individual cost paid by λ^* to serve the item type i. This argument applies to all item types served by λ^*.

Lastly, we consider the investment intervals belonging to requests in Q_{λ^*} that begin after time t^*. Once again we consider two cases: the intervals of level $l < l^*$ and the intervals of level $l \geq l^*$. For the intervals of level $l < l^*$, we consider a fixed level l and look at the latest starting interval at time t. There must have been a normal service at time t that created this interval and this implies that the witness set of this normal service cost less than σ_{l+1} since it was a normal service as opposed to an upgrade service. Since all requests in Q_{λ^*} must arrive before t^*, the witness set of the service at time t is a superset of the intervals of level l beginning after time t^*. Moreover, the intervals created by the service at time t must cost at most σ_l since the service can only invest at most σ_l. Overall, the intervals of level l that begin after time t^* cost at most $\sigma_{l+1} + \sigma_l < 2\sigma_{l+1}$. Summing up over all levels $l < l^*$ and using the geometric property of σ, we get a total investment interval cost of $4\sigma_l^*$ which is 4 times the shared cost paid by λ^*. For the intervals of level $l \geq l^*$ we once again fix a level $l \geq l^*$ and item type i. The cost of each of these item i, level l intervals must be added to the same investment counter without the counter being reset to 0. This is because all requests in Q_{λ^*} arrive before t^*, so if at any time after t^*, the level l investment counter for item i is reset to 0, all requests on item i and of level at most l must have been served and thus there is no way for a level l intervals to contribute to an investment counter after it has been reset. Since the investment counter by design has a maximum value of δ_l, this implies that the total cost of the level l, item i investment intervals is at most δ_l. Fixing the item i, summing up across all levels $l \geq l^*$ and utilising the geometric property of δ, we get that the investment intervals of item i and of level $l \geq l^*$ cost at most $2\delta_{l^*}$ which is 2 times what λ^* pays for the item i. We can once again apply this argument for all item types i served by λ^*.

Hence, it follows that for any service λ^*, the cost of the investment intervals in Q_{λ^*} can be charged to the cost of λ^* and by applying this argument across all services made by OPT, the lemma follows. ◀

▶ **Lemma 25.** *The charged cost of the normal services is at most* 18 OPT.

Proof. This follows from Lemmas 20 and 24. ◀

We conclude by combining the above ingredients to show that the algorithm is constant-competitive.Let P, U, T, N denote the charged service costs of primary services, upgrade services, tail services and normal services, respectively. Using Lemma 12 and the fact that the total charged service cost is $P + U + T + N$, we get

$$\begin{aligned}
\text{ALG} &\leq 2(P + U + T + N) \\
&\leq 2(4P + 4U + N) && \text{(Lemma 16)} \\
&\leq 2(4P + 13N) && \text{(Lemma 18)} \\
&\leq 24\,\text{OPT} + 26N && \text{(Lemma 19)} \\
&\leq 492\,\text{OPT} && \text{(Lemma 25)}
\end{aligned}$$

Hence, we get that $\text{ALG} \leq O(1) \times \text{OPT}$ as desired.

References

1. Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 1:1–1:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1.
2. Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized steiner problem. *Theor. Comput. Sci.*, 324(2-3):313–324, 2004. doi:10.1016/j.tcs.2004.05.021.
3. Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay - clairvoyance is not required. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.8.
4. Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. *Theory Comput. Syst.*, 64(4):572–592, 2020. doi:10.1007/s00224-019-09963-7.
5. Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. *ACM Trans. Algorithms*, 17(3):23:1–23:31, 2021. doi:10.1145/3459925.
6. Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 301–320. SIAM, 2021. doi:10.1137/1.9781611976465.20.
7. Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00013.
8. Yossi Azar and Noam Touitou. Beyond tree embeddings - a deterministic framework for network design with deadlines or delay. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.
9. Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

10 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54. SIAM, 2014. doi:10.1137/1.9781611973402.4.

11 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In Leah Epstein and Thomas Erlebach, editors, *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, volume 11312 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2018. doi:10.1007/978-3-030-04693-4_4.

12 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In Roberto Solis-Oba and Rudolf Fleischer, editors, *Approximation and Online Algorithms - 15th International Workshop, WAOA 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, volume 10787 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2017. doi:10.1007/978-3-319-89441-6_11.

13 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In Zvi Lotker and Boaz Patt-Shamir, editors, *Structural Information and Communication Complexity - 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, volume 11085 of *Lecture Notes in Computer Science*, pages 237–248. Springer, 2018. doi:10.1007/978-3-030-01325-7_22.

14 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244. SIAM, 2017. doi:10.1137/1.9781611974782.80.

15 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online Make-to-Order Joint Replenishment Model: Primal-Dual Competitive Algorithms. *Oper. Res.*, 61(4):1014–1029, 2013. doi:10.1287/opre.2013.1188.

16 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2018. doi:10.1007/978-3-319-77404-6_19.

17 Sin-Shuen Cheung. The submodular facility location problem and the submodular joint replenishment problem. In Evripidis Bampis and Ola Svensson, editors, *Approximation and Online Algorithms - 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014, Revised Selected Papers*, volume 8952 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 2014. doi:10.1007/978-3-319-18263-6_7.

18 Nico P. Dellaert and M. Teresa Melo. Heuristic procedures for a stochastic lot-sizing problem in make-to-order manufacturing. *Ann. Oper. Res.*, 59(1):227–258, 1995. doi:10.1007/BF02031749.

19 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344. ACM, 2016. doi:10.1145/2897518.2897557.

20 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. *Theor. Comput. Sci.*, 754:122–129, 2019. doi:10.1016/j.tcs.2018.07.004.

21 Naveen Garg, Rohit Khandekar, Goran Konjevod, R. Ravi, F. Sibel Salman, and Amitabh Sinha II. On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design problem. In *Integer Programming and Combinatorial Optimization, 8th International IPCO Conference, Utrecht, The Netherlands, June 13-15, 2001, Proceedings*, pages 170–184, 2001. doi:10.1007/3-540-45535-3_14.

22 Fabrizio Grandoni and Giuseppe F. Italiano. Improved approximation for single-sink buy-at-bulk. In *Algorithms and Computation, 17th International Symposium, ISAAC 2006, Kolkata, India, December 18-20, 2006, Proceedings*, pages 111–120, 2006. doi:10.1007/11940128_13.

23 Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation for the single sink edge installation problem. *SIAM J. Comput.*, 38(6):2426–2442, 2009. doi:10.1137/050643635.

24 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020. doi:10.1145/3357713.3384277.

25 Anupam Gupta, Amit Kumar, and Tim Roughgarden. Simpler and better approximation algorithms for network design. In *35th STOC*, pages 365–372, 2003.

26 Anupam Gupta, R. Ravi, Kunal Talwar, and Seeun William Umboh. LAST but not least: Online spanners for buy-at-bulk. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 589–599. SIAM, 2017. doi:10.1137/1.9781611974782.38.

27 Raja Jothi and Balaji Raghavachari. Improved approximation algorithms for the single-sink buy-at-bulk network design problems. *J. Discrete Algorithms*, 7(2):249–255, 2009. doi:10.1016/j.jda.2008.12.003.

28 F. Sibel Salman, Joseph Cheriyan, R. Ravi, and S. Subramanian. Buy-at-bulk network design: Approximating the single-sink edge installation problem. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana.*, pages 619–628, 1997. URL: http://dl.acm.org/citation.cfm?id=314161.314397.

29 Kunal Talwar. The single-sink buy-at-bulk LP has constant integrality gap. In *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings*, pages 475–486, 2002. URL: http://link.springer.de/link/service/series/0558/bibs/2337/23370475.htm, doi:10.1007/3-540-47867-1_33.

30 Noam Touitou. Nearly-tight lower bounds for set cover and network design with deadlines/delay. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.53.

31 Seeun Umboh. Online network design algorithms via hierarchical decompositions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1373–1387, 2015. doi:10.1137/1.9781611973730.91.

Limitations of Local Quantum Algorithms on Random MAX-k-XOR and Beyond

Chi-Ning Chou
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Peter J. Love
Department of Physics and Astronomy, Tufts University, Medford, MA, USA

Juspreet Singh Sandhu
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA

Jonathan Shi
Department of Computing Sciences, Bocconi University, Milan, Italy

Abstract

We introduce a notion of *generic local algorithm*, which strictly generalizes existing frameworks of local algorithms such as *factors of i.i.d.* by capturing local *quantum* algorithms such as the Quantum Approximate Optimization Algorithm (QAOA).

Motivated by a question of Farhi et al. [arXiv:1910.08187, 2019], we then show limitations of generic local algorithms including QAOA on random instances of constraint satisfaction problems (CSPs). Specifically, we show that any generic local algorithm whose assignment to a vertex depends only on a local neighborhood with $o(n)$ other vertices (such as the QAOA at depth less than $\varepsilon \log(n)$) cannot arbitrarily-well approximate boolean CSPs if the problem satisfies a geometric property from statistical physics called the coupled overlap-gap property (OGP) [Chen et al., Annals of Probability, 47(3), 2019]. We show that the random MAX-k-XOR problem has this property when $k \geq 4$ is even by extending the corresponding result for diluted k-spin glasses.

Our concentration lemmas confirm a conjecture of Brandao et al. [arXiv:1812.04170, 2018] asserting that the landscape independence of QAOA extends to logarithmic depth – in other words, for every fixed choice of QAOA angle parameters, the algorithm at logarithmic depth performs almost equally well on almost all instances. One of these lemmas is a strengthening of McDiarmid's inequality, applicable when the random variables have a highly biased distribution, and may be of independent interest.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Combinatorics; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Algorithms, Spin Glasses, Hardness of Approximation, Local Algorithms, Concentration Inequalities, Overlap Gap Property

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.41

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2108.06049` [11]

Funding *Chi-Ning Chou*: Supported by NSF awards CCF 1565264 and CNS 1618026.
Peter J. Love: Supported by NSF STAQ award PHY-1818914 and DARPA ONISQ program award HR001120C0068.
Juspreet Singh Sandhu: Supported by NSF STAQ award PHY-1818914 and DARPA ONISQ program award HR001120C0068.
Jonathan Shi: Supported by European Research Council (ERC) award No. 834861.

Acknowledgements We thank Jonathan Wurtz for many insightful discussions about QAOA. We are grateful to Amartya Shankha Biswas for patiently explaining the *factors of i.i.d.* framework to us. We would also like to thank Antares Chen for many invigorating and profound discussions

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 41; pp. 41:1–41:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

which culminated as the open problem proposed in Problem 5.1. Lastly, we would like to thank Boaz Barak for providing detailed and helpful feedback on a prior version of this manuscript, and David Gamarnik for his explanations on the state of the art results in the research area.

1 Introduction

Recent developments [3, 26, 16] of noisy intermediate-scale quantum (NISQ) devices [40] have brought us to the door of near-term quantum computation. As experimentalists can now build programmable quantum simulators up to 256 qubits [16], this motivates an important theoretical question: what computational advantage can such a NISQ device provide?

One of the constraints of NISQ devices is the inability to create high-fidelity global entanglement. This motivates the study of the power of quantum algorithms that are *local*. A leading candidate in this regime of quantum algorithms is the *Quantum Approximate Optimization Algorithm (QAOA)* [19] at shallow depths. While there have been some recent results [29, 5, 33] that formally examine the QAOA algorithm at depth $p = 1$ or 2, very few results exist for super-constant depth QAOA [17, 18].

Given the imminent quest of demonstrating quantum computational advantage, it is important to clarify for what optimization problems can near-term quantum algorithms (such as *local* quantum algorithms) reliably be expected to demonstrate computational advantage.

We show that local *quantum* algorithms, a large natural class of NISQ algorithms, are obstructed by a geometric property of the solution space known as the *coupled Overlap-Gap Property* [10]. We conjecture that this property is satisfied by most CSPs (Conjecture 5.1). Specific problems known to have this property include the diluted k-spin glass Hamiltonian (equivalent to a max-cut problem on random k-hypergraphs) [10], independent set on random graphs [17], planted clique [25], and many other problems that so far seem to elude efficient algorithms and be algorithmically hard [22]. In this manuscript, we also demonstrate that the random Max-k-XOR problem has this property (see subsection 2.4).

Our results lift the continuous coupled interpolation techniques of Chen et al. [10] to the generic quantum-inclusive setting, with stronger and more general concentration of measure statements about local quantum algorithms, extending the techniques of Farhi et al. [17].

Critical to our approach is a new definition of local algorithms we term *generic local algorithms* (See subsection 2.1). Previous work relating statistical-physics-derived OGPs to local algorithms leveraged the *factors of i.i.d.* framework for local algorithms, which fails to contain local quantum algorithms, as we demonstrate in Proposition 2.3. Our definition of *generic local algorithms* subsumes local quantum and classical algorithms (see Proposition 2.3 and Proposition 4.3) but still satisfies strong concentration properties (see Theorems 5.3 and 5.4 in the full version), allowing techniques for local classical algorithms [10] to apply to the quantum case. Two of our core technical contributions involve showing that the random MAX-k-XOR problem has a coupled Overlap Gap Property (see subsection 2.4) by extending the techniques of Chen et al. [10] and deriving a strengthened version of McDiarmid's inequality for highly-biased random variables using a martingale argument (see Lemma 2.7).

The rest of the paper is organized as follows: In subsection 1.1 we introduce the relevant spin glass literature, defining the notion of a diluted k-spin glass; in subsection 1.2 we introduce the relevant prior work; in subsection 1.3 we state our main theorems (informally); in subsection 1.4 we briefly explain the architecture of our proof and compare our techniques with those of Chen et al. [10] and Farhi et al. [17]; in subsection 1.5 we introduce the necessary mathematical preliminaries and notation, including the models of CSPs we work with and a rigorous defintion of local classical algorithms; in section 2 we state our separation of different

families of local algorithms and our main concentration lemmas; in section 5 we conclude by summarizing our results and mentioning many natural open problems closely related to and/or motivated by our work. Complete proofs and technical details are delegated to the appendices.

1.1 Diluted k-spin glasses, maximum cuts of sparse hypergraphs, and Max-k-XOR

Spin glass theory is a central theoretical framework in statistical physics. The Sherrington-Kirkpatrick model (SK model) [43] is one of the most well studied mathematical models in the theory and consists of two variables: spins $\{\sigma_i\}_{i\in[n]}$ and interactions $\{J_{i,j}\}_{i,j\in[n]}$. A spin σ_i takes values in $\{\pm 1\}$ and the interaction $J_{i,j}$ between two spins σ_i, σ_j is a real-valued variable that captures whether the physical system prefers the two spins to be the same ($J_{i,j} > 0$) or different ($J_{i,j} < 0$). The goal is to understand what spin configurations $\sigma \in \{-1, 1\}^n$ maximize the following quantity (a.k.a. Hamiltonian):

$$H(\sigma) = \sum_{i,j} J_{i,j}\sigma_i\sigma_j \,.$$

The setting is easily generalized to higher order interactions, i.e., $J_{i_1,\ldots,i_k}$ acting on k spins, and this is known as the k-spin model. See [35] for a comprehensive survey.

There is a natural correspondence between spin glass theory and combinatorial optimization problems. In a combinatorial optimization problem (e.g., MAX-CUT), a variable corresponds to a spin and a constraint corresponds to an interaction. Through this correspondence, the maximization of the above Hamiltonian $H(\sigma)$ serves as a proxy for maximizing the number of satisfied constraints in the combinatorial optimization problem.

A spin glass model additionally specifies a particular distribution on the interactions $\{J_{i,j}\}$ for all $i, j \in [n]$. The quantity of interest is the asymptotic maximum value

$$H^* := \lim_{n\to\infty} \frac{1}{n} \max_{\sigma} H(\sigma)\,,$$

(a.k.a. the ground state energy density), as well as spin configurations σ with $H(\sigma) \approx H^*$. There are many well-studied spin glass models in physics and various mathematical insights about these have been discovered over the years [12, 14, 42, 37]. For example, for the SK model [43], Parisi [38] proposed the infamous *Parisi Variational Principle* to capture the exact value of H^*. This was later rigorously proved by Talagrand [44] and again by Panchenko [36] in greater generality. These successes give hope to design local algorithms that simulate the physical system and output a final configuration as an approximation to the corresponding combinatorial optimization problem.

While traditional spin glass models consider the underlying non-trivial interactions as either lying on a certain physically-realistic graph (e.g., the non-zero $J_{i,j}$ form a 2D-grid) or being a *mean field approximation* (for example, where every $J_{i,j}$ is non-trivial), the applications in combinatorial optimization often require the underlying constraint graphs to be sparse and arbitrary. We use two methods of bridging the gap between the two settings:

- By studying the *diluted k-spin glass model* where one first samples a sparse hypergraph and then assigns non-trivial interactions on top of its hyperedges. Intuitively, approximating the H^* of the diluted k-spin glass corresponds to approximating the maximum cut over random sparse hypergraphs. This correspondence is made more precise in Definition 1.5.

- Using the techniques of *Gaussian interpolation* [27] and *Poisson interpolation* [21, 10] from statistical physics to relate the behavior of random dense spin glass models to random sparse CSPs. More specifically, we relate the random Max-k-XOR problem to mean-field p-spin glasses (Section 8 in the full version) by modifying the Guerra-Tonnineli interpolation used in Chen et al. [10].

Table 1 A dictionary between spin glass models and combinatorial optimization problems.

Spin glass models	Combinatorial optimization problems
Spins $\sigma \in \{-1,1\}^n$	An assignment to boolean variables
Interactions $\{J_{i_1,\ldots,i_k}\}_{i_1,\ldots,i_k\in[n]}$	Constraints (i.e., hyperedges)
Hamiltonian $H(\sigma)$	Value of an assignment (i.e., $\mathsf{val}_\Psi(\sigma)$)
Ground state energy H^*	Optimal value (i.e., val_Ψ)
Mean field model (e.g., SK model)	The underlying hypergraph *not* being a lattice
Diluted spin glass model	The underlying hypergraph being sparse

1.2 Prior Work

1.2.1 Constraint-satisfaction problems & hardness for classical algorithms

CSPs (described formally in Definition 1.4) are a natural class of combinatorial optimization problems that have been studied extensively in theoretical computer science [7, 32]. Many NP-Complete problems such as k-SAT, k-NAE-SAT, MAX-CUT and k-XOR, can be framed as CSPs. Consequently, unless $\mathcal{P} = \mathcal{NP}$, finding optimal solutions to these problems is infeasible. A natural question then is to understand how well can approximate answers to instances of these problems be constructed by efficient algorithms. Under the now widely believed *Unique Games Conjecture* [31], upper bounds on the approximability of CSPs are known [30, 41]. These bounds, however, are only worst-case and do not necessarily explicitly demonstrate a family of instances of a CSP that are hard to approximate. Additionally, they remain conditional on a positive resolution to the Unique Games Conjecture, which is still a difficult open problem in the field. In the average-case regime, the goal is to ask how well a *typical* instance of a CSP can be approximated, where the instance is chosen from a "natural" distribution over the set of instances. Perhaps surprisingly, great insight has been drawn about the algorithmic hardness (or lack thereof) about random instances of many CSPs based on work originating in the Statistical Physics community, particularly in Spin Glass Theory [34, 21, 39]. This was so because the problem of finding spin configurations of particles in many spin glass models that put a system in the ground state could naturally be interpreted as a CSP. Various iterative algorithms were proposed to study the problem of explicitly finding near-ground states of *typical* instances of various spin glass models [46, 9]. It was observed that these algorithms either consistently got better with the number of iterations, or hit a threshold which they could not exceed. To understand this, the work of Achlioptas et al. [1] studied the solution geometry of the k-SAT problem and found that most good solutions were in well separated clusters. Additionally, most variables in a good solution could only take a single value (i.e., they were "frozen"). This observation was stated as an intuitive reason for the failure of local algorithms on random instances of k-SAT. Gamarnik et al. [24] made this more formal and precise by showing that *no* classical local algorithm (described formally as *factors of i.i.d.*, see Definition 1.10) could approximate the

MAX-IND-SET problem arbitrarily well on sparse random graphs. Critical to their argument was the fact that *all* (not *most*) nearly-optimal solutions to the problem satisfied the *Overlap Gap Property* - they were in well separated clusters. In various works that followed up, many problems have been shown to have near-optimal solutions conform to this solution geometry and algorithmic hardness for various families of classical algorithms has been established [10, 23, 22].

1.2.2 The Overlap Gap Property & QAOA

A depth-p $QAOA$ algorithm, abbreviated as QAOA_p, applies a length-p sequence of unitaries to each hyperedge and outputs the measurement result of the final state. The goal of $QAOA$ is to approximate the optimal solutions on average-case (i.e., random) instances. For example, the work of Farhi et al. [19] showed that $QAOA_1$ achieved 0.6924-approximation for MAX-CUT on triangle-free 3-regular graphs while such an approximation ratio was not then known to be achievable by local classical algorithms.

Shortly after $QAOA_p$ (defined formally in Definition 2.2) was proposed as a way to solve hard optimization problems and possibly establish quantum computational advantage for MAX-CUT on 3-regular graphs [19], a local classical algorithm was designed that would outperform $QAOA_p$ on these graphs at depth 1 [29]. Consequently, because of a flurry of follow up results, QAOA has been shown to be outmatched by local classical algorithms up to depth 2 [33, 5] for the MAX-CUT problem on d-regular graphs with large girth. In fact, under the widely-believed conjecture in the Spin-Glass Theory community that the SK model does not satisfy the *Overlap Gap Property* [4], an AMP algorithm was recently proposed that outputs arbitrarily good cuts for large (but constant) degree random regular graphs [2]. To analyze the performance of $QAOA_p$ on a problem that possesses an OGP, Farhi et al. [17] was established that $QAOA_p$ with depth $p \leq \varepsilon \log(n)$ could not output independent sets of size better than .854 times the optimal for sparse random graphs. This work suggested that the OGP may broadly prove to be an obstacle for $QAOA_p$ while it is local as much as it does for various classical algorithms. However, MAX-IND-SET is not a (maximum) CSP and, additionally, the prior work [17] does not give an analysis that generalizes to CSPs. Our work establishes this generalization (see Theorem 1.1) and also immediately positively resolves the "landscape independence" conjecture of $QAOA_{\varepsilon \log(n)}$ (see Theorem 1.3) proposed by Brandao et al. [8]. This immediately suggests that quantum advantage is unlikely to be found up to this depth for CSPs with an OGP, and we conjecture that *almost all* CSPs will have an OGP (see subsection 5.1).

1.3 Our results

Our main result establishes that *any* local quantum algorithm, including $QAOA_p$ with $p \leq \varepsilon \log(n)$, is obstructed from arbitrarily approximating *any* CSP that satisfies a "coupled" *Overlap Gap Property* (described in subsubsection 1.5.3). The precise family of CSPs we will work over will be notated as (k, d)-$\mathsf{CSP}(f)$, where k denotes the number of variables in a single clause, d denotes the average number of clauses any variable appears in, and f denotes the predicate applied to each clause. For a formal definition, refer to Definition 1.4. We state the informal version of our results, the formal versions of which may be found in Theorems 4.2 and 4.3 in the full version of this paper.

▶ **Theorem 1.1** (Obstruction to $QAOA_p$ given coupled OGP, informal)**.** *Given a uniformly random instance Ψ of a (k,d)-CSP(f) that satisfies a coupled OGP, with high probability a depth-p $QAOA_p$ circuit with*

$$p \leq \frac{\log(n)}{2\log(d(k-1)/\ln(2))} - 1$$

cannot output a solution that is better than a $(1-\varepsilon_0)$-approximation for some $\varepsilon_0 > 0$.

In particular, this immediately implies an obstruction for approximating maximum cuts of random sparse hypergraphs, as a coupled OGP is known to exist for that problem [10]. Furthermore, though the obstruction in Theorem 1.1 is stated for $QAOA_p$, it will apply to any generic local algorithm, and will, therefore, also apply to any local quantum algorithm. This is stated more precisely in Theorem 2.8.

▶ **Theorem 1.2** (Obstruction to generic local algorithms on random Max-k-XOR, informal)**.** *For every even $k \geq 4$, there exists $d_0 \in \mathbb{N}$ and the following holds. There exists $\varepsilon_0 > 0$ such that if $QAOA_p$ outputs a solution $\sigma \in \{-1,1\}^n$ with $H(\sigma)$ being $(1-\varepsilon_0)$-close to the H^* of a random* Max-k-XOR *instance of average degree $d \geq d_0$ with probability at least* 0.99*, then $p = \Omega(\log n)$.*

This is stated formally in Corollary 4.4 in the full version, and answers a question of Farhi et al. [20], where the authors ask if $QAOA_p$ would perform well on k-spin generalizations of the SK model, citing Max-k-XOR in particular [10]. The above result is immediately implied by a proof of a coupled Overlap-Gap Property for the Max-k-XOR problem, stated in Lemma 2.9 and proved in Section 8 in the full version.

To prove Theorem 1.1, two key lemmas about the concentration of output of local quantum algorithms need to be proved (Theorems 5.3 and 5.4 in the full version). A corollary to these two lemmas is that the quality of solution output by $QAOA_p$ (with p as stated in Theorem 1.1) concentrates heavily around the expected value. More specifically, if we let $\mathsf{val}_\Psi(\sigma)$ denote the number of clauses of Ψ satisfied by an assignment $\sigma \in \{\pm 1\}^n$ to the variables, then this value has small deviation on almost all instances.

▶ **Theorem 1.3** (Landscape-independence of $QAOA_{\varepsilon\log(n)}$, informal)**.** *Given a random instance Ψ of a (k,d)-CSP(f) and a $QAOA_p$ circuit with depth p as stated in Theorem 1.1, the solution σ output by $QAOA_p$ with value $\mathsf{val}_\Psi(\sigma)$ concentrates as,*

$$\Pr\left[\left|\mathsf{val}_\Psi(\sigma) - \mathbb{E}[\mathsf{val}_\Psi(\sigma)]\right| \geq \delta n\right] \leq e^{O(n^\gamma)},$$

for every $\delta > 0$ and some $\gamma > 0$, and the probability taken over both the input distribution and internal randomness of the algorithm.

The theorem above immediately confirms a conjecture by Brandao et al. [8] about the "landscape independence" of $QAOA_p$ upto depth $\varepsilon\log(n)$. The "landscape independence" of $QAOA_p$ is a term which asserts that the algorithm performs almost equally well on almost all instances.

1.4 Technical overview

1.4.1 Chen et al. [10] analysis

Chen et al. [10] establish a coupled overlap-gap property (OGP) for the maximum cut of random hypergraphs. The property says that for two "coupled" random instances and any nearly optimal solutions $\sigma_1, \sigma_2 \in \{-1,1\}^n$ of these, the solutions either have large or small

overlap on the assignment values to the variables, i.e., there exists an interval $0 < a < b < 1$ such that $\langle \sigma_1, \sigma_2 \rangle / n \notin [a, b]$. The coupled OGP holds over an interpolation of a pair of hypergraphs $\{(G_1(t), G_2(t))\}_{t\in[0,1]}$ with the following three properties: for every $t \in [0, 1]$, denote $\sigma_1(t)$ and $\sigma_2(t)$ as the outputs of a factors i.i.d. algorithm on inputs $G_1(t)$ and $G_2(t)$ respectively. (i) when $t = 0$, $(G_1(0), G_2(0))$ are independent random hypergraphs and $\langle \sigma_1(0), \sigma_2(0) \rangle / n < a$ with high probability; (ii) when $t = 1$, $G_1(1) = G_2(1)$ are the same random hypergraph and $\langle \sigma_1(1), \sigma_2(1) \rangle / n = 1$ with high probability; (iii) for each $t \in [0, 1]$, the correlation $\langle \sigma_1(t), \sigma_2(t) \rangle / n$ between the two solutions is highly concentrated (with respect to the randomness of $G_1(t), G_2(t)$ and the algorithm) to a value $R(t)$, and $R(t)$ is a continuous function of t. By the intermediate value theorem, this contradicts the OGP if the solutions are nearly optimal and hence no such factors of i.i.d. algorithm can exist. Note that it is also important to assert that the hamming weight and the objective function values output by the algorithm also concentrate.

1.4.2 Our analysis

The key part of the proof in Chen et al. [10] that does not work for $QAOA_p$ is item (iii) of step 2. Specifically, local quantum algorithms are not factors of i.i.d. algorithms and hence their concentration analysis on the correlation between solutions to coupled instances does not apply. Intuitively, this is because local quantum circuits can induce entanglement between qubits in a local neighborhood which cannot be explained by a local hidden variable theory [6]. We overcome this issue by first generalizing the notion of factors of i.i.d. algorithms to what we call *generic local algorithms* (Definition 2.1).

To establish concentration of overlap for quantum local algorithms, the challenge lies in how to capture the local correlations of $G_1(t)$ and $G_2(t)$. We achieve this by defining a new notion of a random vector being *locally independent* (Definition 5.2 in the full version). This structure enables us to show concentration on a fixed instance over multiple runs of the generic local algorithm with respect to its internal randomness (Theorem 5.3 in the full version). Finally, to establish concentration between a pair of correlated instances ($G_1(t)$ and $G_2(t)$), we strengthen McDiarmid's inequality for biased distributions (Lemma 2.7) and this allows the concentration analysis of the correlation function $R(t)$ to pull through (Theorem 5.1 in the full version). We complete the analysis by showing that the hamming weight and objective function values output by a generic p-local algorithm also concentrate on any (k, d)-CSP(f), obtained as corollaries to the main concentration lemmas (Corollaries 5.10 and 5.13 in the full version).

1.4.3 Comparison with Chen et al. [10] and Farhi et al. [17]

We augment the techniques of Farhi et al. [17] to handle a coupled OGP over a continuous interpolation, as opposed to the coupled OGP in Farhi et al. [17] which is over a fixed discrete interpolation. The advantage of this is to enable the use of a broader family of coupled OGPs provable using statistical mechanics methods, whereas the coupled OGP of Farhi et al. [17] requires reasoning about explicit sequences of instances in a way that does not clearly generalize from their independent set analysis to the setting of CSPs.

Our statements additionally show stronger concentration than those of Chen et al. [10] which is necessary to demonstrate that polynomially many runs of the algorithm will (with high probability) not succeed. Lastly, we extend the coupled OGP from the setting of diluted k-spin glasses shown by Chen et al. [10] to the setting of random Max-k-XOR.

1.5 Preliminaries & notation

1.5.1 Constraint-satisfaction problems & hypergraphs

Constraint satisfaction problems are a class of optimization problems where a set of constraints on the underlying variables need to be satisfied in tandem. We restrict our attention to the setting where the variables are boolean valued and the number of constraints are *sparse*. Furthermore, every constraint involves k variables, where k is a constant independent of the number of variables.

▶ **Definition 1.4** (Random (k,d)-CSP(f))**.** *A (signed) random (k,d)-CSP(f) instance with a local constraint function $f : \{-1,1\}^k \to \{0,1\}$ is constructed as follows:*

1. *Choose $r \sim \text{Poisson}(dn/k)$.*
2. *Sample r clauses of size k by choosing each clause C_i independently as a collection of k variables uniformly at random from $\{x_1, \ldots, x_n\}^k$, and, in the case of a signed random CSP, random signs $s_{i,1}, \ldots, s_{i,k} \in \{\pm 1\}$.*

To each clause C_i there are k variables associated: $\{x_{i_1}, \ldots, x_{i_k}\}$. A clause is satisfied if there is some assignment to every $x_{i_j} \in \{-1,1\}$, such that, $f(x_{i_1}, \ldots, x_{x_k}) = 1$ (or $f(s_{i,1}x_{i_1}, \ldots, s_{i,k}x_{x_k}) = 1$ if signed). The value of an assignment $\sigma \in \{-1,1\}^n$ is defined as $\mathsf{val}_\Psi(\sigma) := \#\{C_i : f(\sigma_{i_1}, \ldots, \sigma_{i_k}) = 1\}$ (or $\#\{C_i : f(s_{i,1}\sigma_{i_1}, \ldots, s_{i,k}\sigma_{i_k}) = 1\}$ if signed). The optimal value of Ψ is defined as $\mathsf{val}(\Psi) := \max_\sigma \mathsf{val}_\Psi(\sigma)$.

When unspecified, we will be referring to unsigned CSPs. In a diluted k-spin glass, the underyling particles can be thought of as vertices of a hypergraph with $\frac{dn}{k}$ hyperedges. Each hyperedge is a tuple of k-vertices, and any interaction $J_{i_1,\ldots,i_k}$ over a tuple $(i_1, \ldots, i_k)$ is -1 if $(i_1, \ldots, i_k)$ is a hyperedge of G and 0 otherwise.

▶ **Definition 1.5** (Random k-uniform hypergraph)**.** *A* random *k-uniform hypergraph consists of choosing a number of edges $|E| \sim \text{Poisson}(\frac{dn}{k})$ and then choosing hyperedges $e_1, \ldots, e_{|E|}$ independently and uniformly at random from the set $\{1, \ldots, n\}^k$ of all vertex k-tuples.*

The underlying hamiltonian of a diluted k-spin glass then is,

$$H^G_{n,k,d}(\sigma) = -\sum_{e \in E} \sigma_{e_1} \ldots \sigma_{e_k}\,, \tag{1}$$

which on maximization corresponds to the MAX-CUT of a random k-uniform hypergraph. Observe that a (k,d)-CSP(f) can be encoded as a diluted k-spin glass by choosing the variables to be the set of vertices, the clauses to be the hyperedges, and the hamiltonian to be the same as in Equation 1 with the additional requirement that f acts on the variables in every hyperedge.

1.5.2 Vanishing Local Neighborhoods of Sparse Random Hypergraphs

We state a bound on sufficiently local neighborhoods of random sparse k-uniform hypergraphs. This bound is used in the obstruction result to precisely quantify the size of the neighborhood $p(n)$ up to which a $p(n)$-generic local algorithm does not "see the whole hypergraph" around any vertex.

▶ **Lemma 1.6** (Vanishing local neighborhoods of random sparse k-uniform hypergraphs). *Let $k \geq 2$ and $d \geq 2$ and $\tau \in (0,1)$. Then there exists $a > 0$ and $0 < A < 1$, such that, for n large enough and p satisfying*

$$2p + 1 \leq \frac{(1-\tau)\log n}{\log\left(\frac{d(k-1)}{\ln 2}\right)},$$

the following are true:

$$\Pr_{G\sim\mathcal{H}_{n,d,k}}[\max_i B_G(v_i, 2p) \geq n^A] \leq e^{-n^a}, \quad \text{and} \quad \Pr_{G\sim\mathcal{H}_{n,d,k}}[\max_i B_G(v_i, p) \geq n^{\frac{A}{2}}] \leq e^{-n^{\frac{a}{2}}}.$$

Intuitively, the above lemma says that the local neighborhood of each vertex is vanishingly small with high probability. To prove Lemma 1.6, we utilize a modified version of the proof of Farhi et al. [17, Neighborhood Size Theorem] to handle the case of sparse random hypergraphs and we defer the complete proof to Appendix B in the full version.

1.5.3 The Overlap Gap Property

We say that a (k,d)-CSP(f) (signed or unsigned) satisfies a *coupled overlap-gap property (OGP)* if, given two instances Ψ, Ψ' constructed so that they share a random t-fraction of clauses with the remaining $(1-t)$-fraction chosen independently, any two "good" solutions σ of Ψ and σ' of Ψ' are either very similar of dissimilar.

▶ **Definition 1.7** (Coupled OGP, informal). *A signed or unsigned (k,d)-CSP(f) satisfies a coupled OGP if there exists $\varepsilon_0 > 0$ and $0 < a < b < 1$ such that the following hold for every $t \in [0,1]$: Given two (k,d)-CSP(f) instances Ψ, Ψ' constructed so that they share a random t-fraction of their clauses and have the remaining $(1-t)$-fraction of clauses chosen independently and uniformly at random, then for every $0 < \varepsilon < \varepsilon_0$, the overlap between any $(1-\varepsilon)$-optimal solution σ of Ψ and σ' of Ψ' satisfies*

$$\frac{1}{n}\langle\sigma,\sigma'\rangle = \frac{1}{n}\left(\sum_{i=1}^{n}\sigma_i\sigma_i'\right) \notin [a,b]$$

with high probability.

A formal definition of the interpolation procedure described in Definition 1.7 and a complete and formal statement of a coupled OGP are provided in section 3.

1.5.4 Local classical algorithms

A local classical algorithm takes as input a hypergraph G and a label set S, runs a stochastic process $\{X^G(t)\}_t$ that associates to each vertex v a label $X_v^G(t) \in S$ at time t, and outputs an assignment σ_v to each vertex v according to its final label. While there is a huge design space for local classical algorithms, a *factors of i.i.d. algorithm* of radius p has the following restrictions:

1. The initial label for each vertex v is set to be one from an i.i.d. set of random variables $X_v^G(0)$.
2. For each vertex v, the assignment σ_v is a random variable that only depends on the labels from a p-neighborhood of v and is updated via a stochastic process.
3. The assignment function (also known as the *factor*) for each vertex is the same.
4. The label assignment of two vertices v, v' in hypergraphs G and G' are equivalent if the two vertices have *isomorphic* p-neighborhoods.

As we shall see, in subsection 2.1, we relax the first three conditions to include a larger class of local algorithms, which subsume local quantum algorithms. We now state the definition of the p-neighborhood of a vertex in a hypergraph, which generalizes the p-neighborhood of a graph by considering two vertices v and w to be adjacent if they belong to the same hyperedge e.

▶ **Definition 1.8** (p-neighborhood and hypergraphs with radius p)**.** *Let G be a hypergraph, $v \in V(G)$, and $p \in \mathbb{N}$. The p-neighborhood of v is defined as*

$$B_G(v,p) := \{w \in V(G) \,|\, w \text{ can be reached in } p \text{ steps from } v\}.$$

Let G be a hypergraph, $v \in V(G)$, and $p \in \mathbb{N}$. We say (G,v) has radius p if $B_p(G_v) = V(G)$. Further, let $k \in \mathbb{N}$, we define

$$\mathcal{G}_p := \{(G,v) \,|\, (G,v) \text{ has radius } p \text{ and } G \text{ is connected, finite, and } k\text{-uniform}\}$$

be the collections of hypergraphs with radius at most p.

To capture the fact that classical local algorithms assign the value of a vertex v by only looking at a stochastic process of the p-neighborhood, we define a factor f as a measurable function which gives a label $\sigma_v \in \{\pm 1\}$ to every vertex $v \in V(G)$. We restrict to the case where the label set $S = [0,1]$.

▶ **Definition 1.9** (Factor of radius p)**.** *Let $p \in \mathbb{N}$. We define the collection of all $[0,1]$-labelled hypergraphs of radius at most p as*

$$\Lambda_p := \left\{(G,v,X) \,|\, (G,v) \in \mathcal{G}_p \text{ and } X \in [0,1]^{V(G)}\right\}$$

We say $(G_1,v_1,X_1),(G_2,v_2,X_2) \in \Lambda_r$ are isomorphic if there exists a hypergraph isomorphism $\phi : V(G_1) \to V(G_2)$ such that (i) $\phi(v_1) = \phi(v_2)$ and (ii) $X_1 = X_2 \circ \phi$.

Finally, we say $f : \Lambda_p \to \{-1,1\}$ is a factor of radius p function if

1. *f is measurable.*
2. *$f(G_1,v_1,X_1) = f(G_2,v_2,X_2)$ for every isomorphic $(G_1,v_1,X_1),(G_2,v_2,X_2) \in \Lambda_r$.*

Intuitively, the output distribution of a factors of i.i.d. algorithm with radius p on a vertex v is determined by the p-neighborhood of v.

▶ **Definition 1.10** (Factors of i.i.d., [10, Section 2])**.** *Let $k,p \in \mathbb{N}$. A factors of i.i.d. algorithm A with radius p is associated with a factor of radius p function f with the following property. On input a k-uniform hyper graph G, the algorithm A samples a random labeling $X = \{X(v)\}_{v \in V(G)}$ where $X(v)$'s are i.i.d. uniform random variables on $[0,1]$. The output of A is $\sigma \in \{-1,1\}^{V(G)}$ where $\sigma_v := f(B_p(G,v), v, \{X(w)\}|_{w \in B_p(G,v)})$ for each $v \in V(G)$.*

Common local algorithms such as Glauber dynamics and Belief Propagation are examples of factors of i.i.d. algorithms.

2 Main Technical Theorems

The main results include showing that $QAOA_p$ is a generic p-local algorithm, a concept which subsumes *factors of i.i.d.* local algorithms and local quantum algorithms, and then establishing that this more general class of algorithms also have strong concentration properties. The concentration allows the original interpolation argument of Chen et al. [10] stated in subsubsection 1.4.1 to continue to pose an obstruction to generic p-local algorithms from outputting arbitrarily good solutions on almost all instances of a (k,d)-CSP(f) that has a coupled OGP.

2.1 Generic local algorithms

Generic local algorithms, including local quantum algorithms, can be defined from an information-based perspective. Specifically, given a parameter p that characterizes the locality of the information an algorithm has about the input, we can assert that a *generic* p-local algorithm makes independent decisions on appropriately far-away vertices and that all decisions for a vertex are sufficiently local.

▶ **Definition 2.1** (Generic local algorithms, informal). *Let $p \in \mathbb{N}$ and let S be a finite label set. We say an algorithm A (which takes a hypergraph G as an input) is generic p-local if the following hold:*

- **(Local distribution determination).** *For every set of vertices $L \subset V$, the joint marginal distribution of the labels $(A(G)_v)_{v\in L}$ depends only on the union of the p-neighborhoods of $v \in L$ in G.*
- **(Local independence).** *$A(G)_v$ is statistically independent of the joint distribution of $\{A(G)_{v'}\}$ for every v' that is farther than a distance of $2p$ from v.*

The definition makes no assumptions about how the randomness of the algorithm is instantiated, what sort of function is used to decide the label locally, or whether it is the same for every vertex. A formal defintion is provided in section 4, along with a procedure to partially sample from the run of the algorithm on correlated instances. We now briefly introduce the $QAOA_p$ circuit.

▶ **Definition 2.2** ($QAOA_p$ algorithm, [19]). *The $QAOA$ circuit parametrized by angle vectors $\hat{\gamma} = (\gamma_1, \ldots, \gamma_p)$ and $\hat{\beta} = (\beta_1, \ldots, \beta_p)$ looks as follows,*

$$U_p(\hat{\beta}, \hat{\gamma}) = \prod_{j=1}^{p} e^{-i\beta_j \sum_{k=1}^{n} X_k} e^{-i\gamma_j H_c(G)} .$$

In the circuit above, X_i is the Pauli-X matrix acting on the i-th qubit (with identity action everywhere else) and $H_C(G)$ is the hamiltonian that encodes the problem instance. For us, $H_C(G) = H^G_{n,k,d}$ where every k-spin interaction is replaced by a Pauli-Z interaction $Z_{e_1} \otimes \cdots \otimes Z_{e_k}$. Typically, the initial state on which the circuit is applied is a *symmetric product state*, most notably $|0\rangle^{\otimes n}$ or $|+\rangle^{\otimes n}$. The expected value that $QAOA$ outputs after applying the circuit on some initial state $|\psi_0\rangle$ is,

$$\langle\psi_0| U^\dagger H_c(G) U |\psi_0\rangle .$$

We will notate by $QAOA(\hat{\beta}, \hat{\gamma})$ a $QAOA$ circuit of depth $2p$ with angle parameters $\hat{\beta}$ and $\hat{\gamma}$. In our regime, we will work with *any* collection of *fixed* angles $(\hat{\beta}, \hat{\gamma})$.

As stated below, generic p-local algorithms strictly generalize the *factors of i.i.d.* framework as well as $QAOA_p$.

▶ **Proposition 2.3** (Generic local strictly generalizes factors of i.i.d.). *There exists a generic p-local algorithm as defined in Definition 2.1 that is not a p-local factors of i.i.d. algorithm as defined in Definition 1.10.*

▶ **Proposition 2.4** ($QAOA_p$ is generic p-local). *For every $p > 0$ and fixed angle vectors $\hat{\beta}$ and $\hat{\gamma}$, $QAOA_p(\hat{\beta}, \hat{\gamma})$ is generic p-local under Definition 2.1.*

The proofs for the two propositions above are provided in section 4.

2.2 Concentration of outputs of local quantum algorithms

Having shown that generic p-local algorithms capture classical & quantum local algorithms, we now first show that the outputs of these algorithms concentrate even when run on multiple fixed correlated instances. Additionally, the outputs concentrate over the input distribution from which the instances are drawn. The proof for the lemmas in this subsection are delegated to Section 5 in the full version.

2.2.1 Concentration over internal randomness of the algorithm

We first show that the output of generic p-local algorithms on sufficiently "local" functions of the underlying spins concentrates heavily, even when run on multiple correlated instances. We state an informal version of the lemma. For a formal and detailed version, please refer to Theorem 5.3 in the full version. "Spins" here refer to the assignment of a variable.

▶ **Lemma 2.5** (Concentration of local functions of spins, informal)**.** *Let $z,\ m \in \mathbb{N}$ and $A < 1$. Let σ be the output of a generic p-local algorithm on a fixed hypergraph G. Let $v_{j,i}$ denote the vertex corresponding to the i-th vertex (or variable) in the j-th hyperedge (or clause) with $j \in [m]$ and $i \in [z]$. Let $r = (r_1, \ldots, r_m)$ be a random vector of a correlated hypergraph G' with a certain joint independence structure (Definition 5.2 in the full version). Now, consider a sum $X = \sum_{j\in[m]} h(\sigma_{v_{j,1}}, \ldots, \sigma_{v_{j,z}})r_j$, where $|h| \leq 1$. Suppose that each vertex v occurs at most C times among the different $v_{j,i}$. Then, provided the p-neighborhood of every vertex in $G \cup G'$ has at most n^A vertices:*

$$\Pr_{\sigma,r}[|X - \mathop{\mathbb{E}}_{\sigma,r}[X]| \geq \delta n] \leq e^{-\Omega(\delta^2 m/(Czn^A))}\,.$$

The lemma above states that even when choosing two correlated instances G and G' that share, say t-fraction of their clauses, provided every vertex sees not too many vertices in the local neighborhood of the combined input $G \cup G'$, the value of sufficiently local functions h acting on the output of the algorithm concentrate. To get concentration of overlap between two solutions σ of G and σ' of G', we choose $h(\sigma_{v_{j,1}}, \ldots, \sigma_{v_{j,z}}) = \sigma_{v_j}$, $m = n$ and $r = \sigma'$. To get concentration for $\mathsf{val}_G(\sigma)$, we choose $h = f$ for the underlying (k,d)-$\mathsf{CSP}(f)$ and $m = \frac{dn}{k}$. When working to obtain concentration on a single instance, it suffices to set $G' = G$ and r to be the all 1s vector.

2.2.2 Concentration over instances

The prior lemma merely asserts that the output is concentrated for *fixed* correlated instances. However, this still doesn't imply that the output will be concentrated *across* random input instances. To do so, we need concentration over the distribution from which the inputs are sampled. This is accomplished by Theorem 5.4 in the full version, an informal version of which is stated below.

▶ **Lemma 2.6** (Concentration of differences of coupled hypergraphs, informal)**.** *Let f be a function of two hypergraphs over n vertices such that*

$$|f(G_1, G_2) - f(G_1', G_2')| \leq r(n)\,,$$

for some r whenever (G_1, G_2) differs from (G_1', G_2') by the addition and/or removal of a single hyperedge $e \in [n]^k$ from one or both hypergraphs. Furthermore, assume that the largest p-nieghborhood of G_1, G_2, G_1', G_2' has no more than $n^{\frac{A}{2}}$ vertices for some $A < 1$. Then $\exists\ a \in (0,1)$, such that

$$\Pr_{\substack{G_1,G_2\sim\\ \mathcal{H}_{n,k,d,t}}}\left[\left|f(G_1,G_2) - \mathop{\mathbb{E}}_{G_1,G_2} f(G_1,G_2)\right| \geq \delta n\, r(n)\right] \leq 2\exp\left(\frac{-3k\delta^2 n}{12(2-t)d + 2k\delta}\right) + 2\exp\left(-n^{a/2}\right).$$

The notation $G_1, G_2 \sim \mathcal{H}_{n,k,d,t}$ indicates that G_1 and G_2 are t-correlated *random* hypergraphs. This means that t-fraction of their hyperedges are chosen i.i.d. from $[n]^k$ and these are common. The remaining $(1-t)$ fraction of edges are chosen independently for each. For a formal definition of the coupled distribution, refer to Definition 3.3. The above lemma relies on two critical facts:

1. For $p \leq \varepsilon \log(n)$ (for an appropriately chosen ε), the p-neighborhood of a *random* k-uniform hypergraph has no more than $n^{\frac{A}{2}}$ vertices (Lemma 1.6).
2. A strengthening of *McDiarmid's Inequality* (Lemma 2.7) for biased distributions. This strengthening is proved via a martinagle argument.

2.2.3 Strengthened McDiarmid's Inequality

We end by stating the strengthened version of McDiarmid's Inequality (Lemma 2.7) for biased distributions, which is a generic concentration statement that may be of independent interest. The inequality adds a Bernstein-like variance term to the tail-bound that controls the likelihood of deviation, where the variance term depends on a precise quantification of how unlikely certain very rare events are.

▶ **Lemma 2.7** (McDiarmid's inequality for biased distributions). *Suppose that $X_1, \ldots, X_n$ are sampled i.i.d. from a distribution D over a finite set $\mathcal{X}$, such that D assigns probability $1-p$ to a particular outcome $\chi_0 \in \mathcal{X}$. Let $f : \mathcal{X}^n \to \mathbb{R}$ satisfy a bounded-differences inequality, so that*

$$|f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots x_n) - f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots x_n)| \leq c$$

for all $x_1, \ldots, x_n, x_i' \in \mathcal{X}$. Then

$$\Pr\left[|f(X_1, \ldots, X_n) - \mathbb{E}\, f(X_1, \ldots, X_n)| \geq \varepsilon\right] \leq 2 \exp\left(\frac{-\varepsilon^2}{2np(2-p)c^2 + 2c\varepsilon/3}\right).$$

The above inequality is robust in that if there are highly unlikely or likely (think $p = o_n(1)$) events that cause large deviations, then these deviations can be absorbed into the Bernstein-like variance term in the tail bound term. This means that one can get strong concentration bounds even in the presence of a vanishingly small set of events that cause large deviations. This property ends up being critical when showing concentration of outputs of generic local algorithms over the input distribution.

2.3 Obstruction of generic local algorithms by the Overlap Gap Property

We state our main theorem (informally) that obstructs all generic p-local algorithms on random instances Ψ of a (k,d)-CSP(f) that satisfies the coupled Overlap-Gap Property.

▶ **Theorem 2.8** (Obstruction to Generic p-Local Algorithms given coupled OGP, informal). *Let f be a boolean predicate defined on k variables and $p(n)$ be such that it satisfies the requirements of Lemma 2.5 and Lemma 2.6. Furthermore, assume that (k,d)-CSP(f) satisfies the coupled Overlap Gap Property. Then, given an instance $\Psi \sim (k,d)$-CSP(f) and a generic $p(n)$-local algorithm $\mathcal{A}$ that outputs a solution σ on input Ψ, with high probability, the solution will be no more than $(1-\varepsilon_0)$-optimal for some $\varepsilon_0 > 0$.*

The theorem above is stated in full precision and formality in Theorem 4.3 in the full version and proved formally in Section 6 of the full version. The theorem effectively obstructs any algorithm that makes assignments for variables by looking at $o(n)$ sized

local neighborhoods *irrespective* of how these decisions are made and what concrete model of randomness is used, provided the problem exhibits a coupled OGP. For instance, the obstruction would hold even for quantum algorithms that assign labels based on the local neighborhoods of qubits *and* start and end in a state that is moderately locally entangled - By this, we mean a state that exhibits entanglement only between any qubit i and $o(n)$ qubits in its $p(n)$-sized local neighborhood.

2.4 Overlap Gap Property for MAX-k-XOR

In Section 8 in the full version, we establish that the random Max-k-XOR problem obeys a coupled OGP, so that the obstruction implied by Theorem 2.8 holds for this problem.

▶ **Lemma 2.9** (Coupled OGP for random Max-k-XOR). *The random* Max-k-XOR *problem satisfies Definition 1.7.*

This is proven by means of a Guerra-Tonelli interpolation [28, 10], where a dense mean-field k-spin glass model is continuously transformed to the random Max-k-XOR model. Treated as a physical system, the thermodynamic equilibrium at each point in the interpolation is calculated at a low enough temperature so that the equilibrium states are approximately optimal. The change in equilibrium energy over the course of the interpolation is then shown to be small.

A coupled OGP is shown in this way by interpolating a pair of systems constrained so that the overlap between the two states of the systems is bounded.

3 Coupled Interpolation & Overlap-Gap Properties

We now state the OGP formally as it holds for the diluted k-spin glass model in both uncoupled and coupled form. To do so, we begin by introducing the notion of an overlap between two spin-configurations σ_1 and σ_2, which is equivalent to the number of spins that are the same in both configurations subtracted by the number of different spins, normalized by the number of particles in the system. Formally,

▶ **Definition 3.1** (Overlap between spin configuration vectors). *Given any two vectors $\sigma_1, \sigma_2 \in \{-1, 1\}^n$, the overlap between them is defined as,*

$$R(\sigma_1, \sigma_2) = \frac{1}{n}\langle \sigma_1, \sigma_2 \rangle = \frac{1}{n} \sum_{i \in [n]} (\sigma_1)_i (\sigma_2)_i \,.$$

We first state the OGP for diluted k-spin glasses about the overlap gaps in a *single instance.*

▶ **Theorem 3.2** (OGP for Diluted k-Spin Glasses, [10, Theorem 2]). *For every even $k \geq 4$, there exists an interval $0 < a < b < 1$ and parameters $d_0 > 0$, $0 < \eta_0 < P(k)$ and $n_0 > 1$, such that, for $d \geq d_0$, $n \geq n_0$ and $L = L(\eta_0, d)$, with probability at least $1 - Le^{-n/L}$ over the random hypergraph $G \sim \mathcal{H}_{n,d,k}$, whenever two spins σ_1, σ_2 satisfy*

$$\frac{H^G(\sigma_i)}{n} \geq M(k,d)\left(1 - \frac{\eta_0}{P(k)}\right),$$

then also, $|R(\sigma_1, \sigma_2)| \notin (a, b)$.

A more general version of the OGP excludes, with high probability, a certain range of overlaps between any two solutions of two different instances jointly drawn from a coupled random process. We first introduce this process, and then state the *coupled* version of the OGP as proven in Chen et al [10].

▶ **Definition 3.3** (Coupled Interpolation, [10, Section 3.2]). *The coupled interpolation $\mathcal{H}_{d,k,n,t}$ generates a coupled pair of hypergraphs $(G_1, G_2) \sim \mathcal{H}_{d,k,n,t}$ as follows:*

1. *First, a random number is sampled from* Poisson(tdn/k)*, and that number of random k-hyperedges are uniformly drawn from the set $[n]^k$ and put into a set E.*
2. *Then, two more random numbers are independently sampled from* Poisson$((1-t)dn/k)$*, and those numbers of random k-hyperedges are independently drawn from $[n]^k$ to form the sets E_1 and E_2 respectively.*
3. *Lastly, the two hypergraphs are constructed as $G_1 = (V, E \cup E_1)$ and $G_2 = (V, E \cup E_2)$.*

▶ **Theorem 3.4** (OGP for Coupled Diluted k-Spin Glasses, [10, Theorem 5]). *For every even $k \geq 4$, there exists an interval $0 < a < b < 1$ and parameters $d_0 > 0$, $0 < \eta_0 < P(k)$ and $n_0 > 1$, such that, for any $t \in [0,1]$, $d \geq d_0$, $n \geq n_0$ and constant $L = L(\eta_0, d)$, with probability at least $1 - Le^{-n/L}$ over the hypergraph pair $(G_1, G_2) \sim \mathcal{H}_{n,d,k,t}$, whenever two spins σ_1, σ_2 satisfy*

$$\frac{H^{G_i}(\sigma_i)}{n} \geq M(k,d)\left(1 - \frac{\eta_0}{P(k)}\right),$$

then their overlap satisfies $|R(\sigma_1, \sigma_2)| \notin [a, b]$.

We also provide a corresponding coupled OGP for random Max-k-XOR in Theorem 8.12 in the full version.

4 Locality and Shared Randomness

4.1 Generic p-local algorithms

We introduce a concept of "local random algorithm" which will allow for different runs of the same local algorithm to "share their randomness", even when run on mostly-different instances. Later we will demonstrate that QAOA is a local algorithm under this definition.

▶ **Definition 4.1** (Generic local algorithms). *We consider randomized algorithms on hypergraphs whose output $A(G) \in S^V$ assigns a label from some set S to each vertex in V. Such an algorithm is generic p-local if the following hold.*

- **(Local distribution determination).** *For every set of vertices $L \subset V$, the joint marginal distribution of its labels $(A(G)_v)_{v \in L}$ is identical to the joint marginal distribution of $(A(G')_v)_{v \in L}$ whenever $\bigcup_{v \in L} B_G(v,p) \cong_L \bigcup_{v \in L} B_{G'}(v,p)$, and,*
- **(Local independence).** *$A(G)_v$ is statistically independent of the joint distribution of $A(G)_{v'}$ over all $v' \notin B_G(v, 2p)$.*

Consequently, it will be possible to sample $A(G)_v$ without even knowing what the hypergraph looks like beyond a distance of p away from v.

This definition is more general than the factors of i.i.d. concept used in probability theory [24, 10]. Our definition, for instance, encompasses local quantum circuits whereas factors of i.i.d. algorithms satisfy Bell's inequalities and do not capture quantum mechanics.

▶ **Proposition 4.2** (Generic local strictly generalizes factors of i.i.d. (Restatement of Proposition 2.3)). *There exists a generic 1-local algorithm as defined in section 4 that is not a 1-local factors of i.i.d. algorithm as defined in Definition 1.10.*

A proof of this proposition is provided in Appendix A in the full version, and consists of setting up a Bell's inequality experiment within the framework of a generic 1-local algorithm.

4.2 Locality properties of QAOA for hypergraphs

We show that any QAOA circuit of depth p with some *fixed* angle parameters $(\hat{\beta}, \hat{\gamma})$ is a p-local algorithm. This allows us to describe a process to sample outputs of this circuit when it is run on two different input hypergraphs.

▶ **Proposition 4.3.** *For every $p > 0$, angle vectors $\hat{\beta}$ and $\hat{\gamma}$, $QAOA_p(\hat{\beta}, \hat{\gamma})$ is generic p-local under Definition 4.1.*

Proof. To see this, consider the structure of QAOA: we start with a product state $|\psi_0\rangle$ where each qubit corresponds to a vertex in the hypergraph, apply the unitary transformation $U = U_p(\hat{\beta}, \hat{\gamma})$ to the state, and then measure each vertex v in the computational basis with the Pauli-Z operator $\sigma_z(v)$. Equally valid and equivalent is the Heisenberg picture interpretation of this process, where we keep the product state $|\psi_0\rangle$ fixed but transform the measurements according to the reversed unitary transformation $U^\dagger$, so that we end up taking the measurements $U^\dagger \sigma_z(v) U$ on the fixed initial state.

Because the $\sigma_z(v)$ operators all commute with each other, their unitarily transformed versions $U^\dagger \sigma_z(v) U$ also mutually commute, and the measurements can be taken in any order without any change in results. Let $M(v) = U^\dagger \sigma_z(v) U$ and $M(L) = \{U^\dagger \sigma_z(u) U \mid u \in L\}$.

To show that QAOA satisfies the first property of generic p-local algorithms, we need to show that the marginal distribution of its assignments to any set $L' \subseteq V$ of vertices depends only on the union of the p-distance neighborhoods of L'. To show this, since we are allowed to take the measurements in any order, take the measurements in $M(L')$ before any other measurement. Then since the action of the unitary $U = U_p(\hat{\beta}, \hat{\gamma})$ on qubits in L' does not depend on any feature of the hypergraph outside of a radius of p around L', the operators $M(L')$ are fully determined by the p-local neighborhoods of L', and since we take them before every other measurement, the qubits are simply in their initial states when we make these measurements, thus the distribution of outputs is fully determined.

The same type of reasoning shows that the assignment to each $v \in V$ is statistically independent of the assignments to any set of vertices outside of a $2p$-distance neighborhood of v. Take $L'' \subset V \setminus B(v, 2p)$. Then $M(v)$ acts on a radius-p ball around v, and each measurement in $M(L'')$ acts on a radius-p ball around a vertex in L'', and by taking $\{M(v)\} \cup M(L'')$ before any of the measurements in $M(\{B(v, 2p) \setminus \{v\}\})$, we ensure that the qubits being measured by $M(v)$ are disjoint from and unentangled with those measured by anything in $M(L'')$. Hence the measurement $M(v)$ is independent of all measurements in $M(L'')$. We conclude that QAOA_p is a generic p-local algorithm. ◀

4.3 Shared randomness between runs of a generic local algorithm

We describe a process to sample the outputs of a generic local algorithm when run twice on two different hypergraphs, so that the two runs of the algorithm can share randomness when the hypergraphs have some hyperedges in common. This is not meant as a constructive algorithm, but a statistical process with no guarantee of feasible implementation.

The idea is to start with two t-coupled hypergraphs, which for large enough n, are likely to have some set of vertices L^+ whose p-neighborhoods are identical between the two hypergraphs. Since these vertices have identical p-neighborhoods, a generic p-local algorithm behaves identically on the vertices in L^+. We pick a random t^+ fraction of the elements of L^+, and assign the same labels to those vertices in the two coupled instances. Then the remaining labels on each hypergraph are assigned by generic p-local algorithms, conditioned on the output being consistent with the already assigned labels.

The formal definition may be found in the full version of this paper, where it is combined with the coupled OGP to demonstrate a contradiction if a generic local algorithm achieves a good approximation.

5 Conclusion & Future Work

The full version of this work conclusively establishes the coupled OGP as an obstruction to all local quantum algorithms on *any* (k, d)-CSP(f). In doing this, the work hints at and leaves open many interesting questions for future work in areas that are at the intersection of Quantum inapproximability, Statistical Physics, Random Graph Theory, Combinatorial Optimization and Average-Case Complexity.

5.1 Which CSPs have an OGP?

While various sparse CSPs such as k-SAT, unsigned max-k-XOR and k-NAE-SAT have been shown to exhibit clustering in their solution spaces at different clause-to-variable ratios [1, 15, 10], it is not known whether this property is pervasive to most CSPs or something that happens to a select few. Therefore, in order to understand the complexity landscape of CSPs on typical instances better, the following open question is interesting to investigate:

▶ **Conjecture 5.1** (Random Predicate CSPs and coupled OGP)**.** *Given a function f chosen uniformly at random from the set of functions $\mathcal{B}_k = \{g \mid g : \{\pm 1\}^k \to \{0, 1\}\}$, (k, d)-CSP(f) has a coupled-OGP for sufficiently large k and d with high probability (over the choice of f and instance $\Psi \sim (k, d)$-CSP(f)).*

Notice that the conjecture above is specifically interested in the solution geometry of a CSP in the *unsatisfiable* regime (large d). A positive resolution to the above conjecture will make the obstructions stated in Theorem 2.8 hold for *almost all* CSPs.
Another question of interest is which properties about a predicate f can be identified which would conclusively imply that a random instance Ψ of a (k, d)-CSP(f) will have an OGP.

▶ **Problem 5.1** (Properties of coupled-OGP predicates)**.** *Can we enumerate a set of necessary and sufficient conditions on f to be such that (k, d)-CSP(f) satisfies a coupled-OGP for sufficiently large k and d?*

5.2 Beyond log-depth obstructions for $QAOA_p$?

Work on obstructing $QAOA_p$ using an OGP heavily relies on the locality of the algorithm at shallow depths. It is interesting to investigate whether this obstruction can be extended beyond the $\varepsilon \log(n)$-depth regime to make this a non-local obstruction. Recent work [23, 45] suggests that the OGP may actually result in stronger obstructions than just local ones, and it would be interesting to see if these techniques can be generalized to the setting of $QAOA_p$ to yield obstructions that are non-local.

▶ **Problem 5.2** (Poylogarithmic obstructions to $QAOA_p$ in the OGP regime)**.** *Given a $QAOA_p$ circuit with depth $p \leq \varepsilon (\log(n))^c$ for some $c > 1$, does there exist $\varepsilon_0 > 0$, such that $QAOA_p$ is obstructed on a (k, d)-CSP(f) with a coupled OGP from outputting solutions that are better than (1 - ε_0) approximations to the optimal?*

5.3 A Quantum OGP and lifting "classical" obstructions

The idea of the OGP obstructing families of algorithms that are *stable* under small perturbations to the input [23] motivates the idea of a quantized version of the OGP, to apply to *quantum* CSPs. To define such a property over quantum states, however, there would need to be a metric that is very similar to the classical hamming distance over $\mathbb{F}_2$ and has the property that it is invariant over permutations of the canonical basis, while still quantifying entanglement in a desired way. One such possible metric is a *quantum* version of the Wasserstein distance of first order that was proposed by De-Palma et al. [13]. In particular, given a natural generalization of Definition 1.7 to a quantized setting using a quantum version of the Wasserstein distance of first order, it is interesting to investigate if a larger family of quantum circuits up to some depth $p(n)$ can be obstructed by a family of d-local hamiltonians $\{H_n\}_{n\geq n_0}$ that possess a qOGP (quantized Overlap-Gap Property). A result of this type could imply a way to generically "lift" classical obstructions for *stable* classical algorithms to a corresponding family of quantum algorithms.

5.4 Message-Passing algorithm for MAX-CUT of *all* d-regular graphs?

Finding an efficient classical algorithm that can output cuts that are arbitrary approximations of the optimal ones for d-regular graphs is a long-standing open problem in Random Graph Theory and Theoretical Computer Science. Recently, this problem was nearly completely solved by Alaoui et al. [2] as they constructed a Message-Passing algorithm for random regular graphs of very large degree under the widely believed no-OGP assumption about the SK model. However, the problem does not provide a complete solution as it needs the degree d to be larger than $O(\frac{1}{\varepsilon})$ in order to output a $(1-\varepsilon)$-optimal cut. A natural question is whether, under a no-OGP assumption, the result can be extended to output $(1-\varepsilon)$-optimal cuts for d-regular graphs for *any* $d \geq 3$.

▶ **Conjecture 5.2** (AMP algorithm for Random d-Regular Graphs). *There exists a poly$(n, \frac{1}{\varepsilon})$ time algorithm A that outputs a (1 - ε)-approximate cut of a random d-regular graph G with high probability under a "no-OGP" assumption for* any $d \geq 3$.

Note that the approach of Alaoui et al. [2] critically relies on the Guerra-Tonnineli interpolation between the $\mathcal{G}_{n,d}$ model and the SK-model which will *only* work for $d \geq O\left(\frac{1}{\varepsilon}\right)$. Consequently, a solution that works for *all* $d \geq 3$ will require a fundamentally different approach. A natural question that is motivated by the above conjecture is to then investigate if there is any range of degree for which the MAX-CUT problem over d-regular graphs possesses an OGP. Given the belief that the SK model does not exhibit an OGP, this would only be an interesting question in the relatively low-degree regime.

▶ **Problem 5.3** (Random d-Regular Graphs don't have an OGP). *Does the* MAX-CUT *problem on random d-regular graphs have an OGP for some $d \geq 3$? If so, for what $\{d_0, d_1\} \subset \mathbb{N}$ does the problem exhibit an OGP?*

The study of $QAOA_p$ was initiated on MAX-CUT for d-regular graphs: positive answers to the conjectures above would resolve the question of quantum advantage on the problem.

References

1 Dimitris Achlioptas and Federico Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 130–139, 2006.

2 Ahmed El Alaoui, Andrea Montanari, and Mark Sellke. Local algorithms for maximum cut and minimum bisection on locally treelike regular graphs of large degree. *arXiv preprint*, 2021. arXiv:2111.06813.

3 Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, et al. Hartree-Fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, 2020. URL: https://www.science.org/doi/abs/10.1126/science.abb9811.

4 Antonio Auffinger, Wei-Kuo Chen, and Qiang Zeng. The sk model is infinite step replica symmetry breaking at zero temperature. *Communications on Pure and Applied Mathematics*, 73(5), 2020.

5 Boaz Barak and Kunal Marwaha. Classical algorithms and quantum limitations for maximum cut on high-girth graphs. *arXiv preprint*, 2021. arXiv:2106.05900.

6 John S Bell. On the einstein-podolsky-rosen paradox. *Physics Physique Fizika*, 1(3):195, 1964.

7 Sally C Brailsford, Chris N Potts, and Barbara M Smith. Constraint satisfaction problems: Algorithms and applications. *European journal of operational research*, 119(3):557–581, 1999.

8 Fernando Brandao, Michael Broughton, Edward Farhi, Sam Gutmann, and Hartmut Neven. For fixed control parameters the quantum approximate optimization algorithm's objective function value concentrates for typical instances. *arXiv preprint*, 2018. arXiv:1812.04170.

9 Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures & Algorithms*, 27(2):201–226, 2005.

10 Wei-Kuo Chen, David Gamarnik, Dmitry Panchenko, et al. Suboptimality of local algorithms for a class of max-cut problems. *Annals of Probability*, 47(3):1587–1618, 2019.

11 Chi-Ning Chou, Peter J Love, Juspreet Singh Sandhu, and Jonathan Shi. Limitations of local quantum algorithms on random max-k-xor and beyond. *arXiv preprint v3*, 2021. arXiv:2108.06049.

12 Francesco Concetti. The full replica symmetry breaking in the ising spin glass on random regular graph. *Journal of Statistical Physics*, 173(5):1459–1483, 2018.

13 Giacomo De Palma, Milad Marvian, Dario Trevisan, and Seth Lloyd. The quantum wasserstein distance of order 1. *IEEE Transactions on Information Theory*, 67(10):6627–6643, 2021.

14 Amir Dembo, Andrea Montanari, and Subhabrata Sen. Extremal cuts of sparse random graphs. *The Annals of Probability*, 45(2):1190–1217, 2017.

15 Jian Ding, Allan Sly, and Nike Sun. Satisfiability threshold for random regular nae-sat. *Communications in Mathematical Physics*, 341(2):435–489, 2016.

16 Sepehr Ebadi, Tout T Wang, Harry Levine, et al. Quantum phases of matter on a 256-atom programmable quantum simulator. *Nature*, 595(7866):227–232, 2021.

17 Edward Farhi, David Gamarnik, and Sam Gutmann. The quantum approximate optimization algorithm needs to see the whole graph: A typical case. *arXiv preprint*, 2020. arXiv:2004.09002.

18 Edward Farhi, David Gamarnik, and Sam Gutmann. The quantum approximate optimization algorithm needs to see the whole graph: Worst case examples. *arXiv preprint*, 2020. arXiv:2005.08747.

19 Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint*, 2014. arXiv:1411.4028.

20 Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Leo Zhou. The quantum approximate optimization algorithm and the sherrington-kirkpatrick model at infinite size. *arXiv preprint*, 2019. arXiv:1910.08187.

21 Silvio Franz and Michele Leone. Replica bounds for optimization problems and diluted spin systems. *Journal of Statistical Physics*, 111(3):535–564, 2003.

22 David Gamarnik and Aukosh Jagannath. The overlap gap property and approximate message passing algorithms for p-spin models. *The Annals of Probability*, 49(1):180–205, 2021.

23 David Gamarnik, Aukosh Jagannath, and Alexander S Wein. Low-degree hardness of random optimization problems. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 131–140. IEEE Computer Society, 2020.

24 David Gamarnik and Madhu Sudan. Limits of local algorithms over sparse random graphs. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 369–376, 2014.

25 David Gamarnik and Ilias Zadik. The landscape of the planted clique problem: Dense subgraphs and the overlap gap property. *arXiv preprint*, 2019. arXiv:1904.07174.

26 Ming Gong, Shiyu Wang, Chen Zha, Ming-Cheng Chen, He-Liang Huang, Yulin Wu, Qingling Zhu, Youwei Zhao, Shaowei Li, Shaojun Guo, et al. Quantum walks on a programmable two-dimensional 62-qubit superconducting processor. *Science*, 372(6545):948–952, 2021.

27 Francesco Guerra and Fabio Lucio Toninelli. The thermodynamic limit in mean field spin glass models. *Communications in Mathematical Physics*, 230(1):71–79, 2002.

28 Francesco Guerra and Fabio Lucio Toninelli. The high temperature region of the viana–bray diluted spin glass model. *Journal of statistical physics*, 115(1):531–555, 2004.

29 Matthew B Hastings. Classical and quantum bounded depth approximation algorithms. *arXiv preprint*, 2019. arXiv:1905.07047.

30 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007.

31 Subhash Khot and Nisheeth K Vishnoi. On the unique games conjecture. In *FOCS*, volume 5, page 3. Citeseer, 2005.

32 Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32–32, 1992.

33 Kunal Marwaha. Local classical max-cut algorithm outperforms $p = 2$ qaoa on high-girth regular graphs. *Quantum*, 5:437, 2021.

34 Marc Mézard and Giorgio Parisi. The bethe lattice spin glass revisited. *The European Physical Journal B-Condensed Matter and Complex Systems*, 20(2):217–233, 2001.

35 Dmitry Panchenko. Introduction to the sk model. *arXiv preprint*, 2014. arXiv:1412.0170.

36 Dmitry Panchenko. The parisi formula for mixed p-spin models. *The Annals of Probability*, 42(3):946–958, 2014.

37 Dmitry Panchenko and Michel Talagrand. Bounds for diluted mean-fields spin glass models. *Probability Theory and Related Fields*, 130(3):319–336, 2004.

38 Giorgio Parisi. A sequence of approximated solutions to the sk model for spin glasses. *Journal of Physics A: Mathematical and General*, 13(4):L115, 1980.

39 Giorgio Parisi, Federico Ricci-Tersenghi, and Tommaso Rizzo. Diluted mean-field spin-glass models at criticality. *Journal of Statistical Mechanics: Theory and Experiment*, 2014(4):P04013, 2014.

40 John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

41 Prasad Raghavendra. Optimal algorithms and inapproximability results for every csp? In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 245–254, 2008.

42 Subhabrata Sen. Optimization on sparse random hypergraphs and spin glasses. *Random Structures & Algorithms*, 53(3):504–536, 2018.

43 David Sherrington and Scott Kirkpatrick. Solvable model of a spin-glass. *Physical review letters*, 35(26):1792, 1975.

44 Michel Talagrand. The parisi formula. *Annals of mathematics*, pages 221–263, 2006.

45 Alexander S Wein. Optimal low-degree hardness of maximum independent set. *Mathematical Statistics and Learning*, 2022.

46 Jonathan Yedidia, William Freeman, Yair Weiss, et al. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003.

Fully-Dynamic $\alpha + 2$ Arboricity Decompositions and Implicit Colouring

Aleksander B. G. Christiansen
Technical University of Denmark, Lyngby, Denmark

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Abstract

The arboricity α of a graph is the smallest number of forests necessary to cover its edges, and an arboricity decomposition of a graph is a decomposition of its edges into forests. The best near-linear time algorithm for arboricity decomposition guarantees at most $\alpha + 2$ forests if the graph has arboricity α (Blumenstock and Fischer [12]).

In this paper, we study arboricity decomposition for *dynamic graphs*, that is, graphs that are subject to insertions and deletions of edges. We give an algorithm that, provided the arboricity of the dynamic graph never exceeds α, maintains an $\alpha + 2$ arboricity decomposition of the graph in $\text{poly}(\log n, \alpha)$ update time, thus matching the number of forests currently obtainable in near-linear time for static (non-changing) graphs.

Our construction goes via dynamic bounded out-degree orientations, and we present a fully-dynamic algorithm that explicitly orients the edges of the dynamic graph, such that no vertex has an out-degree exceeding $\lfloor(1+\varepsilon)\alpha\rfloor + 2$. Our algorithm is deterministic and has a worst-case update time of $O(\varepsilon^{-6}\alpha^2 \log^3 n)$. The state-of-the-art explicit, deterministic, worst-case algorithm for bounded out-degree orientations maintains a $\beta \cdot \alpha + \log_\beta n$ out-orientation in $O(\beta^2\alpha^2 + \beta\alpha \log_\beta n)$ time [30].

As a consequence, we get an algorithm that maintains an implicit vertex colouring with $4 \cdot 2^\alpha$ colours, in amortised poly-$\log n$ update time, and with $O(\alpha \log n)$ worst-case query time. Thus, at the expense of $\log n$-factors in the update time, we improve on the number of colours from $2^{O(\alpha)}$ to $O(2^\alpha)$ compared to the state-of-the-art for implicit dynamic colouring [27].

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graphs, bounded arboricity, graph colouring, data structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.42

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2203.06039` [16]

Funding *Aleksander B. G. Christiansen*: Partially supported by the VILLUM Foundation grant 37507 "Efficient Recomputations for Changeful Problems".
Eva Rotenberg: Partially supported by Independent Research Fund Denmark grants 2020-2023 (9131-00044B) "Dynamic Network Analysis" and 2018-2021 (8021-00249B) "AlgoGraph", and the VILLUM Foundation grant 37507 "Efficient Recomputations for Changeful Problems".

Acknowledgements We thank Krzysztof Nowicki for helpful discussions and ideas.

1 Introduction

Graph colouring is a well-studied problem in computer science and discrete mathematics and has many applications such as planar routing and network optimization [17]. A proper colouring of a graph $G = (V, E)$ on n vertices is an assignment of colours to each vertex in $V(G)$ such that no neighbours receive the same colour. We are interested in minimising the number of colours used. The minimum number of colours that can be used to properly colour G, is called the *chromatic number* of G. It is NP-hard to even approximate the chromatic

number to within a factor of $n^{1-\varepsilon}$ for all $\varepsilon > 0$ [43, 29], but colourings with respect to certain parameters can be efficiently computed. For instance, it is well known that if a graph is uniformly sparse in the sense that we can decompose it into k forests, then we can efficiently compute a colouring: the sparsity of the graph ensures that every subgraph has a vertex of degree at most $2k - 1$, allowing us to compute a $2k$ colouring of the graph in linear time by colouring the vertices in a clever order. The minimum number of forests that the graph can be decomposed into is called the *arboricity* of G. In the past decades, much work has gone into the study of *dynamic algorithms* that are able to efficiently update a solution, as the problem undergoes updates. A general question about dynamic problems is: which (near-) linear-time solvable problems have polylogarithmic updatable solutions?

We study the problem of maintaining a proper colouring of a dynamic graph with bounded arboricity. This class of graphs encompasses, for instance, dynamic planar graphs where $\alpha \leq 3$. Here, the graph undergoes changes in the form of insertions and deletions of edges and one needs to maintain a proper colouring of the vertices with fast update times. We distinguish between two scenarios: one where, as is the case for dynamic planar graphs, we have access to an upper bound on the arboricity α_{max} throughout all updates, and one where we do not. Note that due to insights presented in [39], we can often turn an algorithm for the first scenario into an algorithm for the second by scheduling updates to $O(\log n)$ (partial) copies of the graph, thus incurring only an $O(\log n)$ overhead in the update time.

Barba et al. [6] showed that one cannot hope to maintain a proper, explicit vertex-colouring of a dynamic forest with a constant number of colours in poly-logarithmic update time. Consequently, we cannot maintain explicit colourings where the number of colours depend entirely on α with poly-logarithmic update time - even if we know an upper bound on α. This motivated Henzinger et al. [27] to initiate the study of *implicit* colourings. Here, instead of storing the colours of vertices explicitly in memory, a queryable data structure is provided which after some computations returns the colour of a vertex. If one queries the colours of two neighbouring vertices between updates, the returned colours must differ. Now, we can circumvent the lower bound by using known data structures for maintaining information in dynamic forest to 2-colour dynamic forests in poly-logarithmic update time. Henzinger et al. [27] use this to colour graphs via an *arboricity decomposition* i.e. a decomposition of the graph into forests. They present a dynamic algorithm that maintains an implicit proper $2^{O(\alpha)}$-colouring of a dynamic graph G with arboricity α. Their algorithm adapts to α, but in return it hides a constant (around 40) in the asymptotic notation. Even if one has an upper bound α_{max} on α, the currently best obtainable colouring uses $2^{4(\alpha_{max}+1)}$ colours by combining the arboricity decomposition algorithm from Henzinger et al. [27] with an algorithm of Brodal & Fagerberg [13] that maintains a $2(\alpha_{max} + 1)$ bounded out-degree orientation. Both of these algorithms use a lot of colours. Even for planar graphs with arboricity at most 3, $2^{16} > 60.000$ colours are used. This is quite far from 4 colours, which is always sufficient [3, 38], or the 5 colouring that can be computed in linear time [35, 14, 20].

Dynamic arboricity decompositions. Both colouring algorithms go via dynamic α'-bounded out-orientations. Here, the goal is to orient the edges of the graph while keeping out-degrees low. These are then turned into dynamic $2\alpha'$-arboricity decompositions. By 2-colouring each forest, such a decomposition yields a $2^{2\alpha'}$ colouring. Thus the lower α' is, the fewer colours we use. There has been a lot of work on maintaining dynamic low out-orientations [13, 8, 30, 26, 41], and much of this work aim to improve update complexity by relaxing the allowed out-degree. Motivated by implicit colourings, we provide a different trade-off, providing a lower α' value within polylog(n, α) update time. Specifically, a $\lfloor(1 + \varepsilon)\alpha\rfloor + 2$

dynamic out-orientation with $O(\log^3(n)\alpha^2/\varepsilon^6)$ update-time adaptive to α, and an $\alpha+2$ dynamic arboricity decomposition with $O(\text{poly}(\log n, \alpha_{max}))$ update time, when we have an upper bound α_{max} on the arboricity. Our algorithm maintaining the arboricity decomposition matches the number of forests obtained by the best static algorithm running in near-linear time [12].

These algorithms may also be interesting as they go below the 2α barrier on out-edges and forests respectively. In the static case there exist simple and elegant algorithms computing $2\alpha-1$ out-orientations and arboricity decompositions in linear time [4, 19]. For exact algorithms, the state-of-the-art algorithms spend time $\tilde{O}(m^{10/7})$ [34] or $\tilde{O}(m\sqrt{n})$ [33] for the out-orientation problem, and $\tilde{O}(m^{3/2})$ for the arboricity decomposition problem [21, 22]. Even statically computing an $\alpha+1$ out-orientation [31] resp. an $\alpha+2$ arboricity decomposition [12] takes $\tilde{O}(m)$ time. In the dynamic case, the out-orientation with the lowest bound on the out-degree with $O(\text{poly}(\log n, \alpha))$ update time seem to be the algorithm of Brodal & Fagerberg [13] that achieves $2(\alpha_{max}+1)$ out-degree. In [13] it is also noted that determining exactly the complexity of maintaining a d out-orientation for $d \in [\alpha, 2\alpha]$ is a 'theoretically interesting direction for further research'. We make some progress in this direction by showing how to maintain a $\lfloor(1+\varepsilon)\alpha\rfloor+2$ out-orientation with $\text{poly}(\log n, \alpha, \varepsilon^{-1})$ update time. Thus, if α is a constant, we may carefully choose ε to obtain a polylogarithmic $\alpha+2$ out-orientation.

1.1 Results

Let G be a dynamic graph with n vertices undergoing insertion and deletions of edges, and let α be the current arboricity of the graph; that is α might change, when edges are inserted and deleted. If we at all times have an upper bound α_{max} on α, we say that G is undergoing an α_{max} preserving sequence of updates. We have the following:

▶ **Theorem 1.** *For $1 > \varepsilon > 0$, there exists a fully-dynamic algorithm maintaining an explicit $((1+\varepsilon)\alpha+2)$-bounded out-degree orientation with worst-case insertion time $O(\log^3 n \cdot \alpha^2/\varepsilon^6)$ and worst-case deletion time $O(\log^3 n \cdot \alpha/\varepsilon^4)$*

Using pseudoforest decompositions, we obtain a fully dynamic, implicit colouring algorithm:

▶ **Corollary 2.** *Given a dynamic graph with n vertices, there exists a fully dynamic algorithm that maintains an implicit $2 \cdot 3^{(1+\varepsilon)\alpha}$ colouring with an amortized update time of $O(\log^4 n \cdot \alpha^2/\varepsilon^6)$ and a query time of $O(\alpha \log n)$.*

By moving edges between pseudoforests, we can turn the pseudoforest decomposition into a forest decomposition. This also gives a colouring algorithm using fewer colours.

▶ **Theorem 3.** *Given an initially empty and dynamic graph undergoing an arboricity α_{max} preserving sequence of updates, there exists an algorithm maintaining a $\lfloor(1+\varepsilon)\alpha\rfloor+2$ arboricity decomposition with amortized update time $O(\text{poly}(\log n, \alpha_{max}, \varepsilon^{-1}))$. In particular, setting $\varepsilon < \alpha_{max}^{-1}$ yields $\alpha+2$ forests with an amortized update time of $O(\text{poly}(\log n, \alpha_{max}))$.*

▶ **Corollary 4.** *Given a dynamic graph with n vertices, there exists a fully dynamic algorithm that maintains an implicit $4 \cdot 2^{\alpha}$ colouring with an amortized update of $O(\text{polylog}\, n)$ and a query time of $O(\alpha \log n)$.*

Finally, we modify an algorithm of Brodal & Fagerberg [13] so that it maintains an acyclic out-orientation.

▶ **Theorem 5.** *Given an initially empty and dynamic graph G undergoing an arboricity α_{max} preserving sequence of insertions and deletions, there exists an algorithm maintaining an acyclic $(2\alpha_{max}+1)$ out-degree orientation with an amortized insertion cost of $O(\alpha_{max}^2)$, and an amortized deletion cost of $O(\alpha_{max}^2 \log n)$.*

Table 1 Different dynamic algorithms for maintaining out-orientations. We state the bounds in terms of the arboricity α of the graphs, since many of the results referenced do the same.

Reference	Out-degree	Update time	α
Brodal & Fagerberg [13]	$2(\alpha+1)$	$O(\alpha + \log n)$ am.	fixed
Kopelowitz et al. [30]	$\beta\alpha + \log_\beta n$	$O(\beta^2\alpha^2 + \beta\alpha \log n)$	adaptive
He et al. [26]	$O(\alpha\sqrt{\log n})$	$O(\sqrt{\log n})$ am.	fixed
Berglin & Brodal [8]	$O(\alpha + \log n)$	$O(\log n)$	adaptive
Henzinger et al. [27]	40α	$O(\log^2 n)$ am.	adaptive
Kowalik [32]	$O(\alpha \log n)$	$O(1)$ am.	fixed
New (Thm. 1)	$(1+\varepsilon)\alpha + 2$	$O(\log^3(n)\alpha^2/\varepsilon^6))$	adaptive

1.2 Related Work

Dynamic colouring. Barba et al. [6] give algorithms for the dynamic recolouring problem, and show that c-colouring a dynamic forests incurs $\Omega(n^{\frac{1}{c(c-1)}})$ recolourings per update. Solomon & Wein give improved trade-offs between update time and recolourings and give a deterministic dynamic colouring algorithm parametrized by the arboricity α, using $O(\alpha^2 \log n)$ colours with $O(1)$ amortized update time [41]. Henzinger et al. [27] introduced the study of implicit colouring of sparse graphs in order to circumvent the explicit lower bound of Barba et al. [6]; they maintain an implicit colouring using $2^{O(\alpha)}$ colours, with $O(\log^3 n)$ update time and $O(\alpha \log n)$ query-time. Recently, there has also been a lot of work on the dynamic colouring problem parameterised by the maximum degree [9, 10, 28].

Bounded out-degree orientations. Much of the work with respect to bounded out-degree orientations has gone into either 1) statically computing bounded out-degree orientations with the minimum (or close to it) out-degree [21, 37, 11, 1], or 2) dynamically maintaining bounded out-degree orientations with efficient updates [13, 30, 41, 32, 8], but allowing weaker guarantees on the minimum out-degree (see Table 1). Note that the constant of 40 was extracted from an equation in the proof of Lemma 18 (on page 15) in [27]. The current state-of-the-art for exact, static algorithms have running time $\tilde{O}(m^{10/7})$ [34] and $\tilde{O}(m\sqrt{n})$ [33]. Kowalik [31] also gave an algorithm computing a $\lceil(1+\varepsilon)\alpha\rceil$ out-orientation in $\tilde{O}(m \cdot \varepsilon^{-1})$ time.

Arboricity decompositions. A lot of work has been put into producing efficient static algorithms for computing arboricity decompositions [21, 22, 18, 37] (see [12] for an overview). The fastest static algorithm runs in $\tilde{O}(m^{3/2})$ time [21, 22]. Also approximation algorithms have been studied in the static case. There exists a linear-time 2-approximation algorithm [4, 19]. Furthermore, Blumenstock & Fischer provide an algorithm computing a $\lceil(1+\varepsilon)\alpha\rceil+1$ arboricity decomposition in $\tilde{O}(m \cdot \varepsilon^{-1})$ time. Bannerjee et al. [5] provide an $\tilde{O}(m)$ dynamic algorithm maintaining the exact arboricity α of a dynamic graph, and show a lower bound of $\Omega(\log n)$ for dynamically maintaining arboricity. Henzinger et al. [27] provide a dynamic algorithm for maintaining a $2\alpha'$ arboricity decomposition, given access to any black box dynamic α' out-degree orientation algorithm. (See Table 2.)

Other related work. Motivated by the problem of finding a densest subgraph, Sawlani & Wang [39] gave an (implicit) dynamic approximation algorithm for maintaining a $(1+\varepsilon)\rho$ fractional out-degree orientation, where ρ is the maximum subgraph density. In order to tune the parameters in the algorithm, they use multiple (partial) copies of the same graph, where each copy has a different estimate of the maximum density of the graph.

Table 2 Overview of dynamic algorithms for maintaining arboricity decompositions. Note that applying Lemma 27 to Theorem 5 gives an arboricity decomposition, since the orientation is acyclic.

Reference	Forests	Update time	Uses Lemma from [27]	α
Bannerjee et al. [5]	α	$\tilde{O}(m)$	No	adaptive
Brodal & Fagerberg [13]	$4(\alpha+1)$	$O(\alpha+\log n)$ am.	Yes	fixed
Henzinger et al. [27]	80α	$O(\log^2 n)$ am.	Yes	adaptive
New (Thm. 5)	$2(\alpha+1)$	$O(\alpha^2 \log n)$ am.	Uses Lemma 27	fixed
New (Thm. 3)	$\alpha+2$	$O(\text{polylog}\, n)$ am.	No	adaptive

Computing near optimal out-orientations and arboricity decompositions has also been studied from a distributed point of view. Barenboim & Elkin gave a $(2+\varepsilon)$-approximation in [7]. This has since then been improved to $(1+\varepsilon)$-approximations [24, 25, 42, 23].

1.3 Summary of techniques

It is quite simple to compute 2-approximations of arboricity decompositions and bounded out-degree orientations in the static case: It follows from a Theorem of Nash-Williams [36] that a graph with arboricity α is $2\alpha-1$ *degenerate* i.e. every subgraph of the graph has a vertex of degree at most $2\alpha-1$. By continuously removing a vertex with minimum degree and assigning all remaining edges incident to it as out-edges, one obtains an acyclic $(2\alpha-1)$ out-orientation. By partitioning the edges into $2\alpha-1$ partitions s.t. no vertex has two out-edges in the same partition, one obtains a $(2\alpha-1)$-arboricity decomposition.

However, in order to get static algorithms computing close to optimal out-orientations and arboricity decompositions, one typically formulates the problems as combinatorial optimization problems. These can be difficult to approximate – even in the static case, and thus perhaps even more so dynamically. To achieve the low out-orientation algorithm, we build upon a technique of Sawlani & Wang [39]: if one allows edges to be partially assigned to both endpoints, one gets a relaxed version of the out-orientation problem which we will refer to as the *fractional out-orientation problem*. Here edges are assigned partially to both endpoints and one seeks to minimise the maximum *load* of a vertex, where a vertex's load is the sum of the loads contributed by each edge incident to the vertex.

A novelty in our approach lies in what we call *refinements* of fractional orientations: the edges that assign a substantial load to both endpoints form a subgraph – a refinement – of the original graph. We show how to remove cycles from the refinement without changing the loads of any vertices by reassigning edges along cycles. When the refinement is acyclic, the remaining edges have (almost) decided on which endpoint, they prefer. This ensures that we can naively 'round' all of the edges not in the refinement to become out-edges of the vertices they assign the most load. By 2-orienting the refinement, we obtain an algorithm for maintaining an out-orientation with close to the optimal number of out-edges.

Given an α' out-orientation, it is straighforward to split it into α' pseudoforests i.e. graphs where each component has at most one cycle: partition the edges into α' partitions such that no vertex has out-degree more than one in each partition. Every pseudoforest can be represented as a forest and a matching – simply put exactly one edge from each cycle into the matching. Blumenstock & Fischer [12] show that if one chooses the pseudoforests and the representation of the pseudoforsts in a clever way, then the union of all of the matchings form a forest. Thus by combining these techniques one can go from an α' orientation to an $\alpha'+1$ arboricity decomposition.

It is the same idea that underlies our dynamic arboricity decomposition algorithm: we maintain a refinement and a low out-orientation of the remaining edges as before. Then we partition the oriented edges into pseudoforests and finally we alter these pseudoforests and their representations to arrive at a low arboricity decomposition. The key challenge in making this process dynamic is that the altered pseudoforests must be obtainable by partitioning the current out-orientation; otherwise it is unclear how to maintain the pseudoforests as the graph is updated. In order to achieve this, we modify the approach of Blumenstock & Fischer such that the obtained pseudoforests stay faithful to the underlying orientation. However to do so, we have to alter the underlying orientation to accommodate our choice of pseudoforests. To be able to alter the orientation maintained by the out-orientation algorithm, we have to be careful to keep auxiliary datastructures updated and ensure that certain invariants are maintained. To achieve this, we show how to update the datastructures lazily, and we use a potential based argument to show that we can afford to maintain the required invariants.

Paper Outline. In Section 2, we first show how to 2-orient forests and then we recall the techniques of Sawlani & Wang [39] and Kopelowitz et al. [30]. In order to maintain the refinement, we need to represent some parts of the fractional out-orientation implicitly, hence in Section 2 we also make precise exactly how the fractional out-orientation can be accessed. In Section 3 we introduce refinements and show how to maintain them dynamically thus obtaining Theorem 1 and Corollary 2. In Section 4 we show Theorem 3 and Corollary 4. We begin by briefly discussing our approach, before we in Section 4.1 recall the techniques of Henzinger et al. [27] and Blumenstock & Fischer [12]. In Sections 4.2 to 4.7 we describe our new dynamic algorithm for maintaining an arboricity decomposition with close to the optimal number of forests. Finally, Section 5 is dedicated to Theorem 5. Due to space-constraints some proofs are left out (they can be found in the full-version [16]).

2 Preliminaries & Warm-up

Nash-Williams [36] showed that the arboricity α of a graph G satisfies $\alpha = \lceil \max_{J \subset G} \frac{|E(J)|}{|V(J)|-1} \rceil$. The *pseudoarboricity* α_p is the minimum number of pseudoforests that the edges of G can be partitioned into. The *maximum (subgraph) density* or the *fractional pseudoarboricity* ρ is defined as $\rho = \max_{J \subset G} \frac{|E(J)|}{|V(J)|}$. We have $\alpha_p = \lceil \rho \rceil$ [37]. Note that α and α_p are numerically very close, and that $\alpha_p(G)$ is the lowest maximum out-degree achievable when orienting the edges of an undirected graph G [31]. For an undirected graph G and a vertex $v \in V(G)$, we let $d(v)$ be the degree of v and $N(v)$ the neighbourhood of v. If G is directed, $d^+(v)$ denotes the out-degree of v and $N^+(v)$ the out-neighbourhood of v.

Explicit 2-out orientation of dynamic forests. We begin by considering the simpler problem of orienting the edges of dynamic forests so as to minimise the maximum out-degree of vertices. If we want an implicit out-orientation of a dynamic forest H, we can root each tree in H arbitrarily, and get an out-degree of 1 upon query-time using data structures for maintaining information in a dynamic forest (see for example [2, 40]). However, if we want the out-orientation to be explicit, there is a naive lower bound for maintaining a 1-out orientation: Take a path of length n. Deleting the edge between vertex $n/2$ and $n/2 + 1$, yields two sub-paths of length roughly $n/2$. No matter the 1-orientation of these paths, we can reconnect them so as to necessitate $\Omega(n)$ reorientations to restore a 1-orientation. This may be repeated to defy even the hope of an improved amortised analysis.

On the contrary, to obtain a 2-orientation, one can use dynamic heavy path decompositions to obtain an $O(\log(n))$ update time algorithm via orienting light edges towards the root, and heavy edges arbitrarily. For a rooted-tree T, we have the notion of parents and children of the vertices. The *parent* of v is the first vertex from v on the v-to-root path in T. The *children* of v are all neighbours of v that are not the parent of v. A *heavy child* w of v is then a child of v such that the sub-tree of T rooted at w contains more than half of the vertices of the sub-tree rooted at v. The heavy children in T induces *heavy edges* going from a vertex to its heavy child, and *light edges* going from a vertex to its non-heavy children. Every root-to-leaf path then contains at most $O(\log n)$ light edges. The heavy edges form the desired paths, and the light edges can be assigned to the endpoint that is a child of the other endpoint. Sleator and Tarjan [40] showed how to maintain such a heavy-light decomposition in $O(\log n)$ worst case update time, and with $O(1)$ overhead one can keep all light edges oriented towards the root. As such, we have:

▶ **Lemma 6.** *There exists a fully-dynamic algorithm maintaining an explicit 2-out orientation of an n-vertex dynamic forest with $O(\log n)$ worst-case update time.*

Fractional Out-degree Orientations. We will obtain a low-bounded out-degree orientation by deterministically rounding a *fractional out-degree orientation*. Here, the orientation problem is relaxed so that the edges are allowed to be assigned partially to each end-point, and the goal is to compute an orientation such that the maximum total load assigned to a vertex is minimized. A formal definition is as follows:

▶ **Definition 7.** *A* fractional α'-bounded out-degree orientation O *of a graph G is a pair of variables $X_e^u, X_e^v \in [0,1]$ for each edge $e = uv \in E(G)$ s.t. the following holds:*

1. $\forall e = uv \in E(G)$: $X_e^u + X_e^v = 1$
2. $\forall v \in V(G)$: $s(v) = \sum_{e:v\in e} X_e^v \leq \alpha'$

If furthermore $X_e^u, X_e^v \in \gamma^{-1} \cdot \mathbb{Z}$ for all $e \in E(G)$, we say that O is a (γ, α')-orientation.

In particular, an α'-bounded out-degree orientation is just a $(1, \alpha')$-orientation. We think of $s(v)$ as the load on vertex v, and α' as an upper bound on the allowed vertex load. The γ-parameter underlines the fact that we wish to discretise the fractional loads on edges to rational loads. If one does so in a symmetric manner for each edge, one can view a (γ, α')-orientation of a graph G as a $(1, \gamma\alpha')$-orientation of G^γ, where we define G^γ to be G, where every edge is replaced by γ copies. For an edge $e = uv \in E(G)$, we denote by B_e the bundle of γ edges representing e in G^γ. If G^γ is oriented, we denote by B_e^u the bundle of edges oriented $u \to v$. Since the copies of e in B_e are identical, we only care about the size of B_e^u, and not which copies of e it contains. Hence:

▶ **Observation 8.** *For a graph G, there is a natural bijection (up to symmetry) between $(1, \gamma \cdot \alpha')$ orientations of G^γ and (γ, α')-orientations of G.*

In light of this observation, we shall use these two descriptions interchangeably, and in some cases we shall refer to the same orientation as being both a $(1, \gamma\alpha')$-orientation of G^γ and a (γ, α')-orientation of G. We follow the approaches of Sawlani & Wang [39] and Kopelowitz et al. [30], so we repeat the following:

▶ **Definition 9** ([39])**.** *Given a $(1, \alpha')$-orientation of a graph G^γ, we say that an edge $u \to v \in E(G)$ is* η-valid *if $s(u) - s(v) \leq \eta$ and* η-invalid *otherwise. If also $s(v) - s(u) \leq \eta$, we say that $e = uv \in G$ is* doubly η-valid*. Furthermore, if $s(v) - s(u) \leq -\eta/2$ we say that e is an η-tight out-edge of u and an η-tight in-edge of v.*

Note that if $u \to v$ is η-invalid, then $s(u) - s(v) > \eta$ and so $-\eta > s(v) - s(u)$, so uv is η-tight. Note that typically $\eta = 1$.

▶ **Definition 10** ([39] Def. 3.5). *A* maximal η-tight chain from v *is a path of η-tight edges $v_0v_1, \dots v_{k-1}v_k$, such that $v_0 = v$ and v_k has no η-tight out-edges.*

A maximal η-tight chain to v *is a path of η-tight edges $v_0v_1, \dots v_{k-1}v_k$, such that $v_k = v$ and v_0 has no η-tight in-edges.*

▶ **Lemma 11** (Implicit in [39]). *Inserting an η-valid edge oriented $u \to v$ and reorienting a maximal η-tight chain from u will η-invalidate no η-valid edges.*

Deleting an edge oriented $u \to v$ and reorienting a maximal η-tight chain to u will η-invalidate no η-valid edges.

▶ Remark 12. Note that a maximal η-tight chain has length at most $\frac{2 \cdot \max_v s(v)}{\eta}$. Indeed, each time we follow an η-tight out-edge the load on the vertex increases by at least $\eta/2$.

If every edge is η-valid, Sawlani & Wang say that the orientation is *locally η-stable*. Kopelowitz et al. show the following guarantees for locally 1-stable orientations, where we, for ease of notation, define $\Delta^+ := (1+\varepsilon)\alpha\gamma + \log_{(1+\varepsilon)}(n)$:

▶ **Lemma 13** (Implicit in [30]). *If every edge in G^γ is 1-valid, then $\max_v s(v) \leq \Delta^+$.*

Implicit orientations. We are interested in maintaining a fractional out-degree orientation in which the fractional orientation of edges allow us to 'round' the fractional orientation to a low out-degree orientation. We are interested in two properties: first of all the maximum load of a vertex should be low, and second of all many of the edges should have either X_e^u or X_e^v close to 1, so that a naive rounding strategy does not increase the load of a vertex by much. By Lemma 13, if we ensure that the orientation is locally 1-stable, then we get an upper bound on the maximum vertex load. In order to ensure the second property, we redistribute load along cycles without breaking local stability. Our algorithm has two phases. A phase for inserting/deleting edges in a manner that η-invalidates no edges, thus ensuring the first property, and a second phase for redistributing load along edges in order to ensure that the orientation also has the second property. In order for these two phases to work (somewhat) independently, we think of each phase as having implicit access to the orientation; that is the insertion/deletion algorithm might have to pay a query cost in order to identify the precise fractional load of an edge or neighbourhood of a vertex.

▶ **Definition 14.** *An algorithm on an n vertex dynamic and oriented graph has implicit $(|L|, q(n))$ access to an orientation, if it has access to:*

1. *Operations for querying and changing fractional loads of edges in $O(\log n)$ time.*
2. *A query that returns a list containing a superset of all neighbours of a vertex that have changed status as in- or out-neighbour, since the last time the query was called on this vertex. The list should have length $\leq |L|$ and the query should run in $O(q(n))$ time.*

Implicitly Accessing Orientations. In this section, we outline how to modify the algorithm of Kopelowitz et al. [30] to run on G^γ and to support implicit access to the orientation. The ideas presented here are not new; they arise in [30] and [39], but we present them for completeness.

We think of the algorithm as being run on G^γ for some γ to be specified later. We think of each edge $e \in G$ as γ copies in G^γ, but in practice we only store e along with counters $|B_e^u|$, $|B_e^v|$ denoting the number of copies oriented in each direction. Now, we wish to run the

algorithm from [30] in order to insert/delete each copy of an edge one-by-one. This algorithm inserts/deletes a copy of an edge in G^γ using Lemma 11 with $\eta = 1$. We identify a tight chain from u by continuously looking at all out-neighbours and following tight out-edges, until the chain becomes maximal. We use max-heaps, stored at each vertex, to identify maximally tight chains to u. Since we only have implicit access to the orientation, we have to first process the list of possible changes to in- and out-neighbours before trying to identify the next tight edge. Furthermore, when we reorient said chains, we have to access the fractional load of each edge on the chain, before changing it. Hence, we have:

▶ **Theorem 15** (implicit in [30]). *Given implicit $(|L|, q(n))$ access to an orientation with $\max_v s(v) \leq \Delta^+$, there exists an algorithm that can insert and delete edges from the orientation without creating any new 1-invalid edges. The algorithm has worst-case insertion time of $O(\gamma \cdot \Delta^+(\Delta^+ + \log(n)(|L|+1) + q(n)))$ and a worst case deletion time of $O(\gamma \cdot \Delta^+(\log(n)(|L|+1) + q(n)))$.*

▶ Remark 16. Each insertion/deletion of a copy of an edge in G^γ with $\max_v s(v) \leq \Delta^+$ changes the load of at most $O(\Delta^+)$ edges. Indeed, we only change the load of edges on tight chains (and potentially one new edge), so the statement follows from Remark 12.

Scheduling Updates. Some of the algorithms rely on upper bounds on the arboricity. This is, however, not as limiting a factor as one might think, if we are willing to settle for implicit algorithms. In this section we describe how to use the algorithm of Sawlani & Wang [39] to schedule updates to $O(\log n)$ different copies of a graph such that each copy satisfies different density constraints. Here, we describe the main ideas behind the algorithm, and in the appendix of the full version [16], we paraphrase the ideas in more details.

Sawlani & Wang [39] maintain a fractional out-orientation of a graph G by using an algorithm similar to Theorem 15 to insert and delete edges in G^γ. By allowing η to scale with the maximum density ρ of G, they are able to make the update time independent of the actual value of ρ, provided that they have accurate estimates of ρ. By using $O(\log n)$ copies of G – each with different estimates ρ_{est} of ρ, they are able to at all times keep the copy where $\rho_{est} \leq \max_v s(v) < 2\rho_{est}$ fully updated. They call this copy the *active* copy. Similar to Remark 12 they observe that one can safely insert an edge uv in a copy where at least one of $s(u)$ or $s(v)$ is below $2\rho_{est}$. If, however, this is not the case, one cannot afford to update the copy. Sawlani & Wang resolve this issue by scheduling the updates so that they are only performed, when we can afford to do them. We can use this algorithm as a scheduler for our algorithms: We also run our algorithm on $O(\log n)$ copies of G. Whenever the algorithm from Theorem 1.1 in [39] has fully inserted or deleted an edge in a copy, we insert or delete the edge in our corresponding copy. Whenever our algorithm is queried, we then use the structure from the currently active copy to answer the query. Hence, we have:

▶ **Theorem 17** (Implicit in [39] as Theorem 1.1). *There exists a fully dynamic algorithm for scheduling updates that at all times maintains a pointer to a fully-updated copy with estimate ρ_{est} where $(1-\varepsilon)\rho_{est}/2 \leq \alpha(G) < 4\rho_{est}$. Furthermore, the updates are scheduled such that a copy G' with estimate ρ' satisfies $\alpha(G') \leq 4\rho'$. The algorithm has amortised $O(\log^4(n)/\varepsilon^6)$ update times.*

3 Dynamic Low Out-Orientations

In this section, we work towards the second goal: an orientation where many edges assign most of their load to one endpoint. To realise this, we introduce refinements of fractional orientations and show how to dynamically maintain them. The basic idea is to maintain

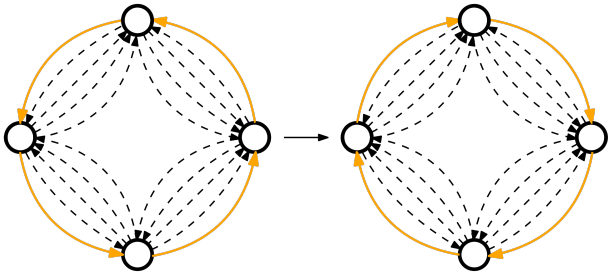


Figure 1 Inverting a cycle. We can reorient copies of edges forming a directed cycle without changing the load of any vertex. By reorienting edges in the refinement, at least one edge on the cycle becomes almost completely oriented towards one endpoint, and hence it can be expelled.

an orientation such that the fractional load of each vertex is small and such that the edges, that distribute their loads somewhat equally between both endpoints, form a forest. This property is nice, if we want to transform our orientation to a bounded out-degree orientation, since all edges outside of the forest almost already have decided on an orientation, and we can 2-orient trees using techniques from Section 2. Definition 18 formalises this idea:

▶ **Definition 18.** *Let O be a (γ, α')-orientation of a graph G. Then H is a (δ, μ)-refinement of G wrt. O if:*

1. $V(H) = V(G)$
2. *For all $e = uv \in E(G) : X_e^u, X_e^v \in (\delta, 1-\delta)$ implies that $e \in H$.*
3. *If $e \in H$, then $X_e^u, X_e^v \in [\delta - \mu, 1 - \delta + \mu]$*

The basic idea behind our algorithm is to maintain a refinement that is a forest. Whenever a cycle C occurs in the refinement, we can redistribute the fractional loads along the cycle so as to not change $s(v)$ for any $v \in C$, but such that an edge of C does not satisfy condition 2. in Definition 18 (see Figure 1). Thus we can remove this edge and again obtain an acyclic refinement with respect to this new orientation. Hence, we have the following observation:

▶ **Observation 19.** *Suppose $1 > \delta > \gamma^{-1} + \mu \geq 2\gamma^{-1} > 0$. Let H be (δ, μ)-refinement of a graph G wrt. some (γ, α')-orientation O of G. Then there exists a (δ, μ)-refinement of G, say H', wrt. some (γ, α')-orientation O' of G, such that H' is a forest.*

Note that if every edge of a graph G is η-valid, then an edge $e = uv \in G$ can only distribute its load somewhat evenly between u and v, if $s(u)$ and $s(v)$ are approximately the same. This implies that e is actually doubly η-valid:

▶ **Observation 20.** *If every edge in G^γ is η-valid, then every edge of a (δ, μ)-refinement H of G with $1 > \delta > \gamma^{-1} + \mu \geq 2\gamma^{-1}$ is doubly η-valid.*

Since the redistribution of fractional load of edges along a cycle does not change the load $s(v)$ of any vertex v, performing the redistribution from Observation 19 η-invalidates no edges.

3.1 The algorithm

As outlined earlier, our algorithm has two phases. In the first phase, we will insert and delete edges without η-invalidating any edges. We do this using the algorithm from Section 2. In the second phase, we examine all of the edges, whose fractional load was altered in phase 1. These edges might need to enter or exit H, depending on their new load. If such an insertion in H creates a unique cycle, we remove it as described in Observation 19. More precisely, the algorithm works as follows:

1. Insert (delete) γ copies of e into G^γ one at a time using a phase I algorithm from Section 2. Whenever a copy of an edge $f \in E(G)$ is reoriented in phase I, we push f onto a stack Q. If e is deleted in G, we also remove e from H.
2. When all γ copies of e are inserted, we set $g = uv = \texttt{pop}(Q)$ and update H as follows until Q is empty:
   - If $g \in H$ and $X_g^u \in (\delta, 1-\delta)$, we update the weight of g in H to match that of G^γ.
   - If $g \in H$ and $X_g^u \notin (\delta, 1-\delta)$, we remove g from H.
   - If $g \notin H$ and $X_g^u \in (\delta, 1-\delta)$, we push g onto a new stack S.
   - If $g \notin H$ and $X_g^u \notin (\delta, 1-\delta)$, we do nothing.
3. After processing all of Q, H together with the edges in S form a (δ, μ)-refinement of G. We now process each edge $h = uv \in S$ as follows:
   - If u, v are not in the same tree in H, we insert h into H.
   - Otherwise, u, v are in a unique cycle C in H. We update the weights along C, locate an edge wz along C with $X_{wz}^w, X_{wz}^z \notin (\delta, 1-\delta)$ and remove it from H. If $uv \neq wz$, we insert uv into H.
   - Finally, we update B_{wz}^w, B_{wz}^z in $G - H$ to match the weights wz had in H.

Since only edges from Q can enter S, we have the following Observation:

▶ **Observation 21.** *Let $S_{\max}$ and $Q_{\max}$ denote the maximum size of the stacks above during an insertion or a deletion. Then we have $S_{\max} \leq Q_{\max} \leq T$, where T is the total number of edges whose fractional orientation are altered during an insertion or a deletion.*

Furthermore, Observations 19 and 20 and Theorem 15 imply the invariants:

▶ **Invariant 22.** *Under the orientation induced by H for edges in $E(H)$ and by $G - H$ for edges in $E(G - H)$, every edge in $E(G)$ is η-valid.*

▶ **Invariant 23.** *H is both a (δ, μ)-refinement and a forest.*

3.2 Implementing updates

Since we maintain the invariant that H is a forest, we can use data structures for maintaining information in fully dynamic forests to store and update H:

▶ **Lemma 24** (Implicit in [2]). *Let F be a dynamic forest in which every edge $e = wz$ is assigned a pair of variables $X_e^w, X_e^z \in [0, 1]$ s.t. $X_e^w + X_e^z = 1$. Then there exists a data structure supporting the following operations, all in $O(\log |F|)$-time:*

- *link$(u, v, X_{uv}^u, X_{uv}^v)$: Add the edge uv to F and set $X_{uv}^u, X_{uv}^v = 1 - X_{uv}^u$ as indicated.*
- *cut(u, v): Remove the edge uv from F.*
- *connected(u, v): Return* true *if u, v are in the same tree, and* false *otherwise.*
- *add_weight(u, v, x): For all edges wz on the path $u \ldots wz \ldots v$ between u and v in F, set $X_{wz}^w = X_{wz}^w + x$ and $X_{wz}^z = X_{wz}^z - x$.*
- *min_weight(u, v): Return the minimum X_{wz}^w s.t. wz is on the path $u \ldots wz \ldots v$ in F.*
- *max_weight(u, v)): Return the maximum X_{wz}^w s.t. wz is on the path $u \ldots wz \ldots v$ in F.*

Note that using non-local search as described in [2], one can also locate the edges of minimum/maximum weight in $O(\log |F|)$-time. The lemma also shows that we can process an edge in Q in $O(\log n)$-time.

▶ **Observation 25.** *We can access and change the fractional load of $e \in H$ in time $O(\log n)$. We can do the same for $e \in G - H$ in $O(1)$ time, since these loads are not stored in top trees.*

To process an edge $uv \in S$ creating a cycle C in the (δ, μ) refinement H, we do as follows. Depending on the argument minimizing $l(C) = \mu + \min\{min\_weight(u,v) - \delta, 1 - \delta - max\_weight(u,v), X^u_{uv} - \delta, 1 - \delta - X^v_{uv}\}$, we either add or subtract $l(C)$ to every edge in C. We determine and remove the edge that minimized $l(C)$ from H. Thus we can process an edge in S in $O(\log n)$-time. Finally, if $\mu > \gamma^{-1}$ then every edge in $S \cup H$ has at least one copy in G^γ pointing in each direction both before and after the inversion of a cycle. Hence, no vertex receives any new in- nor out-neighbours. Since inverting a cycle does not change the load of any vertex, we need not update any priority queues for the insertion/deletion algorithm. Hence, we do not have to return any lists and so $|L| = q(n) = 0$.

3.3 Conclusions

▶ **Theorem 26.** *Suppose $1 > \delta > \gamma^{-1} + \mu > 2\gamma^{-1} > 0$, $\varepsilon > 0$. Then, there exists a dynamic algorithm that maintains a $(\gamma, (1+\varepsilon)\alpha + \gamma^{-1}\log_{(1+\varepsilon)} n)$-orientation of a dynamic graph G with arboricity α as well as a (δ, μ)-refinement H of G wrt. this orientation such that H is a forest. The fractional orientation of an edge can be computed in time $O(\log n)$, insertion takes worst-case $O(\gamma \cdot (\Delta^+)^2)$ time and deletion takes worst-case $O(\gamma \cdot \Delta^+ \cdot \log(n))$ time.*

Proof. Apply Theorem 15 for insertion/deletion. Note that $|L| = 0$ and $q(n) = 0$. The time spent repairing H after each insertion/deletion is in $O(\gamma\Delta^+ \log n)$ by Remark 16 and Observations 21 and 25, since we can process an edge from both Q and S in $O(\log n)$ time. Finally, Observation 20 and the Invariants 22 and 23 show correctness of the algorithm. ◀

Now tuning the parameters of Theorem 26, rounding edges in $G - H$ and 2-orienting H yields Theorem 1. By naively rounding $G - H$ in Theorem 26 and splitting the orientation using Lemma 27, we get an algorithm for dynamically maintaining a decomposition into $\lfloor(1+\varepsilon)\alpha'\rfloor$ pseudoforests and a single forest. Applying the colouring techniques from the full version yields Corollary 2.

4 Forests

We begin this section by outlining the main ideas for turning a dynamic low out-orientation into a dynamic low arboricity decomposition. Given a dynamic α'-bounded out-degree orientation, one can, with very little overhead, split it into α' 1-bounded out-degree orientations using a (slight modification) of an algorithm by Henzinger et al. [27]. Now, given this dynamic pseudoforest partition, we wish to apply the ideas of Blumenstock & Fischer [12] in order to turn the α' pseudoforests into $\alpha' + 1$ forests. The main technical challenge of making this process dynamic is the following: the algorithm from [27] relies heavily on each vertex having out-degree no more than 1 in each pseudoforest. However, the approach of Blumenstock & Fischer [12] is to move edges between pseudoforests, showing no regards as to why an edge was placed in a pseudoforest to begin with. Hence, if one naively applies this approach on top of the pseudoforest partition, one could potentially ruin the invariant that every vertex has out-degree no more than 1 in each pseudoforest, causing the algorithm of Henzinger et al. [27] to fail. We tackle this problem in steps. First, we show that if we were somehow able to invert the orientations of cycles, then we can make the moves of Blumenstock & Fischer's approach *faithful* to the degree condition of the pseudoforest algorithm of Henzinger et al. [27]. If we invert orientations along cycles in the pseudoforests, the out-degree of no vertex in the pseudoforests is changed. However, if we wish to perform these operations, we will have to do it in a manner that still allows us to maintain the underlying α'-bounded out-degree orientation. If the cycles are doubly η-valid, we invert the

cycles using Lemma 24. We do as in Section 3, but this time we add or subtract $1-\delta$ along the cycles. This ensures that every edge on the cycle now prefers the other endpoint, and so is naively rounded to the opposite direction without ending in H. The problem is that we have no guarantee that all edges are doubly η-valid. If an edge is only singly η-valid, then redistributing the load along a cycle containing this edge causes the edge to become invalid. However, by Lemma 11, we can delete such invalid edges and reinsert them again to restore the invariant that all edges are η-valid. We use a potential based argument to show that we can afford to perform these operations.

4.1 Ideas of Henzinger et al. and Blumenstock & Fischer

An α'-bounded out-degree orientation, can be split into α' pseudoforests by partitioning the edges such that each vertex has out-degree at most one in each partition. Then every connected component P_C in a partition is a pseudoforest. Indeed, $|E(P_C)| \leq |P_C|$ since every vertex has out-degree at most one. Hence, there can be at most one cycle in P_C. This idea is implicit in [27] by Henzinger et al. Note that we can store each pseudotree as a top tree with one extra edge with only $O(\log n)$ overhead per operation.

▶ **Lemma 27** (Implicit in [27]). *Given black box access to an algorithm maintaining an α'-bounded out-degree with update time $T(n)$, there exist an algorithm maintaining an α' pseudoforest decomposition with update time $O(T(n))$.*

Using the ideas of Blumenstock & Fischer [12], we can represent a pseudoforest P by a pair (F, M) s.t. F is a forest and M is matching, by adding exactly one edge from each cycle in P to M. Similarly, we can represent a partition of $E(G)$ into pseudoforests $(P_1, \ldots, P_k)$ by a pair (F, M) s.t. $F = \cup F_i$ and $M = \cup M_i$ and (F_i, M_i) represents P_i for all i.

In order to ensure the guarantees of Lemma 27, we need to maintain the invariant that every vertex has out-degree at most one in every pseudoforest. If this is the case, we say that the partition is *faithful* to the underlying orientation. Blumenstock & Fischer [12] perform operations on $G[M]$ in order to turn it into a forest. They call $G[M]$ the *surplus graph*. Some of the operations they perform, are described in the following lemma:

▶ **Lemma 28** (Implicit in [12]). *Let (F, M) be a faithful representation of a pseudoforest partition of a simple graph G equipped with an α'-bounded out-degree orientation. If $uv \in M_i$ and $vw \in M_j$ with $i \neq j$, then there exists an α'-bounded out-degree orientation with respect to which the partition gained by swapping $P_i \leftarrow P_i \cup \{vw\} - \{uv\}$ and $P_j \leftarrow P_j \cup \{uv\} - \{vw\}$ yields a faithful partition, and $uv \in M_j$ resp. $vw \in M_i$ iff. uv resp. vw are on the uni-cycle in their new pseudoforests.*

Furthermore, if $wx \in M_i$ for some x, then vw is not on a uni-cycle in P_i.

Proof. See [12] Lemma 2. To modify the orientation to accommodate the swaps, note that we can always reverse the direction of at most two cycles, without changing the out-degree of any vertex, such that both of the edges swapped are out-edges of v. Now swapping the two out-edges ensures that the partition stays faithful to the orientation (see Figure 2). ◀

Following Blumenstock & Fischer we note that if $e_1, \ldots, e_k$ is a path in a surplus graph $G[M]$ such that e_1 and e_k belong to the same matching M_i, then we can use the moves from Lemma 28 to restore colourfulness (see [12] Lemma 3). The key is that we can move the other edge in M_i towards e_1, and then after $O(k)$ switches, we are sure to end up in the furthermore part of Lemma 28. If a surplus graph contains no such paths, Blumenstock & Fischer say it is a *colourful* surplus graph. They show the following Lemma:

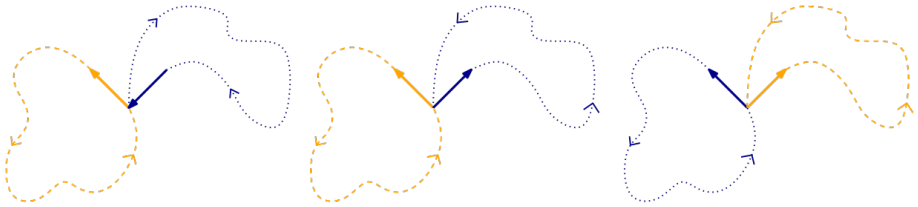


Figure 2 Moving edges between pseudoforests (represented by colour). Before performing the swap, we reorient a cycle so that the swapped edges are out-edges of their common endpoint. This ensures that every vertex has out-degree at most one in each pseudoforest.

▶ **Lemma 29** ([12]). *Suppose J is a colourful component of the surplus graph $G[M]$ of a graph G. Then for all $v \in J$ there exists an index i s.t. $N_{F_i}(v) \cap J = \emptyset$ and $J \cap M_i \neq \emptyset$.*

These Lemmas motivate the following approach: use Lemma 28 to ensure that the surplus graph is always colourful. Next use Lemma 28 to remove any cycles from the surplus graph.

4.2 Our algorithm for maintaining dynamic arboricity decompositions

Assume that we have an upper bound α_{max} on the arboricity throughout the entire update sequence. The algorithm works roughly as follows:

1. Run the algorithm from Theorem 26.
2. Naively round the orientation of each edge in $G - H$.
3. Split the rounded out-degree orientation on $G - H$ into pseudoforests.
4. Whenever an edge enters or moves between pseudoforests, we push it to a queue R.

We process each edge in $e \in R$ as follows: Put e into a pseudoforest. If e completes a cycle in a pseudoforest add it to $G[M]$. When e enters $G[M]$, we determine if it sits in a colourful component. If it doesn't, we apply Lemma 28 until all components in $G[M]$ are colourful. If a non-doubly valid cycle is reoriented in this process, we remove the singly-valid edges from the pseudoforests and add them to R. This ensures that the two edges from the same matching that we were trying to separate into two different components, are indeed separated. We will later bound the total number of edges pushed to R. If, on the other hand, the component is colourful, e may sit in a cyclic component. Then we apply Lemma 29 to remove the unique cycle. This may create a new non-colourful component, which we handle as before.

In the following, we describe the necessary data structures and sub-routines needed to perform these operations.

4.3 Operations on the surplus graph

In this section, we assume all cycles are doubly η-valid. In Section 4.5, we handle cases where this is not the case. Assuming that $G[M]$ is both colourful and acyclic, we can insert an edge in $G[M]$ and restore these invariants by performing switches according to Lemmas 28 and 29. Indeed, after inserting an edge, we can run, for example, a DFS on the component in $G[M]$ to determine if it is colourful. If it is not, we locate a path $e_1, \ldots, e_k$ such that $e_1, e_k \in M_i$. Then we apply Lemma 28 to e_{i-1} and e_i beginning with $i = k$, until an edge from M_i is removed from $G[M]$. Note that this is certain to happen when e_1 and e_3 belong to the same pseudoforest. We continue locating and handling paths until the component becomes colourful. If the component is colourful, but not acyclic, we choose a vertex v on the cycle and apply Lemma 29 to determine a pseudoforest represented in the component in which v is connected to no other vertex in the cyclic component. Then we determine a path

in the surplus graph between an edge in said pseudoforest and v. Now we move the edge in this pseudoforest to v using Lemma 28. If the edge is removed from $G[M]$ or the path is disconnected, we repeat the process. When such an edge is incident to v, we switch it with an edge on the cycle. Finally, we replace it in M with the unique neighbouring edge that is also incident to v in the cycle that put it in M. Now $G[M]$ is acyclic, but it may not be colourful. If this it the case, we repeat the arguments above until it becomes colourful. Note that these moves never create a cycle.

▶ **Lemma 30.** *After inserting an edge into $G[M]$, we can restore acyclicity and colourfulness in $O(\alpha^3 \log^2(n))$ time.*

4.4 Recovering neighbours

For each vertex, we will lazily maintain which of its out-edges belong to which pseudoforest. This costs only $O(1)$ overhead, when actually moving said edges. However, whenever we invert a cycle, these edges may change. Since the cycles can be long, we can only afford to update this information lazily, whenever the insertion/deletion algorithm determines the new out-neighbours of a vertex. When this happens, we say the vertex is *accessed*. Whenever an edge has its fractional load changed via a cycle inversion, it is always changed by the same amount. Hence, we make the following observation:

▶ **Observation 31.** *Between two accesses of a vertex v, the only possible new in-neighbours are the edges which were out-neighbours at the last access of v, and the only new out-neighbours are the vertices that are out-neighbours at the current access of v.*

Thus, we can recover exactly which incident edges might have changed in- and/or out-neighbour status from v, since the last time v was accessed by the insertion/deletion algorithm. To do so, we maintain that each top tree is rooted in the unique vertex, which has out-degree 0, when the underlying orientation is restricted to the tree. This ensures that we can recover v's unique out-edge in a pseudoforest by finding the first edge on the unique v-to-root path in the top tree. We maintain this information as follows:

- When we $link(u, v)$ with an edge oriented $u \to v$, we set the root of the new tree to be that of the tree containing v.
- When we $cut(u, v)$ with an edge oriented $u \to v$, we set the root of the tree containing u to be u and that containing v to be the same as the old tree.
- When we invert the orientation along a cycle originally oriented $u \to v \to \cdots \to u$, we change the root from v to u.
- When we perform a Lemma 29 swap, we also update the root accordingly.

Note that each update is accompanied by an operation costing $O(\log n)$ time, so the overhead for maintaining this information is only $O(1)$. With this information, we can recover the old out-neighbours as the stored out-neighbours, and the new out-neighbours by taking the first edge on the path from v to the root. Hence, we have shown:

▶ **Lemma 32.** *We can supply each vertex with a query returning a list L of neighbours which might have their status changed in time $O(\alpha \log n)$. Furthermore, $|L| = O(\alpha)$.*

4.5 Non doubly η-valid cycles

If a cycle is not doubly η-invalid, we still switch the orientation as before, but now we have to fix invalid edges. Assuming we know which edges have become η-invalidated, we fix them as follows: For every invalid edge, we first remove the edge from the pseudoforest it resides in.

This has two consequences. Firstly, the algorithm from Lemma 27 might move $O(1)$ edges between pseudoforests, and secondly, we also have to move an edge from the surplus graph back down as a normal edge in the pseudoforest it comes from. All of the (re)moved edges are pushed to the queue R. Then, we delete all invalid copies of edges in G^γ, and reinsert them. Now, all edges are valid again, and so we continue processing edges in R as described in Section 4.2. If an edge now belongs to H, we do not insert it into any pseudoforest.

It is important to note that the second consequence i.e. that we remove an edge from $G[M]$, either makes a Lemma 28 switch successful by removing one of the edges from M_i, or it removes an edge on one of the at most two paths between edges in M_i. In this case, we try to locate a second path, and handle it as before. This happens at most once: the component has at most one cycle, and hence at most two paths between two vertices. We ascribe the cost of deleting and reinserting invalid edges to the potential in Lemma 33 that bounds the total number of copies of edges that are inserted into G^γ. This cost is not ascribed to the algorithm maintaining $G[M]$. Set $\Delta^+_{max} = (1+\varepsilon)\alpha_{max}\gamma + \log_{(1+\varepsilon)} n$, we have:

▶ **Lemma 33.** *The total amount of insertions and deletions performed by the insertion / deletion algorithm over the entire update sequence is in* $O(\frac{\gamma\Delta^+_{max}}{\eta}(\Delta^+_{max} \cdot i + d))$

Lemma 33 allows us to bound the total number of edges moved between pseudoforests:

▶ **Lemma 34.** *We move at most* $O(\frac{\gamma(\Delta^+_{max})^3}{\eta}(i+d))$ *edges between pseudoforests.*

Note that this implies that the total no. of insertions into R is $O(\frac{\gamma(\Delta^+_{max})^3}{\eta}(i+d))$.

4.6 Locating singly η-valid edges

When we are accessing an edge, we can check if it is doubly η-valid or not (this information depends only on the load on the endpoints), and maintain this information in a dynamic forest using just 1 bit of information per edge. This allows us to later locate these edges using non-local searches in top trees. However, when edges go between being singly η-valid and doubly η-valid through operations not accessing said edge, we are not able to maintain this information. This can happen in two ways: either 1) a vertex has its load lowered causing an in-going edge to now become doubly η-valid or an outgoing edge to become singly η-valid or 2) a vertex has its load increased causing similar issues. We say an edge is *clean* if we updated the validity bit of an edge, the last time the out-degree of an endpoint of the edge was altered. Otherwise, we say it is *dirty*. Now if all edges on a cycle are clean, we can use top trees to direct searches for the edges that become invalidated by inverting the cycle.

We maintain a heavy-light decomposition of every forest using dynamic st-trees [40] to help us ensure that we can clean all edges in a cycle in time $O(\log^2 n)$. The idea is to maintain the invariant that all heavy edges are clean. Now we can clean a cycle by cleaning the at most $O(\log n)$ light edges on said cycle. In order to realise this invariant, whenever the degree of a vertex is changed, we need to update all of its incident heavy edges in all of the heavy-light decompositions. Since a vertex is incident to at most two heavy edges in each forest, we have to update $O(\alpha)$ heavy edges. The following holds:

▶ **Observation 35.** *We can locate singly η-valid edges on a clean cycle in time $O(\log n)$ per edge, if we spend $O(\log n)$ overhead updating the bit indicating double validity.*

▶ **Observation 36.** *We can insert and delete edges in the heavy-light decomposition in worst case $O(\log^2 n)$ time.*

▶ **Lemma 37.** *We can check if a cycle is doubly valid in time $O(\log^2 n)$.*

4.7 Conclusion

Theorem 3 and Corollary 4 follow from Lemma 38 (see full version for details).

▶ **Lemma 38.** *Consider a sequence of updates with i insertions and d deletions.*

1. *The insertion/deletion algorithm spends $O(\log^6(n) \cdot \alpha_{max}^4 \cdot \varepsilon^{-12}(i+d))$ time to update the fractional out-degree orientation and the refinement.*
2. *The algorithm maintaining the pseudoforests spends $O(\log^6(n) \cdot \alpha_{max}^3 \cdot \varepsilon^{-8}(i+d))$ time.*
3. *The algorithm maintaining the surplus graph spends $O(\log^6(n) \cdot \alpha_{max}^6 \cdot \varepsilon^{-8}(i+d))$ time.*

We have shown how to maintain an $\alpha+2$ arboricity decomposition of a fully dynamic graph as it undergoes an arboricity α preserving sequence of updates in $\text{poly}(\log n, \alpha)$ time per update. We have also shown how to maintain an $\lfloor(1+\varepsilon)\alpha\rfloor+2$ out-orientation of a fully dynamic graph in $\text{poly}(\log n, \alpha)$ time per update. These algorithms are the first dynamic algorithms to go below 2α forests and out-edges, respectively, and the number of forests matches the best near-linear static algorithm by Blumenstock and Fischer [12]. We apply these algorithms to get new trade-offs for implicit colouring algorithms for bounded arboricity graphs. In particular, we maintain $4 \cdot 2^\alpha$ and $2 \cdot 3^\alpha$ implicit colourings in $\text{poly}(\log n, \alpha)$ time per update. This improves upon the $2^{40\alpha}$ colours of the previous most colour-efficient algorithm maintaining $\text{poly}(\log n, \alpha)$ update time [27]. In particular, this reduces the number of colours for planar graphs from 2^{120} to 32. An interesting direction for future work is to see, if one can reduce the number of forests even further in the static case, while still achieving near-linear running time. Also, even though our algorithms use few colours and forests, the update times contain quite high polynomials in both $\log n$ and α. Is it possible to get more efficient update times without using more forests? Finally, for constant α, we get $\alpha+2$ out-edges. Brodal & Fagerberg [13] showed that one cannot get α out-edges with faster than $\Omega(n)$ update time (even amortised). The question remains, can one get $\alpha+1$?

5 Acyclic Orientations and Arboricity Decompositions

In this section, we briefly sketch the algorithm in Theorem 5 (Note that this section is partially based on the master's thesis by Christiansen [15, Chapter 9], see the full version for proofs). We modify an algorithm by Brodal & Fagerberg [13]. Specifically, we change how an edge is inserted. The algorithm maintains a list of out-edges $\text{out}(u)$ for each vertex $u \in G$. An edge e is in $\text{out}(u)$ if and only if $e \in E(G)$ and e is oriented away from u. As a result $d^+(u) = |\text{out}(u)|$. All of these lists are initialized to be empty. The algorithm ensures that the maximum out-degree of the vertices in G is d for some constant $d > 2\alpha$ to be specified later. The algorithm handles deletions and insertions in the following way:

Deletion: If e incident to x, y is deleted, we search $\text{out}(x)$ and $\text{out}(y)$ for e, and delete it.

Insertion: When an edge e is inserted, an arbitrary endpoint u of e is chosen, and e is added to $\text{out}(u)$. Now every edge in $\text{out}(u)$ is oriented in the other direction (also e) i.e. we delete $f = (u, v)$ from $\text{out}(u)$ and add f to $\text{out}(v)$ instead for all edges $f \in \text{out}(u)$. Now u has out-degree at most d, but the reorientations of an edge $f = (u, v)$ might increase $|\text{out}(v)|$ above d. The algorithm then proceeds by reorienting all out-edges out of v. It continues this process until all vertices have out-degree at most d.

Note that this process terminates: Since G has arboricity α, it has an orientation O such that the maximum out-degree in G is α. Call an edge in $E(G)$ *good*, if it is oriented the same way by both the algorithm and O and *bad* if it isn't. Now, inserting e could, in the worst case, make the algorithm change the orientation of α good edges. However from here on,

every vertex whose edges are reoriented will increase the total number of good edges by at least $d - 2\alpha \geq 1$, so the process terminates. The algorithm differs from the one presented in [13] in only one way. When an edge $e = uv$ is inserted, we always turn u into a sink. In [13], this only happens if u's out-degree increases above d. This small modification ensures no cycles are created: when an edge is inserted, one of its endpoints is turned into a sink, and so this edge is in no cycle. Also, turning a vertex into a sink does not create any cycles.

References

1 Oswin Aichholzer, Franz Aurenhammer, and Günter Rote. *Optimal graph orientation with storage applications*. SFB-Report , SFB 'Optimierung und Kontrolle'. TU Graz, 1995. Reportnr.: F003-51.

2 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, October 2005. doi:10.1145/1103963.1103966.

3 K. Appel and W. Haken. A proof of the four color theorem. *Discret. Math.*, 16(2):179–180, 1976. doi:10.1016/0012-365X(76)90147-3.

4 Srinivasa Rao Arikati, Anil Maheshwari, and Christos D. Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discret. Appl. Math.*, 78(1-3):1–16, 1997. doi:10.1016/S0166-218X(97)00007-3.

5 Niranka Banerjee, Venkatesh Raman, and Saket Saurabh. Fully dynamic arboricity maintenance. In *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2019. doi:10.1007/978-3-030-26176-4_1.

6 L. Barba, J. Cardinal, M. Korman, S. Langerman, A. van Renssen, M. Roeloffzen, and S. Verdonschot. Dynamic graph coloring. *Algorithmica*, 81(4):1319–1341, 2019.

7 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 25–34, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1400751.1400757.

8 Edvin Berglin and Gerth Stolting Brodal. A Simple Greedy Algorithm for Dynamic Graph Orientation. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ISAAC.2017.12.

9 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.

10 Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic (Δ+1)-coloring in constant update time. *CoRR*, abs/1910.02063, 2019. arXiv:1910.02063.

11 Markus Blumenstock. Fast algorithms for pseudoarboricity. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*, pages 113–126. SIAM, 2016. doi:10.1137/1.9781611974317.10.

12 Markus Blumenstock and Frank Fischer. A constructive arboricity approximation scheme. In *SOFSEM 2020: Theory and Practice of Computer Science - 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*, volume 12011 of *Lecture Notes in Computer Science*, pages 51–63. Springer, 2020. doi:10.1007/978-3-030-38919-2_5.

13 Gerth Stolting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *In Proc. 6th International Workshop on Algorithms and Data Structures (WADS)*, pages 342–351. Springer-Verlag, 1999.

14 Norishige Chiba, Takao Nishizeki, and Nobuji Saito. A linear 5-coloring algorithm of planar graphs. *J. Algorithms*, 2(4):317–327, 1981. doi:10.1016/0196-6774(81)90031-6.

15 Aleksander B. G. Christiansen. Dynamic algorithms for implicit vertex-colouring of graphs with bounded arboricity. Master's thesis, Technical University of Denmark, Kgs. Lyngby, Denmark, October 2021.

16 Aleksander B. G. Christiansen and Eva Rotenberg. Fully-dynamic α+2 arboricity decomposition and implicit colouring. *CoRR*, abs/2203.06039, 2022. doi:10.48550/arXiv.2203.06039.

17 O. Coudert. Exact coloring of real-life graphs is easy. *Proceedings of 34th Design Automation Conference. ACM*, 35(1):121–126, 1997.

18 Jack Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, page 67, 1965.

19 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inf. Process. Lett.*, 51(4):207–211, August 1994. doi:10.1016/0020-0190(94)90121-X.

20 Greg N. Frederickson. On linear-time algorithms for five-coloring planar graphs. *Inf. Process. Lett.*, 19(5):219–224, 1984. doi:10.1016/0020-0190(84)90056-5.

21 Harold Gabow and Herbert Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 407–421, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/62212.62252.

22 Harold N. Gabow. Algorithms for graphic polymatroids and parametric s-sets. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 88–97, USA, 1995. Society for Industrial and Applied Mathematics.

23 Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2505–2523. SIAM, 2017. doi:10.1137/1.9781611974782.166.

24 David G. Harris. Distributed local approximation algorithms for maximum matching in graphs and hypergraphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 700–724, 2019. doi:10.1109/FOCS.2019.00048.

25 David G. Harris, Hsin-Hao Su, and Hoa T. Vu. On the locality of nash-williams forest decomposition and star-forest decomposition. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 295–305, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467908.

26 Meng He, Ganggui Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *ISAAC*, 2014.

27 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020. arXiv:2002.10142.

28 Monika Henzinger and Pan Peng. Constant-time dynamic (Δ+1)-coloring. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.53.

29 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2006. doi:10.1007/11786986_21.

30 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2014. doi:10.1007/978-3-662-43951-7_45.

31 Łukasz Kowalik. Approximation scheme for lowest outdegree orientation and graph density measures. In *Proceedings of the 17th International Conference on Algorithms and Computation*, ISAAC'06, pages 557–566, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11940128_56.

32 Łukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.*, 102(5):191–195, May 2007. doi:10.1016/j.ipl.2006.12.006.

33 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}$(vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433, 2014. doi:10.1109/FOCS.2014.52.

34 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262, 2013. doi:10.1109/FOCS.2013.35.

35 D. Matula, Y. Shiloach, and R. Tarjan. Two linear-time algorithms for five-coloring a planar graph, 1980. Reportnr.: STAN-CS-80-830.

36 C. St.J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, 1964. doi:10.1112/jlms/s1-39.1.12.

37 Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982. doi:10.1002/net.3230120206.

38 Neil Robertson, Daniel P. Sanders, Paul D. Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 571–575. ACM, 1996. doi:10.1145/237814.238005.

39 Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 181–193. ACM, 2020. doi:10.1145/3357713.3384327.

40 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 114–122, New York, NY, USA, 1981. Association for Computing Machinery. doi:10.1145/800076.802464.

41 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3), June 2020. doi:10.1145/3392724.

42 Hsin-Hao Su and Hoa T. Vu. Distributed Dense Subgraph Detection and Low Outdegree Orientation. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2020.15.

43 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 681–690, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132612.

Expander Random Walks: The General Case and Limitations

Gil Cohen
Department of Computer Science, Tel Aviv University, Israel

Dor Minzer
Department of Mathematics, Massachusetts Institute of Technology, Cmabridge, MA, USA

Shir Peleg
Department of Computer Science, Tel Aviv University, Israel

Aaron Potechin
Department of Computer Science, University of Chicago, IL, USA

Amnon Ta-Shma
Department of Computer Science, Tel Aviv University, Israel

Abstract

Cohen, Peri and Ta-Shma [11] considered the following question: Assume the vertices of an expander graph are labelled by ± 1. What "test" functions $f : \{\pm 1\}^t \to \{\pm 1\}$ can or cannot distinguish t independent samples from those obtained by a random walk? [11] considered only balanced labellings, and proved that for all symmetric functions the distinguishability goes down to zero with the spectral gap λ of the expander G. In addition, [11] show that functions computable by AC^0 circuits are fooled by expanders with vanishing spectral expansion.

We continue the study of this question. We generalize the result to all labelling, not merely balanced ones. We also improve the upper bound on the error of symmetric functions. More importantly, we give a matching lower bound and show a symmetric function with distinguishability going down to zero with λ but not with t. Moreover, we prove a lower bound on the error of functions in AC^0 in particular, we prove that a random walk on expanders with constant spectral gap does not fool AC^0.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains

Keywords and phrases Expander Graphs, Random Walks, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.43

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2021/091/`

Funding *Gil Cohen*: Supported by ERC starting grant 949499 and by the Israel Science Foundation grant 1569/18.
Shir Peleg: The research leading to these results has received funding from the Israel Science Foundation (grant number 514/20) and from the Len Blavatnik and the Blavatnik Family foundation.
Amnon Ta-Shma: The research leading to these results has received funding from the Israel Science Foundation (grant number 952/18)

Acknowledgements The authors thank Amir Yehudayoff for pointing out that the analysis of the error of the parity function is tight if the second eigenvector of the graph is Boolean. This remark, several years later, matured to the current paper. We thank Venkat Guruswami and Vinayak Kumar for discussing their results with us. We thank Ron Peled for discussions on the CLT in TVD, and for pointing us to results on local convergence for independent processes. We thank Oded Goldreich for the vanishing with t and λ notation.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 43; pp. 43:1–43:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Expanders are sparse undirected graphs that have many desirable pseudorandom properties. A formal definition can be given in several equivalent ways and here we consider the algebraic definition where an undirected graph $G = (V, E)$ is a λ-*spectral expander* if the second largest eigenvalue of its normalized adjacency matrix M is bounded above by λ. For simplicity, we only consider regular graphs, in which case M is also the random walk matrix of G. Expander graphs are among the most useful combinatorial objects in theoretical computer science, pivotal in derandomization [18, 29], complexity theory [37, 1, 12] and coding theory [32, 22, 33, 13] to name a few. Many works in the literature have studied explicit constructions of expander graphs (see, e.g., [24, 25, 7, 30, 6, 26]) and utilized their pseudorandom properties. We refer the reader to the excellent expositions [17, 35] and to Chapter 4 of [36].

Expanders can be thought of as spectral sparsifiers of the clique. Let $\mathbf{J}$ be the normalized adjacency matrix of the n-vertex complete graph with self-loops, i.e., the $n \times n$ matrix with all entries equal to $\frac{1}{n}$. One can express the normalized adjacency matrix M of G as $M = (1-\lambda)\mathbf{J} + \lambda E$ for some operator E with spectral norm bounded by 1. As such, one can hope to substitute a sample of two *independent* vertices with the "cheaper" process of sampling an edge from an expander and using its two (highly correlated) end-points. This is captured, e.g., by the expander mixing lemma [2]. This idea also appears in many derandomization results, [18, 3, 28, 29, 31, 9].

A useful generalization of the above is to consider not just an edge but rather a length $t-1$ random walk (where the length is measured in edges) on the expander as a replacement to t independent samples of vertices. For concreteness, consider a labelling $\mathsf{val} : V \to \{\pm 1\}$ of the vertices with mean $\mu = \mathbf{E}[\mathsf{val}(V)]$. Quite a lot is known about random walks on expanders. Next, we elaborate on the hitting property of expanders [1, 10, 19, 5] as well as the expander Chernoff bound [1, 10, 19, 14, 16].

The hitting property states that for every set $A \subset V$, a length $t-1$ random walk is contained in A with probability at most $(\mu + \lambda)^t$. For $\lambda \ll \mu$, this bound is close to μ^t - the probability of the event with respect to t independent samples. The expander hitting property corresponds to a random walk "fooling" the AND function, that is, for every λ-spectral expander and every labelling val as above, the AND function cannot distinguish with good probability labels obtained by t independent samples from labels obtained by taking a length $t-1$ random walk. The fundamental expander Chernoff bound states that the number of vertices in A visited by a random walk is highly concentrated around its measure $|A|/|V|$. The expander Chernoff bound corresponds to fooling functions indicating whether the normalized Hamming weight of the input is concentrated around some number μ. Perhaps surprisingly, it was shown that even the highly sensitive PARITY function is fooled by a random walk on expanders (this was noted independently by Alon in 1993 for arbitrarily long walks, Wigderson and Rozenman in 2004 for length 1 walks, and [33] where the result appears).

Sometimes a random walk *is not* a good replacement to independent samples. To see this, suppose G is a λ-spectral expander for some constant λ, that has a cut $A \subset V$ with $|A| = \frac{|V|}{2}$ and $|E(A, \overline{A})| \geqslant \mu|A|$ for $\mu \geqslant \frac{1}{2} + \widetilde{\Omega}(\lambda)$. Such graphs exist (see [15, Section 7]). If one samples t independent vertices $(v_1, \ldots, v_t)$ from the graph, we expect (v_i, v_{i+1}) to cross the cut about half the time, and by the Chernoff bound the actual number of cut crossings is highly concentrated around the mean. In contrast, when we take a random walk on the graph we expect to cross the cut a μ-fraction of the time, and intuitively the number of cut

crossings should be concentrated around μ.[1] Thus, the simple test function that counts the number of times we cross the cut and apply a threshold at $\frac{1}{2} + \tau$ for some $\tau = \widetilde{\Theta}(\lambda)$ should distinguish with probability close to 1 between a random walk and independent samples.

This brings to the forefront a natural question that was recently raised by [11] (see also the work of Guruswami and Kumar [15] who considered a related question).

What test functions does a random walk on an expander fool?

Formally, we compare two distributions on the set $\{\pm 1\}^t$. The first "ideal" distribution is obtained by sampling independently and uniformly at random t vertices $v_1, \ldots, v_t$ and returning $(\mathsf{val}(v_1), \ldots, \mathsf{val}(v_t))$. If we let $\mu = \mathbf{E}[\mathsf{val}(V)]$, the latter induces the distribution U_t^μ in which the t bits are independent and each has mean μ. The second distribution, denoted by $\mathrm{RW}_{G,\mathsf{val}}$, is obtained by taking a length $t-1$ random walk on the graph, namely, sample v_1 uniformly at random from V, and then for $i = 2, 3, \ldots, t$, sample v_i uniformly at random from the set of neighbors of v_{i-1}, and return $(\mathsf{val}(v_1), \ldots, \mathsf{val}(v_t))$. Denote

$$\mathcal{E}_{G,\mathsf{val}}(f) = \left|\mathbf{E}\, f(\mathrm{RW}_{G,\mathsf{val}}) - \mathbf{E}\, f(U_t^\mu)\right|.$$

Informally, $\mathcal{E}_{G,\mathsf{val}}(f)$ measures the distinguishability between these two distributions as observed by the test function f on the graph G with respect to the labelling val. We wish to have a discussion that holds uniformly on all λ-spectral expanders (on any number of vertices) and for every labelling. The bound, however, is expected to depend on the expectation μ of the labelling. We denote by $\mathcal{E}_{\lambda,\mu}(f)$ the supremum of $\mathcal{E}_{G,\mathsf{val}}(f)$ over all λ-spectral expanders G, on any number of vertices, and all labelling functions $\mathsf{val} : V \to \{\pm 1\}$ with $\mathbf{E}[\mathsf{val}(V)] = \mu$.

The work [11] focuses on the case $\mu = 0$. One result shows that

$$\mathcal{E}_{\lambda,0}(\mathsf{MAJ}) \leqslant O\left(\frac{\lambda^2}{\sqrt{t}}\right) \tag{1.1}$$

Their main result states that for each balanced labelling, for every symmetric function $f : \{\pm 1\}^t \to \{\pm 1\}$,

$$\mathcal{E}_{\lambda,0}(f) = O(\lambda \cdot \log^{3/2}(1/\lambda)). \tag{1.2}$$

This readily implies, for the specific case of balanced labelling, a central limit theorem with respect to the total variation distance, that vanishes as $\lambda \to 0$, thus strengthens previous results that considered the Kolmogorov distance [20, 23, 21] instead of the total variation distance.

To summarize the state of knowledge so far:

- Every symmetric function is fooled with error probability going down to zero with the spectral gap λ (see Equation (1.2)), where $\mu = 0$.
- The MAJ function is fooled with error probability going down to zero with t even when λ is fixed (see Equation (1.1)); and,
- The PARITY, AND, OR functions are fooled with error probability going down to zero *exponentially* fast with t even when λ is fixed.

Accordingly, let us say an error function vanishes with λ, if the error function is vanishing as $\lambda \to 0$. Similarly, we say an error function vanishes with t, if for some fixed $\lambda \geqslant 0$, it is going down to zero together with t.

[1] To show such a concentration one needs to invoke a Chernoff bound for a walk on the corresponding directed line graph.

[11] further considers non-symmetric functions. In particular, they analyze test functions that are computable by AC^0 circuits and prove that if f is computable by a size-s depth-d circuit then

$$\mathcal{E}_{\lambda,0}(f) = O(\sqrt{\lambda} \cdot (\log s)^{2(d-1)}). \tag{1.3}$$

Thus, for balanced labelling, every test function in AC^0 cannot distinguish t independent labels from those obtained by a random walk on a λ-spectral expander provided λ is taken sufficiently small. This result can be thought of as an analog of Braverman's celebrated result [8] (see also [34]) that studies the pseudorandomness of k-wise independent distributions with respect to AC^0 test functions. However, for it to be meaningful, the spectral gap λ should be small.

1.1 Our contribution

The work of [11] leaves several open problems. First, and foremost, while [11] show the error function of any symmetric function vanishes with λ, it leaves open the possibility that a better convegence exists and, perhaps, the error function of any symmetric function vanishes with t, i.e., for some fixed λ, the error function goes down to zero together with the walk length t. Indeed, this is the case with the AND, OR and PARITY functions, where the error vanishes exponentially fast with t, and the MAJ function where the error goes down polynomially in t (see Equation (1.1)). Similarly, one may ask whether the error of AC^0 functions decays faster than Equation (1.3) and allows for larger spectral gaps λ then dictated by the above bound.

Our first result is that there exists a symmetric function for which the error function does not vanish with t:

▶ **Theorem 1.** *There exists a family of symmetric functions $(f_t)_{t\in\mathbb{N}}$ where $f_t : \{\pm1\}^t \to \{\pm1\}$ such that for every λ there is a λ-spectral expander $G = (V, E)$, and a labelling $\mathsf{val} : V \to \{\pm1\}$ with $\mathbf{E}[\mathsf{val}(V)] = 0$, such that for all t, $\mathcal{E}_{G,\mathsf{val}}(f_t) = \Omega(\lambda)$.*

To explain how we obtain such a lower bound on a function f, we first review how [11] obtained their upper bound. The key idea in [11] is to expand the test function f under consideration in the Fourier basis. The question of fooling general test functions then reduces to the study of test functions that are Fourier characters. Now, let G denote the adjacency matrix of the graph (i.e, $M = \frac{1}{d}G$). Also, for a labelling $\ell : V \to \{\pm1\}$ let us denote by P the diagonal matrix with $\ell(i)$ in the i'th element on the diagonal. One can check that for the parity function $\chi_{[t]} : \{\pm1\}^t \to \{\pm1\}$, $\chi_{[t]}(x) = \prod_{i=1}^t x_i$, we get $\mathbf{E}[\chi_{[t]}(\mathrm{RW}_{G,\mathsf{val}})] = \mathbf{1}^T \left(\prod_{i=1}^t PG\right) \mathbf{1},$, where $\mathbf{1} = (\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$.

In general,

$$\mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}})] = \mathbf{1}^T \left(\prod_{i=1}^t P^{\delta_S(i)} G\right) \mathbf{1}, \tag{1.4}$$

where $\delta_S(i)$ is 1 if $i \in S$ an 0 otherwise. In [11] it is shown how to upper bound this expression for any λ-expander G and μ-biased function val.

For the proof of Theorem 1 we choose a λ-expander G and a labelling function val such that we can *exactly* express Equation (1.4) in terms of λ, S. To do so, we first choose G to be a Cayley graph over an Abelian group, and we use the fact that the eigenvectors of such a graph correspond to the characteristic functions of the underlying group, regardless of

the set of generators used. One disadvantage in choosing a Cayley graph over an Abelian group is that it cannot give constant degree exapnders, though this is not a concern to us because with logarithmic degree we can have vanishing second eigenvalue. Next, we choose the underlying group to be $\mathbb{Z}_2^n$. This guarantees that the characteristic functions of $\mathbb{Z}_2^n$, and therefore also all the entries in all eigenvectors, are Boolean, i.e., either 1 or -1. Finally, we choose the labelling function val to correspond to the entries of the eigenvalue with the second largest eigenvalue.

The above choices guarantee that $P\mathbf{1} = v_2$ and $Pv_2 = \mathbf{1}$ (because $P^2 = I$). Also $G\mathbf{1} = \mathbf{1}$ and $Gv_2 = \lambda v_2$. It follows that no matter what S is, $\left(\prod_{i=1}^{t} P^{\delta_S(i)} G\right)\mathbf{1}$ belongs to the two dimensional subspace $\mathrm{Span}\,(\mathbf{1}, v_2)$ and, furthermore, has a closed expression as a function of t, λ and S.

We finally choose a function f for which we can estimate the expression we get. We choose f to have high mass on its second Fourier level. It turns out that we can take f to be, e.g., the threshold function that returns one if the number of ones exceeds the mean by one standard deviation, and this function has error function that is of the order λ, and, therefore, in particular, vanishes with λ but not with t. It is interesting to note that, in contrast, the MAJ function, that has threshold at the mean, vanishes with t.

Next, using the same graph and labelling we also prove that constant spectral expansion does not suffice to fool AC^0 circuits. In fact, the bound obtained by [11] is tight up to a polynomial. Let $\mathsf{AC}(d)$ denote the class of all languages with polynomial size boolean circuit of depth at most d. Then:

▶ **Theorem 2.** *There exists a constant $\varepsilon > 0$ such that the following holds. For every integer $d \geqslant 3$ there exist $t_d, c_d \in \mathbb{N}$, and a family of functions $(h_t)_{t_d \leqslant t \in \mathbb{N}} \subset \mathsf{AC}(d)$ such that the following holds. For every $\lambda \geqslant \frac{c_d}{\log^{d-2} t}$ there is a λ-spectral expander $G = (V, E)$ and a labelling $\mathsf{val} : V \to \{\pm 1\}$ with $\mathbf{E}[\mathsf{val}(V)] = 0$ such that $\mathcal{E}_{G,\mathsf{val}}(h_t) \geqslant \varepsilon$.*

The choice of function f here is more complicated. The key idea is that two adjacent bits obtained by such a random walk are λ correlated. Thus, evaluating a function f on the parity of consecutive bits obtained by a random walk is the same as applying the noise operator $T_\lambda(f)$ (see Claim 17 for an exact statement). Having this key fact, we construct small depth functions that are highly sensitive to small noise. We first start with the Tribes function composed with XOR on two adjacent bits. This gives a function in $\mathsf{AC}(3)$ with large distinguishability. We then give a recursive construction of a family of functions $h_d \in \mathsf{AC}(d+1)$ for every d, where in each step we increase the depth by one and the noise sensitivity of h_d by a logarithmic factor. This gives the desired dependence of $\mathcal{E}_{G,\mathsf{val}}(h_d)$ on d.

Finally, we also tighten and simplify the upper bounds given in [11]. We prove:

▶ **Theorem 3.** *For every symmetric function $f : \{\pm 1\}^t \to \{\pm 1\}$, all $\mu \in (-1, 1)$ and $0 < \lambda < \frac{1-|\mu|}{128e}$ it holds that*

$$\mathcal{E}_{\lambda,\mu}(f) \leqslant \frac{124}{\sqrt{1-|\mu|}} \cdot \lambda.$$

Theorem 3 improves upon the corresponding theorem in [11] in two ways:

1. First, the results in [11] are obtained only for balanced test functions f. In contrast, Theorem 3 holds for every test function f with arbitrary bias μ.
2. Second, the bound stated in Theorem 3 improves upon the bound in Equation (1.2) by removing the $\log^{3/2}(1/\lambda)$ factor.

The extension of the results of [11] to arbitrary bias μ is obtained by modifying the Fourier basis we work with. For a given bias μ we choose a basis that consists of $\prod_{i \in S} \frac{x_i - \mu}{\sqrt{1-\mu^2}}$ for all $S \subseteq [t]$. The improvement of the poly-logarithmic factor is achieved by using a more direct Fourier analysis argument. The proof strategy of [11] is to bound the error of weight indicator functions, and use it to handle weights around the mean. Then the argument invokes the expander Chernoff bound for bounding the remaining weights. Our approach does not go through analyzing weight indicator functions nor it uses the expander Chernoff bound. Instead, we use a very simple bound on the Fourier mass of symmetric functions, which gives a simpler and better analysis.

1.2 Open problems

We conclude the introduction with several open problems that follow from our work.

1. Can one combine the distribution obtained by a random walk on an expander with another pseudorandom distribution to obtain stronger results for functions in AC^0. For example, does permuting the values of the random walk with a pairwise independent permutation yields a distribution that better fools AC^0?
2. As explained before, our lower bounds are obtained for a graph G that is a Cayley graph over an Abelian group. It is well-known that such a Cayley graph with constant expansion gap, has degree that depends on the number of vertices. Thus, a natural question is whether we can give similar lower bounds for constant degree graphs.
3. Continuing this line of thought, it is still possible that there is a family of graphs that fools all symmetric functions with error going down to zero with t. I.e., that while for some graphs (like Cayley graphs over $\mathbb{Z}_2^n$) there are bad labelling functions, for some other expander graphs, no such bad labellings exist. Similarly, it is possible that for some specific expanders better bounds exist for test functions in AC^0. Finding such graphs is a compelling goal that might require studying additional properties of graphs beyond expansion.
4. Finally, there is still a polynomial gap between the value of λ that fools functions in AC^0 and the corresponding lower bound we obtain. Any progress towards closing this gap will be interesting.

1.3 Paper organization

In Section 2 we give some background, mainly on Fourier Analysis. In Section 3 we recall the basic framework of [11], except that we do it for arbitrary bias μ rather than just bias $\mu = 0$. In Section 4 we choose the graph and labelling function that we use for the lower bounds, and for which we can compute exactly the error induced by characters. In Section 5 we prove Theorem 1 and show that threshold function at one standard deviation away from the mean has error that goes down to zero with λ but not with t. In Section 6 we prove a special case of Theorem 2 for the case of $d = 3$. The full proof of Theorem 2 appears in the full version of the paper. Then we turn to give a better upper bound on the error function and in show a better and tight upper bound with a simpler proof. Finally we show the threshold function about the mean (if $\mu = 0$ it is MAJ) and weight indicator functions do vanish with t. The last two results appear in the full version of the paper.

2 Preliminaries

We let $[n] = \{1, \ldots, n\}$, $\mathbb{1} \in \mathbb{R}^n$ denote the all 1s vector, i.e., $\mathbb{1} = (1, \ldots, 1)^T \in \mathbb{R}^n$. We let $\mathbf{1} \in \mathbb{R}^n$ denote the normalized vector of $\mathbb{1}$, i.e $\mathbf{1} = \frac{1}{\sqrt{n}} \cdot \mathbb{1}$, we also use $\mathbf{J} := \mathbf{1}\mathbf{1}^\mathsf{T}$. When we write $\|\cdot\|$ we always refer to the L_2-norm. Unless stated otherwise, $\log x = \log_2 x$. Throughout the paper, we make use of the following well known inequalities about binomial coefficients. Let $a \geqslant b \geqslant 1$ be integers. Then, $(\frac{a}{b})^b \leqslant \binom{a}{b} \leqslant (\frac{ea}{b})^b$.

2.1 Fourier analysis

Consider the space of functions $f : \{\pm 1\}^t \to \mathbb{R}$, along with the inner product

$$\langle f, g \rangle = 2^{-t} \sum_{x \in \{\pm 1\}^t} f(x)g(x).$$

It is a well-known fact that the set $\{\chi_S(x) \mid S \subseteq [t]\}$, where $\chi_S(x) = \prod_{i \in S} x_i$, forms an orthonormal basis with respect to this inner product, which is called the Fourier basis. Thus every function $f : \{\pm 1\}^t \to \mathbb{R}$ can be uniquely represented as $f(x) = \sum_{S \subseteq [t]} \widehat{f}(S)\chi_S(x)$, where $\widehat{f}(S) \in \mathbb{R}$.

In this work we consider other bases, with respect to a similar inner product. Let $\mu \in [-1, 1]$, and denote by U_t^μ the distribution over $\{\pm 1\}^t$ where each bit is chosen independently with expectation μ. Define $\langle f, g \rangle_\mu = \mathbf{E}_{x \sim U_t^\mu}[f(x)g(x)]$. Denote by $\sigma = \sqrt{1 - \mu^2}$, and let $\chi_S^\mu(x) = \prod_{i \in S} \frac{x_i - \mu}{\sigma}$. It is easy to see that the set $\{\chi_S^\mu(x) \mid S \subseteq [t]\}$, forms an orthonormal basis with respect to this new inner product, which is called the *μ-biased Fourier basis*. To see this, note that, by design, for $S \neq \emptyset$, $\mathbf{E}[\chi_S^\mu] = 0$ and $\mathbf{E}[(\chi_S^\mu)^2] = 1$. Similarly to the standard Fourier basis, every function $f : \{\pm 1\}^t \to \mathbb{R}$ can be uniquely represented as

$$f(x) = \sum_{S \subseteq [t]} \widehat{f_\mu}(S)\chi_S^\mu(x),$$

where $\widehat{f_\mu}(S) \in \mathbb{R}$.

We say that a function $f : \{\pm 1\}^t \to \mathbb{R}$ is symmetric if for every permutation $\sigma \in S_t$, $f(x_1, \ldots, x_t) = f(x_{\sigma(1)}, \ldots, x_{\sigma(t)})$. It is not hard to show that if f is symmetric, then for every $S_1, S_2 \subseteq [t]$, with $|S_1| = |S_2|$, $\widehat{f_\mu}(S_1) = \widehat{f_\mu}(S_2)$. This allows us to use the following definition for symmetric functions: $\widehat{f_\mu}(k) = \left|\widehat{f_\mu}([k])\right|$, which is the absolute value of the Fourier coefficients of any weight k character. For more details on biased Fourier analysis see Chapter 8 of [27].

3 The basic framework extended to arbitrary balanced tests

[11] reduced the analysis of the error function of a *balanced* test function f to the analysis of the error function of characters. In this section we restate this framework, but do it in a more general way that applies to any test function f, no matter how balanced it is.

Let $G = (V, E)$ be a regular λ-spectral expander, and let $\mathsf{val} : V \to \{\pm 1\}$ be a labelling of the vertices of G with $\mathbf{E}[\mathsf{val}(V)] = \mu$. Let $t \geqslant 1$ be an integer. We want to compare two distributions on $\{\pm 1\}^t$:

- The distribution obtained by sampling t vertices $v_1, \ldots, v_t$ uniformly and independently at random, and outputting the ordered tuple $(\mathsf{val}(v_1), \ldots, \mathsf{val}(v_t))$. Note that this is the same distribution as sampling a sequence of t elements in $\{\pm 1\}$ independently from a μ-biased distribution, that is a distribution with expectation μ. We denote this distribution by U_t^μ.

- $\mathrm{RW}_{G,\mathsf{val}}$ is the distribution obtained by sampling a random length $t-1$ path $v_1,\ldots,v_t$ in G and outputting the ordered tuple $(\mathsf{val}(v_1),\ldots,\mathsf{val}(v_t))$. Equivalently, sample v_1 uniformly at random from V. Then, for $i=2,3,\ldots,t$, sample v_i uniformly at random from the neighbours of v_{i-1}.

Let $f:\{\pm1\}^t\to\{\pm1\}$ be a test function. Expand f in the μ-biased Fourier basis,

$$f(x)=\sum_{S\subseteq[t]}\widehat{f_\mu}(S)\chi_S^\mu(x).$$

▶ **Lemma 4.** *Let $G=(V,E)$ be a regular λ-spectral expander, and let $\mathsf{val}:V\to\{\pm1\}$ be a labelling of the vertices of G with $\mathbf{E}[\mathsf{val}(V)]=\mu$. Then, for every function $f:\{\pm1\}^t\to\mathbb{R}$,*

$$\mathcal{E}_{G,\mathsf{val}}(f)\leqslant\sum_{\substack{S\subseteq T\\ S\neq\emptyset}}|\widehat{f_\mu}(S)|\mathcal{E}_{G,\mathsf{val}}(\chi_S^\mu).$$

Proof. Since $\mathbf{E}[\mathsf{val}]=\mu$, for $S\neq\emptyset$, $\mathbf{E}[\chi_S^\mu(U_t^\mu)]=0$ and thus $\mathbf{E}[f(U_t^\mu)]=\widehat{f_\mu}(\emptyset)$. Hence,

$$\mathcal{E}_{G,\mathsf{val}}(f)=|\mathbf{E}\,f(\mathrm{RW}_{G,\mathsf{val}})-\mathbf{E}\,f(U_t^\mu)|=\Big|\sum_{\substack{S\subseteq T\\ S\neq\emptyset}}\widehat{f_\mu}(S)\,\mathbf{E}[\chi_S^\mu(\mathrm{RW}_{G,\mathsf{val}})]\Big|.$$

For $S\neq\emptyset$, $\mathcal{E}_{G,\mathsf{val}}(\chi_S^\mu)=|\,\mathbf{E}[\chi_S^\mu(\mathrm{RW}_{G,\mathsf{val}})]|$. The proof follows by the triangle inequality. ◀

Lemma 4 motivates us to consider parity test functions which we do next. We start by introducing some notation. For an integer $k\geqslant 2$, we define the family $\mathcal{F}_k$ of subsets of $[k-1]$ that, informally, consists of all subsets for which at least one of every two consecutive elements participate in the set. We also require the "end points" $1,k-1$ to participate in the set. Formally, we define

$$\mathcal{F}_k=\{I\subseteq[k-1]\mid\{1,k-1\}\subseteq I\ \text{ and }\ \forall j\in[k-2]\ \ \{j,j+1\}\cap I\neq\emptyset\}\,. \tag{3.1}$$

So, for example, $\mathcal{F}_6$ consists of the elements $\{1,3,5\}$, $\{1,2,4,5\}$ as well as of all subsets of $[5]$ that have as a subset any one of these two elements, namely, $\{1,2,3,5\}$, $\{1,3,4,5\}$ and $\{1,2,3,4,5\}$. We extend the definition in the natural way to $k=0,1$ by setting $\mathcal{F}_0=\mathcal{F}_1=\emptyset$.

▶ **Definition 5.** *For integers $t\geqslant 1$, $2\leqslant k\leqslant t$ and $j\in[k-2]$ define the map*

$$\Delta_j:\binom{[t]}{k}\to\mathbb{N}$$

as follows. Let $S\subseteq[t]$ of size $k\geqslant 2$ and denote $S=\{s_1,\ldots,s_k\}$ where $s_1<\cdots<s_k$. For $i\in[k-1]$ write $\delta_i=s_{i+1}-s_i$. Define

$$\Delta_j(S)=\min(\delta_j,\delta_{j+1}).$$

▶ **Definition 6.** *For an integer $t\geqslant 1$ define the map $\Delta:\binom{[t]}{\geqslant 2}\to\mathbb{N}$ as follows. Let $S\subseteq[t]$ of size $k\geqslant 2$. For $k=2$ we define $\Delta(S)=\Delta_1(S)$, and for $k\geqslant 3$,*

$$\Delta(S)=\sum_{i=1}^{k-2}\Delta_i(S). \tag{3.2}$$

We prove:

▶ **Proposition 7.** *Let $G = (V, E)$ be a regular λ-spectral expander and $\mathsf{val} : V \to \{\pm 1\}$ a labelling of the vertices of G with $\mathbf{E}[\mathsf{val}(V)] = \mu$. Then, for every $1 \leqslant k \leqslant t$ and $S \subseteq [t]$ of size k,*

$$\mathcal{E}_{G,\mathsf{val}}(\chi_S^\mu) \leqslant \left(\frac{1+|\mu|}{1-|\mu|}\right)^{\frac{k-1}{2}} \cdot \sum_{I \in \mathcal{F}_k} \lambda^{\sum_{j\in I} \Delta_j(S)} \leqslant \left(\frac{1+|\mu|}{1-|\mu|}\right)^{\frac{k-1}{2}} 2^k \cdot \lambda^{\Delta(S)/2}.$$

We remark that for sets of size $|S| = 1$ the sum is taken over the empty index set $\mathcal{F}_1$ and so equals 0. We also note that when $|\mu| = 1$ the error is trivially zero, while our bound tends to infinity.

Proof. Consider any non-empty subset $S \subseteq [t]$ of size $|S| = k$. As $\mathbf{E}[\chi_S(U_t^\mu)] = 0$ we have that

$$\mathcal{E}_{G,\mathsf{val}}(\chi_S^\mu) = \left|\mathbf{E}[\chi_S^\mu(\mathrm{RW}_{G,\mathsf{val}})]\right|.$$

We wish to express the right hand side algebraically. Let $n = |V|$ and identify V with $[n]$ in an arbitrary way. Let P be the $n \times n$ diagonal matrix with

$$P_{v,v} = \frac{\mathsf{val}(v) - \mu}{\sqrt{1-\mu^2}}$$

for every $v \in [n]$. We slightly abuse notation and denote the random walk matrix (that is, the normalized adjacency matrix) of G also by G. Define $\delta_S(i) = 1$ if $i \in S$ and $\delta_S(i) = 0$ otherwise and observe that

$$\mathbf{E}[\chi_S^\mu(\mathrm{RW}_{G,\mathsf{val}})] = \mathbf{1}^T \left(\prod_{i=1}^{t} P^{\delta_S(i)} G\right) \mathbf{1},$$

where, recall, $\mathbf{1}$ is the all one vector normalized by $\frac{1}{\sqrt{n}}$. Indeed, informally, at the i'th step we take a random step using G and then, depending on i being an element of I or not, we multiply by P or by I, respectively. Thus, we can write

$$\mathbf{E}[\chi_S^\mu(\mathrm{RW}_{G,\mathsf{val}})] = \mathbf{1}^T G^{t-s_k} \left(\prod_{i=1}^{k-1} P G^{\Delta_i}\right) P G^{s_1} \mathbf{1} = \mathbf{1}^T \left(\prod_{i=1}^{k-1} P G^{\Delta_i}\right) P \mathbf{1}, \tag{3.3}$$

where we have used the regularity of G, namely, $G\mathbf{1} = \mathbf{1}$.

Next, we use the spectral decomposition of G. As G is a λ-spectral expander we know that $G = \mathbf{J} + \lambda E$ where $\|E\| \leqslant 1$. Similarly, As G^ℓ is a λ^ℓ-spectral expander we have that $G^\ell = \mathbf{J} + \lambda^\ell E_\ell$ for some operator E_ℓ with bounded norm $\|E_\ell\| \leqslant 1$. Thus,

$$\prod_{i=1}^{k-1} P G^{\Delta_i} = \sum_{I \subseteq [k-1]} \prod_{i=1}^{k-1} P B_i(I), \tag{3.4}$$

where

$$B_i(I) = \begin{cases} \lambda^{\Delta_i} E_{\Delta_i} & i \in I; \\ \mathbf{J} & \text{otherwise.} \end{cases}$$

For $I \subseteq [k-1]$ let

$$e_I = \mathbf{1}^T \left(\prod_{i=1}^{k-1} P B_i(I)\right) P \mathbf{1}.$$

Equations (3.3) and (3.4) imply that

$$\mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}})] = \sum_{I\subseteq[k-1]} e_I. \tag{3.5}$$

Not all subsets $I \subseteq [k-1]$ contribute non-zero values e_I to the sum. Indeed, if $k-1 \notin I$ then $B_{k-1}(I) = \mathbf{J}$ and so

$$\begin{aligned} e_I &= \mathbf{1}^T \left(\prod_{i=1}^{k-2} PB_i(I)\right)(P\mathbf{J})P\mathbf{1} \;= \mathbf{1}^T \left(\prod_{i=1}^{k-2} PB_i(I)\right)(P\mathbf{1}\mathbf{1}^T)P\mathbf{1} \\ &= \mathbf{1}^T \left(\prod_{i=1}^{k-2} PB_i(I)\right) P\mathbf{1}(\mathbf{1}^T P\mathbf{1}) = 0, \end{aligned}$$

because

$$\mathbf{1}^T P\mathbf{1} = \frac{1}{\sqrt{1-\mu^2}} \cdot \sum_{i\in[n]} \frac{\mathsf{val}(i)-\mu}{n} = \frac{\mathbf{E}[\mathsf{val}(V)]-\mu}{\sqrt{1-\mu^2}} = 0.$$

Similarly $e_I = 0$ for I not containing 1. Moreover, if $j, j+1$ are both not contained in I for some $j \in [k-2]$ then

$$\begin{aligned} e_I &= \mathbf{1}^T \left(\prod_{i=1}^{j-1} PB_i(I)\right)(PB_j(I))(PB_{j+1}(I)) \left(\prod_{i=j+2}^{k-2} PB_i(I)\right) P\mathbf{1} \\ &= \mathbf{1}^T \left(\prod_{i=1}^{j-1} PB_i(I)\right)(P\mathbf{J})(P\mathbf{J}) \left(\prod_{i=j+2}^{k-2} PB_i(I)\right) P\mathbf{1} = 0, \end{aligned}$$

Because

$$(P\mathbf{J})(P\mathbf{J}) = (P\mathbf{1}\mathbf{1}^T)(P\mathbf{1}\mathbf{1}^T) = P\mathbf{1}(\mathbf{1}^T P\mathbf{1})\mathbf{1}^T = 0.$$

Thus, any subset $I \subseteq [k-1]$ that may contribute to the sum in Equation (3.5) is contained in $\mathcal{F}_k$ as defined in Equation (3.1).

Next, we look at $I \in \mathcal{F}_k$. We have that

$$e_I = \mathbf{1}^T \left(\prod_{i=1}^{k-1} PB_i(I)\right) P\mathbf{1} \;\leqslant \prod_{i=1}^{k-1} \|PB_i(I)\| \;\leqslant \|P\|^{k-1} \prod_{i\in I} \|B_i(I)\|. \tag{3.6}$$

Recall that for every $i \in I$, $B_i(I) = \lambda^{\Delta_i} E_{\Delta_i}$ and that $\|E_{\Delta_i}\| \leqslant 1$. Thus, $\prod_{i\in I} \|B_i(I)\| \leqslant \prod_{i\in I} \lambda^{\Delta_i}$. Also, Let M be the $n \times n$ diagonal matrix defined by $M_{v,v} = \mathsf{val}(v)$ for all $v \in [n]$. Note that $P = \frac{1}{\sqrt{1-\mu^2}}(M - \mu I)$. As $\|M\| = 1$, using the triangle inequality we get

$$\|P\| \leqslant \frac{\|M\| + \|\mu I\|}{\sqrt{1-\mu^2}} \leqslant \frac{1+|\mu|}{\sqrt{1-\mu^2}} = \sqrt{\frac{1+|\mu|}{1-|\mu|}}. \tag{3.7}$$

Equation (3.6) and Equation (3.7) together imply that $e_I \leqslant \left(\frac{1+|\mu|}{1-|\mu|}\right)^{\frac{k-1}{2}} \prod_{i\in I} \lambda^{\Delta_i}$. This proves the first inequality in the proposition.

To prove the second inequality consider $I \in \mathcal{F}_k$, and notice that

$$2\sum_{i\in I} \Delta_i \geqslant \sum_{i=1}^{k-2} \delta_i\Delta_i + \delta_{i+1}\Delta_{i+1} \geqslant \sum_{i=1}^{k-2} \min(\Delta_i, \Delta_{i+1}),$$

because for every $i \in [k-2]$, at least one of $i, i+1$ is in I. To complete the proof of the second inequality notice also that $|\mathcal{F}_k| \leqslant 2^{k-1}$. ◀

4 Choosing the graph

In this section we choose an expander graph for which we obtain a precise analytic formula for the expectation of characters under the input distribution given by the random walk.

▷ Claim 8. Let $G = ([n], E)$ be a regular graph with second largest eigenvalue λ_2 and corresponding eigenvector v_2. Further assume all coordinates in v_2 have ± 1 values. Define $\mathsf{val}_2 : [n] \to \{\pm 1\}$ by $\mathsf{val}_2(i) = v_2(i)$. Let $S \subseteq [n]$, $|S| = k$. Let P be the diagonal matrix corresponding to val_2, that is, $P_{i,i} = \mathsf{val}_2(i) = v_2(i)$. Then,

$$\left(\prod_{i=1}^{k-1} PG^{\Delta_i}\right) P\mathbb{1} = \begin{cases} \lambda^{\sum_{i=1}^{(k-2)/2} \Delta_{2i+1}} \mathbb{1} & k \in \mathbb{N}_{even}, \\ \lambda^{\sum_{i=1}^{(k-1)/2} \Delta_{2i+1}} v_2 & k \in \mathbb{N}_{odd}. \end{cases}$$

Proof. We will prove the claim by induction. For the base case $k = 1$ it holds that $\prod_{i=1}^{k-1} PG^{\Delta_i} = I$, and the statement follows as $IP\mathbb{1} = v_2 = \lambda^0 v_2$. For the induction step, note that

$$\left(\prod_{i=1}^{k} PG^{\Delta_i}\right) P\mathbb{1} = PG^{\Delta_k} \left(\prod_{i=1}^{k-1} PG^{\Delta_i}\right) P\mathbb{1}.$$

If $k \in \mathbb{N}_{even}$ than $k-1 \in \mathbb{N}_{odd}$ and, using the induction hypothesis we get that

$$PG^{\Delta_k} \left(\prod_{i=1}^{k-1} PG^{\Delta_i}\right) P\mathbb{1} = PG^{\Delta_k} \lambda^{\sum_{i=1}^{(k-2)/2} \Delta_{2i+1}} v_2 \ = \lambda^{\sum_{i=1}^{k/2} \Delta_{2i+1}} \mathbb{1},$$

which is what we wanted to prove. The proof in the case that $k \in \mathbb{N}_{odd}$ is similar. ◁

▶ **Definition 9.** *For $S \subseteq [t]$ denote $\Delta_{odd}(S) = \sum_{i=1}^{\lfloor (|S|-1)/2 \rfloor} \Delta_{2i+1}(S)$.*

▶ **Corollary 10.** *Let $G = ([n], E)$ and $\mathsf{val}_2 : [n] \to \{\pm 1\}$ be as above. Then,*

$$\mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}_2})] = \begin{cases} \lambda^{\Delta_{odd}(S)} & |S| \in \mathbb{N}_{even}, \\ 0 & |S| \in \mathbb{N}_{odd}. \end{cases}$$

Proof. Note that $Pv_2 = \mathbb{1}$ and $P\mathbb{1} = v_2$. As before, it holds that

$$\mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}_2})] = \mathbf{1}^T \left(\prod_{i=1}^{k-1} PG^{\Delta_i}\right) P\mathbf{1} = \frac{1}{n} \mathbb{1}^{\mathsf{T}} \left(\prod_{i=1}^{k-1} PG^{\Delta_i}\right) P\mathbb{1}.$$

Using Claim 8, we conclude that,

$$\mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}_2})] = \begin{cases} \lambda^{\Delta_{odd}(S)} \cdot \frac{1}{n} \mathbb{1}^{\mathsf{T}} \mathbb{1} & k \in \mathbb{N}_{even}, \\ \lambda^{\sum_{i=1}^{k-1/2} \Delta_{2i+1}} \cdot \frac{1}{n} \mathbb{1}^{\mathsf{T}} v_2 & k \in \mathbb{N}_{odd}. \end{cases}$$

The fact that G is regular implies that $\mathbb{1}^{\mathsf{T}} v_2 = 0$, which finishes the case that k is odd; the case that k is even is handled similarly by noting that $\mathbb{1}^{\mathsf{T}} \mathbb{1} = n$. ◀

We now give an example to such a graph G. Cayley graphs over an Abelian group commute and share an orthonormal basis of eigenvectors, which is known to be the set of all characters of the group. If the group is $\mathbb{Z}_2^n$, the eigenvectors have entries that are 2nd roots of unity, i.e., have ± 1 entries as desired. The eigenvalues have a direct correspondence to the set of generators of the Cayley graph. Building on that, [4] proved that for every $0 < \lambda < 1$ of the form $\frac{1}{m}$ for $m \in \mathbb{N}$ and $m \leqslant n \in \mathbb{N}$, there is a Cayley graph on the n dimensional boolean cube, with $\lambda_2 = \lambda$. The degree of this graph depends both on n and λ.

From now on we let G be a regular expander with second largest eigenvalue λ and corresponding eigenvector with ± 1 entries, and we let val_2 reflect that eigenvector.

5 A lower bound for symmetric functions

In this section we prove the following theorem.

▶ **Theorem 11.** *Let $0 < c_0 \leqslant 1$, and let G, val_2 be as in the previous section, for $0 < \lambda < \frac{c_0^2}{12800 \cdot e}$. Let $f\colon \{\pm 1\}^t \to \{\pm 1\}$ be a symmetric function with $\left|\widehat{f}(2)\right| \geqslant \frac{c_0}{\sqrt{\binom{t}{2}}}$. Then,*

$$\mathcal{E}_{G,\mathsf{val}_2}(f) \geqslant 0.001 c_0 \lambda.$$

The idea behind the proof is to show that when choosing G, val_2 as in Section 4, the upper bound given by [11] is tight (up to the redundant poly logarithmic factor). We will use the following claim from [11].

▶ **Lemma 12** ([11], Lemma 4.4)**.** *Denote*

$$\beta_k = \sum_{\substack{S \subseteq [t] \\ |S|=k}} \mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}})]. \quad \textit{Then,} \quad \beta_k \leqslant 2^k \binom{t-1}{\lfloor \frac{k}{2} \rfloor} \left(\frac{\lambda}{1-\lambda}\right)^{\lceil \frac{k}{2} \rceil}. \tag{5.1}$$

Using these notations we are now ready to prove Theorem 11.

Proof of Theorem 11. Denote by $\mathcal{B}_2 = \{\{i, i+1\} \mid i \in [t-1]\}$, note that $|\mathcal{B}_2| = t-1$ and that for every $S \in \mathcal{B}_2$ it holds that $\Delta_{odd}(S) = 1$. Recall that $\mathcal{E}_{G,\mathsf{val}_2}(\chi_S) = 0$ if $|S| = 1$, therefore,

$$\mathcal{E}_{G,\mathsf{val}_2}(f) = \left| \sum_{S \subseteq [t], |S| \geqslant 2} \widehat{f}(|S|) \, \mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}})] \right| \tag{5.2}$$

$$\geqslant \left|\widehat{f}(2)\right| \left| \sum_{S \subseteq [t], |S|=2} \mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}})] \right| - \left| \sum_{S \subseteq [t], |S|>2} \widehat{f}(|S|) \mathcal{E}_{G,\mathsf{val}_2}(\chi_S) \right|. \tag{5.3}$$

However, by Corollary 10,

$$\left|\widehat{f}(2)\right| \left| \sum_{S \subseteq [t], |S|=2} \mathbf{E}[\chi_S(\mathrm{RW}_{G,\mathsf{val}})] \right| \geqslant \left|\widehat{f}(2)\right| \sum_{S \in \mathcal{B}_2} \lambda \; \geqslant c_0 \sqrt{2} \sqrt{\frac{t-1}{t}} \lambda \; \geqslant \frac{c_0}{\sqrt{2}} \lambda.$$

Furthermore,

$$\left| \sum_{S \subseteq [t], |S|>2} \widehat{f}(S) \mathcal{E}_{G,\mathsf{val}_2}(\chi_S) \right| \leqslant \sum_{k \geqslant 3} \left|\widehat{f}(k)\right| \beta_k \leqslant \sum_{k \geqslant 3} \frac{1}{\sqrt{\binom{t}{k}}} 2^k \binom{t-1}{\lfloor \frac{k}{2} \rfloor} \left(\frac{\lambda}{1-\lambda}\right)^{\lceil \frac{k}{2} \rceil},$$

where in the last inequality we used Lemma 12. The right hand side of the above equation is bounded above by

$$\sum_{k \geqslant 3} (16e)^{k/2} \lambda^{k/2} \leqslant 124 \lambda^{1.5}.$$

We omit the calculations. Assume that $\lambda \leqslant \frac{c_0^2}{128e \cdot 100}$. Then Equation (5.2) yields

$$\mathcal{E}_{G,\mathsf{val}_2}(f) \geqslant \frac{c_0}{\sqrt{2}} \lambda - 124 \lambda^{1.5} \geqslant 0.04 c_0 \lambda. \qquad \blacktriangleleft$$

In order to prove Theorem 1, we are left with providing a function f that satisfies the conditions of Theorem 11. Next, we show that the threshold function at one standard deviation distance from the mean has non-vanishing error in t.

We use the following definitions and claim. For integers t and $w \in \{0,1,\ldots,t\}$ let $\mathbf{1}_w : \{\pm 1\}^t \to \{0,1\}$ be the function indicating whether the weight of the input is w. That is, $\mathbf{1}_w(x_1,\ldots,x_t) = 1$ if $|\{i \in [t] \mid x_i = 1\}| = w$ and $\mathbf{1}_w(x_1,\ldots,x_t) = 0$ otherwise. We also define $\mathbf{1}_{>w} : \{\pm 1\}^{t+1} \to \{0,1\}$ be the function indicating whether the weight of the input is greater w. That is, $\mathbf{1}_w(x_1,\ldots,x_t) = 1$ if $\sum_i x_i > w$ and $\mathbf{1}_w(x_1,\ldots,x_t) = 0$ otherwise.

▷ Claim 13. For every $S \subseteq [t]$, it holds that

$$\widehat{(\mathbf{1}_w)_\mu}(S) = \frac{\widehat{(\mathbf{1}_{>w})_\mu}(S \cup \{0\})}{\sqrt{1-\mu^2}}.$$

Proof.

$$\begin{aligned}
\mathbf{1}_w(x_1,\ldots,x_t) &= \mathbf{1}_{>w}(1,x_1,\ldots,x_t) - \mathbf{1}_{>w}(0,x_1,\ldots,x_t) \\
&= \sum_{S\subseteq\{0,\ldots,t\}} \widehat{(\mathbf{1}_{>w})_\mu}(S)\chi^\mu_S(1,x_1,\ldots,x_t) - \sum_{S\subseteq\{0,\ldots,t\}} \widehat{(\mathbf{1}_{>w})_\mu}(S)\chi^\mu_S(0,x_1,\ldots,x_t) \\
&= \sum_{S\subseteq\{0,\ldots,t\}} \widehat{(\mathbf{1}_{>w})_\mu}(S)(\chi^\mu_S(1,x_1,\ldots,x_t) - \chi^\mu_S(0,x_1,\ldots,x_t)) \\
&\quad \sum_{\substack{S\subseteq\{0,\ldots,t\}\\ 0\in S}} \widehat{(\mathbf{1}_{>w})_\mu}(S)\frac{1}{\sqrt{1-\mu^2}}\chi^\mu_{S\setminus\{0\}}(x_1,\ldots,x_t),
\end{aligned}$$

and the claim follows. ◁

Proof of Theorem 1. Take $f = \mathbf{1}_{>w}$ for $w = \frac{t-\sqrt{t}}{2}$. We claim that $\left|\widehat{f}(2)\right| > \frac{c_0}{\sqrt{\binom{t}{2}}}$, for some absolute constant $c_0 > 0$ and therefore by Theorem 11, $\mathcal{E}_{G,\mathsf{val}}(f_t) \geqslant c \cdot \lambda$ for some constant c. Indeed, by Claim 13 we have $\widehat{f}(2) = \widehat{\mathbf{1}_w}(1)$ for $\mathbf{1}_w : \{\pm 1\}^{t-1} \to \{0,1\}$. To compute $\widehat{\mathbf{1}_w}(1)$ we apply [11, Claim 4.9] for $w = \frac{t-\sqrt{t}}{2}$ and get

$$\left|\widehat{\mathbf{1}_w}(1)\right| = \left|\frac{1}{2^{t-1}}\frac{\binom{t-1}{w}}{\binom{t-1}{1}}\sum_{\ell=0}^{\lfloor\frac{1}{2}\rfloor}(-1)^{1-\ell}\binom{w}{\ell}\binom{t-2w-1}{1-2\ell}\right| = \frac{1}{2^{t-1}}\frac{\binom{t-1}{w}}{t-1}(t-1-2w).$$

Substituting $w = \frac{t-\sqrt{t}}{2}$, together with the fact that $\binom{t-1}{\frac{t-\sqrt{t}}{2}} \geqslant \Omega\left(\frac{1}{\sqrt{t}}2^t\right)$, concludes the proof. ◀

6 A lower bound for AC^0 tests

In this section we use the noise operator. The following definitions and claims appear in [27].

▶ **Definition 14.** *Let $\rho \in [-1,1]$. For a fixed $x \in \{\pm 1\}^t$ we write $y \sim N_\rho(x)$ to denote the random string y that is drawn as follows: for each $i \in [t]$ independently,*

$$y_i = \begin{cases} x_i & \text{with probability } \frac{1+\rho}{2}, \\ -x_i & \text{with probability } \frac{1-\rho}{2}. \end{cases}$$

▶ **Definition 15.** *Let $\rho \in [-1,1]$. The noise operator T_ρ is the linear operator on functions $\{\pm 1\}^t \to \mathbb{R}$ defined by $T_\rho f(x) = \mathbf{E}_{y \sim N_\rho(x)} f(y)$. The fact that the operator is linear follows directly from the linearity of the expectation.*

Notice that $T_1(f) = f$ whereas $T_0(f)$ is the constant function $T_0(f) = \mathbf{E}\, f$. We make use of the following lemma.

▶ **Lemma 16.** *For every function $f : \{\pm 1\}^t \to \mathbb{R}$ it holds that: $\widehat{T_\rho f}(S) = \widehat{f}(S)\rho^{|S|}$.*

The starting point of this section is to connect the expectation of f under a random walk and the noise function $T_\lambda(f)$, we prove this claim in the full version of the paper.

▷ Claim 17. For $f\colon \{\pm 1\}^t \to \mathbb{R}$ define $\tilde{f}\colon \{\pm 1\}^{2t} \to \mathbb{R}$ by

$$\tilde{f}(x_1, x_2, \ldots, x_{2t-1}, x_{2t}) = f(x_1 \cdot x_2, \ldots, x_{2t-1} \cdot x_{2t}).$$

Then, $\mathbf{E}[\tilde{f}(\mathrm{RW}_{G,\mathsf{val}_2})] = (T_\lambda f)(\mathbb{1})$.

Proof. For $\{s_1, \ldots, s_k\} = S \subseteq [t]$ denote $2S\colon = \{2s_1 - 1, 2s_1, \ldots, 2s_k - 1, 2s_k\} \subseteq [2t]$. Note that $\Delta_{odd}(2S) = |S|$.

$$\begin{aligned}
\tilde{f}(x_1, x_2, \ldots, x_{2t-1}, x_{2t}) &= f(x_1 \cdot x_2, \ldots, x_{2t-1} \cdot x_{2t}) \\
&= \sum_{S \subseteq [t]} \widehat{f}(S) \chi_S(x_1 \cdot x_2, \ldots, x_{2t-1} \cdot x_{2t}) \\
&= \sum_{S \subseteq [t]} \widehat{f}(S) \chi_{2S}(x_1, x_2, \ldots, x_{2t-1}, x_{2t}).
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbf{E}[\tilde{f}(\mathrm{RW}_{G,\mathsf{val}_2})] &= \sum_{S \subseteq [t]} \widehat{f}(S)\, \mathbf{E}[\chi_{2S}(\mathrm{RW}_{G,\mathsf{val}_2})] \\
&= \sum_{S \subseteq [t]} \widehat{f}(S) \lambda^{|S|} \\
&= \sum_{S \subseteq [t]} \widehat{f}(S) \lambda^{|S|} \chi_S(\mathbb{1}),
\end{aligned}$$

which is equal to $T_\lambda(f)(\mathbb{1})$ by Lemma 16. For the second equality we used Corollary 10. ◁

6.1 A lower bound for the Tribes function composed with IP

We now construct a function in $\mathsf{AC}(3)$, that satisfies Theorem 2. Later on we extend the construction inductively to obtain the general theorem. The idea behind the depth-3 construction is the following. We look for a function $f = f(x_1, \ldots, x_t) \in \mathsf{AC}(2)$ such that

$$|\, \mathbf{E}[f(U_t)] - T_\lambda(f)(\mathbb{1})| \geqslant \lambda \cdot \log t. \tag{6.1}$$

We then look at $\tilde{f}(y_1, \ldots, y_{2t}) = f(y_1 \cdot y_2, \ldots, y_{2t-1} \cdot y_{2t}) \in \mathsf{AC}(3)$ and note that:

- $\mathbf{E}[\tilde{f}(U_t)] = \mathbf{E}[f(U_t)]$ as the product of two uniform ± 1 bits is uniform; However,
- by Claim 17, $\mathbf{E}[\tilde{f}(\mathrm{RW}_{G,\mathsf{val}_2})] = T_\lambda(f)(\mathbb{1})$.

Together,

$$\mathcal{E}_\lambda(\tilde{f}) \geqslant \mathcal{E}_{G,\mathsf{val}_2}(\tilde{f}) = |\,\mathbf{E}[\tilde{f}(U_t)] - \mathbf{E}[\tilde{f}(\mathrm{RW}_{G,\mathsf{val}_2})]| = |\,\mathbf{E}[f(U_t)] - T_\lambda(f)(\mathbf{1})] \geqslant \lambda \cdot \log t,$$

which in turns implies Theorem 2, for $d = 3$.

We take f to be the Tribes function. Fix t; we choose parameters r, h such that $r \cdot h \leqslant t$ by taking $h = \log(t) - \log\log(t)$[2] and $r = \lfloor \frac{t}{\log t} \ln(2) \rfloor$. Partition $[t]$ into disjoint sets $I_1, \ldots, I_r$, each of size h. We define $f : \{\pm 1\}^t \to \{0,1\}$ to be the Tribes function on t bits and define g to be the related function

$$f(z_1, \ldots, z_t) = \bigvee_{i \in [r]} \bigwedge_{j \in I_i} z_j, \qquad g(z_1, \ldots, z_t) = \bigwedge_{i \in [r]} \bigvee_{j \in I_i} z_j.$$

Here, -1 is interpreted as "true", 1 is interpreted as "false". Note that $f, g \in \mathsf{AC}(2)$.

As before, we choose G to be a Cayley graph on the boolean hypercube with $\lambda_2 = \lambda$ and $\mathsf{val} = \mathsf{val}_2$.

▷ Claim 18. The functions f and g are almost balanced with respect to the uniform distribution. Quantitatively, $\mathbf{E}[f], \mathbf{E}[g] \in \left[\frac{1}{2} - O\left(\frac{\log t}{t}\right), \frac{1}{2} + O\left(\frac{\log t}{t}\right)\right]$.

Proof. From De Morgan's identity we have $g(x_1, \ldots, x_t) = 1 - f(\overline{x_1}, \ldots, \overline{x_t})$, so $\mathbf{E}[g] = 1 - \mathbf{E}[f]$ and so it is enough to prove the statement for f. To this end write

$$\begin{aligned}\mathbf{E}[f] = \mathbf{Pr}[f = 1] &= 1 - \prod_{i=1}^{r} \mathbf{Pr}\left[\bigwedge_{j \in I_i} z_j = 0\right] \\ &= 1 - \prod_{i=1}^{r} \left(1 - \mathbf{Pr}\left[\bigwedge_{j \in I_i} z_j = 1\right]\right) = 1 - \left(1 - \frac{1}{2^h}\right)^r.\end{aligned}$$

Using the fact that $1 - \varepsilon = e^{-\varepsilon + O(\varepsilon^2)}$ we obtain that

$$\begin{aligned}1 - \left(1 - \frac{1}{2^h}\right)^r &= 1 - e^{-2^{-h} r + O(2^{-2h} r)} \\ &= 1 - e^{-\ln 2 + O\left(\frac{\log t}{t}\right)} = \frac{1}{2} + \Theta\left(\frac{\log t}{t}\right),\end{aligned}$$

as desired. ◁

Denote by μ_p the product distribution over $\{\pm 1\}^t$, wherein for each $i \in [t]$ we have that $\mathbf{Pr}[z_i = -1] = p$. Abusing notation denote $\mu_p(f) = \mathbf{E}_{x \sim \mu_p}[f(x)]$.

▷ Claim 19. Let $p = \frac{1-\varepsilon}{2}$ and assume $\varepsilon \geqslant \frac{k}{\log(t)}$. Then,

$$\mu_p(f), \mu_p(g) \leqslant e^{-k/10}.$$

Proof. First, we analyze $\mu_p(f)$. By definition it is equal to

$$\begin{aligned}\mathop{\mathbf{Pr}}_{\mu_p}[f = 1] &= 1 - (1 - p^h)^r = 1 - (1 - 2^{-h}(1 - \varepsilon)^h)^r \\ &= 1 - \left(1 - 2^{-h}\left(1 - \frac{k}{\log t}\right)^h\right)^r \leqslant 1 - \left(1 - 2^{-h} e^{-k}\right)^r.\end{aligned}$$

[2] Recall that $\log t = \log_2 t$

Using $(1-\delta)^r \geqslant 1 - r\delta$, we get that the above expression is bounded by $r2^{-h}e^{-k} \leqslant e^{-k}$. Next, we upper bound $\mu_p(g)$. By definition, it is equal to

$$\mathop{\mathbf{Pr}}_{\mu_p}[g=1] \leqslant (1-(1-p)^h)^r = (1-2^{-h}(1+\varepsilon)^h)^r = \left(1-2^{-h}\left(1+\frac{k}{\log t}\right)^h\right)^r.$$

Using $(1+\delta)^r \geqslant \delta r$ for $\delta > 0$, we get that this is at most

$$(1-2^{-h}k)^r \leqslant e^{-r2^{-h}k} \leqslant e^{-k/10}. \qquad \lhd$$

We now prove Theorem 2 for $d=3$. We take $h(x_1, y_1, \ldots, x_t, y_t) = f(x_1 \cdot y_1, \ldots, x_t \cdot y_t)$. $h \in \mathsf{AC}(3)$ because $f \in \mathsf{AC}(2)$.

- On the one hand, by Claim 17, $\mathbf{E}[h(\mathrm{RW}_{G,\mathsf{val}_2})] = T_\lambda(f)(\mathbb{1}) = \mu_{\frac{1-\lambda}{2}}(f)$. By Claim 19, and using $\lambda \geqslant \frac{k}{\log t}$, we get that $\mathbf{E}[h(\mathrm{RW}_{G,\mathsf{val}_2})] < e^{-k/10}$.
- On the other hand, By Claim 18, $\mathbf{E}[h] = \mathbf{E}[f] \geqslant \frac{1}{2} - O(\frac{\log t}{t})$.

Together, h is as desired.

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in logspace. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 132–140, 1987.

2 Noga Alon and Fan RK Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72(1-3):15–19, 1988.

3 Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 512–519. IEEE, 1995.

4 Noga Alon and Yuval Roichman. Random cayley graphs and expanders. *Random Struct. Algorithms*, 5(2):271–285, 1994. doi:10.1002/rsa.3240050203.

5 Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Randomness in interactive proofs. *Computational Complexity*, 3(4):319–354, 1993.

6 Avraham Ben-Aroya and Amnon Ta-Shma. A combinatorial construction of almost-Ramanujan graphs using the zig-zag product. *SIAM J. Comput.*, 40(2):267–290, 2011. doi:10.1137/080732651.

7 Yonatan Bilu and Nathan Linial. Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica*, 26(5):495–519, 2006. doi:10.1007/s00493-006-0029-7.

8 Mark Braverman. Polylogarithmic independence fools AC^0 circuits. *J. ACM*, 57(5):Art. 28, 10, 2010. doi:10.1145/1754399.1754401.

9 Mark Braverman, Gil Cohen, and Sumegha Garg. Pseudorandom pseudo-distributions with near-optimal error for read-once branching programs. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1197734.

10 Aviad Cohen and Avi Wigderson. Dispersers, deterministic amplification, and weak random sources. In *30th Annual Symposium on Foundations of Computer Science*, pages 14–19. IEEE Computer Society, 1989.

11 Gil Cohen, Noam Peri, and Amnon Ta-Shma. Expander random walks: a fourier-analytic approach. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1643–1655. ACM, 2021. doi:10.1145/3406325.3451049.

12 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):Art. 12, 44, 2007. doi:10.1145/1236457.1236459.

13 Irit Dinur, Shai Evra, Ron Livne, Alexander Lubotzky, and Shahar Mozes. Locally testable codes with constant rate, distance, and locality. *CoRR*, abs/2111.04808, 2021. arXiv:2111.04808.

14 David Gillman. A chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, 1998.

15 Venkatesan Guruswami and Vinayak M Kumar. Pseudobinomiality of the sticky random walk. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

16 Alexander D Healy. Randomness-efficient sampling within nc. *Computational Complexity*, 17(1):3–37, 2008.

17 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S.)*, 43(4):439–561, 2006. doi:10.1090/S0273-0979-06-01126-8.

18 Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 356–364, 1994.

19 Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *FOCS*, volume 89, pages 248–253, 1989.

20 Claude Kipnis and SR Srinivasa Varadhan. Central limit theorem for additive functionals of reversible markov processes and applications to simple exclusions. *Communications in Mathematical Physics*, 104(1):1–19, 1986.

21 Benoît Kloeckner. Effective limit theorems for Markov chains with a spectral gap. *arXiv preprint*, 2017. arXiv:1703.09623.

22 Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *J. ACM*, 64(2):Art. 11, 42, 2017. doi:10.1145/3051093.

23 Pascal Lezaud. Chernoff and Berry-Esséen inequalities for markov processes. *ESAIM: Probability and Statistics*, 5:183–201, 2001.

24 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

25 Grigorii Aleksandrovich Margulis. Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problemy peredachi informatsii*, 24(1):51–60, 1988.

26 Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. Explicit near-ramanujan graphs of every degree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 510–523, 2020.

27 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. doi:10.1017/CBO9781139814782.

28 Ran Raz, Omer Reingold, and Salil Vadhan. Error reduction for extractors. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 191–201. IEEE, 1999.

29 Omer Reingold. Undirected ST-connectivity in log-space. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 376–385. ACM, New York, 2005. doi:10.1145/1060590.1060647.

30 Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 3–13. IEEE, 2000.

31 Eyal Rozenman and Salil Vadhan. Derandomized squaring of graphs. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 436–447. Springer, 2005.

32 Michael Sipser and Daniel A Spielman. Expander codes. *IEEE transactions on Information Theory*, 42(6):1710–1722, 1996.

33 Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *STOC'17—Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 238–251. ACM, New York, 2017. doi:10.1145/3055399.3055408.

34 Avishay Tal. Tight bounds on the fourier spectrum of ac0. In *32nd Computational Complexity Conference (CCC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

35 Luca Trevisan. Lecture notes on graph partitioning, expanders and spectral methods. *University of California, Berkeley*, 2017. URL: https://lucatrevisan.github.io/books/expanders-2016.pdf.

36 Salil P Vadhan. *Pseudorandomness*, volume 7. Now, 2012.

37 Leslie G. Valiant. Graph-theoretic properties in computational complexity. *J. Comput. System Sci.*, 13(3):278–285, 1976. doi:10.1016/S0022-0000(76)80041-4.

LCC and LDC: Tailor-Made Distance Amplification and a Refined Separation

Gil Cohen
Department of Computer Science, Tel Aviv University, Israel

Tal Yankovitz
Department of Computer Science, Tel Aviv University, Israel

Abstract

The Alon-Edmonds-Luby distance amplification procedure (FOCS 1995) is an algorithm that transforms a code with vanishing distance to a code with constant distance. AEL was invoked by Kopparty, Meir, Ron-Zewi, and Saraf (J. ACM 2017) for obtaining their state-of-the-art LDC, LCC and LTC. Cohen and Yankovitz (CCC 2021) devised a procedure that can amplify inverse-polynomial distances, exponentially extending the regime of distances that can be amplified by AEL. However, the improved procedure only works for LDC and assuming rate $1 - \frac{1}{\text{poly}\log n}$.

In this work we devise a distance amplification procedure for LCC with inverse-polynomial distances even for vanishing rate $\frac{1}{\text{poly}\log\log n}$. For LDC, we obtain a more modest improvement and require rate $1 - \frac{1}{\text{poly}\log\log n}$. Thus, the tables have turned and it is now LCC that can be better amplified. Our key idea for accomplishing this, deviating from prior work, is to tailor the distance amplification procedure to the code at hand.

Our second result concerns the relation between linear LDC and LCC. We prove the existence of linear LDC that are not LCC, qualitatively extending a separation by Kaufman and Viderman (RANDOM 2010).

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Locally Correctable Codes, Locally Decodable Codes, Distance Amplifications

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.44

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2021/136/` [6]

Funding The research leading to these results has received funding from the ERC starting grant 949499, the Israel Science Foundation grant 1569/18 and from the Azrieli Faculty Fellowship.

1 Introduction

1.1 Distance amplification

It is a recurrent theme in coding theory that the construction of a code is done in two steps. In the first step, a code with weak parameters is constructed, and typically it is the distance of the code that is unsatisfactory. In the second step, one transforms the code obtained in the first step to a code with the desired parameters, where typically, in the process, the other parameters deteriorate only slightly. When the distance is the unsatisfactory parameter, the second step is referred to as a distance amplification step.

Examples that fall into this framework include the breakthrough constructions of near-optimal small-bias sets by Ta-Shma [19], and the state-of-the-art construction of locally decodable codes (LDC), locally correctable codes (LCC), and locally testable codes (LTC) by Kopparty, Meir, Ron-Zewi, and Saraf [17]. A prominent example from the (adjacent) PCP literature is Dinur's celebrated proof of the PCP Theorem by gap amplification [8]. It is interesting to note that in all the above cases the first step is done using algebraic machinery whereas the second step is based on combinatorial arguments.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 44; pp. 44:1–44:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.2 LDC and LCC

Informally, a linear (q, δ) *locally decodable code (LDC)* is a code, given by an $\mathbb{F}$-linear encoding function $\mathsf{Enc} : \mathbb{F}^k \to \mathbb{F}^n$, where $\mathbb{F}$ is a finite field, that is also equipped with a "local decoder". The latter is a randomized algorithm, denoted by Dec, with the following guarantee. Given an oracle access to $z \in \mathbb{F}^n$ that is within relative Hamming distance δ from some codeword $\mathsf{Enc}(x)$, and given $i \in [k]$, $\mathsf{Dec}^z(i) = x_i$ with high probability. Moreover, Dec makes at most q queries to z. That is, every message symbol can be decoded, with high probability, by querying only few symbols of a corrupted codeword. A (q, δ) *locally correctable code (LCC)* is the variant in which one wishes to decode (or, more precisely, correct) the codeword symbols rather than the message symbols.

Locally decodable codes were defined by Katz and Trevisan [15] who proved that asymptotically good LDC require $q = \Omega(\log n)$ queries. Whether or not this bound is tight is a major open problem. An intensive research effort is devoted to the study and construction of LDC and LCC. Of particular interest is the study of asymptotically good LDC and LCC [18, 13, 14, 17, 12, 7] where the goal is to minimize the query complexity.

In their seminal work, Kopparty, Meir, Ron-Zewi and Saraf [17] constructed LDC and LCC with sub-polynomial query complexity. For the first step, a code with vanishing distance $\delta = \frac{1}{\mathrm{poly}(\log n)}$ was used [18], having the desired query complexity, namely, $q = 2^{\tilde{O}(\sqrt{\log n})}$. Then, in the second step the authors invoked a distance amplification procedure due to Alon, Edmonds and Luby [2, 1], which was originally introduced in the context of linear-time erasure codes, and observed that it converts an LDC (resp. LCC) with distance δ and query complexity q to an LDC (resp. LCC) with constant distance and query complexity $q_{\mathrm{new}} = q \cdot \mathrm{poly}(\frac{1}{\delta})$.

1.3 Improved distance amplification for LDC

Motivated by the key role that the distance amplification procedure plays in [17], Cohen and Yankovitz [7] asked whether much lower distances can be amplified. Indeed, AEL's procedure is mostly relevant in the regime $\delta = \frac{1}{\mathrm{poly}(\log n)}$. In [7], the authors devised an improved procedure that can amplify distances as low as $\frac{1}{n^\alpha}$ for any constant $\alpha < 1$ with a fairly low cost in query complexity, $q_{\mathrm{new}} = q^{O(\log \log n)}$ [1] (and even for $\alpha = 1 - o(1)$ at a small additional cost in query complexity). However, their improved distance amplification procedure has two drawbacks:

1. Unlike the AEL distance amplification procedure, the improved procedure was only shown to work for LDC (and it may or may not work for LCC).
2. Second, to amplify the distance, the original LDC must have rate close to one, more precisely, rate $1 - \frac{1}{\mathrm{poly}(\log n)}$.

2 Our contribution

We turn to present the two results of this work.

[1] $\mathrm{poly}(\log \log n)$ factors in the exponent of the query complexity can be safely ignored given that, at present, the lowest known query complexity is $2^{\tilde{\Theta}(\sqrt{\log n})}$. Such an overlook will matter only when (and if) the query complexity will go below quasi-poly-logarithmic.

2.1 Tailor-made distance amplification procedure

Our first contribution is a distance amplification procedure for LCC that can amplify distances as low as those handled by [7] (for LDC). Moreover, our procedure works even for vanishing rate LCC.

▶ **Theorem 1** (Distance amplification for LCC; informal). *Let $h \geqslant 1 \geqslant \alpha > 0$ be any constants. There exists a transformation that takes a q-query LCC with distance $\frac{1}{n^{\alpha}}$ and rate $\frac{1}{(\log\log n)^h}$ to an asymptotically good LCC with query complexity*

$$q_{\mathsf{new}} = q^{O((\log\log n)^{2h+2})}.$$

We chose to state our result in a somewhat informal manner. For the formal statement, see Corollary 37.

An example usage of Theorem 1 is given by the next corollary. The corollary shows the implication of a case that an LCC with query complexity meeting the Katz-Trevisan bound is shown to exist - only with a vanishing rate and distance.

▶ **Corollary 2** (Informal). *If there exists a q-query LCC for $q = \log n$, with distance $\frac{1}{\sqrt{n}}$ and rate $\frac{1}{\log\log n}$, then there exists an asymptotically good LCC with query complexity*

$$q_{\mathsf{new}} = (\log n)^{O((\log\log n)^4)}.$$

We now turn to give further details on the result.

Explicitness

In the statement of Theorem 1 we ignore the issue of explicitness. Indeed, understanding LDC and LCC is already interesting in the information-theoretic level. Having said that, our transformation is fairly explicit: It is a zero error randomized transformation that runs in polynomial-time. More precisely, for every "failure" parameter $\varepsilon > 0$, our transformation runs in time $\mathrm{poly}(n) \cdot \log \frac{1}{\varepsilon}$ and produces an LCC with probability at least $1 - \varepsilon$; otherwise, it declares failure. We find this aspect to be a minor issue as, recall, LCC are anyhow randomized in nature. Nonetheless, it will be interesting to obtain a deterministic transformation with matching parameters.

Codes vs. family of codes

A second issue that we chose to sweep under the rug in the statement of Theorem 1 is that the transformation operates on the level of family of codes rather than on the level of individual codes. That is, in order to produce an asymptotically good LCC of a given block-length n, our transformation requires as input a sufficiently dense family of codes. By that we mean that the consecutive block-lengths in the family are not too far apart. The density of the resulted family of codes is the same as that of the original family.

Amplifying lower distances

Like [7], we can even amplify sub-polynomial distances, in particular, distances of the form $1/n^{1-1/g(n)}$ for an increasing function g, and assuming a certain technical relation between g and the rate. In particular, for every constant $m \geqslant 1$ we can handle $g(n) = (\log\log n)^m$, and end up with query complexity

$$q_{\mathsf{new}} = q^{O((\log\log n)^{2h+2m+2})}.$$

We note that constructing a code for $g(n) = \log n$ is trivial.

Amplifying the distance of LDC

We also obtain an improvement for LDC by devising a distance amplification procedure that requires rate $1 - \frac{1}{\text{poly}(\log\log n)}$, modestly improving upon the $1 - \frac{1}{\text{poly}(\log n)}$ rate required by [7]. The reason that we can do much better for LCC is due to the rate amplification procedure of [7] that, informally, can amplify rate ρ LCC with q queries to constant rate LCC with query complexity $q_{\mathsf{new}} = q^{\text{poly}(\frac{1}{\rho})}$. Such a transformation is not known for LDC.

2.1.1 Proof idea

In this section we give a short and informal account on our proof technique, and start by contrasting our technique with prior work. Both the AEL distance amplification procedure, as was used in [17], and the one given by [7] are based on samplers and further involve a "small" code, that is, a code with logarithmic block-length. The latter improves upon the former by using unbalanced samplers (rather than balanced ones, or expander graphs as was used originally [1, 2]) and using a recursive construction. To obtain our result, we deviate from prior work and tailor the distance amplification procedure to the LCC at hand. That is, our procedure is "white box" - it produces a new code with improved distance by first examining the structure of the given code. To tailor the procedure to the LCC at hand, we do not work directly with the definition of LCC as it lacks sufficient structure to work with. Instead, we work with a more combinatorial characterization of LCC as was used in [7]. We turn to elaborate on this.

Let $C \subseteq \mathbb{F}^n$ be a linear (q, δ)-LCC. One can prove the following structural result. With every coordinate $i \in [n]$ one can associate a set, called a *query set*, $A_i = \{Q_1^i, \ldots, Q_m^i\}$ of $m = \delta n/q$ disjoint subsets of $[n]$, each of size at most q, such that the following holds: For every $c \in C$ and $t \in [m]$, c_i can be deduced from $c_{Q_t^i}$. Assume from here on, for simplicity, that $\delta = 1/\sqrt{n}$ and so $m = \sqrt{n}/q$. Denote $\bar{A}_i = \bigcup_{t=1}^m Q_t^i$ and note that $|\bar{A}_i| \leqslant \sqrt{n}$.

For our distance amplification procedure, we make use of a special partition π of $[n]$ into $\sqrt{n}$ parts $P_1, \ldots, P_{\sqrt{n}}$, each of size $\sqrt{n}$. We say that such a partition is a *d-splitter* for C (more precisely, a d-splitter for the query sets $A_1, \ldots, A_n$ obtained from C) if for every $s \in [\sqrt{n}]$ and $i \in [n]$, $|P_s \cap \bar{A}_i| \leqslant d$. We wish to minimize d and thus consider a max load balls into bins like problem: For every $i \in [n]$ we place a ball with color i at each of the coordinates in $\bar{A}_i$. Note that a coordinate $j \in [n]$ may contain many balls of different colors. Indeed, the average number of balls at coordinate $j \in [n]$ is $\sqrt{n}$. Our goal is to choose the partition π in such a way that every part P_t will contain at most d balls of the same color. It is easy to show that a d-splitter for C exists with $d = O(\frac{\log n}{\log\log n})$.

We construct a new code $C' \subseteq \mathbb{F}^n$ as follows. We take C' to be the code $C' \subseteq C$ with the property that for every part P_s of π, when C' is projected to the coordinate set P_s, the obtained vectors consist of codewords of a code $C_{\sqrt{n}}$ having block length $\sqrt{n}$, which is a q'-query LCC. That is to say, we require that for every $c \in C'$ and $s \in [\sqrt{n}]$, $c_{P_s} \in C_{\sqrt{n}}$. Observe that C' can be constructed by adjoining to the parity checks of C, the parity checks of $C_{\sqrt{n}}$ when restricted to each block in π.

We show that if $C_{\sqrt{n}}$ is a smooth LCC, which means that it queries each coordinate with roughly the same probability, then so is C'. Moreover, C' has query complexity qq'. Thus, C can be transformed into a smooth LCC of length n given that a smooth LCC of length $\sqrt{n}$ is at hand. This calls for a recursive construction which results with a smooth LCC with query complexity $q^{O(\log\log n)}$. After obtaining a smooth code, the final step is to invoke the AEL distance amplification to end up with a good LCC. This final step has a minor effect on the query complexity.

The above recursive construction must start with LCC of rate $1 - \frac{1}{\mathrm{poly}(\log\log n)}$. This is due to the rate deterioration throughout the $\log\log n$ recursive calls. For amplifying rate $\frac{1}{\mathrm{poly}(\log\log n)}$ LCC, as stated in Theorem 1, we invoke the rate amplification procedure of [7] before running the recursive construction described above. This has some effect on the density of the LCC family that the recursion has access to which requires some care.

2.2 Refined separation between LDC and LCC

Understanding the relation between LDC and LCC is fundamental. Currently the only regime in which the state of affairs is better understood is the 2-query regime [3, 4, 5]. In the constant-query regime for $q \geqslant 3$, q-LDC with sub-exponential length are known [20, 11, 9] whereas it is not known if this can be matched for q-LCC. Recall that in the constant-rate regime, the state of the art result of [17] achieves sub-polynomial query complexity and holds for LDC and LCC alike.

In the general case, clearly, a systematic LCC is an LDC. As every linear code can be made systematic (by applying Gaussian elimination to its generating matrix), a linear LCC induces a linear LDC with the same parameters. Thus, informally, LCC are stronger than LDC, at least for linear codes.

Are LDC and LCC "equivalent"?

As for the converse, Kaufman and Viderman [16] observed that an LDC is not necessarily an LCC. Their proof starts with an LDC. If it is not an LCC to begin with, we are done. If it is an LCC, the proof goes on by transforming it to a new code by appending to it one additional entry that does not involve low-weight constraints (namely, every vector in the dual code that does not vanish on the new entry is of large weight). In this way, one obtains an LDC with an entry that cannot be corrected with few queries. Such an entry can be shown to exist by a counting argument. This argument can be extended to produce many new bits that cannot be corrected.

While, formally, the argument above establishes the existence of LDC that are not LCC, it has a drawback which makes it somewhat less appealing. In the resulted code, the adjoined bits that cannot be corrected are not needed for decoding the original bits. This means that if one is given a code that is not an LCC because of the above transformation, with the task of taking such a code and "convert" it to an LCC, this could be done easily: simply by removing these coordinates, and this clearly would not harm the code's dimension. This raises the question: Can any linear LDC be so "easily" converted to an LCC of similar dimension and query complexity?

The thought that the answer to this question may turn out to be in the affirmative is not far fetched in the case of linear codes. Indeed, we know that the locality features of linear codes "come from" linear relations between different bits of the codeword and of the message. For example, if a linear code $\mathsf{Enc} : \mathbb{F}^k \to \mathbb{F}^n$ is a q-query LDC, and in particular the i-th bit of each message m can be deduced from a subset $Q \subseteq [n]$ that consists of at most q coordinates of $c = \mathsf{Enc}(m)$, then there exists a linear map $f_{i,Q}$ which satisfies $m_i = f_{i,Q}(c_Q)$ for any m. Likewise, if m_i can as well be deduced from another subset $Q' \subseteq [n]$, $|Q'| \leqslant q$ (as is expected due to the distance guarantee), then there is a linear map $f_{i,Q'}$ satisfying $m_i = f_{i,Q'}(c_{Q'})$ for every m. It follows that in such a case, for every codeword c, $f_{i,Q}(c_Q) = f_{i,Q'}(c_{Q'})$. Since $f_{i,Q}$ and $f_{i,Q'}$ are linear maps (that, we may assume, depend on all their parameters) this means that for every $j \in Q \triangle Q'$, there exists a linear map g_j satisfying $c_j = g_j(c_{Q_j})$ for every codeword c, where $Q_j = (Q \cup Q')\backslash\{j\}$.

Therefore, by the mere fact that $j \in [n]$ is sometimes used in the local decoding process of $i \in [k]$, it is implied that it is possible to "correct" the j-th coordinate by reading only a few locations of the codeword (at most $2q-1$). Thus, the question of whether local decoding implies local correction is in place, in the case of linear codes, and especially so in the setting where k is close to n.

In light of this, the fact that in the separating result of [16] between linear LDC and LCC, the coordinates which are shown to be uncorrectable are not used by the local decoding process, calls for the question of whether there exists a linear LDC with uncorrectable coordinates that are crucial for the decoding process.

Our result

The second contribution of this work is a proof for the existence of an LDC that is not an LCC in the following stronger sense: It contains entries that cannot be corrected which are crucial for the local decoder. This raises the question of what we mean by coordinates that are "crucial". The mere fact that it is possible for a set of coordinates to be queried by the local decoding process should not qualify them as such, as what allows for a code to be locally decodable or locally correctable is that there are many options to decode or correct each symbol. Thus, a more suitable interpretation for a "crucial" set of coordinates $J \subseteq [n]$ is the following: If every coordinate $j \in J$ is "zeroed out" from the code (i.e., for every codeword c we override c_j with zero) then the transformed code is no longer locally decodable. With this we are ready to present our separation.

▶ **Theorem 3** (Separation of LDC and LCC; Informal)**.** *Let $C : \mathbb{F}^k \to \mathbb{F}^n$ for $|\mathbb{F}| > 2$ and $k = \Theta(n)$ be a linear q-query LDC. Then, there exists a linear q^2-query LDC $\widehat{C} : \mathbb{F}^{k^2} \to \mathbb{F}^{n^2}$ with the following property. There exists a subset of coordinates $J \subseteq [n^2]$ in which every coordinate cannot be locally corrected with query complexity $\sqrt{n}$ and correction radius $1/\sqrt{n}$. Moreover, if every coordinate $j \in J$ is zeroed out from the code, then the relative distance of the obtained code is $\widetilde{O}(1/\sqrt{n})$ (and so it is certainly not an LDC).*

For the formal, more general, statement, see Theorem 46. Note that our result does not cover the binary field and it is an interesting question whether it can be extended to include that case.

Proof idea

The underlying idea of the proof of Theorem 3 is an operation on two codes to which we call *weighted tensoring*. The weighted tensoring of codes is similar to the standard tensoring of codes. In the case of standard tensoring, the encoding of the tensor of two codes is done by taking a matrix as input and applying the first code to each column and then applying the second code to each row in the resulted matrix. In the encoding of a weighted tesnor, before the second step, each entry of the matrix is multiplied by a non-zero field element, or weight.

We consider the case of weighted tensoring which is done with *random* weights. We show that while the code resulted from this is an LDC (assuming that the two input codes were so), with high probability there is a set of coordinates in the code that cannot be locally corrected, while being crucial for the decoding. The analysis showing that the set of coordinates cannot be locally corrected is done by considering the affect of the weights on the dual code. A probabilistic argument is then used to show that the argued codes exist.

Discussion

We end this section with a short discussion to clarify a potentially confusing point. While LCC are, in a sense, more powerful than LDC (indeed, our second contribution, Theorem 3, attempts to formalize that better), our first result, given by Theorem 1, transforms a vanishing rate LCC with polynomially-small distance to an asymptotically good LCC–a result that is not known for LDC. So, how can it be that we can do this for LCC and not for the weaker LDC?

Of course, this should cause no confusion as the latter is a *transformation* that works for LCC and not LDC, not a *construction* nor it is even a proof of existence. Put differently, although the transformation generates the stronger object, the transformation is also given it as its input.

3 Preliminaries

3.1 Notations and conventions

Unless stated otherwise, all logarithms are taken to the base 2. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \ldots, n\}$. For ease of readability, we sometimes avoid the use of floor and ceiling. This does not affect the stated results. We use $\mathbb{F}$ to denote a field, and any referenced field is assumed to be finite and of a constant size. When n and $\mathbb{F}$ are clear from context, we use $e_i \in \mathbb{F}^n$ to denote the i-th vector of the standard basis. For $q \in \mathbb{N}$, we use H_q to denote the q-ary entropy function, and H to denote the binary entropy function. For a vector $v \in \mathbb{F}^n$, we denote by $|v|$ the *hamming weight* of v, which is the number of its non-zero coordinates $|v| = |\{j \in [n] \mid v_j \neq 0\}|$, and the *support* of v is $\mathsf{supp}(v) = \{j \in [n] \mid v_j \neq 0\}$. For two vectors $u, v \in \mathbb{F}^n$, we denote their (absolute) hamming distance by $\mathsf{dist}(u, v)$. For a linear subspace $L \subseteq \mathbb{F}^n$, we denote by $L^{\leqslant q}$ the set of vectors of weight at most q. For two vector $u, v \in \mathbb{F}^n$, we use $\langle u, v \rangle$ to denote the inner product of u and v, $\sum_{i=1}^{n} u_i v_i \in \mathbb{F}$. For a vector $v \in \mathbb{F}^n$ and a sequence $I = (i_1, \ldots, i_m) \in [n]^m$, we denote by v_I the vector $(v_{i_1}, \ldots, v_{i_m}) \in \mathbb{F}^m$. For a linear subspace $L \subseteq \mathbb{F}^n$ and a sequence $I = (i_1, \ldots, i_m) \in [n]^m$, we denote by L_I the subspace $\{v_I \mid v \in L\}$. Note that L_I is indeed a subspace as it is given by a suitable projection.

A *partition* π of size k of $[n]$ is a set $\{P_1, \ldots, P_k\}$ of disjoint subsets of $[n]$, such that $P_1 \cup \cdots \cup P_k = [n]$. A partition $\{P_1, \ldots, P_k\}$ is *ordered* if each P_i is a sequence rather than a set (and the sequences, when viewed as sets, satisfy the same requirements). Throughout this paper, any partition of $[n]$ will be an ordered partition (though we may not state it explicitly) with the sequences defined by the natural increasing order of $\mathbb{N}$.

3.2 Error correcting codes

We start by recalling the definition of an error correcting code, and of a family of error correcting codes. In this work we only consider linear codes.

▶ **Definition 4.** *For $n \in \mathbb{N}$ and $\mathbb{F}$ a field, a* code *of length n over $\mathbb{F}$ is a linear subspace $C \subseteq \mathbb{F}^n$.*[2] *The dimension of the code, denoted by k, is the dimension of C over $\mathbb{F}$, $\dim_{\mathbb{F}} C$. The (non-local) distance of the code, denoted by d, is $\min_{c \in C, c \neq 0} |c|$. The rate of the code, denoted by ρ, is k/n. The (non-local) relative distance of the code, denoted by Δ, is d/n. The elements of C are called* codewords.

[2] We may omit the phrase "over $\mathbb{F}$" if the underlying field is clear from context.

We will also need to consider encodings of codes.

▶ **Definition 5.** *We call a function* $\mathsf{Enc} : \mathbb{F}^k \to \mathbb{F}^n$ *an* encoding *of a code* C *if it is an injective linear map and* $C = \mathsf{Im}(\mathsf{Enc})$.

▶ **Definition 6.** *For a field* $\mathbb{F}$*, a* code family *over* $\mathbb{F}$ *is a set of codes* $C = \{C^n\}$*, which contains at most one code* C^n *of length* n *over* $\mathbb{F}$*, for every possible length* $n \in \mathbb{N}$*. For every* $n \in \mathbb{N}$*, we denote by* $\lceil n \rceil^C$ *the minimal length of a code in the family* C *of length at least* n*, and by* $\lfloor n \rfloor^C$ *the maximal length of a code in the family of length at most* n*. For constants* $n_0 \in \mathbb{N}$*,* $c \geqslant 1$ *and* $d \leqslant 1$*, we say that the family is* (n_0, c, d)-dense *if for every* $n \geqslant n_0$*,* $\lceil n \rceil^C \leqslant cn$ *and* $\lfloor n \rfloor^C \geqslant dn$*.*

▶ **Definition 7.** *For a field* $\mathbb{F}$*, a* code-encoding family *over* $\mathbb{F}$ *is a set of pairs of codes and corresponding encodings* $C = \{(C^k, \mathsf{Enc}^k)\}$*, which contains at most one code* C^k *of dimension* k *over* $\mathbb{F}$*, for every possible dimension* $k \in \mathbb{N}$*. For every* $k \in \mathbb{N}$*, we denote by* $\lceil k \rceil^C$ *the minimal dimension of a code in the family* C *of dimension at least* k*, and by* $\lfloor k \rfloor^C$ *the maximal dimension of a code in the family of dimension at most* k*. For constants* $k_0 \in \mathbb{N}$*,* $c \geqslant 1$ *and* $d \leqslant 1$*, we say that the family is* (k_0, c, d)-dense *if for every* $k \geqslant k_0$*,* $\lceil k \rceil^C \leqslant ck$ *and* $\lfloor k \rfloor^C \geqslant dk$*.*

▶ **Definition 8.** *Let* C *be a code of length* n *over* $\mathbb{F}$*. The* dual code of C *is defined to be its orthogonal subspace* $C^\perp$*.*

▶ **Definition 9.** *Let* C *be a code of length* n *over* $\mathbb{F}$*, let* $i \in [n]$ *and* $B \subseteq [n]$*. We say that* B determines i in C *if there exists a function* $f : \mathbb{F}^{|B|} \to \mathbb{F}$ *such that for every* $c \in C$*,* $c_i = f(c_B)$*.*

We also need the following property of linear codes.

▶ **Fact 10.** *Let* C *be a code of length* n *over* $\mathbb{F}$*. Further let* $i \in [n]$*,* $Q \subseteq [n]$ *and* $x \in \mathbb{F}^{|Q|}$*. Then, one of the following cases must hold.*

1. *There is at most one* $\alpha \in \mathbb{F}$ *for which there exists some* $c \in C$ *satisfying* $c_Q = x$ *and* $c_i = \alpha$*.*
2. *For every* $\alpha \in \mathbb{F}$ *there is an equal number of* $c \in C$ *for which* $c_i = \alpha$*.*

In particular, either no function (even randomized) of c_Q *can predict* c_i *with probability larger than* $1/|\mathbb{F}|$*, when* $c \in C$ *is randomly chosen uniformly, or* c_Q *determines* c_i *for all* $c \in C$*.*

3.3 Locally decodable codes and locally correctable codes

▶ **Definition 11.** *For* $C \subseteq \mathbb{F}^n$*, we say that a procedure* $f : A \to B$ *is with* oracle access to $c \in C$ *if when* f *is run, it gets besides an input* $a \in A$*, access to* $c \in C$*:* f *can query* c_i *for indices* $i \in [n]$*. To describe a specific run of* f *with input* $a \in A$ *and oracle access to* $c \in C$*, we either say that* $f(a)$ is run with oracle access to c*, or write* $f^c(a)$ *for short. We say that* f *is* non-adaptive *if the queries it makes are independent of* $c \in C$*.*

▶ **Definition 12.** *For a code* C *of length* n *and dimension* k *over* $\mathbb{F}$*, and* Enc *and encoding of it,* (C, Enc) *is called a* (q, δ, ε)-LDC *(locally decodable code, abbreviated) if there exists a randomized procedure* $\mathsf{Dec} : [k] \to \mathbb{F}$ *that is given an oracle access to* $z \in \mathbb{F}^n$*, and has the following guarantee. For every* $i \in [k]$*,* $x \in \mathbb{F}^k$ *and* $z \in \mathbb{F}^n$ *satisfying* $\mathsf{dist}(z, \mathsf{Enc}(x)) \leqslant \delta n$*,* $\mathsf{Dec}^z(i) = x_i$ *with probability at least* $1 - \varepsilon$*. Furthermore,* $\mathsf{Dec}^z(i)$ *always makes at most* q *queries to* z*. We further require that* Dec *is non-adaptive. We call* Dec *a* local decoder *(or* decoder*) for* (C, Enc)*, and the parameter* q *is called the* query complexity *of* (C, Enc)*.*

▶ **Definition 13.** *A code-encoding family $C = \{(C^k, \mathsf{Enc}^k)\}$ of codes over $\mathbb{F}$ is called* a family of good $q(k)$-LDC*, or a* a family of good LDC with query complexity $q(k)$*, if every code C^k in the family is a code with rate at least $\rho(k)$, which is a $(q(k), \delta(k), \varepsilon(k))$-LDC, for $\rho(k) = \Omega(1)$, $\delta(k) = \Omega(1)$, and $\varepsilon(k) \leqslant 1/3$.*

We have the following easy fact.

▶ **Fact 14.** *If C is a code of length n and dimension $k > 0$ over $\mathbb{F}$ and Enc is an encoding of it, and if (C, Enc) is a (q, δ, ε)-LDC, then, provided that $\varepsilon < 1 - 1/|\mathbb{F}|$, the (non-local) relative distance of C, Δ, satisfies $\Delta > \delta$.*[3]

▶ **Definition 15.** *A code C of length n over $\mathbb{F}$ is called a* (q, δ, ε)-LCC *(locally correctable code, abbreviated) if there exists a randomized procedure $\mathsf{Cor} : [n] \to \mathbb{F}$ that is given an oracle access to $z \in \mathbb{F}^n$, and has the following guarantee. For every $i \in [n]$, $y \in C$ and $z \in \mathbb{F}^n$ satisfying $\mathsf{dist}(z, y) \leqslant \delta n$, $\mathsf{Cor}^z(i) = y_i$ with probability at least $1 - \varepsilon$. Furthermore, $\mathsf{Cor}^z(i)$ always makes at most q queries to z. We further require that Cor is non-adaptive and that $\mathsf{Cor}(i)$ never queries i*[4]*. We call Cor a* local corrector *(or* corrector*) for C, and the parameter q is called the* query complexity *of C.*

▶ **Definition 16.** *For a code C of length n over $\mathbb{F}$ (not necessarily a (q, δ, ε)-LCC), and $i \in [n]$, we say that i is a* (δ, q, ε)-correctable coordinate *in C if there exists a procedure $\mathsf{Cor} : [n] \to \mathbb{F}$ such that $\mathsf{Cor}(i)$ satisfies the requirements in Definition 15.*

▶ **Definition 17.** *A family $C = \{C^n\}$ of codes over $\mathbb{F}$ is called* a family of good $q(n)$-LCC*, or a* a family of good LCC with query complexity $q(n)$*, if every code C^n in the family is a code with rate at least $\rho(n)$, which is a $(q(n), \delta(n), \varepsilon(n))$-LCC, for $\rho(n) = \Omega(1)$, $\delta(n) = \Omega(1)$, and $\varepsilon(n) \leqslant 1/3$.*

The following well-known fact is an implication of the fact that every linear code has a systematic encoding[5].

▶ **Fact 18.** *If a code C is a (q, δ, ε)-LCC, then there exists an encoding Enc such that (C, Enc) is a (q, δ, ε)-LDC.*

4 Tailor made distance amplification

4.1 Characterization of LCC

In this section, we will need to use two characterizations of LCC, as was given by Definition 15. The first, given next in Definition 19, is of a (q, τ)-LCC, and resembles the definition of smooth codes given by [15] for LDC. A (q, τ)-LCC differs from a (q, δ, ε)-LCC in that its local correction is only required to succeed if it is given a codeword of the code, rather than a possible corrupted codeword. Accordingly, the correction of a (q, τ)-LCC has no "distance" guarantee, but instead it is required not to query any coordinate with too high probability, i.e., probability larger than τ. When we will construct an LCC, it will be easier to first argue that it is a (q, τ)-LCC and use that to show it can be made into a (q, δ, ε)-LCC for any ε and $\delta = \varepsilon/(\tau n)$.

[3] Note that in the case that $\varepsilon < 1/2$ a stronger bound $\Delta > 2\delta$ holds.

[4] The assumption that $\mathsf{Cor}(i)$ never queries i is only for simplicity. Any LCC which defies this assumption can be easily converted to one which does not, with a negligible effect on δ.

[5] An encoding Enc is a *systematic* encoding if for some $f : [k] \to [n]$, for all $x \in \mathbb{F}^k$ and $i \in [k]$, $\mathsf{Enc}(x)_{f(i)} = x_i$.

The second characterization, which will be given in Definition 23, is of what we call a (q,τ)-*query-set LCC*. Informally, a code is (q,τ)-*query-set LCC* if for every coordinate we have a large enough set of disjoint subsets of $[n]$, from which it can be decoded. The distance amplification procedure that we define utilizes these query sets and so the properties of the input code that we will use are that of its characterization as a (q,τ)-query-set LCC. This is, in a sense, a more "combinatorial" characterization of LCC which can be more conveniently used when a manipulation of these objects is needed.

The three characterizations of LCC all imply each other, but some of the transitions are at some cost to the parameters. Indeed, Claim 20 will show that a (q,τ)-LCC is a (q,δ,ε)-LCC for $\delta=\varepsilon/(\tau n)$, Claim 24 will show that a (q,τ)-query-set is a (q,τ)-LCC, and Claim 25 will complete the cycle and show that a (q,δ,ε)-LCC is a (q,τ)-query-set LCC for $\tau=q/(\delta n)$.

▶ **Definition 19.** *A code C of length n over $\mathbb{F}$ is called a (q,τ)-LCC if there exists a randomized procedure $\mathsf{Cor}:[n]\to\mathbb{F}$ that is given an oracle access to $c\in C$, and has the following guarantee. For every $i\in[n]$ and $c\in C$, $\mathsf{Cor}^c(i)=c_i$, with probability 1. Furthermore, $\mathsf{Cor}^c(i)$ always makes at most q queries to c, and for every $j\in[n]$, the probability that c_j is queried by $\mathsf{Cor}^c(i)$ is at most τ. We further require that Cor is non-adaptive and that $\mathsf{Cor}(i)$ never queries i. We call the parameter q* the query complexity *and the parameter τ the* smoothness *of the LCC.*

▷ Claim 20. Let C be a code of length n which is a (q,τ)-LCC. Then, for any $\varepsilon>0$, C is a (q,δ,ε)-LCC with $\delta=\varepsilon/(\tau n)$.

Proof. Let $\varepsilon>0$ and let Cor be a corrector of C. Let $c\in C$ and $z\in\mathbb{F}^n$ such that $\mathsf{dist}(c,z)\leqslant\delta n=\varepsilon/\tau$, and set $B=\{j\in[n]\mid z_j\neq c_j\}$. Fix $i\in[n]$. By the union bound over $j\in B$, except with probability ε, when $\mathsf{Cor}(i)$ is run with oracle access to $c\in C$, it does not make a query to an index in B. If this is the case, then if Cor was given access to z instead of c, it would successfully output c_i, as well. Thus, C is indeed a (q,δ,ε)-LCC as the same corrector Cor can be used with oracle access to strings $z\in\mathbb{F}^n$, and given that $\mathsf{dist}(c,z)\leqslant\delta n$, $\mathsf{Cor}(i)$ is promised to output c_i with probability at least $1-\varepsilon$. ◁

▶ **Definition 21.** *A set $\mathcal{A}=\{A_1,\ldots,A_n\}$ is called an* n-query-set *if for every $i\in[n]$, A_i is a set of disjoint subsets of $[n]\backslash\{i\}$. For every $i\in[n]$ we define $\overline{A_i}=\bigcup_{B\in A_i}B$.*

▶ **Definition 22.** *Let C be a code of length n and let $\mathcal{A}=\{A_1,\ldots,A_n\}$ be an n-query-set. $\mathcal{A}$ is said to be a* query-set for C *if for every $i\in[n]$ and $B\in A_i$, B determines i in C (see Definition 9).*

▶ **Definition 23.** *Let C be a code of length n. C is said to be a* (q,τ)-query-set-LCC *if there exists a set $\mathcal{A}=\{A_1,\ldots,A_n\}$ which is a query-set for C, such that for every $i\in[n]$, $|A_i|\geqslant 1/\tau$ and for every $B\in A_i$, $|B|\leqslant q$.*

▷ Claim 24. Let C be a code of length n over $\mathbb{F}$ which is a (q,τ)-query-set LCC. Then C is a (q,τ)-LCC.

Proof. Let $\mathcal{A}=\{A_1,\ldots,A_n\}$ be a query set that corresponds to C being a (q,τ)-query-set LCC. The following corrector Cor shows that C is a (q,τ)-LCC. Given $i\in[n]$, and oracle access to $c\in C$, $\mathsf{Cor}(i)$ samples uniformly at random some $B\in A_i$ and queries c_B. As B determines i in C, there exists a function f satisfying $f(c_B)=c_i$ for every $c\in C$, and so $\mathsf{Cor}(i)$ uses such a function and outputs its result. Thus, for every $c\in C$, the output of $\mathsf{Cor}(i)$ is always equal to c_i, and note that as any sampled $B\in A_i$ satisfies $|B|\leqslant q$, $\mathsf{Cor}(i)$ always makes at most q queries. Since A_i is of size at least $1/\tau$ and is composed of disjoint subsets of $[n]\backslash\{i\}$, any coordinate is queried by $\mathsf{Cor}(i)$ with probability at most τ, and $\mathsf{Cor}(i)$ never queries i. Thus, C is a (q,τ)-LCC. ◁

▷ Claim 25. Let C be a code of length n over $\mathbb{F}$ which is a (q, δ, ε)-LCC, for $\varepsilon < 1 - 1/|\mathbb{F}|$. Then, C is a (q, τ)-query-set-LCC for $\tau = q/(\delta n)$.

The proof for the claim is similar to the proof in [15] to their Theorem 1 and to the proof in [10] for Theorem 1.1.

Proof for Claim 25. To prove the claim, we need to show that there exists a set $\mathcal{A} = \{A_1, \ldots, A_n\}$ which is a query-set for C, such that for every $i \in [n]$, $|A_i| \geqslant 1/\tau = \delta n/q$ and for every $B \in A_i$, $|B| \leqslant q$. We construct $\mathcal{A}$ with the required properties by constructing each of the subsets separately. Let Cor denote a corrector promised by the fact that C is a (q, δ, ε)-LCC, and let $i \in [n]$. To construct A_i, we construct a sequence of disjoint sets $B^i_1, \ldots, B^i_{m_i} \subseteq [n] \backslash \{i\}$, in an iterative manner. We will eventually set $A_i = \{B^i_1, \ldots, B^i_{m_i}\}$. It will hold that for every j, B^i_j determines i in C, while satisfying $|B^i_j| \leqslant q$, and that $m_i \geqslant \delta n/q$, which will conclude the proof.

The construction of $B^i_1, \ldots, B^i_{m_i} \subseteq [n]$ is done by the following procedure. Start by setting $B^i_0 = \varnothing$. For $j = 1, 2, \ldots$, set $S^i_j = B^i_0 \cup \cdots \cup B^i_{j-1}$. If $|S^i_j| > \delta n$ halt and set $m_i = j - 1$ and $A_i = \{B^i_1, \ldots, B^i_{m_i}\}$. Otherwise, it holds that for every $c \in C$, for every modification of the coordinates in S^i_j to some erroneous values, $\mathsf{Cor}(i)$ correctly outputs c_i with probability at least $1 - \varepsilon$. An equivalent description of this case is the following: for every $c \in C$ and $z : S^i_j \to \mathbb{F}$, define $c^z \in \mathbb{F}^n$ such that for every $r \notin S^i_j$, $c^z_r = c_r$ and for $r \in S^i_j$, $c^z_r = z(r)$. The corrector Cor chooses a set of queries $Q \subseteq [n] \backslash \{i\}$, $|Q| \leqslant q$, according to some distribution[6] and applies some function f_Q on c^z_Q. We know that with probability at least $1 - \varepsilon$, $f_Q(c^z_Q) = c_i$. Since Q is sampled in a manner that is independent of c and z, by an averaging argument, there exists some fixed Q for which when $c \in C$ and $z : S^i_j \to \mathbb{F}$ are chosen randomly in a uniform manner, with probablity at least $1 - \varepsilon$ (this time over the choice of c and z), $f_Q(c^z_Q) = c_i$. Therefore, we can define another function f'_Q that only gets $c_{Q \backslash S^i_j}$, chooses z uniformly at random, and outputs $f_Q(c^z_Q)$. If $c \in C$ is chosen uniformly at random, $f'_Q(c_{Q \backslash S^i_j}) = c_i$ with probability at least $1 - \varepsilon > 1/|\mathbb{F}|$. By Fact 10, this implies that $Q \backslash S^i_j$ determines i in C. We therefore set $B^i_j = Q \backslash S^i_j$[7], and proceed to the next j.

As this process only halts when $|S^i_j| > \delta n$, and for every j, $|S^i_j| \leqslant q(j-1)$, we have that $m_i \geqslant \delta n/q$. Further note that by the choice of each B^i_j, the sets $B^i_1, \ldots, B^i_{m_i}$ are disjoint, and of size at most q, as required. This thus shows how each A_i can be constructed, and the claim follows. ◁

4.2 Splitters for query sets

Splitters for query sets, that are defined as follows, are key ingredients in our distance amplification procedure. Informally, a *c-splitter* for a query set $\mathcal{A} = \{A_1, \ldots, A_n\}$ is partition of $[n]$ which satisfies that for every i, the intersection between $\overline{A_i}$, the union all the sets in A_i that correspond to an index i, and each part of the partition, is not too large, i.e., of size at most c. In the distance amplification procedure, we will describe a corrector which samples a set $B \in A_i$, in some query set $\mathcal{A}$, and then makes queries according to which parts of the c-splitter intersect with B. For the resulted queries to be smooth, we will need the partition to "split" $A_1, \ldots, A_n$, meaning that no part of the partition is too common within any certain A_i.

[6] As the corrector in non-adaptive, $\mathsf{Cor}(i)$ naturally induces a distribution on subsets of $[n]$ which correspond to the possible query sets.

[7] Note that $i \notin B^i_j$, as $i \notin Q$, since $\mathsf{Cor}(i)$ by definition never queries i.

▶ **Definition 26.** *Let $n \in \mathbb{N}$, $\mathcal{A}$ an n-query-set and $c \in \mathbb{N}$. A partition π of $[n]$ is called a* c-splitter of $\mathcal{A}$ *if for every $i \in [n]$ and $P \in \pi$, $|P \cap \overline{A_i}| \leqslant c$.*

The next claim shows that if each A_i is of size at most k, then c-splitters with k parts exist, for c, the bound on the maximal intersection, being equal to roughly the minimal intersection that is possible, up to a constant factor.

▷ Claim 27. Let $n, k, q \in \mathbb{N}$ such that $k/n \leqslant 1$ and $q \geqslant \log n$. Further let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be an n-query-set such that for every $i \in [n]$, $|A_i| \leqslant k$ and for every $B \in A_i$, $|B| \leqslant q$. Then, there exists a partition π of $[n]$ with k parts, each of size n/k, which is a c-splitter of $\mathcal{A}$ for $c = 2eq$.

Proof. The proof is by a probabilistic argument. We randomly choose a partition π with k equally-sized parts in a uniform manner among all such partitions. We bound the probability that π is not a c-splitter for $\mathcal{A}$: this is the case if $|\overline{A_i} \cap P| > c$ for some $i \in [n]$ and P a part of π. Towards this end, we first fix some $i \in [n]$ and $t \in [k]$, and let P_t denote the t-th part of π. We have that for every $j \in \overline{A_i}$ the probability that $j \in P_t$ is $1/k$, and for every fixed subset of $\overline{A_i}$ of size c, the probability that it is contained in P_t is at most $(1/k)^c$ (since for distinct $j, j' \in \overline{A_i}$, the events that $j \in P_t$ and $j' \in P_t$ are negatively correlated). By a union bound over the possible subsets of size c, the probability that $|\overline{A_i} \cap P_t| > c$ is at most

$$\begin{aligned}\binom{|\overline{A_i}|}{c}(1/k)^c &\leqslant \left(\frac{e|\overline{A_i}|}{ck}\right)^c \\ &\leqslant \left(\frac{eq}{c}\right)^c \\ &= \left(\frac{1}{2}\right)^{2eq}.\end{aligned}$$

By taking a union bound over all possible i, t, the probability that there exist $i \in [n]$ and $t \in [k]$ such that $|\overline{A_i} \cap P_t| > c$ is at most $nk\left(\frac{1}{2}\right)^{2eq} \leqslant n^2\left(\frac{1}{2}\right)^{2eq}$, which is less than 1 a $q \geqslant \log n$, and the claim follows. ◁

4.3 The distance amplification procedure

We now turn to define the basic operation behind our distance amplification procedure. This operation "composes"[8] two codes of different lengths, a big code and a small code, in a way that is parameterized by some partition of $[n]$. The result is a code of the same length as the big code, with an improved smoothness (if the partition satisfies certain requirements), as we will have in the claims that follow the definition. The distance amplification procedure (or perhaps, more directly, the smoothness amplification procedure) will be an iterative application of this composition.

▶ **Definition 28.** *Let C_1 be a code of length n_1, C_2 a code of length n_2, π a partition of $[n_1]$ into n_1/n_2 parts of size n_2. We define the* π-composition of C_1 and C_2, *which we denote by $C_1 \odot_\pi C_2$, to be the code $\{c \in C_1 \mid \forall P \in \pi \quad c_P \in C_2\}$.*

A bound on the rate of the composition of two codes is given in the following claim.

[8] Note that the term "composition" here is used in a different sense than the usual composition of two codes in coding theory, which is achieved from the composition of the encoding functions.

▷ Claim 29. If C_1, C_2 are codes with of lengths n_1, n_2 and rates ρ_1, ρ_2 respectively, then $C = C_1 \odot_\pi C_2$ is a code of length n_1 and rate at least $\rho_1 + \rho_2 - 1$.

Proof. That the length of C is n_1 follows from the definition. As for the rate, by inspecting the code dual to C, it can be seen that the dimension of $C^\perp$ is at most

$$d = (1-\rho_1)n_1 + \frac{n_1}{n_2}(1-\rho_2)n_2.$$

From that, the rate of C is at least $1 - d/n_1 = \rho_1 + \rho_2 - 1$. ◁

We now show that if the partition used in the composition is a c-splitter for a query set of the big code, the resulted code has smoothness roughly equal to the product of the two smoothnesses.

▷ Claim 30. Let C_1 be a code of length n_1 and C_2 a code of length n_2 which is a (q_2, τ_2)-LCC. Let $\mathcal{A} = \{A_1, \ldots, A_{n_1}\}$ be a query-set for C_1 such that for every i, $|A_i| \geqslant 1/\tau_1$ and for every $B \in A_i$, $|B| \leqslant q_1$. If π is a c-splitter for $\mathcal{A}$, then $C = C_1 \odot_\pi C_2$ is a (q, τ)-LCC for $q = q_1 q_2$ and $\tau = c\tau_1\tau_2$.

Proof. To show that C is a (q, τ)-LCC we need to show a corrector Cor for it. We first set up some notations. Let Cor_2 be a corrector promised by the fact that C_2 is a (q_2, τ_2)-LCC. For every $j \in [n]$, let P_j denote the part of π that contains j, and let $\bar{j}$ denote the index of j in P_j with respect to the natural order. For $i \in [n]$, and $B \in A_i$, let $f_{i,B} : \mathbb{F}^{|B|} \to \mathbb{F}$ denote a function satisfying $f_{i,B}(c_B) = c_i$ for every $c \in C_1$. Such $f_{i,B}$ is guaranteed to exists as $\mathcal{A}$ is a query-set for C_1.

For $i \in [n]$, $\mathsf{Cor}(i)$ with oracle access to $c \in C$ acts as follows: it first samples $B \in A_i$ uniformally at random. Secondly, for every $j \in B$, the procedure obtains c_j by invoking $\mathsf{Cor}_2(\bar{j})$ with oracle access to c_{P_j}. After obtaining c_j for every $j \in B$, $\mathsf{Cor}(i)$ outputs $f_{i,B}(c_B)$.

That $\mathsf{Cor}(i)$ successfully outputs c_i for every $c \in C$ is immediate, and follows from the fact that for every j, c_{P_j} is a codeword of C_2 and so $\mathsf{Cor}_2(\bar{j})$ with access to c_{P_j} correctly outputs c_j, and from the fact $c \in C_1$ and so $f_{i,B}(c_B) = c_i$. Moreover, $\mathsf{Cor}(i)$ makes at most $q_1 q_2$ queries to c, since $|B| \leqslant q_1$ by assumption, and Cor_2 makes at most q_2 queries.

It remains to bound the probability that a coordinate $r \in [n]$ is queried by $\mathsf{Cor}(i)$ for $i \in [n]$. Let p be the probability that $\mathsf{Cor}(i)$ queries r. Fix $B \in A_i$. Conditioned on the event that B was sampled by $\mathsf{Cor}(i)$ in the first step, r is queried by $\mathsf{Cor}(i)$ if one of the calls to $\mathsf{Cor}_2(\bar{j})$, with oracle access to c_{P_j}, queries c_r for some $j \in B$. That probability is at most $|B \cap P_r|\tau_2$. Indeed, this follows by taking the union bound over the different $j \in B$, noting that if $j \notin P_r$, c_r cannot be queried by $\mathsf{Cor}_2(\bar{j})$, and using that Cor_2 queries any coordinate with probability bounded above by τ_2. Therefore,

$$\begin{aligned}
p &\leqslant \sum_{B \in A_i} \mathbf{Pr}[B \text{ is sampled by } \mathsf{Cor}(i)] \cdot |B \cap P_r|\tau_2 \\
&= \sum_{B \in A_i} \frac{1}{|A_i|} \cdot |B \cap P_r|\tau_2 \\
&\leqslant \sum_{B \in A_i} \tau_1 \cdot |B \cap P_r|\tau_2 \\
&= \tau_1\tau_2 |P_r \cap \overline{A_i}| \\
&\leqslant c\tau_1\tau_2.
\end{aligned}$$

Note that we used the assumptions that $|A_i| \geqslant 1/\tau_1$, and that π is a c-splitter for $\mathcal{A}$. We thus have that $p \leqslant c\tau_1\tau_2$, which concludes the proof. ◁

The following lemma concludes the properties of the code that is achieved by the composition of two codes, when done with the c-splitter that is given by Claim 27.

▶ **Lemma 31.** *Let $n \in \mathbb{N}$. Assume there exists a code C_1 of length n over $\mathbb{F}$, with rate ρ_1, which is a (q_1, τ_1)-query-set-LCC for $q_1 \geqslant \log n$. Further assume that there exists a code C_2 of length $n\tau_1$ over $\mathbb{F}$, with rate ρ_2, which is a (q_2, τ_2)-LCC. Then, there exists a code C of length n, with rate $\rho_1 + \rho_2 - 1$, which is a $(q_1 q_2, 2eq_1\tau_1\tau_2)$-LCC.*

Proof. As C_1 is a (q_1, τ_1)-query-set-LCC, there exists an n-query-set $\mathcal{A} = \{A_1, \ldots, A_n\}$ in which for every i, $|A_i| \geqslant 1/\tau_1$ and for every $B \in A_i$, $|B| \leqslant q_1$. In particular, there exists a query set $\mathcal{A}' = \{A'_1, \ldots, A'_n\}$ in which every A'_i is of size exactly $1/\tau_1$ (which is achieved by, for each A_i, arbitrarily removing sets $B \in A_i$ until it is of size $1/\tau_1$). By Claim 27 invoked with $k = 1/\tau_1$, there exists a partition π of $[n]$, in which every part is of size $\tau_1 n$, which is a c-splitter for $\mathcal{A}'$, with $c = 2eq_1$. We take $C = C_1 \odot_\pi C_2$ to be the code with the claimed properties. Indeed, by Claim 29, C is of length n, and has rate at least $\rho_1 + \rho_2 - 1$. Furthermore, by applying Claim 30, and using that π is a c-splitter for $\mathcal{A}'$, we get that C is a (q, τ)-LCC for $q = q_1 q_2$ and $\tau = 2eq_1\tau_1\tau_2$, and the lemma follows. ◀

The following lemma, or more precisely, its proof, composes the distance amplification procedure. It assumes a family of codes which are LCC, and describes the properties of the code that is obtained by an iterative application of the composition, where at each iteration a code of the family is composed with the "current" code.

▶ **Lemma 32.** *Assume there exists a family of codes $C = \{C^n\}$ over $\mathbb{F}$, in which every code C^n of length n in the family is a code of rate $\rho(n) = 1 - r(n)$, which is a $(q(n), \tau(n))$-query-set-LCC for $q(n) \geqslant \log n$. Then, for every $t \in \mathbb{N}$, there exists a code family $C' = \{(C')^n\}$ over $\mathbb{F}$ which has a code $(C')^n$ of length n for every n which is a code length in C, and $(C')^n$ has the following properties. Define $n_1 = n$ and for $i = 2, \ldots, t+1$ let $n_i = \left\lceil \tau(n_{i-1}) n_{i-1} \right\rceil^C$. Then, $(C')^n$ has rate $\rho'(n) = 1 - \sum_{i=1}^{t} r(n_i)$, and is a $(q'(n), \tau'(n))$-LCC for $q'(n) = \prod_{i=1}^{t} q(n_i)$ and*

$$\tau'(n) = (2e)^{t-1} \frac{n_{t+1}}{n} \prod_{i=1}^{t-1} q(n_i).$$

Proof. To show the existence of a code family with the claimed properties, we describe how for every n that is a length of a code in the family C, a code of the same length, of the family C', can be constructed. Let C^n be a code of length n of the family C. Set $n_1 = n$ and for $i = 2, \ldots, t+1$, $n_i = \left\lceil \tau(n_{i-1}) n_{i-1} \right\rceil^C$, as defined in the claim. We construct a sequence of codes $C'_1, \ldots, C'_t$, where for each $i \in [t]$, C'_i is a code of length n_i and rate ρ'_i, which is a (q'_i, τ'_i)-LCC. We start by setting $C'_t = C^{n_t}$, and for $i = t-1, \ldots, 1$, we take C'_i to be a code which is the result of applying Lemma 31 on C^{n_i} and C'_{i+1}. Note that C^{n_i} is a $(q(n_i), \tau(n_i))$-query-set-LCC and C'_{i+1} is a code of length $n_{i+1} \geqslant \tau(n_i) n_i$, and so in particular C^{n_i} is indeed of smoothness n_{i+1}/n_i, as required for the lemma to be applicable. From Lemma 31 it follows that C'_i is a code of rate

$$\rho'_i = \rho(n_i) + \rho'_{i+1} - 1 = \rho'_{i+1} - r(n_i)$$

which is a (q'_i, τ'_i)-LCC for

$$q'_i = q(n_i) q'_{i+1},$$
$$\tau'_i = 2eq(n_i) \tau'_{i+1} \frac{n_{i+1}}{n_i}.$$

Recall that $C'_t = C^{n_t}$ and so $\rho'_t = 1 - r(n_t)$, $q'_t = q(n_t)$ and $\tau'_t = \tau(n_t)$. It follows inductively that for every $i \in [t]$,

$$\rho'_i = 1 - \sum_{j=i}^{t} r(n_j),$$

$$q'_i = \prod_{j=i}^{t} q(n_j),$$

and

$$\begin{aligned}\tau'_i &= (2e)^{t-i}\left(\prod_{j=i}^{t}\frac{n_{j+1}}{n_j}\right)\left(\prod_{j=i}^{t-1} q(n_j)\right)\\ &= (2e)^{t-i}\frac{n_{t+1}}{n_i}\left(\prod_{j=i}^{t-1} q(n_j)\right).\end{aligned}$$

We set C'_1, which is indeed a code of length n, to be the code $(C')^n$ of C', and from the account given above it follows that its rate, query complexity and smoothness are as stated, i.e., that $q'_1 = q'(n)$, $\rho'_1 = \rho'(n)$ and $\tau'_1 = \tau'(n)$. We thus have that C' is a family of codes with rate at least $\rho(n)$ that are $(q(n), \tau(n))$-LCC, and the lemma follows. ◀

4.4 Corollaries

In this part we present two corollaries of our distance amplification procedure that is given by Lemma 32. As a special case of the first corollary, Corollary 34, we will have that if one has a sufficiently dense code family of $(q(n), \delta(n), \varepsilon(n))$-LCC which is of high rate, meaning that each code has rate $\rho(n)$ that approaches 1 "fast enough", but with $\delta(n)$ that is only polynomially small in n, $\delta(n) = 1/n^{\alpha}$, for some constant $\alpha \in (0, 1)$, then there exists a good family of LCC with query complexity $q(n)^{O(\log\log n)}$. In the general case, a weaker guarantee on $\delta(n)$ can also be handled by Corollary 34, meaning that a sub-polynomial $\delta(n)$ can also be amplified. More precisely, Corollary 34 will state that if $\delta(n) = 1/n^{1-1/g(n)}$ for a (non-decreasing) function $g(n)$, then a family of good LCC can be constructed, with query complexity $q(n)^{O(g(n)\log\log n)}$. The requirement of the rate function $\rho(n)$, which we described as approaching 1 "fast enough", in more detail comes down to the requirement that $\rho(n) \geqslant 1 - 1/(g(n)(\ln\ln n)^2)$.

The second corollary, Corollary 37, addresses the case that the family of $(q(n), \delta(n), \varepsilon(n))$-LCC one starts with is of a much smaller rate, either of a constant rate or of a vanishing rate of $(1/\ln\ln n)^h$ for some constant h. In the case that $\delta(n) = 1/n^{\alpha}$ for some constant $\alpha \in (0, 1)$ and $\rho(n) \geqslant (1/\ln\ln n)^h$, as a special case Corollary 37 we will have that there exists a family of good LCC with query complexity $q(n)^{\mathrm{poly}(\log\log n)}$. Here too, sub-polynomial $\delta(n)$ can also handled by the corollary, as in a more general case, it is shown by Corollary 37 that if $\delta(n) = 1/n^{1-1/g(n)}$ for a non-decreasing $g(n) \leqslant \log n$, and if $\rho(n)$ is at least $(1/\ln\ln n)^h$ for some constant h, then a family of good LCC can be constructed, with query complexity $q(n)^{g(n)\mathrm{poly}(\log\log n)}$. The precise statement Corollary 37 is more generally stated and handles a few more cases that may be of interest.

We remark that while in any case that Corollary 34 can be applied so can Corollary 37 be used, the reason that we state both corollaries is that if one starts with an LCC that satisfies the requirement of Corollary 34 then using it, rather than using Corollary 37, would result in a better bound on the resulted query complexity. We further remark that the proof for

Corollary 37 builds on Corollary 34. Lastly, another reason that Corollary 34 is of interest is that it has an analogous corollary in the case of LDC (see Corollary 35), unlike Corollary 37 (whose proof relies on properties specific to LCC).

The proofs for the corollaries can be found in the full version of the paper (see [6]).

4.4.1 From high rate and low distance LCC to good LCC

The proof for the first corollary relies on the following lemma which states that any family of (q, τ)-LCC with constant rate can be converted to a family of good LCC by paying a multiplicative factor of $\text{poly}(\tau n)$ in query complexity. This lemma follows from the AEL distance amplification procedure [2, 1] and from the adaptation of it by [17] for LDC and LCC. To derive this lemma with certain parameters, some adaptations to these techniques are needed, and so we provide a full proof for Lemma 33 in the appendix of the full version.

▶ **Lemma 33.** *Let $C = \{C^n\}$ be a code family over $\mathbb{F}$ in which every code C^n is a $(q(n), \tau(n))$-LCC with rate $\rho(n) = \Omega(1)$. Then, there exists a code family $C' = \{(C')^n\}$ over $\mathbb{F}$ which has a code $(C')^n$ of length n for every C^n in C, such that $(C')^n$ is a $(q'(n), \delta'(n), \varepsilon)$-LCC for $q'(n) = O(q(n)(n\tau(n))^2)$, $\delta'(n) = \Omega(1)$ and $\varepsilon = 1/3$, with rate $\rho'(n) = \Omega(1)$.*

We now state our first corollary.

▶ **Corollary 34.** *Let $q(n) \geqslant \log n$[9] and $g(n) > 1$ be two non-decreasing functions. Assume there exists a family of codes $C = \{C^n\}$ over $\mathbb{F}$ that is (n_0, c, d)-dense, in which every code C^n of length n has rate*

$$\rho(n) \geqslant 1 - \frac{1}{g(n)(\ln \ln n)^2},$$

and either C^n is a $(q(n), \delta(n), \varepsilon(n))$-LCC, for $\varepsilon(n) < 1 - 1/|\mathbb{F}|$ and $\delta(n) = 1/n^{1-1/g(n)}$, or it is a $(q(n), \tau(n))$-query-set-LCC, for $\tau(n) = q(n)/n^{1/g(n)}$. Then, there exists a family of codes $C' = \{(C')^n\}$ over $\mathbb{F}$ that is (n_0, c, d)-dense, which is a family of good LCC with query complexity $q_{new}(n) = q(n)^{O(g(n) \ln \ln n)}$.

Note that Corollary 34 allows for the code family C in the hypothesis to be one of two types, either a family of (q, δ, ε)-LCC or a family of (q, τ)-query-set-LCC. For the proof, what we actually need is that C is of the second type. However, if one starts with a family C which is known to be of the first (more standard) type, with the specified $\delta(n)$, by Claim 25 it will follow that C is a family of query-set-LCC with the same smoothness $\tau(n)$ that is stated in the corollary in the second case. The corollary explicitly allows both of the types because it is also possible that the base code is already known to be a query-set-LCC, as would be the case in the proof of Corollary 37, which uses Corollary 34. It is preferable to avoid going back and forth between the types, as this has some cost in the resulted parameters.

We further state a corollary analogous to Corollary 34, that holds in the case of LDC. The proof for this corollary is straightforward given the result regarding LCC, and follows the same lines.

[9] We remark that while we assume for simplicity that $q(n) \geqslant \log n$, by the Katz-Trevisan bound (instantiated for the case of rate and distance as specified by the corollary), lifting this assumption would not yield an improvement in the obtained query complexity in any case.

▶ **Corollary 35.** *Let $n(k) > k$, $q(k) \geqslant \log n(k)$ and $g(k) > 1$ be non-decreasing functions. Assume there exists a code-encoding family $C = \{(C^k, \mathsf{Enc}^k)\}$ over $\mathbb{F}$ that is (k_0, c, d)-dense, in which every code C^k of dimension k has rate*

$$\rho(k) \geqslant 1 - \frac{1}{g(k)(\ln \ln k)^2} > \frac{1}{2},$$

and either (C^k, Enc^k) is a $(q(k), \delta(k), \varepsilon(k))$-LDC, for $\varepsilon(k) < 1 - 1/|\mathbb{F}|$ and $\delta(k) = 1/n(k)^{1-1/g(k)}$. Then, there exists a code-encoding family $C' = \{((C')^k, (\mathsf{Enc}')^k)\}$ over $\mathbb{F}$ that is (k_0, c, d)-dense, which is a family of good LDC with query complexity $q_{new}(k) = q(k)^{O(g(k) \ln \ln k)}$.

4.4.2 From low rate and low distance LCC to good LCC

The proof for our second corollary uses the following proposition from [7]. This proposition is basically Proposition 4.14 in [7] but for (q, τ)-query-set-LCC rather than for a different object[10]. That the proposition indeed applies to (q, τ)-query-set-LCC is quite immediate with the account given in [7].

▶ **Proposition 36** (Implicit in [7])**.** *Let C be a code of length n over $\mathbb{F}$ with rate ρ that is a (q, τ)-query-set-LCC. Then, for every $\ell \in \mathbb{N}$, there exists a code C' of length $n' = n^\ell$ with rate $1 - (1 - \rho)^\ell$, which is a (q', τ)-query-set-LCC for $q' = q^\ell$.*

We now state our second corollary.

▶ **Corollary 37.** *Let $h \geqslant 1$ be an arbitrary constant, $q(n) \geqslant \log n$ and $g(n) \in [1, \log n]$ non-decreasing functions, and $\rho(n)$ a non-increasing function, satisfying*

$$\frac{1}{(\ln \ln n)^h} \leqslant \rho(n) \leqslant 1 - \frac{1}{g(n)(\ln \ln n)^2}$$

for every n. Assume further that

$$\frac{1}{\rho(n+1)}(\ln g(n+1) + \ln \ln \ln(n+1)) - \frac{1}{\rho(n)}(\ln g(n) + \ln \ln \ln n) = O\left(\frac{1}{\log n}\right).$$

Assume there exists a family of codes $C = \{C^n\}$ over $\mathbb{F}$ that is $(n_0, 1, 1)$-dense[11], in which every code C^n of length n is a code of rate $\rho(n)$, which is a $(q(n), \delta(n), \varepsilon(n))$-LCC, for $\varepsilon(n) < 1 - 1/|\mathbb{F}|$ and

$$\delta(n) = \frac{1}{n^{1-1/g(n)}}.$$

Then, there exists a family of codes $C' = \{(C')^n\}$ over $\mathbb{F}$, which is a family of good LCC with query complexity $q_{new}(n) = q(n)^{e(n)}$ for

$$e(n) = O\left(\frac{1}{\rho(n)^2}(\ln g(n) + \ln \ln \ln n)^2 g(n) \ln \ln n\right).$$

[10] "dual SLR" in the terminology of [7].

[11] Note that if one starts with a code family C that is (n_0, c, d) for some constants c, d, then it can be easily converted to a $(n_0, 1, 1)$-dense family, with a constant multiplicative cost to the rate and with little affect to the obtained parameters.

5 LDC are not LCC via random weighted tensor codes

In this section we argue that there exist linear codes which are LDC but not LCC, in the following strong sense. Not only are these codes LDC while not being LCC even for a weak requirement of very high query complexity and very low correction radius, moreover, this negative property that local correction with such parameters is impossible is maintained in any puncturing of the code. We will be able to show this to be the case because in the codes that we construct the uncorrectable coordinates are crucial for the distance of the code, and in particular for the LDC feature of the code, thus any attempt to remove them while keeping these properties, fails. In this section here we state the main claims required for proving the result, the proofs for which can be found in the full version of the paper (see [6]).

We start with a few preliminaries for this section. In what follows we will sometimes need to conveniently convert a pair of indices $i_1 \in [m_1]$, $i_2 \in [m_2]$ to an index $i \in [m_1 m_2]$, and so we set the following convention. Where $m_1, m_2 \in \mathbb{N}$ are clear from context and $i_1 \in [m_1]$, $i_2 \in [m_2]$, we denote by $(i_1; i_2)$ the index $(i_2 - 1)m_1 + i_1 \in [m_1 m_2]$.

▶ **Definition 38** (Trivial coordinates). *For a code C of length n over $\mathbb{F}$, we say that a coordinate $j \in [n]$ is* trivial *(in C) if for every $c \in C$, $c_j = 0$.*

▶ **Definition 39** (Puncturing of codes). *Let C be a code of length n and dimension k over $\mathbb{F}$ and let $J \subseteq [n]$. For every codeword $c \in C$, we define the vector $(y_1, \ldots, y_n) \in \mathbb{F}^n$, where $y_j = c_j$ if $j \notin J$ and $y_j = 0$ otherwise, to be the J-*puncturing of c, *and we denote it by $c_{\setminus J}$. We define $\{c_{\setminus J} \mid c \in C\}$ to be the J-*punctured code C, and denote it by $C_{\setminus J}$. *Note that $C_{\setminus J}$ is indeed a code. Furthermore, given an encoding Enc of C, we define $\mathsf{Enc}_{\setminus J} : \mathbb{F}^k \to \mathbb{F}^n$ by $\mathsf{Enc}_{\setminus J}(x) = \mathsf{Enc}(x)_{\setminus J}$ for all $x \in \mathbb{F}^k$.*

5.1 Weighted tensors

We turn to define an operation to which we call the *weighted tensor* of two codes and state several of its properties. The codes of Theorem 3 will be constructed using a weighted tensor. This operation gets two input codes (more precisely, two codes and respective encodings), and a matrix of non-zero entries, and results in a new code. To define the result of the operation, we will define a new encoding function which depends on the encodings of the two input codes and on the weight matrix. We will then take the resulted code to be the image of that encoding. We begin by describing the encoding function of the *weighted tensor*.

Let $\mathsf{Enc}_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{n_1}$ and $\mathsf{Enc}_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{n_2}$ be a linear maps, and let $B \in \mathbb{F}^{n_1 \times k_2}$ be a matrix with non-zero entries. We define the following function $\mathsf{Enc} : \mathbb{F}^{k_1 k_2} \to \mathbb{F}^{n_1 n_2}$ that acts as follows on input $x \in \mathbb{F}^{k_1 k_2}$.

Action of Enc on x

1. Identify x with a matrix $X \in \mathbb{F}^{k_1 \times k_2}$ where for $i_1 \in [k_1]$, $i_2 \in [k_2]$, $X_{i_1, i_2} = x_{(i_1; i_2)}$.
2. Use Enc_1 to encode each column of X and set X' to be the resulted matrix, $X' \in \mathbb{F}^{n_1 \times k_2}$.
3. For each $j_1 \in [n_1], i_2 \in [k_2]$ multiply the element X'_{j_1, i_2} by B_{j_1, i_2} and set X'' to be the resulted matrix.
4. Use Enc_2 to encode each row of X'' and set X''' to be the resulted matrix, $X''' \in \mathbb{F}^{n_1 \times n_2}$.
5. Output $x' \in \mathbb{F}^{n_1 n_2}$ where for $j_1 \in [n_1], j_2 \in [n_2]$, $x'_{(j_1; j_2)} = X'''_{j_1, j_2}$.

▷ Claim 40. If Enc_1 and Enc_2 are injective then so is Enc.

▷ Claim 41. Let $A^1 \in \mathbb{F}^{n_1 \times k_1}$ and $A^2 \in \mathbb{F}^{n_2 \times k_2}$ be the generating matrices of Enc_1 and Enc_2, respectively. Then, for every $x \in \mathbb{F}^{k_1 k_2}$, $\mathsf{Enc}(x) = Ax$, where $A \in \mathbb{F}^{n_1 n_2 \times k_1 k_2}$ is the matrix where for $i_1 \in [k_1], i_2 \in [k_2], j_1 \in [n_1], j_2 \in [n_2]$ we have that $A_{(j_1;j_2),(i_1;i_2)} = A^1_{j_1,i_1} A^2_{j_2,i_2} B_{j_1,i_2}$. In particular, Enc is a linear map.

We can now define the weighted tensor operation.

▶ **Definition 42.** *Let Enc_1, Enc_2, B and Enc be as above. Let C_1 be a code of length n_1 and dimension k_1 over $\mathbb{F}$ such that Enc_1 is an encoding of it, and let C_2 be a code of length n_2 and dimension k_2 over $\mathbb{F}$ such that Enc_2 is an encoding of it. Let C be the image of Enc. We define the B-*weighted tensor of (C_1, Enc_1) and (C_2, Enc_2) *to be the pair (C, Enc), and denote $(C, \mathsf{Enc}) = (C_1, \mathsf{Enc}_1) \otimes_B (C_2, \mathsf{Enc}_2)$.*

▷ Claim 43. Let $(C, \mathsf{Enc}) = (C_1, \mathsf{Enc}_1) \otimes_B (C_2, \mathsf{Enc}_2)$. Then C is a code of length $n = n_1 n_2$ and dimension $k = k_1 k_2$ over $\mathbb{F}$, and Enc is an encoding of it.

5.2 Local decodability and correctablity of weighted tensors

The weighted tensor of two LDC is an LDC with comparable parameters, regardless of the weight matrix, as we have in the following claim.

▷ Claim 44. Let (C_1, Enc_1) be a $(q_1, \delta_1, \varepsilon_1)$-LDC, where C_1 is a code of length n_1 and dimension k_1 over $\mathbb{F}$. Let (C_2, Enc_2) be a $(q_2, \delta_2, \varepsilon_2)$-LDC, where C_2 is a code of length n_2 and dimension k_2 over $\mathbb{F}$, and let $B \in \mathbb{F}^{n_1 \times k_2}$ be a matrix with no zero entries. Then, $(C, \mathsf{Enc}) = (C_1, \mathsf{Enc}_1) \otimes_B (C_2, \mathsf{Enc}_2)$ is a $(q_1 q_2, \delta_1 \delta_2, 1 - (1 - \varepsilon_1)(1 - \varepsilon_2)^{q_1})$-LDC.

In the next claim we argue that the weighted tensor of two codes, when performed with a randomly chosen weight matrix is, with high probability, not locally correctable. In particular, there exists a subset of the coordinates which cannot be locally corrected even with a small correction radius guarantee, and cannot be removed from the code either if its decodablity is to be preserved.

▷ Claim 45. Let (C_1, Enc_1) be a $(q_1, \delta_1, \varepsilon_1)$-LDC of length n_1 and dimension k_1 over $\mathbb{F}$, and let (C_2, Enc_2) be a $(q_2, \delta_2, \varepsilon_2)$-LDC of length n_2 and dimension k_2 over $\mathbb{F}$. Assume that C_1 and C_2 have no non-trivial coordinates. Let $B \in \mathbb{F}^{n_1 \times k_2}$ be a random matrix of non-zero weights, chosen uniformly and independently, and let $(C, \mathsf{Enc}) = (C_1, \mathsf{Enc}_1) \otimes_B (C_2, \mathsf{Enc}_2)$. For every $t < k_2$ and $\tilde{q}, \tilde{\tilde{q}} \in \mathbb{N}$, $\delta \geqslant \tilde{q}/n_1$, $\delta' \geqslant t/k_2$ and $\varepsilon < 1 - 1/|\mathbb{F}|$, with probability at least $1 - n_1 n_2 \binom{n_1 n_2}{\tilde{q}} |\mathbb{F}|^{\tilde{q}} / (|\mathbb{F}| - 1)^t$ over the choice of B, C satisfies the following. There exists a set $\bar{J} \subseteq [n]$ such that every $j \in \bar{J}$ is not $(\tilde{q}, \delta, \varepsilon)$-locally correctable in C. Further, the relative (non-local) distance of $C_{\setminus \bar{J}}$ is less than t/k_2.

The main theorem of this part is an immediate consequence of Claim 45.

▶ **Theorem 46.** *Let (C_0, Enc_0) be a $(q_0, \delta_0, \varepsilon_0)$-LDC for a code C_0 of dimension k_0 and length n_0 over $\mathbb{F}$ for $|\mathbb{F}| > 2$, such that $\varepsilon_0 < 1 - 1/|\mathbb{F}|$, $k_0^{1/2} > 10 \log n_0$, and assume that C_0 has no trivial coordinates. Then, there exists a $(q_0^2, \delta_0^2, 1 - (1 - \varepsilon_0)^{q_0 + 1})$-LDC (C, Enc) for a code C of dimension $k = k_0^2$ and length $n = n_0^2$ over $\mathbb{F}$ satisfying the following property. There exists a set $J \subseteq [n]$ of coordinates such that every $j \in J$, j is not $(k^{1/4}, k^{1/4}/n^{1/2}, \varepsilon)$-locally correctable in C, for any $\varepsilon < 1 - 1/|\mathbb{F}|$. Moreover, the relative distance of $C_{\setminus J}$ is less than $5 \log(n)/k^{1/4}$ (in particular for any $\tilde{\tilde{q}} \in \mathbb{N}$ and $\varepsilon < 1 - 1/|\mathbb{F}|$, $C_{\setminus J}$ is not a $(\tilde{\tilde{q}}, 5 \log(n)/k^{1/4}, \varepsilon)$-LDC).*

References

1 Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 512–519. IEEE, 1995.

2 Noga Alon and Michael Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Transactions on Information Theory*, 42(6):1732–1736, 1996.

3 Boaz Barak, Zeev Dvir, Amir Yehudayoff, and Avi Wigderson. Rank bounds for design matrices with applications to combinatorial geometry and locally correctable codes. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 519–528, 2011.

4 Arnab Bhattacharyya, Zeev Dvir, Amir Shpilka, and Shubhangi Saraf. Tight lower bounds for 2-query lccs over finite fields. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 638–647. IEEE, 2011.

5 Arnab Bhattacharyya, Sivakanth Gopi, and Avishay Tal. Lower bounds for 2-query lccs over large alphabet. *arXiv preprint*, 2016. `arXiv:1611.06980`.

6 Gil Cohen and Tal Yankovitz. Lcc and ldc: Tailor-made distance amplification and a refined separation. *Electronic Colloquium on Computational Complexity (ECCC)*, 136, 2021.

7 Gil Cohen and Tal Yankovitz. Rate amplification and query-efficient distance amplification for linear lcc and ldc. In *36th Computational Complexity Conference (CCC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

8 Irit Dinur. The PCP theorem by gap amplification. In *Proc. 38th ACM Symp. on Theory of Computing*, pages 241–250, 2006.

9 Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM Journal on Computing*, 40(4):1154–1178, 2011.

10 Zeev Dvir and Kalina Petrova. Lecture 1: Introduction. Lecture notes: `https://www.cs.princeton.edu/~zdvir/LDCnotes/LDC1.pdf`, 2016.

11 Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM Journal on Computing*, 41(6):1694–1703, 2012.

12 Sivakanth Gopi, Swastik Kopparty, Rafael Oliveira, Noga Ron-Zewi, and Shubhangi Saraf. Locally testable and locally correctable codes approaching the gilbert-varshamov bound. *IEEE Transactions on Information Theory*, 64(8):5813–5831, 2018.

13 Alan Guo, Swastik Kopparty, and Madhu Sudan. New affine-invariant codes from lifting. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 529–540. ACM, 2013.

14 Brett Hemenway, Rafail Ostrovsky, and Mary Wootters. Local correctability of expander codes. *Information and Computation*, 243:178–190, 2015.

15 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 80–86, 2000.

16 Tali Kaufman and Michael Viderman. Locally testable vs. locally decodable codes. In *Approximation, randomization, and combinatorial optimization*, volume 6302 of *Lecture Notes in Comput. Sci.*, pages 670–682. Springer, Berlin, 2010. `doi:10.1007/978-3-642-15369-3_50`.

17 Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *Journal of the ACM (JACM)*, 64(2):11, 2017.

18 Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *Journal of the ACM (JACM)*, 61(5):28, 2014.

19 A. Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 238–251. ACM, 2017.

20 Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM (JACM)*, 55(1):1–16, 2008.

Metastability of the Potts Ferromagnet on Random Regular Graphs

Amin Coja-Oghlan
Faculty of Computer Science, TU Dortmund, Germany

Andreas Galanis
Department of Computer Science, University of Oxford, UK

Leslie Ann Goldberg
Department of Computer Science, University of Oxford, UK

Jean Bernoulli Ravelomanana
Faculty of Computer Science, TU Dortmund, Germany

Daniel Štefankovič
Department of Computer Science, Univerity of Rochester, NY, USA

Eric Vigoda
Computer Science, University of California Santa Barbara, CA, USA

Abstract

We study the performance of Markov chains for the q-state ferromagnetic Potts model on random regular graphs. While the cases of the grid and the complete graph are by now well-understood, the case of random regular graphs has resisted a detailed analysis and, in fact, even analysing the properties of the Potts distribution has remained elusive. It is conjectured that the performance of Markov chains is dictated by metastability phenomena, i.e., the presence of "phases" (clusters) in the sample space where Markov chains with local update rules, such as the Glauber dynamics, are bound to take exponential time to escape, and therefore cause slow mixing. The phases that are believed to drive these metastability phenomena in the case of the Potts model emerge as local, rather than global, maxima of the so-called Bethe functional, and previous approaches of analysing these phases based on optimisation arguments fall short of the task.

Our first contribution is to detail the emergence of the metastable phases for the q-state Potts model on the d-regular random graph for all integers $q, d \geq 3$, and establish that for an interval of temperatures, delineated by the uniqueness and a broadcasting threshold on the d-regular tree, the two phases coexist. The proofs are based on a conceptual connection between spatial properties and the structure of the Potts distribution on the random regular graph, rather than complicated moment calculations. This significantly refines earlier results by Helmuth, Jenssen, and Perkins who had established phase coexistence for a small interval around the so-called ordered-disordered threshold (via different arguments) that applied for large q and $d \geq 5$.

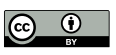
Based on our new structural understanding of the model, we obtain various algorithmic consequences. We first complement recent fast mixing results for Glauber dynamics by Blanca and Gheissari below the uniqueness threshold, showing an exponential lower bound on the mixing time above the uniqueness threshold. Then, we obtain tight results even for the non-local and more elaborate Swendsen-Wang chain, where we establish slow mixing/metastability for the whole interval of temperatures where the chain is conjectured to mix slowly on the random regular graph. The key is to bound the conductance of the chains using a random graph "planting" argument combined with delicate bounds on random-graph percolation.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains

Keywords and phrases Markov chains, sampling, random regular graph, Potts model

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.45

Category Track A: Algorithms, Complexity and Games



49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 45; pp. 45:1–45:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version *Full Version* (theorem numbering here matches v1): arxiv.org/abs/2202.05777

Funding *Amin Coja-Oghlan*: supported by DFG CO 646/3 and 646/4.
Jean Bernoulli Ravelomanana: supported by DFG CO 646/4.
Daniel Štefankovič: supported by NSF grant CCF-2007287.
Eric Vigoda: supported by NSF CCF-2007022.

1 Introduction

1.1 Motivation

Spin systems on random graphs have turned out to be a source of extremely challenging problems at the junction of mathematical physics and combinatorics [36, 37]. Beyond the initial motivation of modelling disordered systems, applications have sprung up in areas as diverse as computational complexity, coding theory, machine learning and even screening for infectious diseases; e.g. [1, 14, 22, 34, 38, 40, 41]. Progress has been inspired largely by techniques from statistical physics, which to a significant extent still await a rigorous justification. The physicists' sophisticated but largely heuristic tool is the Belief Propagation message passing scheme in combination with a functional called the Bethe free energy [35]. Roughly speaking, the fixed points of Belief Propagation are conjectured to correspond to the "pure states" of the underlying distribution, with the Bethe functional gauging the relative weight of the different pure states. Yet at closer inspection matters are actually rather complicated. For instance, the system typically possesses spurious Belief Propagation fixed points without any actual combinatorial meaning, while other fixed points need not correspond to metastable states that attract dynamics such as the Glauber Markov chain [11, 15]. Generally, the mathematical understanding of the connection between Belief Propagation and dynamics leaves much to be desired.

In this paper we investigate the ferromagnetic Potts model on the random regular graph. Recall, for an integer $q \geq 3$ and real $\beta > 0$, the Potts model on a graph $G = (V, E)$ corresponds to a probability distribution $\mu_{G,\beta}$ over all possible configurations $[q]^V$, commonly referred to as the Boltzmann/Gibbs distribution; the weight of a configuration σ in the distribution is defined as $\mu_{G,\beta}(\sigma) = \mathrm{e}^{\beta\mathcal{H}_G(\sigma)}/Z_\beta(G)$ where $\mathcal{H}_G(\sigma)$ is the number of edges that are monochromatic under σ, and $Z_\beta(G) = \sum_{\tau\in[q]^V} \mathrm{e}^{\beta\mathcal{H}_G(\tau)}$ is the normalising factor of the distribution. In physics jargon, β corresponds to the so-called inverse-temperature of the model, $\mathcal{H}_G(\,\cdot\,)$ is known as the Hamiltonian, and $Z_\beta(\,\cdot\,)$ is the partition function. Note, since $\beta > 0$, the Boltzmann distribution assigns greater weight to configurations σ where many edges join vertices of the same colour; thus, the pairwise interactions between vertices are ferromagnetic.

The Potts model on the d-regular random graph has two distinctive features. First, the local geometry of the random regular graph is essentially deterministic. For any fixed radius ℓ, the depth-ℓ neighbourhood of all but a tiny number of vertices is just a d-regular tree. Second, the ferromagnetic nature of the model precludes replica symmetry breaking, a complex type of long-range correlations [35]. Given these, it is conjectured that the model on the random regular graph has a similar behaviour to that on the clique (the so-called mean field case), and there has already been some preliminary evidence of this correspondence [4, 20, 19, 22, 29]. On the clique, the phase transitions are driven by a battle between two subsets of configurations (phases): (i) the paramagnetic/disordered phase, consisting of configurations where every colour appears roughly equal number of times, and (ii) the ferromagnetic/ordered phase, where one of the colours appears more frequently than

the others. It is widely believed that these two phases also mark (qualitatively) the same type of phase transitions for the Potts model on the random regular graph, yet this has remained largely elusive.

The main reason that this behaviour is harder to establish on the random regular graph is that it has a non-trivial global geometry which makes both the analysis of the distribution and Markov chains significantly more involved (to say the least). In particular, the emergence of the metastable states in the distribution, which can be established by way of calculus in the mean-field case, is out of reach with single-handed analytical approaches in the random regular graph and it is therefore not surprising that it has resisted a detailed analysis so far. Likewise, the analysis of Markov chains is a far more complicated task since their evolution needs to be considered in terms of the graph geometry and therefore much harder to keep track of.

Our main contribution is to detail the emergence of the metastable states, viewed as fixed points of Belief Propagation on this model, and their connection with the dynamic evolution of the two most popular Markov chains, the Glauber dynamics and the Swensen-Wang chain. We prove that these natural fixed points, whose emergence is directly connected with the phase transitions of the model, have the combinatorial meaning in terms of both the pure state decomposition of the distribution and the Glauber dynamics that physics intuition predicts they should. The proofs avoid the complicated moment calculations and the associated complex optimistion arguments that have become a hallmark of the study of spin systems on random graphs [2]. Instead, building upon and extending ideas from [3, 16], we exploit a connection between spatial mixing properties on the d-regular tree and the Boltzmann distribution. Our metastability results for the Potts model significantly refine those appearing in the literature, especially those in [22, 29] which are more relevant to this work, see Section 1.6 for a more detailed discussion.

We expect that this approach might carry over to other examples, particularly other ferromagnetic models. Let us begin by recapitulating Belief Propagation.

1.2 Belief Propagation

Suppose that $n, d \geq 3$ are integers such that dn is even and let $\mathbb{G} = \mathbb{G}(n, d)$ be the random d-regular graph on the vertex set $[n] = \{1, \ldots, n\}$. For an inverse temperature parameter $\beta > 0$ and an integer $q \geq 3$ we set out to investigate the Boltzmann distribution $\mu_{\mathbb{G},\beta}$; let us write $\boldsymbol{\sigma}_{\mathbb{G},\beta}$ for a configuration drawn from $\mu_{\mathbb{G},\beta}$.

A vital step toward understanding the Boltzmann distribution is to get a good handle on the partition function $Z_\beta(\mathbb{G})$. Indeed, according to the physicsts' cavity method, Belief Propagation actually solves both problems in one fell swoop [35]. To elaborate, with each edge $e = uv$ of $\mathbb{G}$, Belief Propagation associates two messages $\mu_{\mathbb{G},\beta,u\to v}, \mu_{\mathbb{G},\beta,v\to u}$, which are probability distributions on the set $[q]$ of colours. The message $\mu_{\mathbb{G},\beta,u\to v}(c)$ is defined as the marginal probability of v receiving colour c in a configuration drawn from the Potts model on the graph $\mathbb{G} - u$ obtained by removing u. The semantics of $\mu_{\mathbb{G},\beta,v\to u}$ is analogous.

Under the assumption that the colours of far apart vertices of $\mathbb{G}$ are asymptotically independent, one can heuristically derive a set of equations that links the various messages together. For a vertex v, let ∂v be the set of neighbours of v, and for an integer $\ell \geq 1$ let $\partial^\ell v$ be the set of vertices at distance precisely ℓ from v. The *Belief Propagation equations* read

$$\mu_{\mathbb{G},\beta,v\to u}(c) = \frac{\prod_{w\in\partial v\setminus\{u\}} 1 + (\mathrm{e}^\beta - 1)\mu_{\mathbb{G},\beta,w\to v}(c)}{\sum_{\chi\in[q]}\prod_{w\in\partial v\setminus\{u\}} 1 + (\mathrm{e}^\beta - 1)\mu_{\mathbb{G},\beta,w\to v}(\chi)} \qquad (uv \in E(\mathbb{G}),\ c \in [q]). \tag{1}$$

The insight behind (1) is that once we remove v from the graph, its neighbours $w \neq u$ are typically far apart from one another because $\mathbb{G}$ contains only a negligible number of short cycles. Hence, we expect that in $\mathbb{G} - v$ the spins assigned to $w \in \partial v \setminus \{u\}$ are asymptotically independent. From this assumption it is straightforward to derive the sum-product-formula (1).

A few obvious issues spring to mind. First, for large β it is not actually true that far apart vertices decorrelate. This is because at low temperature there occur q different ferromagnetic pure states, one for each choice of the dominant colour. To break the symmetry between them one could introduce a weak external field that slighly boosts a specific colour or, more bluntly, confine oneself to a conditional distribution on subspace where a specific colour dominates. In the definition of the messages and in (1) we should thus replace the Boltzmann distribution by the conditional distribution $\mu_{\mathbb{G},\beta}(\cdot \mid S)$ for a suitable $S \subseteq [q]^n$. Second, even for the conditional measure we do not actually expect (1) to hold precisely. This is because for any finite n minute correlations between far apart vertices are bound to remain.

Nonetheless, precise solutions $(\mu_{v\to u})_{uv\in E(\mathbb{G})}$ to (1) are still meaningful. They correspond to stationary points of a functional called the *Bethe free energy*, which connects Belief Propagation with the problem of approximating the partition function [44]. Given a collection $(\mu_{u\to v})_{uv\in E(\mathbb{G})}$ of probability distributions on $[q]$, the Bethe functional reads

$$\begin{aligned}\mathcal{B}_{\mathbb{G},\beta}\left((\mu_{u\to v})_{uv\in E(\mathbb{G})}\right) = \frac{1}{n}\sum_{v\in V(\mathbb{G})}\log\Bigg[\sum_{c\in[q]}\prod_{w\in\partial v}1+(\mathrm{e}^{\beta}-1)\mu_{w\to v}(c)\Bigg]\\ -\frac{1}{n}\sum_{vw\in E(\mathbb{G})}\log\Bigg[1+(\mathrm{e}^{\beta}-1)\sum_{c\in[q]}\mu_{v\to w}(c)\mu_{w\to v}(c)\Bigg].\end{aligned} \tag{2}$$

According to the cavity method the maximum of $\mathcal{B}_{\mathbb{G},\beta}\left((\mu_{u\to v})_{uv\in E(\mathbb{G})}\right)$ over all solutions $(\mu_{u\to v})_{uv\in E(\mathbb{G})}$ to (1) should be asymptotically equal to $\log Z_\beta(\mathbb{G})$ with high probability.

In summary, physics lore holds that the solutions $(\mu_{u\to v})_{uv\in E(\mathbb{G})}$ to (1) are meaningful because they correspond to a decomposition of the phase space $[q]^n$ into pieces where long-range correlations are absent. Indeed, these "pure states" are expected to exhibit metastability, i.e., they trap dynamics such as the Glauber Markov chain for an exponential amount of time. Moreover, the relative probabilities of the pure states are expected to be governed by their respective Bethe free energy. In the following we undertake to investigate these claims rigorously.

Before proceeding, let us mention that ferromagnetic spin systems on random graphs have been among the first models for which predictions based on the cavity method could be verified rigorously. Following seminal work by Dembo and Montanari on the Ising model [18] vindicating the "replica symmetric ansatz", Dembo, Montanari and Sun [20] studied, among other things, the Gibbs unique phase of the Potts ferromagnet on the random regular graph, and Dembo, Montanari, Sly and Sun [20] established the free energy of the model for all β (and d even). More generally, Ruozzi [39] pointed out how graph covers [43] can be used to investigate the partition function of supermodular models, of which the Ising ferromagnet is an example. In addition, Barbier, Chan and Macris [4] proved that ferromagnetic spin systems on random graphs are generally replica symmetric in the sense that the multi-overlaps of samples from the Boltzmann distribution concentrate on deterministic values.

1.3 The ferromagnetic and the paramagnetic states

An obvious attempt at constructing solutions to the Belief Propagation equations is to choose identical messages $\mu_{u\to v}$ for all edges $uv \in E(\mathbb{G})$. Clearly, any solution $(\mu(c))_{c\in[q]}$ to the system

$$\mu(c) = \frac{(1+(\mathrm{e}^\beta-1)\mu(c))^{d-1}}{\sum_{\chi\in[q]}(1+(\mathrm{e}^\beta-1)\mu(\chi))^{d-1}} \qquad (c\in[q]) \tag{3}$$

supplies such a "constant" solution to (1). Let $\mathcal{F}_{d,\beta}$ be the set of all solutions $(\mu(c))_{c\in[q]}$ to (3). The Bethe functional (2) then simplifies to

$$\mathcal{B}_{d,\beta}\big((\mu(c))_{c\in[q]}\big) = \log\Big[\sum_{c\in[q]}\big(1+(\mathrm{e}^\beta-1)\mu(c)\big)^d\Big] - \frac{d}{2}\log\Big[1+(\mathrm{e}^\beta-1)\sum_{c\in[q]}\mu(c)^2\Big]. \tag{4}$$

One obvious solution to (3) is the uniform distribution on $[q]$; we refer to that solution as paramagnetic/disordered and denote it by μ_{p}. Apart from μ_{p}, other solutions to (3) emerge as β increases for any $d \geq 3$. Specifically, let $\beta_u > 0$ be the supremum value of $\beta > 0$ where μ_{p} is the unique solution to (3).[1] Then, for $\beta = \beta_u$, one more solution μ_{f} emerges such that $\mu_{\mathrm{f}}(1) > \mu_{\mathrm{f}}(i) = \frac{1-\mu_{\mathrm{f}}(1)}{q-1}$ for $i = 2,\ldots,q$, portending the emergence of a metastable state and, ultimately, a phase transition. In particular, for any $\beta > \beta_u$, a bit of calculus reveals there exist either one or two distinct solutions μ with $\mu(1) > \mu(i) = \frac{1-\mu(1)}{q-1}$ for $i = 2,\ldots,q$; we denote by μ_{f} the solution of (3) which maximises the value $\mu(1)$ and refer to it as ferromagnetic/ordered. The value β_u is the so-called uniqueness threshold for the Potts model on the d-regular tree, see, e.g., [22] for a more detailed discussion and related pointers.

At the critical value

$$\beta_p = \max\{\beta \geq \beta_u : \mathcal{B}_{d,\beta}(\mu_{\mathrm{p}}) \geq \mathcal{B}_{d,\beta}(\mu_{\mathrm{f}})\} = \log\frac{q-2}{(q-1)^{1-2/d}-1}.$$

the ferromagnetic solution μ_{f} takes over from the paramagnetic solution μ_{p} as the global maximiser of the Bethe functional. For that reason, the threshold β_p is also known in the literature as the ordered-disordered threshold. Yet, up to the threshold

$$\beta_h = \log(1+q/(d-2))$$

the paramagnetic solution remains a local maximiser of the Bethe free energy; later, in Section 2.2 we will see that β_h has a natural interpretation as a tree-broadcasting threshold (and is also a conjectured threshold for uniqueness in the random-cluster representation for the Potts model, see [28] for details).

The relevance of these critical values has been demonstrated in [22] (see also [19] for d even, and [29] for q large), where it was shown that $\frac{1}{n}\log Z_\beta(\mathbb{G})$ is asymptotically equal to $\max_\mu \mathcal{B}_{d,\beta}(\mu)$, the maximum ranging over μ satisfying (3). In particular, at the maximum it holds that $\mu = \mu_{\mathrm{p}}$ when $\beta < \beta_p$, $\mu = \mu_{\mathrm{f}}$ when $\beta > \beta_p$ and $\mu \in \{\mu_{\mathrm{p}}, \mu_{\mathrm{f}}\}$ when $\beta = \beta_p$.

[1] The value does not have a closed-form expression, but there is an equivalent formulation of it given by the equality $\mathrm{e}^{\beta_u} = 1 + \inf_{y>1}\frac{(y-1)(y^{d-1}+q-1)}{y^{d-1}-y}$.

1.4 Slow mixing and metastability

To investigate the two BP solutions further and obtain connections to the dynamical evolution of the model, we need to look more closely how these two solutions $\mu_{\mathrm{p}}, \mu_{\mathrm{f}}$ manifest themselves in the random regular graph. To this end, we define for a given distribution μ on $[q]$ another distribution

$$\nu^{\mu}(c) = \frac{(1+(\mathrm{e}^{\beta}-1)\mu(c))^d}{\sum_{\chi\in[q]}(1+(\mathrm{e}^{\beta}-1)\mu(\chi))^d} \qquad (c\in[q]). \tag{5}$$

Let $\nu_{\mathrm{f}} = \nu^{\mu_{\mathrm{f}}}$ and $\nu_{\mathrm{p}} = \nu^{\mu_{\mathrm{p}}}$ for brevity; of course $\nu_{\mathrm{p}} = \mu_{\mathrm{p}}$ is just the uniform distribution. The distributions ν_{f} and ν_{p} represent the expected Boltzmann marginals within the pure states corresponding to μ_{f} and μ_{p}. Indeed, the r.h.s. of (5) resembles that of (3) except that the exponents read d rather than $d-1$. This means that we pass from messages, where we omit one specific endpoint of an edge from the graph, to actual marginals, where all d neighbours of a vertex are present. For small $\varepsilon > 0$, it will therefore be relevant to consider the sets of configurations

$$S_{\mathrm{f}}(\varepsilon) = \Big\{\sigma\in[q]^n : \sum_{c\in[q]}\Big|\big|\sigma^{-1}(c)\big| - n\nu_{\mathrm{f}}(c)\Big| < \varepsilon n\Big\},$$
$$S_{\mathrm{p}}(\varepsilon) = \Big\{\sigma\in[q]^n : \sum_{c\in[q]}\Big|\big|\sigma^{-1}(c)\big| - n\nu_{\mathrm{p}}(c)\Big| < \varepsilon n\Big\},$$

whose colour statistics are about $n\nu_{\mathrm{f}}$ and $n\nu_{\mathrm{p}}$, respectively; i.e., in S_{p}, all colours appear with roughly equal frequency, whereas in S_{f} colour 1 is favoured over the other $q-1$ colours (which appear with roughly equal frequency).

We are now in position to state our main result for Glauber dynamics. Recall that, for a graph $G=(V,E)$, Glauber is initialised at a configuration $\sigma_0\in[q]^V$; at each time step $t\geq 1$, Glauber draws a vertex uniformly at random and obtains a new configuration σ_t by updating the colour of the chosen vertex according to the conditional Boltzmann distribution given the colours of its neighbours. It is a well-known fact that Glauber converges in distribution to $\mu_{G,\beta}$; the mixing time of the chain is defined as the maximum number of steps t needed to get within total variation distance $\leq 1/4$ from $\mu_{G,\beta}$, where the maximum is over the choice of the initial configuration σ_0, i.e., the quantity $\max_{\sigma_0}\min\{t : d_{\mathrm{TV}}(\sigma_t,\mu_{G,\beta})\leq 1/4\}$.

For metastability, we will consider Glauber launched from a random configuration from a subset $S\subseteq[q]^V$ of the state space. More precisely, let us denote by $\mu_{G,\beta,S} = \mu_{G,\beta}(\cdot\mid S)$ the conditional Boltzmann distribution on S. We call S a *metastable state for Glauber dynamics* on G if there exists $\delta>0$ such that

$$\mathbb{P}\Big[\min\{t : \sigma_t\notin S\}\leq \mathrm{e}^{\delta|V|} \mid \sigma_0\sim\mu_{G,\beta,S}\Big] \leq \mathrm{e}^{-\delta|V|}.$$

Hence, it will most likely take Glauber an exponential amount of time to escape from a metastable state.

▶ **Theorem 1.1.** *Let $d,q\geq 3$ be integers and $\beta>0$ be real. Then, for all sufficiently small $\varepsilon>0$, the following hold w.h.p. over the choice of $\mathbb{G}=\mathbb{G}(n,d)$.*

(i) *If $\beta<\beta_h$, then $S_{\mathrm{p}}(\varepsilon)$ is a metastable state for Glauber dynamics on $\mathbb{G}$.*

(ii) *If $\beta>\beta_u$, then $S_{\mathrm{f}}(\varepsilon)$ is a metastable state for Glauber dynamics on $\mathbb{G}$.*

Further, for $\beta>\beta_u$, the mixing time of Glauber is $\mathrm{e}^{\Omega(n)}$.

Thus, we can summarise the evolution of the Potts model as follows. For $\beta<\beta_u$ there is no ferromagnetic state. As β passes β_u, the ferromagnetic state S_{f} emerges first as a metastable state. Hence, if we launch Glauber from S_{f}, the dynamics will most likely remain

trapped in the ferromagnetic state for an exponential amount of time, even though the Boltzmann weight of the paramagnetic state is exponentially larger (as we shall see in the next section). At the point β_p the ferromagnetic state then takes over as the one dominating the Boltzmann distribution, but the paramagnetic state remains as a metastable state up to β_h. Note in particular that the two states coexist as metastable states throughout the interval (β_u, β_h).

The metastability for the Potts model manifests also in the evolution of the Swendsen-Wang (SW) chain, which is another popular and substantially more elaborate chain that makes non-local moves, based on the random-cluster representation of the model. For a graph $G = (V, E)$ and a configuration $\sigma \in [q]^V$, a single iteration of SW starting from σ consists of two steps.

- *Percolation step:* Let $M = M(\sigma)$ be the random edge-set obtained by adding (independently) each monochromatic edge under σ with probability $p = 1 - \mathrm{e}^{-\beta}$.
- *Recolouring step:* Obtain the new $\sigma' \in [q]^V$ by assigning each component[2] of the graph (V, M) a uniformly random colour from $[q]$; for $v \in V$, we set σ'_v to be the colour assigned to v's component.

We define metastable states for SW dynamics analogously to above. The following theorem establishes the analogue of Theorem 1.1 for the non-local SW dynamics. Note here that SW might change the most-frequent colour due to recolouring step, so the metastability statement for the ferromagnetic phase needs to consider the set $S_{\mathrm{f}}(\varepsilon)$ with its $q-1$ permutations.

▶ **Theorem 1.2.** *Let $d, q \geq 3$ be integers and $\beta > 0$ be real. Then, for all sufficiently small $\varepsilon > 0$, the following hold w.h.p. over the choice of $\mathbb{G} = \mathbb{G}(n, d)$.*

(i) *If $\beta < \beta_h$, then $S_{\mathrm{p}}(\varepsilon)$ is a metastable state for SW dynamics on $\mathbb{G}$.*

(ii) *If $\beta > \beta_u$, then $S_{\mathrm{f}}(\varepsilon)$ together with its $q-1$ permutations is a metastable state for SW dynamics on $\mathbb{G}$.*

Further, for $\beta \in (\beta_u, \beta_h)$, the mixing time of SW is $\mathrm{e}^{\Omega(n)}$.

1.5 The relative weight of the metastable states

At the heart of obtaining the metastability results of the previous section is a refined understanding of the relative weight of the ferromagnetic and paramagnetic states. The following notion of non-reconstruction will be the key in our arguments; it captures the absence of long-range correlations within a set $S \subseteq [q]^n$, saying that, for any vertex v, a typical boundary configuration on $\boldsymbol{\sigma}_{\partial^\ell v}$ chosen according to the conditional distribution on S does not impose a discernible bias on the colour of v (for large ℓ, n; recall, $\partial^\ell v$ is the set of all vertices at distance precisely ℓ from v). More precisely, let $\mu = \mu_{\mathbb{G},\beta}$ and $\boldsymbol{\sigma} \sim \mu$; the Boltzmann distribution exhibits *non-reconstruction given a subset $S \subseteq [q]^n$* if for any vertex v it holds that

$$\lim_{\ell\to\infty} \limsup_{n\to\infty} \sum_{c\in[q]} \sum_{\tau\in S} \mathbb{E}\left[\mu(\tau \mid S) \times |\mu(\boldsymbol{\sigma}_v = c \mid \boldsymbol{\sigma}_{\partial^\ell v} = \tau_{\partial^\ell v}) - \mu(\boldsymbol{\sigma}_v = c \mid S)|\right] = 0,$$

where the expectation is over the choice of the graph $\mathbb{G}$.

[2] Note, isolated vertices count as connected components.

▶ **Theorem 1.3.** *Let $d, q \geq 3$ be integers and $\beta > 0$ be real. The following hold for all sufficiently small $\varepsilon > 0$ as $n \to \infty$.*

- **(i)** *For all $\beta < \beta_p$, $\mathbb{E}[\mu_{\mathbb{G},\beta}(S_{\mathrm{p}})] \to 1$ and, if $\beta > \beta_u$, then $\mathbb{E}\left[\frac{1}{n}\log \mu_{\mathbb{G},\beta}(S_{\mathrm{f}})\right] \to \mathcal{B}_{d,\beta}(\mu_{\mathrm{f}}) - \mathcal{B}_{d,\beta}(\mu_{\mathrm{p}})$.*
- **(ii)** *For all $\beta > \beta_p$, $\mathbb{E}[\mu_{\mathbb{G},\beta}(S_{\mathrm{f}})] \to 1/q$ and, if $\beta < \beta_h$, then $\mathbb{E}\left[\frac{1}{n}\log \mu_{\mathbb{G},\beta}(S_{\mathrm{p}})\right] \to \mathcal{B}_{d,\beta}(\mu_{\mathrm{p}}) - \mathcal{B}_{d,\beta}(\mu_{\mathrm{f}})$.*

Furthermore, the Boltzmann distribution given S_{p} exhibits non-reconstruction if $\beta < \beta_h$ and the Boltzmann distribution given S_{f} exhibits non-reconstruction if $\beta > \beta_u$.

Theorem 1.3 shows that for $\beta < \beta_p$ the Boltzmann distribution is dominated by the paramagnetic state S_{p} for $\beta < \beta_p$. Nonetheless, at β_u the ferromagnetic state S_{f} and its $q-1$ mirror images start to emerge. Their probability mass is determined by the Bethe free energy evaluated at μ_{f}. Further, as β passes β_p the ferromagnetic state takes over as the dominant state, with the paramagnetic state lingering on as a sub-dominant state up to β_h. Finally, both states S_{p} and S_{f} are free from long-range correlations both for the regime of β where they dominate and for those β where they are sub-dominant.

1.6 Discussion

Our slow mixing result for Glauber dynamics when $\beta > \beta_u$ (Theorem 1.1) significantly improves upon previous results of Bordewich, Greenhill and Patel [9] that applied to $\beta > \beta_u + \Theta_q(1)$. Similarly, our slow mixing result for Swendsen-Wang dynamics when $\beta \in (\beta_u, \beta_h)$ (Theorem 1.2) strengthens earlier results of Galanis, Štefankovič, Vigoda, Yang [22] which applied to $\beta = \beta_p$, and by Helmuth, Jenseen and Perkins [29] which applied for a small interval around β_p; both results applied only for q sufficiently large. To obtain our result for all integers $q, d \geq 3$, we need to carefully track how SW evolves on the random regular graph for configurations starting from the ferromagnetic and paramagnetic phases, by accounting for the percolation step via delicate arguments, whereas the approaches of [22, 29] side-stepped this analysis by considering the change in the number of monochromatic edges instead.

Our slow mixing results complement the recent fast mixing result of Blanca and Gheissari [6] for edge dynamics on the random d-regular graph that applies to all $\beta < \beta_u$. Roughly, edge dynamics is the analogue of Glauber dynamics for the random cluster representation of the Potts model (the random-cluster representation has nicer monotonicity properties). The result of [6] already implies a polynomial bound on the mixing time of SW when $\beta < \beta_u$ (due to comparison results by Ullrich that apply to general graphs [42]), and conversely our exponential lower bound on the mixing time of SW for $\beta \notin (\beta_u, \beta_h)$ implies an exponential lower bound on the mixing time of edge dynamics for $\beta \notin (\beta_u, \beta_h)$. The main open questions remaining are therefore showing whether Glauber dynamics for the Potts model mixes fast when $\beta \leq \beta_u$ and whether SW/edge-dynamics mixes fast when $\beta > \beta_h$. Extrapolating from the mean-field case (see discussion below), it is natural to conjecture that our slow mixing results are best-possible, i.e., for $\beta \leq \beta_u$, Glauber mixes rapidly and similarly, for $\beta \notin (\beta_u, \beta_h)$, SW mixes rapidly on the random regular graph.

Theorem 1.3, aside from being critical in establishing the aforementioned slow mixing and metastability results, is the first to establish for all $q, d \geq 3$ the coexistence of the ferromagnetic and paramagnetic phases for all β in the interval (β_u, β_h) and detail the logarithmic order of their relative weight in the same interval. Previous work in [22] showed coexistence for $\beta = \beta_p$ (for all $q, d \geq 3$) and [29] for β in a sub-interval of (β_u, β_h) around β_p (for large q and $d \geq 5$), see also footnote 3. We remark here that the approaches in [22, 29] establish more refined estimates on the deviations from the limiting value of the

log-partition function of the phases (in the corresponding regimes they apply), with [29] characterising in addition the limiting distribution using cluster-expansion methods. One can obtain analogous distributional characterisations for all $q, d \geq 3$ from our methods, once combined with the small subgraph conditioning method of [22]. It should be noted though the approach of [29] which goes through cluster expansion is more direct in that respect. We don't pursue such distributional results here since Theorem 1.3 is sufficient for our slow mixing results.

Together with Theorems 1.1 and 1.2, Theorem 1.3 delineates more firmly[3] the correspondence with the (simpler) mean-field case, the Potts model on the clique. In the mean-field case, there are qualitatively similar thresholds $\beta_u, \beta_p, \beta_h$ and the mixing time for Glauber and SW have been detailed for all β, even at criticality, see [7, 8, 25, 23, 17, 27, 31]. As mentioned earlier, the most tantalising question remaining open is to establish whether the fast mixing of SW for $\beta = \beta_u$ and $\beta \geq \beta_h$ in the mean-field case translates to the random regular graph as well. Another interesting direction is to extend our arguments to the random-cluster representation of the Potts model for all non-integer $q \geq 1$; note that the arguments of [5] and [29] do apply to non-integer q ($q \geq 1$ and q large, respectively). The proof of Theorem 1.3 relies on a truncated second moment computation, an argument that was applied to different models in [16, 13].

We further remark here that, from a worst-case perspective, it is known that sampling from the Potts model on d-regular graphs is #BIS-hard for $\beta > \beta_p$ [22], and we conjecture that the problem admits a poly-time approximation algorithm when $\beta < \beta_p$. However, even showing that Glauber mixes fast on any d-regular graph in the uniqueness regime $\beta < \beta_u$ is a major open problem, and Theorems 1.1 and 1.2 further demonstrate that getting an algorithm all the way to β_p will require using different techniques. On that front, progress has been made on the random regular graph: [29] obtained an algorithm for $d \geq 5$ and q large that applies to all β by sampling from each phase separately (using different tools), see also [10]. Moreover, for $\beta < \beta_p$, Efthymiou [21] gives an algorithm with weaker approximation guarantees but which applies to all $q, d \geq 3$ (see also [5]). In principle, and extrapolating again from the mean-field case, one could use Glauber/SW to sample from each phase on the random regular graph for all $q, d \geq 3$ and all β. Analysing such chains appears to be relatively far from the reach of current techniques even in the case of the random regular graph, let alone worst-case graphs. In the case of the Ising model however, the case $q = 2$, the analogue of this fast mixing question has recently been established for sufficiently large β in [26] on the random regular graph and the grid, exploiting certain monotonicity properties.

Finally, let us note that the case of the grid has qualitatively different behaviour than the mean-field and the random-regular case. There, the three critical points coincide and the behaviour at criticality depends on the value of q; the mixing time of Glauber and SW has largely been detailed, see [7, 33, 24].

[3] Note that the interval-behaviour on the random regular graph (and hence the correspondence with the mean-field case) is already implied to some extent by the interval-result of [29] (for q large and $d \geq 5$). Note however that the interval therein is contained strictly inside (β_u, β_h) and, in particular, its endpoints do not have the probabilistic interpretation of β_u, β_h. Nevertheless, [29] obtains various probabilistic properties of the metastable phases, including a stronger form of correlation decay than that of reconstruction that we consider here.

2 Overview

In this section we give an overview of the proofs of Theorems 1.1–1.3; for now, we will mostly work towards the proof of Theorem 1.3 which gives the main insights/tools that are needed to prove Theorems 1.1 and 1.2.

Fortunately, to prove Theorem 1.3, we do not need to start from first principles. Instead, we build upon the formula for the partition function $Z_\beta(\mathbb{G})$ and its proof via the second moment method from [22]. Additionally, we are going to seize upon facts about the non-reconstruction properties of the Potts model on the random d-regular tree, also from [22]. We will combine these tools with an auxiliary random graph model known as the planted model, which also plays a key role in the context of inference problems on random graphs [15].

Throughout most of the paper, instead of the simple random regular graph $\mathbb{G}$, we will work with the random d-regular multi-graph $\boldsymbol{G} = \boldsymbol{G}(n,d)$ drawn from the pairing model. Recall that $\boldsymbol{G}$ is obtained by creating d clones of each of the vertices from $[n]$, choosing a random perfect matching of the complete graph on $[n] \times [d]$ and subsequently contracting the vertices $\{i\} \times [d]$ into a single vertex i, for all $i \in [n]$. It is well-known that $\mathbb{G}$ is contiguous with respect to $\boldsymbol{G}$ [30], i.e., any property that holds w.h.p. for $\boldsymbol{G}$ also holds w.h.p. for $\mathbb{G}$.

The following notation will be handy. For a graph G and a configuration $\sigma \in [q]^{V(G)}$, define a probability distribution ν^σ on $[q]$ by letting

$$\nu^\sigma(s) = |\sigma^{-1}(s)|/n \qquad (s \in [q]).$$

In words, ν^σ is the empirical distribution of the colours under σ. Similarly, let $\rho^{G,\sigma} \in \mathcal{P}([q] \times [q])$ be the edge statistics of a given graph/colouring pair, i.e.,

$$\rho^{G,\sigma}(s,t) = \frac{1}{2|E(G)|} \sum_{u,v \in V(G)} \mathbf{1}\{uv \in E(G),\, \sigma_u = s,\, \sigma_v = t\}.$$

2.1 Moments and messages

The routine method for investigating the partition function and the Boltzmann distribution of random graphs is the method of moments [2]. The basic strategy is to calculate, one way or another, the first two moments $\mathbb{E}[Z_\beta(\boldsymbol{G})]$, $\mathbb{E}[Z_\beta(\boldsymbol{G})^2]$ of the partition function. Then we cross our fingers that the second moment is not much larger than the square of the first. It sometimes works. But potential pitfalls include a pronounced tendency of running into extremely challenging optimisation problems in the course of the second moment calculation and, worse, lottery effects that may foil the strategy altogether. While regular graphs in general and the Potts ferromagnet in particular are relatively tame specimens, these difficulties actually do arise once we set out to investigate metastable states. Drawing upon [3, 16] to sidestep these challenges, we develop a less computation-heavy proof strategy.

The starting point is the observation that the fixed points of (3) are intimately related to the moment calculation. This will not come as a surprise to experts, and indeed it was already noticed in [22]. To elaborate, let $\nu = (\nu(\sigma))_{\sigma \in [q]}$ be a probability distribution on the q colours. Moreover, let $\mathcal{R}(\nu)$ be the set of all symmetric matrices $(\rho(\sigma,\tau))_{\sigma,\tau \in [q]}$ with non-negative entries such that

$$\sum_{\tau \in [q]} \rho(\sigma,\tau) = \nu(\sigma) \qquad \text{for all } \sigma \in [q]. \tag{6}$$

Relatively standard arguments (e.g., [12, Lemma 2.7]) show that the first moment satisfies

$$\lim_{n\to\infty}\frac{1}{n}\log\mathbb{E}[Z_\beta(\boldsymbol{G})] = \max_{\nu\in\mathcal{P}([q]),\rho\in R(\nu)} F_{d,\beta}(\nu,\rho), \qquad \text{where}$$
$$F_{d,\beta}(\nu,\rho) = (d-1)\sum_{\sigma\in[q]}\nu(\sigma)\log\nu(\sigma) - d\sum_{1\le\sigma\le\tau\le q}\rho(\sigma,\tau)\log\rho(\sigma,\tau) + \frac{d\beta}{2}\sum_{\sigma\in[q]}\rho(\sigma,\sigma). \tag{7}$$

Thus, the first moment is governed by the maximum or maxima, as the case may be, of $F_{d,\beta}$. The function $F_{d,\beta}$ accounts for the contribution to $\mathbb{E}[Z_\beta(\boldsymbol{G})]$ coming from the set $\mathcal{Q}$ which consists of pairs (G,σ) with $\nu^\sigma = \nu + o(1)$ and $\rho^{G,\sigma} = \rho + o(1)$, i.e., the sum $\sum_{(G,\sigma)\in\mathcal{Q}}\mathbb{P}[\boldsymbol{G}=G]\mathrm{e}^{\beta\mathcal{H}_G(\sigma)}$ equals $\mathrm{e}^{nF_{d,\beta}(\nu,\rho)+o(n)}$.

We need to know that the maxima of $F_{d,b}$ are in one-to-one correspondence with the stable fixed points of (3). To be precise, a fixed point μ of (3) is *stable* if the Jacobian of (3) at μ has spectral radius strictly less than one. Let $\mathcal{F}^+_{d,\beta}$ be the set of all stable fixed points $\mu\in\mathcal{F}_{d,\beta}$. Moreover, let $\mathcal{F}^1_{d,\beta}$ be the set of all $\mu\in\mathcal{F}^+_{d,\beta}$ such that $\mu(1) = \max_{\sigma\in[q]}\mu(\sigma)$. In addition, let us call a local maximum (ν,ρ) of $F_{d,\beta}$ *stable* if there exist $\delta, c>0$ such that

$$F_{d,\beta}(\nu',\rho') \le F_{d,\beta}(\nu,\rho) - c\left(\|\nu-\nu'\|^2 + \|\rho-\rho'\|^2\right) \tag{8}$$

for all $\nu'\in\mathcal{P}([q])$ and $\rho'\in\mathcal{R}(\nu')$ such that $\|\nu-\nu'\| + \|\rho-\rho'\| < \delta$. Roughly, (8) provides that the Hessian of $F_{d,\beta}$ is negative definite on the subspace of all possible ν,ρ.

▶ **Lemma 2.2** ([22, Theorem 8]). *Suppose that $d,q\ge 3$ are integers and $\beta>0$ is a real. The map $\mu\in\mathcal{P}([q])\mapsto(\nu^\mu,\rho^\mu)$ defined by*

$$\nu^\mu(\sigma) = \frac{(1+(\mathrm{e}^\beta-1)\mu(\sigma))^d}{\sum_{\tau\in[q]}(1+(\mathrm{e}^\beta-1)\mu(\tau))^d}, \qquad \rho^\mu(\sigma,\tau) = \frac{\mathrm{e}^{\beta\mathbf{1}\{\sigma=\tau\}}\mu(\sigma)\mu(\tau)}{1+(\mathrm{e}^\beta-1)\sum_{s\in[q]}\mu(s)^2} \tag{9}$$

is a bijection from $\mathcal{F}^+_{d,\beta}$ to the set of stable local maxima of $F_{d,\beta}$. Moreover, for any fixed point μ we have $\mathcal{B}_{d,\beta}(\mu) = F_{d,\beta}(\nu^\mu,\rho^\mu)$.

For brevity, let $(\nu_\mathrm{p},\rho_\mathrm{p}) = (\nu^{\mu_\mathrm{p}},\rho^{\mu_\mathrm{p}})$ and $(\nu_\mathrm{f},\rho_\mathrm{f}) = (\nu^{\mu_\mathrm{f}},\rho^{\mu_\mathrm{f}})$. The following result characterises the stable fixed points $\mathcal{F}^1_{d,\beta}$.

▶ **Proposition 2.3** ([22, Theorem 4]). *Suppose that $d\ge 3, \beta>0$.*

- **(i)** *If $\beta<\beta_u$, then (3) has a unique fixed point, namely the paramagnetic distribution ν_p on $[q]$. This fixed point is stable and thus $F_{d,\beta}$ attains its global maximum at $(\nu_\mathrm{p},\rho_\mathrm{p})$.*
- **(ii)** *If $\beta_u<\beta<\beta_h$, then $\mathcal{F}^1_{d,\beta}$ contains two elements, namely the paramagnetic distribution ν_p and the ferromagnetic distribution ν_f; $(\nu_\mathrm{p},\rho_\mathrm{p})$ is a global maximum of $F_{d,\beta}$ iff $\beta\le\beta_p$, and $(\nu_\mathrm{f},\rho_\mathrm{f})$ iff $\beta\ge\beta_p$.*
- **(iii)** *If $\beta>\beta_h$, then $\mathcal{F}^1_{d,\beta}$ contains precisely one element, namely the ferromagnetic distribution ν_f, and $(\nu_\mathrm{f},\rho_\mathrm{f})$ is a global maximum of $F_{d,\beta}$.*

Like the first moment, the second moment boils down to an optimisation problem as well, albeit one of much higher dimension (q^2-1 rather than $q-1$). Indeed, it is not difficult to derive the following approximation (once again, e.g., via [12, Lemma 2.7]). For a probability distribution $\nu\in\mathcal{P}([q])$ and a symmetric matrix $\rho\in\mathcal{R}(\nu)$ let $\mathcal{R}^\otimes(\rho)$ be the set of all tensors $r=(r(\sigma,\sigma',\tau,\tau'))_{\sigma,\sigma',\tau,\tau'\in[q]}$ such that $r(\sigma,\sigma',\tau,\tau') = r(\tau,\tau',\sigma,\sigma')$ for $\tau,\tau',\sigma,\sigma'\in[q]$ and

$$\sum_{\sigma',\tau'} r(\sigma,\sigma',\tau,\tau') = \sum_{\sigma',\tau'} r(\sigma',\sigma,\tau',\tau) = \rho(\sigma,\tau) \quad \text{for all } \sigma,\tau\in[q]. \tag{10}$$

Then, with $H(\cdot)$ denoting the entropy function, we have

$$\lim_{n\to\infty}\frac{1}{n}\log\mathbb{E}[(Z_\beta(\boldsymbol{G}))^2] = \max_{\nu,\rho\in\mathcal{R}(\nu),r\in\mathcal{R}^\otimes(\rho)} F^\otimes_{d,\beta}(\rho,r),\ \text{where} \tag{11}$$

$$F^\otimes_{d,\beta}(\rho,r) = (d-1)H(\rho)+\frac{d}{2}H(r)+\frac{d\beta}{2}\sum_{\sigma,\sigma',\tau,\tau'\in[q]}\big(\mathbf{1}\{\sigma=\tau\}+\mathbf{1}\{\sigma'=\tau'\}\big)\,r(\sigma,\sigma',\tau,\tau').$$

A frontal assault on this optimisation problem is in general a daunting task due to the doubly-stochastic constraints in (10), i.e., the constraint $r\in\mathcal{R}^\otimes(\rho)$. But rather fortunately, to analyse the global maximum (over ν and ρ), these constraints can be relaxed, permitting an elegant translation of the problem to operator theory. In effect, the second moment computation can be reduced to a study of matrix norms. The result is summarised as follows.

▶ **Proposition 2.4** ([22, Theorem 7]). *For all $d,q\geq 3$ and $\beta>0$ we have*

$$\max_{\nu,\rho\in\mathcal{R}(\nu),r\in\mathcal{R}^\otimes(\rho)} F^\otimes_{d,\beta}(\rho,r) = 2\max_{\nu,\rho}F_{d,\beta}(\nu,\rho)$$

and thus $\mathbb{E}[Z_\beta(\boldsymbol{G})^2]=O(\mathbb{E}[Z_\beta(\boldsymbol{G})]^2)$.

Combining Lemma 2.2, Proposition 2.3 and Proposition 2.4, we obtain the following reformulation of [22, Theorem 7], which verifies that we obtain good approximations to the partition function by maximising the Bethe free energy on $\mathcal{F}_{d,\beta}$.

▶ **Theorem 2.5.** *For all integers $d,q\geq 3$ and real $\beta>0$, we have $\lim_{n\to\infty}n^{-1}\log Z_\beta(\mathbb{G}) = \max_{\mu\in\mathcal{F}_{d,\beta}}\mathcal{B}_{d,\beta}(\mu)$ in probability.*

While the *global* maximisation of the function $F^\otimes_{d,\beta}$ and thus the proof of Theorem 2.5 boils down to matrix norm analysis, in order to prove Theorems 1.3 and 1.1 via the method of moments we would in addition need to get a good handle on all the *local* maxima. Unfortunately, we do not see a way to reduce this more refined question to operator norms (and it seems unlikely that one exists). Hence, it would seem that we should have to perform a fine-grained analysis of $F^\otimes_{d,\beta}$ after all. But luckily another path is open to us. Instead of proceeding analytically, we resort to probabilistic ideas. and we harness "quiet-planting" arguments with the notion of non-reconstruction on the Potts model on the d-regular tree. We review the latter in the next section.

2.2 Non-reconstruction on the regular tree

Let $\mathbb{T}_d$ be the infinite d-regular tree with root o. For a probability distribution $\mu\in\{\mu_\mathrm{p},\mu_\mathrm{f}\}$ we define a broadcasting process $\boldsymbol{\sigma}=\boldsymbol{\sigma}_{d,\beta,\mu}$ on $\mathbb{T}_d$ as follows. Initially we draw the color $\boldsymbol{\sigma}_o$ of the root o from the distribution ν^μ. Subsequently, working our way down the levels of the tree, the color of a vertex v whose parent u has been coloured already is drawn from the distribution

$$\mathbb{P}\left[\boldsymbol{\sigma}_v=\sigma\mid\boldsymbol{\sigma}_u\right]=\frac{\mu(\sigma)\mathrm{e}^{\beta\mathbf{1}\{\sigma=\boldsymbol{\sigma}_u\}}}{\sum_{\tau\in[q]}\mu(\tau)\mathrm{e}^{\beta\mathbf{1}\{\tau=\boldsymbol{\sigma}_u\}}}.$$

Naturally, the colours of different vertices on the same level are pairwise independent, but not jointly since there is potentially some correlation with the root. Let $\partial^\ell o$ be the set of all vertices at distance precisely ℓ from o. We say that the broadcasting process has the *strong non-reconstruction property* if $\sum_{\tau\in[q]}\mathbb{E}\Big[\big|\mathbb{P}\left[\boldsymbol{\sigma}_o=\tau\mid\boldsymbol{\sigma}_{\partial^\ell o}\right]-\mathbb{P}\left[\boldsymbol{\sigma}_o=\tau\right]\big|\Big]=\mathrm{e}^{-\Omega(\ell)}$, where the expectation is over the random configuration $\boldsymbol{\sigma}_{\partial^\ell o}$ (distributed according to the broadcasting

process). In words, this says that the information about the spin of the root inferred from the spins of vertices at depth ℓ decays in the broadcasting process; the term "strong" refers that the decay is exponential with respect to the depth ℓ.

▶ **Proposition 2.6** ([22, Theorem 50]). *Let $d, q \geq 3$ be integers and $\beta > 0$ be real.*
(i) *For $\beta < \beta_h$, the broadcasting process $\boldsymbol{\sigma}_{d,\beta,\mu_{\mathrm{p}}}$ has the strong non-reconstruction property.*
(ii) *For $\beta > \beta_u$, the broadcasting process $\boldsymbol{\sigma}_{d,\beta,\mu_{\mathrm{f}}}$ has the strong non-reconstruction property.*

In order to prove Theorems 1.1–1.3 we will combine Proposition 2.6 with reweighted random graph models known as planted models. To be precise, we will consider two versions of planted models, a paramagnetic and a ferromagnetic one. Then we will deduce from Proposition 2.6 that the Boltzmann distribution of these planted models has the non-reconstruction property in a suitably defined sense. In combination with some general facts about Boltzmann distributions this will enable us to prove Theorems 1.1–1.3 without the need for complicated moment computations.

2.3 Second Moment via planting and non-reconstruction

We proceed to introduce the paramagnetic and the ferromagnetic version of the planted model. Roughly speaking, these are weighted versions of the common random regular graph $\mathbb{G}$ where the probability mass of a specific graph is proportional to the paramagnetic or ferromagnetic bit of the partition function. To be precise, for $\varepsilon > 0$, recall the subsets $S_{\mathrm{p}} = S_{\mathrm{p}}(\varepsilon), S_{\mathrm{f}} = S_{\mathrm{f}}(\varepsilon)$ of the configuration space $[q]^n$. Letting

$$Z_{\mathrm{f}}(G) = \sum_{\sigma \in S_{\mathrm{f}}} \mathrm{e}^{\beta \mathcal{H}_G(\sigma)} \text{ and } Z_{\mathrm{p}}(G) = \sum_{\sigma \in S_{\mathrm{p}}} \mathrm{e}^{\beta \mathcal{H}_{\boldsymbol{G}}(\sigma)}, \tag{12}$$

we define random graph models $\hat{\boldsymbol{G}}_{\mathrm{f}}, \hat{\boldsymbol{G}}_{\mathrm{p}}$ by

$$\mathbb{P}\left[\hat{\boldsymbol{G}}_{\mathrm{f}} = G\right] = \frac{Z_{\mathrm{f}}(G)\mathbb{P}\left[\boldsymbol{G} = G\right]}{\mathbb{E}[Z_{\mathrm{f}}(\boldsymbol{G})]}, \qquad \mathbb{P}\left[\hat{\boldsymbol{G}}_{\mathrm{p}} = G\right] = \frac{Z_{\mathrm{p}}(G)\mathbb{P}\left[\boldsymbol{G} = G\right]}{\mathbb{E}[Z_{\mathrm{p}}(\boldsymbol{G})]}. \tag{13}$$

Thus, $\hat{\boldsymbol{G}}_{\mathrm{f}}$ and $\hat{\boldsymbol{G}}_{\mathrm{p}}$ are d-regular random graphs on n vertices such that the probability that a specific graph G comes up is proportional to $Z_{\mathrm{f}}(G)$ and $Z_{\mathrm{p}}(G)$, respectively. Note, the expected value of $Z_{\mathrm{f}}(G)$ and $Z_{\mathrm{p}}(G)$ is captured by the function $F_{d,\beta}$, and we have (see Lemmas 3.2 and 3.3 in the full version)

$$\mathbb{E}[Z_{\mathrm{p}}(G)] = n^{O(1)} \exp(nF_{d,\beta}(\nu_{\mathrm{p}}, \rho_{\mathrm{p}})) \quad \text{and} \quad \mathbb{E}[Z_{\mathrm{f}}(G)] = n^{O(1)} \exp(nF_{d,\beta}(\nu_{\mathrm{f}}, \rho_{\mathrm{f}})). \tag{14}$$

The key ingredient to prove Theorem 1.3 is to quantify the overlap of two typical configurations in the conditional Boltzmann distributions (under S_{f} and S_{p}). To be precise, for a graph $G = (V, E)$, the overlap of two configurations $\sigma, \sigma' \in [q]^V$ is defined as the probability distribution $\nu(\sigma, \sigma') \in \mathcal{P}([q]^2)$ with

$$\nu_{c,c'}(\sigma, \sigma') = \frac{1}{n} \sum_{v \in V(G)} \mathbf{1}\{\sigma_v = c,\, \sigma'_v = c'\} \qquad (c, c' \in [q]).$$

For a graph G let $\boldsymbol{\sigma}_{G,\mathrm{f}}$ denote a sample from the conditional distribution $\mu_{G,\beta}(\cdot \mid S_{\mathrm{f}})$. and define $\boldsymbol{\sigma}_{G,\mathrm{p}}$ similarly for S_{p}. The following lemma studies the overlap for two configurations in the conditional distribution $\mu_{\hat{\boldsymbol{G}}_{\mathrm{p}},\beta}(\cdot \mid S_{\mathrm{p}})$, a similar lemma applies to the ferromagnetic phase S_{f}, see Lemma 3.9 in the full version.

▶ **Lemma 3.8.** *Let $d, q \geq 3$ be integers and $\beta < \beta_h$ be real. Let $\boldsymbol{\sigma}_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}, \boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}$ be independent samples from $\mu_{\hat{\boldsymbol{G}}_{\mathrm{p}},\beta}(\cdot \mid S_{\mathrm{p}})$. Then $\mathbb{E}\big[d_{\mathrm{TV}}\big(\nu(\boldsymbol{\sigma}_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}, \boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}), \nu_{\mathrm{p}} \otimes \nu_{\mathrm{p}}\big)\big] = o(1)$.*

To utilise Lemmas 3.8 and 3.9, we proceed to apply the second moment method to truncated versions of the paramagnetic and ferromagnetic partition functions $Z_{\mathrm{p}}, Z_{\mathrm{f}}$ where we expressly drop graphs that violate the overlap bounds from Lemmas 3.8. Thus, we introduce the events $\mathcal{E}_{\mathrm{p}} = \{G : \mathbb{E}\big[d_{\mathrm{TV}}(\nu(\boldsymbol{\sigma}_{G,\mathrm{p}}, \boldsymbol{\sigma}'_{G,\mathrm{p}}), \nu_{\mathrm{p}} \otimes \nu_{\mathrm{p}})\big] = o(1)\}$ and the analogous event $\mathcal{E}_{\mathrm{f}}$ for the ferromagnetic phase. Consider now the random variables

$$Y_{\mathrm{p}}(G) = Z_{\mathrm{p}}(G) \cdot \mathbf{1}\{G \in \mathcal{E}_{\mathrm{p}}\} \text{ and } Y_{\mathrm{f}}(G) = Z_{\mathrm{f}}(G) \cdot \mathbf{1}\{G \in \mathcal{E}_{\mathrm{f}}\}$$

Combining Lemma 3.8 with the Nishimori identity (17), we obtain

$$\frac{\mathbb{E}[Y_{\mathrm{p}}]}{\mathbb{E}[Z_{\mathrm{p}}]} = \mathbb{P}\left[\hat{\boldsymbol{G}}_{\mathrm{p}} \in \mathcal{E}_{\mathrm{p}}\right] \sim 1 \text{ and } \frac{\mathbb{E}[Y_{\mathrm{f}}]}{\mathbb{E}[Z_{\mathrm{f}}]} = \mathbb{P}\left[\hat{\boldsymbol{G}}_{\mathrm{f}} \in \mathcal{E}_{\mathrm{f}}\right] \sim 1$$

and thus $\mathbb{E}[Y_{\mathrm{p}}] \sim \mathbb{E}[Z_{\mathrm{p}}]$ and $\mathbb{E}[Y_{\mathrm{f}}] \sim \mathbb{E}[Z_{\mathrm{f}}]$. Crucially, estimating the second moments of these two random variables is a cinch because by construction we can avoid an explicit optimisation of the function $F^{\otimes}_{d,\beta}$ from (11). Indeed, because we drop graphs G whose overlaps stray far from the product measures $\nu_{\mathrm{p}} \otimes \nu_{\mathrm{p}}$ and $\nu_{\mathrm{f}} \otimes \nu_{\mathrm{f}}$, respectively, we basically just need to evaluate the function $F^{\otimes}_{d,\beta}$ at $\nu_{\mathrm{p}} \otimes \nu_{\mathrm{p}}$ and $\nu_{\mathrm{f}} \otimes \nu_{\mathrm{f}}$, which is a matter of relatively simple algebra (due to convexity arguments). We thus obtain the following.

▶ **Corollary 3.10.** *Let $d, q \geq 3$ be integers and $\beta > 0$ be real.*
(i) *If $\beta < \beta_h$, then $\mathbb{E}[Y_{\mathrm{p}}(\boldsymbol{G})] \sim \mathbb{E}[Z_{\mathrm{p}}(\boldsymbol{G})]$ and $\mathbb{E}[Y_{\mathrm{p}}(\boldsymbol{G})^2] \leq \exp(o(n))\mathbb{E}[Z_{\mathrm{p}}(\boldsymbol{G})]^2$.*
(ii) *If $\beta > \beta_u$, then $\mathbb{E}[Y_{\mathrm{f}}(\boldsymbol{G})] \sim \mathbb{E}[Z_{\mathrm{f}}(\boldsymbol{G})]$ and $\mathbb{E}[Y_{\mathrm{f}}(\boldsymbol{G})^2] \leq \exp(o(n))\mathbb{E}[Z_{\mathrm{f}}(\boldsymbol{G})]^2$.*

At this stage, one can combine Corollary 3.10 together with (14) to derive the first two parts of Theorem 1.3 (using also the results from Section 2.1).

3 Quiet planting and non-reconstruction

In this section we give an outline of the proof of Lemma 3.8, which was the main ingredient to carry out the second moment method of Section 2.3.

While the planted models defined in (13) are useful for the second-moment argument, working with them directly is rather unwieldy. Fortunately, there is a relatively simple way out using the so-called Nishimori identities; on the way, we will also introduce some of the ingredients that are used for the metastability/slow-mixing results.

To elaborate, we complement the definition (13) of the planted random graphs $\hat{\boldsymbol{G}}_{\mathrm{f}}, \hat{\boldsymbol{G}}_{\mathrm{p}}$ by also introducing a reweighted distribution on graphs for a specific configuration $\sigma \in [q]^n$. Specifically, we define a random graph $\hat{\boldsymbol{G}}(\sigma)$ by letting

$$\mathbb{P}\left[\hat{\boldsymbol{G}}(\sigma) = G\right] = \frac{\mathbb{P}\left[\boldsymbol{G} = G\right] \mathrm{e}^{\beta \mathcal{H}_G(\sigma)}}{\mathbb{E}[\mathrm{e}^{\beta \mathcal{H}_{\boldsymbol{G}}(\sigma)}]}. \tag{15}$$

Furthermore, recalling the truncated partition functions $Z_{\mathrm{f}}, Z_{\mathrm{p}}$ from (12), we introduce reweighted random configurations $\hat{\boldsymbol{\sigma}}_{\mathrm{f}} = \hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon) \in [q]^n$ and $\hat{\boldsymbol{\sigma}}_{\mathrm{p}} = \hat{\boldsymbol{\sigma}}_{\mathrm{p}}(\varepsilon) \in [q]^n$ with distributions

$$\mathbb{P}\left[\hat{\boldsymbol{\sigma}}_{\mathrm{f}} = \sigma\right] = \frac{\mathbf{1}\{\sigma \in S_{\mathrm{f}}\}\, \mathbb{E}[\mathrm{e}^{\beta \mathcal{H}_{\boldsymbol{G}}(\sigma)}]}{\mathbb{E}[Z_{\mathrm{f}}(\boldsymbol{G})]}, \qquad \mathbb{P}\left[\hat{\boldsymbol{\sigma}}_{\mathrm{p}} = \sigma\right] = \frac{\mathbf{1}\{\sigma \in S_{\mathrm{p}}\}\, \mathbb{E}[\mathrm{e}^{\beta \mathcal{H}_{\boldsymbol{G}}(\sigma)}]}{\mathbb{E}[Z_{\mathrm{p}}(\boldsymbol{G})]}. \tag{16}$$

We have the following paramagnetic and ferromagnetic *Nishimori identities*. Nishimori identities were derived in [15] for a broad family of planted models which, however, does not include the planted ferromagnetic models $\hat{\boldsymbol{G}}_{\mathrm{p}}, \hat{\boldsymbol{G}}_{\mathrm{f}}$. Nonetheless, the (simple) proof of Proposition 3.1 is practically identical to that in [15] (and is given in the full version).

▶ **Proposition 3.1.** *We have the distributional equalities*

$$(\hat{\boldsymbol{G}}_{\mathrm{p}}, \boldsymbol{\sigma}_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}) \stackrel{d}{=} (\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}), \hat{\boldsymbol{\sigma}}_{\mathrm{p}}), \qquad (\hat{\boldsymbol{G}}_{\mathrm{f}}, \boldsymbol{\sigma}_{\hat{\boldsymbol{G}}_{\mathrm{f}},\mathrm{f}}) \stackrel{d}{=} (\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}), \hat{\boldsymbol{\sigma}}_{\mathrm{f}}). \tag{17}$$

Proposition 3.1 paves the way for a more hands-on description of the planted models in (13). Indeed, the random graph models $(\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}), \hat{\boldsymbol{\sigma}}_{\mathrm{p}})$ and $(\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}), \hat{\boldsymbol{\sigma}}_{\mathrm{f}})$ are invariant under permutations of the vertices, so $\hat{\boldsymbol{\sigma}}_{\mathrm{p}}$ and $\hat{\boldsymbol{\sigma}}_{\mathrm{f}}$ are uniformly random given their colour statistics, and the random graphs $\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}})$ and $\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{f}})$ themselves are uniformly random and easy to sample given the planted assignment $\hat{\boldsymbol{\sigma}}_{\mathrm{p}}$ or $\hat{\boldsymbol{\sigma}}_{\mathrm{f}}$ and given the edge statistics $\rho^{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}$ and $\rho^{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}),\hat{\boldsymbol{\sigma}}_{\mathrm{f}}}$. Moreover, because $(\nu_{\mathrm{p}}, \rho_{\mathrm{p}})$ and $(\nu_{\mathrm{f}}, \rho_{\mathrm{f}})$ are local maxima of the first moment function $F_{d,b}(\nu, \rho)$, a first moment argument based on (8) allows us to control the vertex-edge colour statistics very accurately, i.e., there exist $c, t_0 > 0$ such that for all $t \in [0, t_0]$

$$\mathbb{P}\left[d_{\mathrm{TV}}\left(\nu^{\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}, \nu_{\mathrm{p}}\right) + d_{\mathrm{TV}}\left(\rho^{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}, \rho_{\mathrm{p}}\right) > t\right] \leq n^{O(1)}\mathrm{e}^{-ct^2 n}, \tag{18}$$

and similarly for the deviations from $(\nu_{\mathrm{f}}, \rho_{\mathrm{f}})$ in the ferromagnetic phase (see Lemmas 3.4 and 3.5 in the full version). At this point we have handy descriptions of the models $(\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}), \hat{\boldsymbol{\sigma}}_{\mathrm{p}})$ and $(\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}), \hat{\boldsymbol{\sigma}}_{\mathrm{f}})$, and therefore, via Proposition 3.1, $(\hat{\boldsymbol{G}}_{\mathrm{p}}, \boldsymbol{\sigma}_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}})$ and $(\hat{\boldsymbol{G}}_{\mathrm{f}}, \boldsymbol{\sigma}_{\hat{\boldsymbol{G}}_{\mathrm{f}},\mathrm{f}})$.

We will next utilise the information on the distribution of $\hat{\boldsymbol{\sigma}}_{\mathrm{p}}, \rho^{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}$ to couple the distribution of the colouring produced by the tree broadcasting process and the colouring that $\hat{\boldsymbol{\sigma}}_{\mathrm{p}}$ induces on the neighbourhood of some particular vertex of $\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}})$, say v. In particular, for $\ell = \lceil \log\log n \rceil$, the ℓ-neighbourhood of v is going to be tree-like, so conditional on the statistics $\nu^{\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}, \rho^{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}$, an inductive coupling (see Lemma 3.6) shows that

$$d_{\mathrm{TV}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p},\partial^\ell v}, \tau_{\partial^\ell o}) = d^\ell\left(d_{\mathrm{TV}}(\nu^{\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}, \nu_{\mathrm{p}}) + d_{\mathrm{TV}}(\rho^{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\hat{\boldsymbol{\sigma}}_{\mathrm{p}}}, \rho_{\mathrm{p}}) + n^{-0.99}\right).$$

From (18), it then follows that the last quantity is $o(n^{-1/5})$ with probability $1 - o(1/n)$. Hence, the colourings $\hat{\boldsymbol{\sigma}}_{\mathrm{p},\partial^\ell v}$ and $\tau_{\partial^\ell o}$ can be coupled such that both are identical with probability $1 - o(n^{-1/5})$. Consequently, from the tree broadcasting results of Proposition 2.6, we obtain that $\sum_{c\in[q]} \mathbb{E}\left|\nu_{\mathrm{p}}(c) - \mu_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\beta}(\boldsymbol{\sigma}_v = c \mid \boldsymbol{\sigma}_{\partial^\ell v} = \hat{\boldsymbol{\sigma}}_{\mathrm{p},\partial^\ell v})\right| < \ell^{-3}$ which translates via the Nishimori identity into

$$\sum_{c\in[q]} \mathbb{E}\left|\nu_{\mathrm{p}}(c) - \mu_{\hat{\boldsymbol{G}}_{\mathrm{p}},\beta}(\boldsymbol{\sigma}_v = c \mid \boldsymbol{\sigma}_{\partial^\ell v} = \boldsymbol{\sigma}_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p},\partial^\ell v})\right| < \ell^{-3}. \tag{19}$$

Proof Sketch of Lemma 3.8. Due to the Nishimori identity (17), it suffices to prove that for a sample $\boldsymbol{\sigma}_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\mathrm{p}}$ from $\mu_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\beta}(\cdot \mid S_{\mathrm{p}})$ that

$$d_{\mathrm{TV}}\big(\nu(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}, \boldsymbol{\sigma}_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\mathrm{p}}), \nu_{\mathrm{p}} \otimes \nu_{\mathrm{p}}\big) = o(1). \tag{20}$$

To see (20), for colors $s, t \in [q]$, we consider the first and second moment of the number of vertices u with $\hat{\boldsymbol{\sigma}}_{\mathrm{p}}(u) = s$ and $\boldsymbol{\sigma}_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\mathrm{p}}(u) = t$. To facilitate the analysis of the second moment, it will be convenient to consider the following configuration $\boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\mathrm{p}}$. Let $\boldsymbol{v}, \boldsymbol{w}$ be two random vertices such that $\hat{\boldsymbol{\sigma}}_{\mathrm{p}}(\boldsymbol{v}) = \hat{\boldsymbol{\sigma}}_{\mathrm{p}}(\boldsymbol{w}) = s$. Also let $\ell = \ell(n) = \lceil \log\log n \rceil$. Now, draw $\boldsymbol{\sigma}''_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\mathrm{p}}$ from $\mu_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\beta}(\cdot \mid S_{\mathrm{p}})$ and subsequently generate $\boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\mathrm{p}}$ by re-sampling the colours of the vertices at distance less than ℓ from $\boldsymbol{v}, \boldsymbol{w}$ given the colours of the vertices at distance ℓ from $\boldsymbol{v}, \boldsymbol{w}$ and the event S_{p}. Then $\boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\mathrm{p}}$ has distribution $\mu_{\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}),\beta}(\cdot \mid S_{\mathrm{p}})$. Moreover, since for two random vertices $\boldsymbol{v}, \boldsymbol{w}$ their ℓ-neighbourhoods are going to be disjoint w.h.p., the reconstruction property in (19) implies that w.h.p. for all $\chi, \chi' \in [q]$

$$\mathbb{P}\left[\boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}(\boldsymbol{v}) = \chi, \boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}(\boldsymbol{w}) = \chi' \mid \hat{\boldsymbol{\sigma}}_{\mathrm{p}}, \hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}), \boldsymbol{v}, \boldsymbol{w}\right] = \nu_p(\chi)\nu_p(\chi') + o(1). \tag{21}$$

Hence, for a colour $t \in [q]$ let $\boldsymbol{X}(s,t)$ be the number of vertices u with $\hat{\boldsymbol{\sigma}}_{\mathrm{p}}(u) = s$ and $\boldsymbol{\sigma}'_{\hat{\boldsymbol{G}}_{\mathrm{p}},\mathrm{p}}(\boldsymbol{v}) = t$. Then (21) shows that w.h.p. $\mathbb{E}\left[\boldsymbol{X}(s,t) \mid \hat{\boldsymbol{\sigma}}_{\mathrm{p}}, \hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}})\right] \sim \frac{n}{q^2}$ and $\mathbb{E}\left[\boldsymbol{X}(s,t)^2 \mid \hat{\boldsymbol{\sigma}}_{\mathrm{p}}, \hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{p}})\right] \sim \frac{n^2}{q^4}$. Thus, (20) follows from Chebyshev's inequality. ◀

4 Metastability and Slow mixing

In this section, we prove Theorems 1.1 and 1.2. Recall from Section 1.3 the paramagnetic and ferromagnetic states $S_{\mathrm{p}}(\varepsilon)$ and $S_{\mathrm{f}}(\varepsilon)$ for $\varepsilon > 0$. For the purposes of this section we will need to be more systematic of keeping track the dependence of these phases on ε. In particular, we will use the more explicit notation $Z^{\varepsilon}_{\mathrm{p}}(G)$ and $Z^{\varepsilon}_{\mathrm{f}}(G)$ to denote the quantities $Z_{\mathrm{p}}(G)$ and $Z_{\mathrm{f}}(G)$, respectively, from (12). The following lemma, based on Theorem 1.3, reflects the fact that ν_{p} and ν_{f} are local maxima of the first-moment function $F_{d,\beta}$.

▶ **Lemma 4.1.** *Let $q, d \geq 3$ be integers and $\beta > 0$ be real. Then, for all sufficiently small constants $\varepsilon' > \varepsilon > 0$, there exists constant $\zeta > 0$ such that w.h.p. over $G \sim \boldsymbol{G}$, it holds that*

1. *If $\beta < \beta_h$, then $Z^{\varepsilon}_{\mathrm{p}}(G) \geq \mathrm{e}^{-n^{3/4}}\mathbb{E}[Z^{\varepsilon}_{\mathrm{p}}(\boldsymbol{G})]$ and $Z^{\varepsilon'}_{\mathrm{p}}(G) \leq (1+\mathrm{e}^{-\zeta n})Z^{\varepsilon}_{\mathrm{p}}(G)$.*
2. *If $\beta > \beta_u$, then $Z^{\varepsilon}_{\mathrm{f}}(G) \geq \mathrm{e}^{-n^{3/4}}\mathbb{E}[Z^{\varepsilon}_{\mathrm{f}}(\boldsymbol{G})]$ and $Z^{\varepsilon'}_{\mathrm{f}}(G) \leq (1+\mathrm{e}^{-\zeta n})Z^{\varepsilon}_{\mathrm{f}}(G)$.*

Theorem 1.1 will follow by way of a conductance argument. Let $G = (V, E)$ be a graph, and P be the transition matrix for the Glauber dynamics defined in Section 1.4. For a set $S \subseteq [q]^V$ define the *bottleneck ratio* of S to be $\Phi(S) = \frac{\sum_{\sigma\in S,\, \tau\notin S}\mu_{G,\beta}(\sigma)P(\sigma,\tau)}{\mu_{G,\beta}(S)}$. The following lemma provides a routine conductance bound (e.g., [32, Theorem 7.3]). For the sake of completeness the proof is included in the full version.

▶ **Lemma 4.2.** *Let $G = (V, E)$ be a graph. For any $S \subseteq [q]^V$ such that $\mu_G(S) > 0$ and any integer $t \geq 0$ we have $\|\mu_{G,S}P^t - \mu_{G,S}\|_{TV} \leq t\Phi(S)$.*

Proof of Theorem 1.1. We prove the statement for the pairing model $\boldsymbol{G}$, the result for $\mathbb{G}$ follows immediately by contiguity. Let $\varepsilon' > \varepsilon > 0$ and $\zeta > 0$ be small constants such that Lemma 4.1 applies, and let $G \sim \boldsymbol{G}$ be a graph satisfying the lemma. Set for convenience $\mu = \mu_{\boldsymbol{G},\beta}$; we consider first the metastability of $S_{\mathrm{f}}(\varepsilon)$ for $\beta > \beta_u$.

Since Glauber updates one vertex at a time it is impossible in one step to move from $\sigma \in S_{\mathrm{f}}(\varepsilon)$ to $\tau \in [q]^n \backslash S_{\mathrm{f}}(\varepsilon')$, i.e., $P(\sigma,\tau) = 0$, and therefore

$$\Phi\big(S_{\mathrm{f}}(\varepsilon)\big) = \frac{\sum_{\sigma\in S_{\mathrm{f}}(\varepsilon)}\sum_{\tau\notin S_{\mathrm{f}}(\varepsilon)}\mu(\sigma)P(\sigma,\tau)}{\mu\big(S_{\mathrm{f}}(\varepsilon)\big)} = \frac{\sum_{\sigma\in S_{\mathrm{f}}(\varepsilon)}\sum_{\tau\in S_{\mathrm{f}}(\varepsilon')\backslash S_{\mathrm{f}}(\varepsilon')}\mu(\sigma)P(\sigma,\tau)}{\mu\big(S_{\mathrm{f}}(\varepsilon)\big)}$$

By reversibility of Glauber, for any $\sigma, \tau \in [q]^n$ we have $\mu(\sigma)P(\sigma,\tau) = \mu(\tau)P(\tau,\sigma)$, and therefore the numerator is upper-bounded by $\mu\big(S_{\mathrm{f}}(\varepsilon')\backslash S_{\mathrm{f}}(\varepsilon)\big)$. Hence, $\Phi\big(S_{\mathrm{f}}(\varepsilon)\big) \leq \frac{\mu\left(S_{\mathrm{f}}(\varepsilon')\backslash S_{\mathrm{f}}(\varepsilon)\right)}{\mu\left(S_{\mathrm{f}}(\varepsilon)\right)} = \frac{Z^{\varepsilon'}_{\mathrm{f}}(G)-Z^{\varepsilon}_{\mathrm{f}}(G)}{Z^{\varepsilon}_{\mathrm{f}}(G)} \leq \mathrm{e}^{-\zeta n}$, where the last inequality follows from the fact that G satisfies Lemma 4.1. Lemma 4.2 now ensures that for all nonnegative integers $T \leq \mathrm{e}^{\zeta n/3}$

$$\big\|\mu\big(\cdot \mid S_{\mathrm{f}}(\varepsilon)\big)P^T - \mu\big(\cdot \mid S_{\mathrm{f}}(\varepsilon)\big)\big\|_{TV} \leq T \cdot \Phi(S_{\mathrm{f}}) \leq \mathrm{e}^{-2\zeta n/3}. \tag{22}$$

Now, consider the Glauber dynamics $(\sigma_t)_{t\geq 0}$ launched from σ_0 drawn from $\mu_{\mathbb{G},\beta,S_{\mathrm{f}}(\varepsilon)}$, and denote by $T_{\mathrm{f}} = \min\{t > 0 : \sigma_t \notin S_{\mathrm{f}}(\varepsilon)\}$ its escape time from $S_{\mathrm{f}}(\varepsilon)$. Observe that σ_t has the same distribution as $\mu(\cdot \mid S_{\mathrm{f}}(\varepsilon))P^t$, so (22) implies that for all nonnegative integers $T \leq \mathrm{e}^{\zeta n/3}$ it holds that $\big|\mathbb{P}[\sigma_T \in S_{\mathrm{f}}(\varepsilon)] - 1\big| < \mathrm{e}^{-2\zeta n/3}$, or equivalently $\mathbb{P}[\sigma_T \notin S_{\mathrm{f}}(\varepsilon)] \leq \mathrm{e}^{-2\zeta n/3}$. By a

union bound over the values of T, we therefore obtain that $\mathbb{P}[T_{\mathrm{f}} \leq \mathrm{e}^{\zeta n/3}] \leq \mathrm{e}^{-\zeta n/3}$, thus proving that $S_{\mathrm{f}}(\varepsilon)$ is a metastable state for $\beta > \beta_u$. Analogous arguments show that $S_{\mathrm{p}}(\varepsilon)$ is a metastable state for $\beta < \beta_h$.

The slow mixing of Glauber for $\beta > \beta_u$ follows from the metastability of $S_{\mathrm{f}}(\varepsilon)$. In particular, from Theorem 1.3 we have that $\big\|\mu\big(\cdot \mid S_{\mathrm{f}}(\varepsilon)\big) - \mu\big\| \geq 3/5$ and therefore, from (22), $\big\|\mu\big(\cdot \mid S_{\mathrm{f}}(\varepsilon)\big)P^T - \mu\big\| \geq 1/2$, yielding that the mixing time is $\mathrm{e}^{\Omega(n)}$. ◀

The key and much more challenging ingredient to establish Theorem 1.2 is to bound the probability that Swendsen-Wang escapes $S_{\mathrm{p}}(\varepsilon)$ and $S_{\mathrm{f}}(\varepsilon)$. More precisely, for a graph G, a configuration $\sigma \in [q]^n$, and $S \subseteq [q]^n$, let $P^G_{SW}(\sigma \to S)$ denote the probability that after one step of SW on G starting from σ, we end up in a configuration in S.

The following proposition shows that for almost all pairs (G, σ) from the ferromagnetic planted distribution $\big(\hat{\boldsymbol{G}}\big(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big), \hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big)$, the probability that SW leads to a configuration in the ferromagnetic phase, slightly enlarged, is $1-\mathrm{e}^{-\Omega(n)}$. Note here that SW might change the dominant colour due to recolouring step, so, for $\varepsilon > 0$, we consider the set of configurations $\tilde{S}_{\mathrm{f}}(\varepsilon)$ that consists of the ferromagnetic phase $S_{\mathrm{f}}(\varepsilon)$ together with its $q-1$ permutations, and the probability that SW escapes from it, starting from a ferromagnetic state.

▶ **Proposition 4.4.** *Let $q, d \geq 3$ be integers and $\beta \in (\beta_u, \beta_h)$. Then, for all sufficiently small constants $\varepsilon' > \varepsilon > 0$, there exists constant $\eta > 0$ such that with probability $1 - \mathrm{e}^{-\eta n}$ over the planted distribution $(G, \sigma) \sim \big(\hat{\boldsymbol{G}}\big(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big), \hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big)$, it holds that $P^G_{SW}\big(\sigma \to \tilde{S}_{\mathrm{f}}(\varepsilon')\big) \geq 1 - \mathrm{e}^{-\eta n}$.*

An analogous Proposition 4.3 applies for the paramagnetic distribution $\big(\hat{\boldsymbol{G}}\big(\hat{\boldsymbol{\sigma}}_{\mathrm{p}}(\varepsilon)\big), \hat{\boldsymbol{\sigma}}_{\mathrm{p}}(\varepsilon)\big)$. The proof of these Propositions requires a delicate analysis of the percolation step in SW since we need probability bounds that are exponentially close to 1. Especially for Proposition 4.4, the presence of a giant component (corresponding to the dominant colour) complicates the arguments significantly since we need to take into account the underlying vertex-edge colour statistics of $\big(\hat{\boldsymbol{G}}\big(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big), \hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big)\big)$ studied in Section 3. Even with Propositions 4.3 and 4.4 at hand, concluding Theorem 1.2 requires a bit of work based on the planting ideas.

Proof Sketch of Theorem 1.2. We consider first the metastability for the ferromagnetic phase when $\beta > \beta_u$. Let $\varepsilon' > \varepsilon > 0$ and $\eta, \zeta > 0$ be small constants such that Lemma 4.1 and Proposition 4.4 apply. Let $\theta = \frac{1}{10}\min\{\eta, \zeta\}$.

Let $\mathcal{Q}$ be the set of d-regular (multi)graphs that satisfy both items in Lemma 4.1. Moreover, let $\mathcal{Q}'$ be the set of d-regular (multi)graphs G such that the set of configurations where SW has conceivable probability of escaping $\tilde{S}_{\mathrm{f}}(\varepsilon')$ has small weight, i.e., the set

$$S_{\mathrm{Bad}}(G) = \big\{\sigma \in \tilde{S}_{\mathrm{f}}(\varepsilon) \,\big|\, P^G_{SW}\big(\sigma \to \tilde{S}_{\mathrm{f}}(\varepsilon')\big) < 1 - \mathrm{e}^{-\eta n}\big\}$$

has aggregate weight $Z_{\mathrm{Bad}}(G) = \sum_{\sigma \in S_{\mathrm{Bad}}(G)} \mathrm{e}^{\beta \mathcal{H}(G)}$ less than $\mathrm{e}^{-\theta n} Z^{\varepsilon}_{\mathrm{f}}(G)$. For a d-regular graph G such that $G \in \mathcal{Q} \cap \mathcal{Q}'$, using arguments analogous to those for Glauber, we have that $\Phi_{SW}\big(\tilde{S}_{\mathrm{f}}(\varepsilon)\big) \leq 10\mathrm{e}^{-\theta n}$. By arguments analogous to those in the proof of Theorem 1.1, we have that $\tilde{S}_{\mathrm{f}}(\varepsilon)$ is a metastable state for graphs $G \in \mathcal{Q} \cap \mathcal{Q}'$. Therefore, to finish the metastability proof for the random graph, it suffices to show that $\mathbb{P}(\boldsymbol{G} \in \mathcal{Q} \cap \mathcal{Q}') = 1 - o(1)$.

To do this, let $\mathcal{G}(n, d)$ be the set of all multigraphs that can be obtained in the pairing model and $\Lambda_{d,\beta}(n) = \big\{(G, \sigma) \,\big|\, G \in \mathcal{G}(n, d),\ \sigma \in \tilde{S}_{\mathrm{f}}(\varepsilon)\big\}$. Let $\mathcal{E}$ be the pairs $(G, \sigma) \in \Lambda_{d,\beta}(n)$ where one step of SW starting from G, σ stays within $\tilde{S}_{\mathrm{f}}(\varepsilon')$ with probability $1 - \mathrm{e}^{-\Omega(n)}$, more precisely $\mathcal{E} = \Big\{(G, \sigma) \in \Lambda_{d,\beta}(n) \,\Big|\, P^G_{SW}\big(\sigma \to \tilde{S}_{\mathrm{f}}(\varepsilon')\big) \geq 1 - \mathrm{e}^{-\eta n}\Big\}$. The aggregate weight corresponding to pairs $(G, \sigma) \notin \mathcal{E}$ can be lower-bounded by

$$\sum_{(G,\sigma) \in \Lambda_{d,\beta} \setminus \mathcal{E}} \mathrm{e}^{\beta \mathcal{H}_G(\sigma)} \geq \sum_{G \in \mathcal{Q} \setminus \mathcal{Q}'} \sum_{\sigma \in \Sigma_{\mathrm{Bad}}(G)} \mathrm{e}^{\beta \mathcal{H}_G(\sigma)} \geq \mathrm{e}^{-\theta n} \sum_{G \in \mathcal{Q} \setminus \mathcal{Q}'} Z^{\varepsilon}_{\mathrm{f}}(G).$$

For graphs $G \in \mathcal{Q}$ we have $Z^{\varepsilon}_{\mathrm{f}}(G) \geq \mathrm{e}^{-n^{3/4}}\mathbb{E}[Z^{\varepsilon}_{\mathrm{f}}(\boldsymbol{G})]$, and therefore

$$\sum_{(G,\sigma)\in\Lambda_{d,\beta}\setminus\mathcal{E}} \mathrm{e}^{\beta\mathcal{H}_G(\sigma)} \geq \mathrm{e}^{-(\theta n+n^{3/4})}\big|\mathcal{Q}\setminus\mathcal{Q}'\big|\ \mathbb{E}\big[Z^{\varepsilon}_{\mathrm{f}}(\boldsymbol{G})\big] = \mathrm{e}^{-(\theta n+n^{3/4})}\big|\mathcal{Q}\setminus\mathcal{Q}'\big|\ \frac{\sum_{(G,\sigma)\in\Lambda_{d,\beta}} \mathrm{e}^{\beta\mathcal{H}_G(\sigma)}}{\big|\mathcal{G}(n,d)\big|}$$

From the definition of $\big(\hat{\boldsymbol{G}}\big(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big), \hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big)$, cf. (15),(16), observe that

$$\frac{\sum_{(G,\sigma)\in\Lambda_{d,\beta}\setminus\mathcal{E}} \mathrm{e}^{\beta\mathcal{H}_G(\sigma)}}{\sum_{(G,\sigma)\in\Lambda_{d,\beta}} \mathrm{e}^{\beta\mathcal{H}_G(\sigma)}} = \mathbb{P}\big[\big(\hat{\boldsymbol{G}}(\hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)), \hat{\boldsymbol{\sigma}}_{\mathrm{f}}(\varepsilon)\big) \in \Lambda_{d,\beta}\setminus\mathcal{E}\big] \leq \mathrm{e}^{-\eta n} \leq \mathrm{e}^{-10\theta n},$$

where the penultimate inequality follows from Proposition 4.4 and the last from the choice of θ. Combining the last two inequalities, we obtain $\mathbb{P}[\boldsymbol{G} \in \mathcal{Q}\setminus\mathcal{Q}'] = o(1)$. Since $\mathbb{P}[\boldsymbol{G} \in \mathcal{Q}] = 1-o(1)$ from Lemma 4.1, it follows that $\mathbb{P}[\boldsymbol{G} \in \mathcal{Q}\cap\mathcal{Q}'] \geq \mathbb{P}[\boldsymbol{G} \in \mathcal{Q}] - \mathbb{P}[\boldsymbol{G} \in \mathcal{Q}\setminus\mathcal{Q}'] \geq 1 - o(1)$. This concludes the proof for the metastability of the ferromagnetic phase $\tilde{S}_{\mathrm{f}}(\varepsilon)$ when $\beta > \beta_u$.

A similar bottleneck-ratio argument shows that $S_{\mathrm{p}}(\varepsilon)$ is a metastable state for $\beta < \beta_h$. The slow mixing of SW for $\beta \in (\beta_u, \beta_h)$ follows from the metastability of $\tilde{S}_{\mathrm{f}}(\varepsilon)$ when $\beta \in (\beta_u, \beta_p]$ and the metastability of $S_{\mathrm{p}}(\varepsilon)$ when $\beta \in [\beta_p, \beta_h)$. In particular, let $S \in \{\tilde{S}_{\mathrm{f}}(\varepsilon), S_{\mathrm{p}}(\varepsilon)\}$ be such that $\big\|\mu\big(\cdot \mid S\big) - \mu\big\| \geq 1/2$, then Lemma 4.2 gives that for $T = \mathrm{e}^{\Omega(n)}$, it holds that $\big\|\mu\big(\cdot \mid S\big)P^T_{SW} - \mu\big\| \geq 1/2 - 1/10$, yielding that the mixing time is $\mathrm{e}^{\Omega(n)}$. ◀

References

1 Emmanuel Abbe. Community detection and stochastic block models: Recent developments. *Journal of Maching Learning Research*, 18(1):6446–6531, 2017.

2 Dimitris Achlioptas, Assaf Naor, and Yuval Peres. Rigorous location of phase transitions in hard optimization problems. *Nature*, 435(7043):759–764, 2005.

3 Victor Bapst and Amin Coja-Oghlan. Harnessing the Bethe free energy. *Random Structures and Algorithms*, 49:694–741, 2016.

4 Jean Barbier, Chun Lam Clement Chan, and Nicolas Macris. Concentration of multi-overlaps for random dilute ferromagnetic spin models. *Journal of Statistical Physics*, 180:534–557, 2019.

5 Antonio Blanca, Andreas Galanis, Leslie Ann Goldberg, Daniel Štefankovič, Eric Vigoda, and Kuan Yang. Sampling in uniqueness from the Potts and random-cluster models on random regular graphs. *SIAM Journal on Discrete Mathematics*, 34(1):742–793, 2020.

6 Antonio Blanca and Reza Gheissari. Random-cluster dynamics on random regular graphs in tree uniqueness. *Communications in Mathematical Physics*, 386(2):1243–1287, 2021.

7 Antonio Blanca and Alistair Sinclair. Dynamics for the mean-field random-cluster model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 528–543, 2015.

8 Antonio Blanca, Alistair Sinclair, and Xusheng Zhang. The critical mean-field Chayes-Machta dynamics. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, 2021.

9 Magnus Bordewich, Catherine Greenhill, and Viresh Patel. Mixing of the Glauber dynamics for the ferromagnetic Potts model. *Random Structures and Algorithms*, 48(1):21–52, 2016.

10 Charlie Carlson, Ewan Davies, Nicolas Fraiman, Alexandra Kolla, Aditya Potukuchi, and Corrine Yap. Algorithms for the ferromagnetic Potts model on expanders. *arXiv preprint*, 2022. `arXiv:2204.01923`.

11 Amin Coja-Oghlan, Oliver Cooley, Mihyun Kang, Joon Lee, and Jean Bernoulli Ravelomanana. The sparse parity matrix. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 822–833, 2022.

12 Amin Coja-Oghlan, Charilaos Efthymiou, and Samuel Hetterich. On the chromatic number of random regular graphs. *Journal of Combinatorial Theory, Series B*, 116:367–439, 2016.

13 Amin Coja-Oghlan, Charilaos Efthymiou, Nor Jaafari, Mihyun Kang, and Tobias Kapetanopoulos. Charting the replica symmetric phase. *Communications in Mathematical Physics*, 359(2):603–698, 2018.

14 Amin Coja-Oghlan, Oliver Gebhard, Max Hahn-Klimroth, and Philipp Loick. Optimal group testing. *Combinatorics, Probability and Computing*, 30(6):811–848, 2021.

15 Amin Coja-Oghlan, Florent Krzakala, Will Perkins, and Lenka Zdeborová. Information-theoretic thresholds from the cavity method. *Advances in Mathematics*, 333:694–795, 2018.

16 Amin Coja-Oghlan, Philipp Loick, Balázs F Mezei, and Gregory B Sorkin. The Ising antiferromagnet and max cut on random regular graphs. *arXiv preprint*, 2020. `arXiv:2009.10483`.

17 Paul Cuff, Jian Ding, Oren Louidor, Eyal Lubetzky, Yuval Peres, and Allan Sly. Glauber dynamics for the mean-field Potts model. *Journal of Statistical Physics*, 149(3):432–477, 2012.

18 Amir Dembo and Andrea Montanari. Ising models on locally tree-like graphs. *The Annals of Applied Probability*, 20(2):565–592, 2010.

19 Amir Dembo, Andrea Montanari, Allan Sly, and Nike Sun. The replica symmetric solution for Potts models on d-regular graphs. *Communications in Mathematical Physics*, 327(2):551–575, 2014.

20 Amir Dembo, Andrea Montanari, and Nike Sun. Factor models on locally tree-like graph. *The Annals of Probability*, 41(6):4162–4213, 2013.

21 Charilaos Efthymiou. On sampling symmetric Gibbs distributions on sparse random graphs and hypergraphs. *arXiv preprint*, 2020. `arXiv:2007.07145`.

22 Andreas Galanis, Daniel Stefankovic, Eric Vigoda, and Linji Yang. Ferromagnetic Potts model: Refined #BIS-hardness and related results. *SIAM Journal of Computation*, 45(6):2004–2065, 2016.

23 Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Swendsen-Wang algorithm on the mean-field Potts model. *Random Structures and Algorithms*, 54(1):82–147, 2019.

24 Reza Gheissari and Eyal Lubetzky. Mixing times of critical two-dimensional Potts models. *Communications on Pure and Applied Mathematics*, 71(5):994–1046, 2018.

25 Reza Gheissari, Eyal Lubetzky, and Yuval Peres. Exponentially slow mixing in the mean-field Swendsen-Wang dynamics. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2018.

26 Reza Gheissari and Alistair Sinclair. Low-temperature Ising dynamics with random initializations. *arXiv preprint*, 2021. `arXiv:2106.11296`.

27 Vivek K Gore and Mark R Jerrum. The Swendsen-Wang process does not always mix rapidly. *Journal of Statistical Physics*, 97(1):67–86, 1999.

28 Olle Häggström. The random-cluster model on a homogeneous tree. *Probability Theory and Related Fields*, 104(2):231–253, 1996.

29 Tyler Helmuth, Matthew Jenssen, and Will Perkins. Finite-size scaling, phase coexistence, and algorithms for the random cluster model on random graphs. *arXiv preprint*, 2020. `arXiv:2006.11580`.

30 Svante Janson, Andrzej Rucinski, and Tomasz Luczak. *Random graphs*. John Wiley and Sons, 2011.

31 Jungkyoung Lee. Energy landscape and metastability of curie–weiss–potts model. *Journal of Statistical Physics*, 187(1):1–46, 2022.

32 David A Levin and Yuval Peres. Markov chains and mixing times. *American Mathematical Society*, 2009.

33 Eyal Lubetzky and Allan Sly. Critical Ising on the square lattice mixes in polynomial time. *Communications in Mathematical Physics*, 313(3):815–836, 2012.

34 Marc Mézard. Mean-field message-passing equations in the Hopfield model and its generalizations. *Physical Review E*, 95(2):022117, 2017.

35 Marc Mezard and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.

36 Marc Mézard and Giorgio Parisi. The Bethe lattice spin glass revisited. *The European Physical Journal B-Condensed Matter and Complex Systems*, 20(2):217–233, 2001.

37 Marc Mézard and Giorgio Parisi. The cavity method at zero temperature. *Journal of Statistical Physics*, 111(1):1–34, 2003.

38 Tom Richardson and Ruediger Urbanke. *Modern coding theory*. Cambridge University Press, 2008.

39 Nicholas Ruozzi. The bethe partition function of log-supermodular graphical models. *Advances in Neural Information Processing Systems*, 25, 2012.

40 Allan Sly. Computational transition at the uniqueness threshold. In *51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 287–296, 2010.

41 Allan Sly and Nike Sun. The computational hardness of counting in two-spin models on d-regular graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 361–369, 2012.

42 Mario Ullrich. Swendsen-Wang is faster than single-bond dynamics. *SIAM Journal on Discrete Mathematics*, 28(1):37–48, 2014.

43 Pascal O Vontobel. Counting in graph covers: A combinatorial characterization of the Bethe entropy function. *IEEE Transactions on Information Theory*, 59(9):6018–6048, 2013.

44 Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:239–269, 2003.

On Computing the k-Shortcut Fréchet Distance

Jacobus Conradi
Department of Computer Science, Universität Bonn, Germany

Anne Driemel
Hausdorff Center for Mathematics, Universität Bonn, Germany

Abstract

The Fréchet distance is a popular measure of dissimilarity for polygonal curves. It is defined as a min-max formulation that considers all direction-preserving continuous bijections of the two curves. Because of its susceptibility to noise, Driemel and Har-Peled introduced the shortcut Fréchet distance in 2012, where one is allowed to take shortcuts along one of the curves, similar to the edit distance for sequences. We analyse the parameterized version of this problem, where the number of shortcuts is bounded by a parameter k. The corresponding decision problem can be stated as follows: Given two polygonal curves T and B of at most n vertices, a parameter k and a distance threshold δ, is it possible to introduce k shortcuts along B such that the Fréchet distance of the resulting curve and the curve T is at most δ? We study this problem for polygonal curves in the plane. We provide a complexity analysis for this problem with the following results: (i) assuming the exponential-time-hypothesis (ETH), there exists no algorithm with running time bounded by $n^{o(k)}$; (ii) there exists a decision algorithm with running time in $\mathcal{O}(kn^{2k+2}\log n)$. In contrast, we also show that efficient approximate decider algorithms are possible, even when k is large. We present a $(3+\varepsilon)$-approximate decider algorithm with running time in $\mathcal{O}(kn^2\log^2 n)$ for fixed ε. In addition, we can show that, if k is a constant and the two curves are c-packed for some constant c, then the approximate decider algorithm runs in near-linear time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Fréchet distance, Partial similarity, Conditional lower bounds, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.46

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2202.11534` [19]

Funding This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – AA 1111/2-2 (FOR 2535 Anticipating Human Behavior).

1 Introduction

With the prevalence of geographical data collection and usage, the need to process and compare polygonal curves stemming from this data arises. A popular versatile distance measure for polygonal curves is the Fréchet distance [27]. The distance measure is very similar to the well-known Hausdorff distance for geometric sets, except that it takes the ordering of points along the curves into account by minimizing over all possible direction-preserving continuous bijections between the two curves. Intuitively, the distance measure can be defined as follows. Imagine two agents independently traversing the two curves with varying speeds. Let δ be an upper bound on the (Euclidean) distance of the two agents that holds at any point in time during the traversal. The Fréchet distance corresponds to the minimum value of δ that can be attained over all possible traversals.

In practice, the distance measure may be distorted by outliers and measurement errors. As a remedy, partial similarity and distance measures have been introduced which are thought to be more robust. Buchin, Buchin and Wang define a *partial Fréchet distance* [14]

which maximizes the portions of the two curves matched to one-another within some given distance threshold. Driemel and Har-Peled suggested the *shortcut Fréchet distance* [21] in the spirit of the well-known edit distance for strings: a set of non-overlapping subcurves can be replaced by straight edges connecting the endpoints (so-called shortcuts) to minimize the Fréchet distance of the resulting curves. Akitaya, Buchin, Ryvkin and Urhausen [3] introduced a variant of the Fréchet distance, where a certain number of "jumps" (backwards and forwards) are allowed during the traversal of the two curves. We note that it has been acknowledged in the literature that partial dissimilarity measures generally do not satisfy metric properties [12, 25, 28].

It is conceivable that computing a partial dissimilarity based on the Fréchet distance should be more difficult than the standard Fréchet distance because of the structure of the optimization problems involved. We briefly discuss what is known for these problems. The continuous Fréchet distance can be computed in $\mathcal{O}(n^2\text{polylog}(n))$ time for two polygonal curves of n vertices [5, 13]. For the discrete Fréchet distance, slightly subquadratic time is possible, as it can be computed in $\mathcal{O}(\frac{n^2\log\log(n)}{\log(n)})$ time [2]. However, assuming the Strong-Exponential-Time-Hypothesis (SETH), there is no algorithm, in either the discrete or the continuous setting, for any $\varepsilon > 0$, with running time in $\mathcal{O}(n^{2-\varepsilon})$ [9, 11, 15]. Following these works, we know that the complexity of computing the standard Fréchet distance is roughly quadratic in the size of the input, both in the discrete and continuous setting, and this holds for any dimension $d \geq 1$. Strongly subquadratic approximation algorithms are possible for restricted classes of curves [6, 7, 10, 22] and for large approximation factors [17, 18].

Compared to the standard Fréchet distance, the overall picture of the computational complexity of the partial variants is much more heterogeneous. De Carufel et al. [20] showed that the problem of computing the partial Fréchet distance is not solvable by radicals over $\mathbb{Q}$ and that the degree of the polynomial equations involved is unbounded in general. On the other hand, some variants of the partial Fréchet distance can be computed exactly in polynomial time [14]. Computing the shortcut Fréchet distance was shown to be NP-hard [16] when shortcuts are allowed anywhere along the curve. On the other hand, the *discrete Fréchet distance with shortcuts* was shown to be computable in strictly subquadratic time by Avraham et al. [8], which is even faster than computing the standard variant without shortcuts. The variant defined by Akitaya et al. [3] turns out to be NP-hard, but allows for fixed-parameter tractable algorithms.

Our contribution. In this paper, we study the computational complexity of a parameterized version of the shortcut Fréchet distance, where the maximum number of shortcuts that may be introduced on the curve is restricted by a parameter k. We show that assuming the Exponential-Time-Hypothesis (ETH), no fixed-parameter tractable running time is possible with k being the parameter. For polygonal curves in the plane, we present an exponential-time exact algorithm and we show that near-linear time approximation algorithms are possible using certain realistic input assumptions on the two curves.

Previous work. Driemel and Har-Peled [21] introduced the shortcut Fréchet distance and described a near-linear time $(3+\varepsilon)$-approximation algorithm for the class of c-packed curves. However, they only allowed shortcuts that start and end at *vertices* of the base curve. Buchin, Driemel and Speckmann [16] showed that, if shortcuts are allowed *anywhere along the curve*, then the problem of computing the shortcut Fréchet distance exactly is NP-hard via reduction from SUBSET-SUM. They also describe a 3-approximation algorithm for the decision problem with running time in $\mathcal{O}(n^3 \log n)$ for the case that shortcuts may start and

end in the middle of edges. Prior to our work, there has been no study of exact algorithms for the shortcut Fréchet distance, with non-restricted shortcuts. Our analysis of the exact problem therefore closes an important gap in the literature. Obtaining the exact algorithm was surprisingly simple, once the relevant techniques were combined in the right way.

1.1 Basic definitions

▶ **Definition 1** (curve). *A curve T is a continuous map from $[0,1]$ to $\mathbb{R}^d$, where $T(t)$ denotes the point on the curve parameterized by $t \in [0,1]$. For $0 \leq s < t \leq 1$ we denote the subcurve of T from $T(s)$ to $T(t)$ by $T[s,t]$. A polygonal curve of complexity n is given by a sequence of n points in $\mathbb{R}^d$. The curve is then defined as the piecewise linear interpolation between consecutive points.*

▶ **Definition 2** (Fréchet distance). *Given two curves T and B in $\mathbb{R}^d$, their Fréchet distance is defined as*

$$d_{\mathcal{F}}(T,B) = \inf_{f,g:[0,1]\to[0,1]} \max_{t\in[0,1]} \|T(f(t)) - B(g(t))\|,$$

where f and g are monotone, continuous, non-decreasing and surjective. We call a pair of such functions (f,g) a traversal. Any such traversal has the cost $\max_{\alpha\in[0,1]} \|T(f(\alpha)) - B(g(\alpha))\|$ associated to it.

In our definition of the Fréchet distance given above, we follow Alt and Godau [4]. Strictly speaking, this definition does not use bijections as for the sake of convenience the strict monotonicity of f and g is relaxed.

▶ **Definition 3** (k-shortcut curve). *We call a line segment between two arbitrary points $B(s)$ and $B(t)$ of a curve B a shortcut on B, where $s < t$ and denote it by $\overline{B}[s,t]$. A k-shortcut curve of B is the result of replacing k subcurves $B[s_i,t_i]$ of B for $1 \leq i \leq k$ by shortcuts $\overline{B}[s_i,t_i]$ connecting their start and endpoint, with $t_i \leq s_{i+1}$ for $1 \leq i \leq k-1$.*

▶ **Definition 4** (k-shortcut Fréchet Distance). *Given two polygonal curves T and B, their k-shortcut Fréchet distance $d_{\mathcal{S}}^k(T,B)$ is defined as the minimum Fréchet distance between T and any k'-shortcut curve of B for some $0 \leq k' \leq k$. In this context, we call B the base curve (where we take shortcuts) and T the target curve (which we want to minimize the Fréchet distance to).*

1.2 Overview of this paper

In Section 3 we present an exact algorithm for deciding if the k-shortcut Fréchet distance is smaller than a given threshold δ. The algorithm can also be used for the non-parameterized variant by setting $k = n$. Our first main result is the following theorem.

▶ **Theorem 5.** *Let T and B be two polygonal curves in the plane with overall complexity n, together with a value $\delta > 0$. There exists an algorithm with running time in $\mathcal{O}\big(kn^{2k+2}\log n\big)$ and space in $\mathcal{O}\big(kn^{2k+2}\big)$ that decides whether $d_{\mathcal{S}}^k(T,B) \leq \delta$.*

Our algorithm for Theorem 5 iterates over the free space diagram by Alt and Godau [5] in k rounds. Within the free space diagram, a direction-preserving continuous bijection between two curves corresponds to a monotone path starting at $(0,0)$ and ending at $(1,1)$. In each round, we compute the set of points in the parametric space of the two curves that are reachable by using one additional shortcut. For computing the set of eligible shortcuts

spanning a fixed set of edges, we make use of the so-called line-stabbing wedge introduced by Guibas et al. [23]. Line-stabbing wedges were also used in the approximation algorithm by Buchin et al [16]. In our case, since we perform exact computations, the reachable space may be fragmented into a number of components, and this number may grow exponentially with the number of rounds.

In Section 5 we give some evidence that this high complexity due to fragmentation is not an artifact of our algorithm, but may be inherent in the problem itself. For this, we assume that the exponential time hypothesis (ETH) holds. The ETH states that 3-SAT in n variables cannot be solved in $2^{o(n)}$ time [24]. Our second main result is the following conditional lower bound.

▶ **Theorem 6.** *Unless ETH fails, there is no algorithm for the k-shortcut Fréchet distance decision problem in $\mathbb{R}^d$ for $d \geq 2$, with running time $n^{o(k)}$.*

Our conditional lower bound of Theorem 6 is obtained via reduction from a variant of the k-SUM problem, which is called k-Table-SUM. In particular, we construct a $(4k+2)$-shortcut Fréchet distance decision instance for a given k-Table-SUM instance. Our construction is based on the NP-hardness reduction by Buchin, Driemel and Speckmann [16]. Their reduction was from SUBSET-SUM and could not be directly applied to obtain our result. The construction implicitly encodes partial solutions for the SUBSET-SUM instance as reachable intervals on the edges of one of the curves. In this way, each shortcut taken by the optimal solution implements a choice for an element to be included in the sum. The previous reduction implemented this in the form of a binary choice, thereby leading to a number of shortcuts that is linear in n. In our case, the number of shortcuts taken should only depend on k and not n. Therefore, we give a new construction for a choice gadget, that allows to choose an element from a set to be included in a partial solution while using only a constant number of shortcuts for this choice. We remark that a tighter bound can be obtained when assuming the k-SUM hypothesis [1].

In light of the above results, it is interesting to consider approximation algorithms and realistic input assumptions for this problem. In Section 4 we show that there is an efficient approximation algorithm for this problem. If we can assume that the input curves are well-behaved, we even obtain a near-linear time algorithm for constant k. To formalize this, we consider the class of c-packed curves, see also [22].

▶ **Definition 7** (c-packed curves)**.** *For $c > 0$, a curve X is called c-packed if the total length of X inside any ball is bounded by c times the radius of the ball.*

The following is our third main result. Since any polygonal curve of complexity n is c-packed for some $c \leq 2n$, the theorem also implies a running time of $\mathcal{O}\big(kn^2\varepsilon^{-5}(\log^2(n\varepsilon^{-1})\big)$ for polygonal curves in the plane – without any input assumptions.

▶ **Theorem 8.** *Let T and B be two c-packed polygonal curves in the plane with overall complexity n, together with values $0 < \varepsilon \leq 1$ and $\delta > 0$. There exists an algorithm with running time in $\mathcal{O}\big(kcn\varepsilon^{-5}\log^2(n\varepsilon^{-1})\big)$ and space in $\mathcal{O}\big(kcn\varepsilon^{-4}\log^2(\varepsilon^{-1})\big)$ which outputs one of the following: (i) $d_{\mathcal{S}}^k(T,B) \leq (3+\varepsilon)\delta$ or (ii) $d_{\mathcal{S}}^k(T,B) > \delta$. In any case, the output is correct.*

The main ideas that go into the proof of Theorem 8 can be sketched as follows. The first observation is that a highly fragmented reachable space that leads to the high running time of the exact algorithm of Theorem 5 can be approximated by limiting the number of shortcuts that the algorithm may take. To show that the algorithm still takes the right

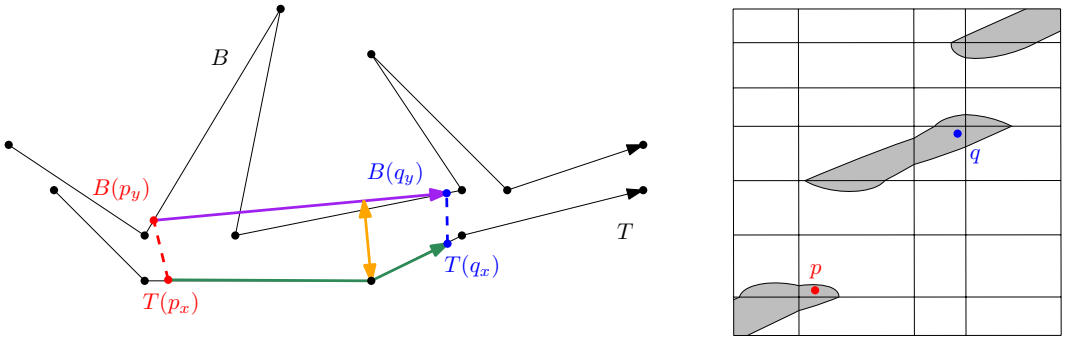


Figure 1 Free space diagram for a base curve B and a target curve T, with the δ-free space in gray. Marked are some grid lines with their corresponding vertex on each curve. The figure also shows a feasible proper tunnel $\tau(p, q)$. The shortcut $\overline{B}[p_y, q_y]$ is shown in purple, and the subcurve $T[p_x, p_y]$ in green. The price of $\tau(p, q)$ is the Fréchet distance of the shortcut and the subcurve.

decisions (within the approximation bounds), we make use of a property of shortcut prices that was first observed by Driemel and Har-Peled [21]. Namely, the price of a shortcut is approximately monotone and it suffices in each round to take the 'shortest' feasible shortcut among all shortcuts that are available. Now, the main challenge as compared to the algorithm in [21] is that this shortcut may still start in the middle of an edge. Thus, we would need to invoke the line-stabbing wedges, but this would be too expensive. Instead, we use a data structure by Driemel and Har-Peled [21] that allows to query the Fréchet distance of a line segment to a subcurve. We combine this with a scheme to simulate an approximation of the output of the line-stabbing wedge with queries to this data structure. In particular, we can approximate the line-stabbing wedge with a convex hull of a set of grid points. However, this is still not enough, as the free-space may have quadratic complexity. To obtain a near-linear running time for small c, we make use of the property of c-packed curves as observed by Driemel, Har-Peled and Wenk [22], that the complexity of the free space diagram of two c-packed curves is only linear in $c \cdot n$ when the curves are appropriately simplified.

2 Preliminaries

Our algorithm uses the free space diagram which was introduced by Alt and Godau [5] for computing the standard Fréchet distance.

▶ **Definition 9** (Free space diagram). *Let T and B be two polygonal curves in $\mathbb{R}^d$. The free space diagram of T and B is the joint parametric space $[0, 1]^2$ together with a not necessarily uniform grid, where each vertical line corresponds to a vertex of T and each horizontal line to a vertex of B (refer to Figure 1). We call the cell of the parametric space corresponding to the ith edge of the target curve and the jth edge of the base curve $C_{i,j}$. The δ-free space of T and B is defined as*

$$\mathcal{D}_\delta(T, B) = \{(x, y) \in [0, 1]^2 \mid \|T(x) - B(y)\| \leq \delta\}$$

This is the set of points in the parametric space whose corresponding points on B and T are at a distance at most δ. Denote by $\mathcal{D}_\delta^{(i,j)}(T, B) = \mathcal{D}_\delta(T, B) \cap C_{i,j}$ the δ-free space inside the cell $C_{i,j}$.

In the following T and B will often be fixed, thus we will simply write $\mathcal{D}_\delta$. It is known that $\mathcal{D}_\delta^{(i,j)}$ is convex and has constant complexity. More precisely, it is an ellipse intersected with the cell $C_{i,j}$. Furthermore the Fréchet distance between two curves is less than or equal

to δ if and only if there exists a monotone path (in x and y) in the free space that starts in the lower left corner $(0,0)$ and ends in the upper right corner $(1,1)$ cf. [5]. In the case of the k-shortcut Fréchet distance we need to also consider shortcuts when traversing the parametric space. When considering any k-shortcut curve B' of B and any traversal (f,g) of B' and T with associated cost δ, then (f,g) induces traversals (f',g') with associated cost at most δ on every shortcut $\overline{B}[s,t]$ and some corresponding subcurve $T[u,v]$ of T. To capture this, we use the notion of *tunnels* which was introduced in [21] and is defined as follows.

▶ **Definition 10** (Tunnel). *A tunnel $\tau(p,q)$ is a pair of points $p=(x_p,y_p)$ and $q=(x_q,y_q)$ in the parametric space of B and T, with $x_p \le x_q$ and $y_p \le y_q$. $\tau(p,q)$ is called feasible if p and q are in $\mathcal{D}_\delta$. We say that a tunnel is proper, if the endpoints of the shortcut do not lie on the same edge of B. We say a tunnel has a price* $\mathrm{prc}(\tau(p,q)) = d_{\mathcal{F}}(T[x_p,x_q],\overline{B}[y_p,y_q])$, *refer to Figure 1.*

▶ **Observation 11.** *Given line segments $\overline{ab}$ and $\overline{cd}$ in $\mathbb{R}^d$, then for the Fréchet distance we have $d_{\mathcal{F}}(\overline{ab},\overline{cd}) = \max(||a-c||, ||b-d||)$.*

▶ **Definition 12** (Reachable space). *We define the (δ,s)-reachable free space of T and B*

$$\mathcal{R}_{\delta,s}(T,B) = \{(x_p,y_p) \in [0,1]^2 \mid d^s_{\mathcal{S}}(T[0,x_p],B[0,y_p]) \le \delta\}$$

and again $\mathcal{R}^{(i,j)}_{\delta,s}(T,B) = \mathcal{R}_{\delta,s}(T,B) \cap C_{i,j}$. We call the intersection $\mathcal{R}^{(i,j)}_{\delta,s}(T,B) \cap C_{a,b}$ for any $(a,b) \in \{(i-1,j),(i,j-1),(i+1,j),(i,j+1)\}$ a reachability interval of the cell $C_{i,j}$. In particular for $(a,b) \in \{(i-1,j),(i,j-1)\}$ we call them incoming reachability intervals and for $(a,b) \in \{(i+1,j),(i,j+1)\}$ we call them outgoing reachability intervals.

Note that the reachability intervals for every cell $C_{i,j}$ and s are contained in $\partial C_{i,j}$, and each reachability interval is described by a (possibly empty) single interval, since any two points in the reachability interval can be connected via a monotone path that stays inside the δ-free space. We will simply write $\mathcal{R}_{\delta,s}$ and $\mathcal{R}^{(i,j)}_{\delta,s}$ whenever T and B are fixed. The k-shortcut Fréchet distance of T and B is at most δ if and only if $(1,1) \in \mathcal{R}_{\delta,k}$.

We want to reduce the problem of deciding the k-shortcut Fréchet distance to the problem of deciding on the existence of a monotone path in the free space diagram with k tunnels starting in $(0,0)$ and ending in $(1,1)$. Note that any tunnel $\tau(p,q)$ with $p=(x_p,y_p)$ and $q=(x_q,y_q)$, that is not proper, induces a traversal of $B[y_p,y_q] = \overline{B}[y_p,y_q]$ and $T[x_p,x_q]$. Thus we can omit the tunnel and replace it with a monotone path from p to q in $\mathcal{D}_\delta$. Therefore, in the following, we only consider monotone paths with proper tunnels.

▶ **Definition 13** (Monotone path with tunnels). *A monotone path with k proper tunnels in the δ-free space of two curves consists of $k+1$ monotone (in x and y) paths in the δ-free space from s_i to t_i for $1 \le i \le k+1$, with $s_1=(0,0)$, such that t_i lies to the left and below s_{i+1}, for $1 \le i \le k$. The k proper tunnels have the form $\tau(t_i,s_{i+1})$ for $1 \le i \le k$.*

▶ **Observation 14.** *Let T and B be two polygonal curves. The set $\mathcal{R}_{\delta,s}(T,B)$ is exactly the set of points $p \in \mathcal{D}_\delta(T,B)$ such that there exists a monotone path ending in p with at most s proper tunnels, each of price at most δ. (By definition, these paths have to start at $(0,0)$).*

To decide whether a cell is reachable by a tunnel, we need to check if there exists a shortcut edge that stabs through an ordered set of disks centered at a subset of the vertices of the other curve. To formalize this, we use the notion of ordered stabbers and line-stabbing wedges as defined by Guibas et al. [23].

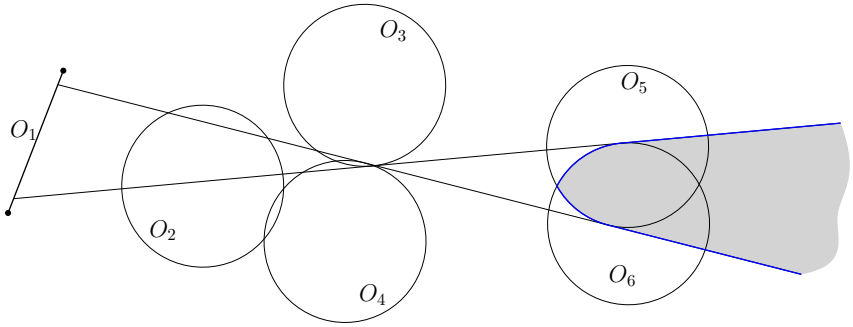


Figure 2 Example for the line-stabbing wedge for a line segment O_1 and disks $O_2, \ldots, O_6$. The line-stabbing wedge is shown in gray, with its boundary in blue.

▶ **Definition 15** (Line-stabbing wedge). *Given a sequence of n convex objects $O_1, \ldots, O_n$, an ordered stabber of this sequence is a line segment $l(x) = (1-x)s + xt$ from s to t, such that points $0 \leq x_1 \leq x_2 \leq \ldots \leq x_n \leq 1$ exist with $p_i = l(x_i) \in O_i$. We call p_i the realising points of l. We say that l stabs through $O_1, \ldots, O_n$. We call the set of points t that are endpoints of ordered stabbers of $O_1, \ldots, O_n$ the line-stabbing wedge of this sequence.*

In their paper, Guibas et al. give an algorithm to compute the line-stabbing wedge for a sequence of n unit disks as well as convex polygons of constant size, with running time $\mathcal{O}(n \log n)$. This line-stabbing wedge is described by $\mathcal{O}(n)$ circular arcs, and two tangents that go to infinity (see Figure 2). These instances can be scaled, such that we can use any sequence of disks $\{\mathrm{b}_\delta(p_1), \ldots, \mathrm{b}_\delta(p_n)\}$, where all disks have the same radius ($\mathrm{b}_\delta(p)$ denotes the closed disk of radius δ centered around p).

▶ **Observation 16.** *Let T, B and δ be given. Denote by v_k the vertices of T. For any feasible tunnel $\tau(p, q)$ with $p = (x_p, y_p) \in C_{a,b}$ and $q = (x_q, y_q) \in C_{i,j}$, it holds that $\overline{B}[y_p, y_q]$ stabs through the ordered set $\{\mathrm{b}_\delta(v_{a+1}), \ldots, \mathrm{b}_\delta(v_i)\}$, if and only if the price of $\tau(p, q)$ is at most δ.*

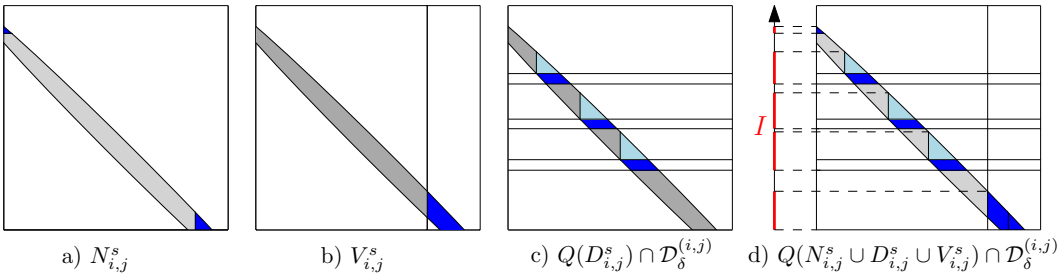
3 Exact decider algorithm

In this section we describe an exact decider algorithm for the k-shortcut Fréchet distance for two polygonal curves. The algorithm can also be used to solve the decision problem of the (unparameterized) shortcut Fréchet distance by setting $k = n$. We first describe the algorithm and then analyse its correctness and running time in Section 3.2. The full correctness proof can be found in [19].

3.1 The Algorithm

We are given a parameter k, a value δ and the two polygonal curves T and B in the plane. Our algorithm iterates over the δ-free space diagram of T and B in k rounds. In each round, based on the computation of the previous round, we compute the set of points that are reachable by using one additional shortcut. The goal is to compute the (δ, s)-reachable space $\mathcal{R}_{\delta,s}(T, B)$ in round s. In each round, we handle the cells of the free space diagram in a row-by-row order, and within each row from left to right. For every cell $C_{i,j}$ we consider three possible ways that a monotone path with proper tunnels can enter.

1. The monotone path could enter the cell $C_{i,j}$ from the neighboring cell $C_{i-1,j}$ to the left or from the neighboring cell $C_{i,j-1}$ below. This does not directly involve a tunnel.



a) $N^s_{i,j}$ b) $V^s_{i,j}$ c) $Q(D^s_{i,j}) \cap \mathcal{D}^{(i,j)}_\delta$ d) $Q(N^s_{i,j} \cup D^s_{i,j} \cup V^s_{i,j}) \cap \mathcal{D}^{(i,j)}_\delta$

Figure 3 Example of the composition of the reachable space within a single free-space cell. The lightblue sets in *c*) (resp. *d*)) contain all the points reachable via monotone paths inside the cell reachable from $D^s_{i,j}$ (resp. $N^s_{i,j} \cup D^s_{i,j} \cup V^s_{i,j}$) computed via $Q(\cdot) \cap \mathcal{D}^{(i,j)}_\delta$. The fragmentation of the reachable space in this cell $P^s_{i,j} = Q(N^s_{i,j} \cup D^s_{i,j} \cup V^s_{i,j}) \cap \mathcal{D}^{(i,j)}_\delta$ results in a large family of intervals I on the edge of B.

2. The monotone path could reach $C_{i,j}$ with a proper tunnel. We distinguish between vertical and diagonal tunnels (compare [16, 21] for a similar distinction).
   - **(i)** The tunnel may start in any cell $C_{a,b}$ with $a < i$ and $b < j$. We call this a diagonal tunnel.
   - **(ii)** The tunnel may start in any cell $C_{i,b}$ for $b < j$. We call this a vertical tunnel.

Using this distinction, we will describe how to compute the set of points reachable by a monotone path with s proper tunnels, for each cell of the diagram. We denote the set computed by the algorithm for cell $C_{i,j}$ in round s with $P^s_{i,j}$. The (δ, s)-reachable space is then obtained by taking the union of these sets over all rounds $\mathcal{R}^{(i,j)}_{\delta,s} = \bigcup_{0 \le s' \le s} P^{s'}_{i,j}$.

After k rounds, the algorithm tests whether the point $(1, 1)$ is contained in our computed set of reachable points. If this is the case, then the algorithm returns "$d^k_{\mathcal{S}}(T, B) \le \delta$", otherwise the algorithm returns "$d^k_{\mathcal{S}}(T, B) > \delta$".

Propagating reachability within a cell. To simplify the description of the algorithm, we use the following set function which receives a set $P \subseteq C_{i,j}$ for some cell $C_{i,j}$ and which extends P to all points above and to the right of it.

$$Q(P) = \{(x, y) \in [0, 1]^2 \mid \exists (a, b) \in P \text{ such that } a \le x \text{ and } b \le y\}$$

We will usually intersect this set with $\mathcal{D}^{(i,j)}_\delta$ to obtain all points that are reachable from a point of P by a monotone path that stays inside the δ-free space of this cell. Figure 3 c) shows an example of the resulting set. Note that the boundary of the resulting set can be described by pieces of the boundary of $\mathcal{D}^{(i,j)}_\delta$, pieces of the boundary of P, and horizontal and vertical line segments.

Step 1: Neighbouring cells. Since we traverse the cells of the diagram in a lexicographical order, we have already computed the (possibly empty) sets $P^s_{i-1,j}$ and $P^s_{i,j-1}$, by the time we handle cell $C_{i,j}$ in round s. Therefore, we can compute the incoming reachability intervals by intersecting $P^s_{i-1,j}$ and $P^s_{i,j-1}$ with $C_{i,j}$. Now we apply the function Q to these sets and denote the result with $N^s_{i,j}$ (refer to Figure 3 a):

$$N^s_{i,j} = \big(Q(P^s_{i-1,j} \cap C_{i,j}) \cup Q(P^s_{i,j-1} \cap C_{i,j})\big) \cap \mathcal{D}^{(i,j)}_\delta$$

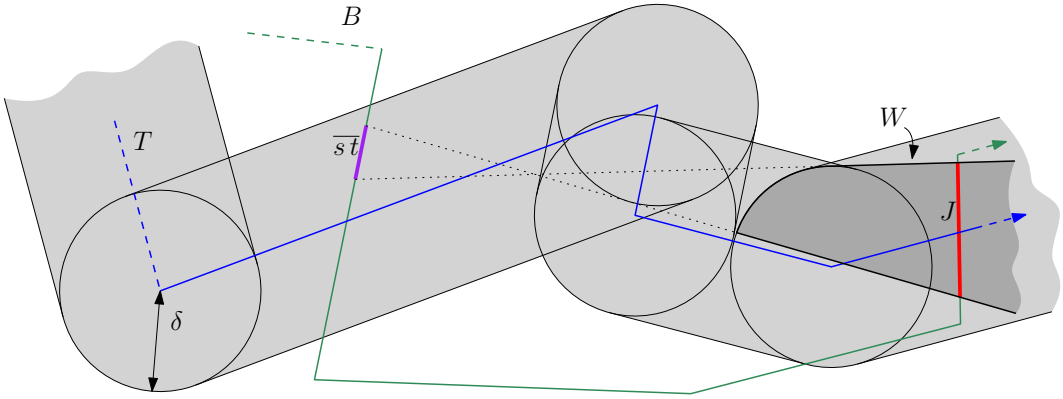


Figure 4 Example of the set J (in red) computed by the DIAGONALTUNNEL procedure.

Step 2 (i): Diagonal tunnels. We invoke the following procedure for every $a < i$ and $b < j$ with $P^{s-1}_{a,b}$. We denote the union of resulting sets of points in $\mathcal{D}^{(i,j)}_\delta$ computed in this step with $D^s_{i,j}$.

The procedure is given a set of points $P^{s-1}_{a,b}$ in the δ-free space $\mathcal{D}^{(a,b)}_\delta$ and computes all points in $\mathcal{D}^{(i,j)}_\delta$ that are endpoints of tunnels starting in $P^{s-1}_{a,b}$ with price at most δ. The procedure first projects $P^{s-1}_{a,b}$ onto the edge e_b of the base curve. The resulting set consists of disjoint line segments $I = \{\overline{s_1\,t_1}, \ldots\}$ along e_b (refer to Figure 3 d)). The procedure then computes the line-stabbing wedge W through $\overline{s_1\,t_1}$ and disks $\mathrm{b}_\delta(v_{a+1}), \ldots, \mathrm{b}_\delta(v_i)$ centered at vertices of T. W is then intersected with the edge e_j, resulting in a set J on e_j corresponding to a horizontal slab in $C_{i,j}$ (compare Figure 3 c) and Figure 4). This resulting set is then intersected with $\mathcal{D}^{(i,j)}_\delta$ to obtain all endpoints of feasible shortcuts with price at most δ starting in $\overline{s_1\,t_1}$. The procedure performs the above steps for every line segment $\overline{s\,t} \in I$ and returns the union of these sets. The resulting set may look as illustrated in Figure 3 c).

Step 2 (ii): Vertical tunnels. Let p denote a point in $\bigcup_{l \le j-1} P^{s-1}_{i,l}$ with minimal x-coordinate, i.e., a leftmost point in this set. A feasible vertical tunnel always has price at most δ, as the Fréchet distance of two line segments is bounded by the maximal euclidean distance of the start points or end points. Therefore, we simply take all points in the δ-free space to the right of p in the cell $C_{i,j}$. To do this, we compute the intersection of a halfplane that lies to the right of the vertical line at p with the δ-free space in $C_{i,j}$. We denote this set with $V^s_{i,j}$. Refer to Figure 3 b) for an example.

Putting things together. Finally, we compute the set $P^s_{i,j}$ by taking the union of the computed sets and extending this set by using the function Q defined above:

$$P^s_{i,j} = Q(N^s_{i,j} \cup D^s_{i,j} \cup V^s_{i,j}) \cap \mathcal{D}^{(i,j)}_\delta$$

It remains to specify the initialization: We define $P^0_{1,0} = P^0_{0,1} = \{(0,0)\}$, if $(0,0) \in \mathcal{D}_\delta$, and otherwise $P^0_{1,0} = P^0_{0,1} = \emptyset$. In addition we define $P^{-1}_{i,j} = \emptyset$ for all i, j.

3.2 Analysis

We argue that the structure of $P^s_{i,j}$ as computed by the algorithm is indeed as claimed. Namely for all i, j and s it holds that $\mathcal{R}^{(i,j)}_{\delta,s} = \bigcup_{0 \le s' \le s} P^{s'}_{i,j}$. The main argument of the proof consists of considering any monotone path with s proper tunnels ending in some cell.

Correctness then follows via induction over the lexicographical ordering of the cells in each round, and by induction over all rounds. The full proof of correctness can be found in [19]. Here, we provide an analysis of the running time of the algorithm.

▶ **Lemma 17.** *Let T and B be two polygonal curves in the plane with overall complexity n, together with a distance threshold $\delta > 0$. The algorithm described in Section 3.1 has running time in $\mathcal{O}(kn^{2k+2} \log n)$ and uses $\mathcal{O}(kn^{2k+2})$ space.*

Proof. Note that the sets $N^s_{i,j}$, $D^s_{i,j}$ and $V^s_{i,j}$ computed by the algorithm are described as intersections of $\mathcal{D}^{(i,j)}_\delta$ with halfplanes and horizontal slabs, and unions of these. For a fixed $P^s_{i,j}$ we define $n_{i,j,s}$ as the total number of such operations from which $P^s_{i,j}$ was obtained. As such, $\mathcal{O}(n_{i,j,s})$ bounds the complexity of this set.

The complexity of $N^s_{i,j}$ and $V^s_{i,j}$ is constant. The complexity of $D^s_{i,j}$ is bounded by the sum of the complexities of all cells to the lower left:

$$n_{i,j,s} \in \mathcal{O}\left(\sum_{0 \le a < i} \sum_{0 \le b < j} n_{a,b,s-1}\right).$$

As $i, j \le n$, and $s \le k$, and $n_{a,b,0} \in \mathcal{O}(1)$ for all a and b, it holds that $n_{i,j,s} \in \mathcal{O}(n^{2k})$.

Computing $D^s_{i,j}$ takes $\mathcal{O}(\sum_{a<i} \sum_{b<j} n_{a,b,s-1} \log n + n^2 \log n) = \mathcal{O}(n^{2k} \log n)$ time. This follows from the fact, that we compute $\mathcal{O}(n)$ line-stabbing wedges, and for every cell $C_{a,b}$ with $a < i$ and $b < j$ we handle $n_{a,b,s-1}$ line segments based on $P^{s-1}_{a,b}$. Computing $N^s_{i,j}$ takes $\mathcal{O}(n_{i-1,j,s} + n_{i,j-1,s}) = \mathcal{O}(n^{2k})$ time, as we need to compute the reachability intervals from neighbouring cells. Computing $V^s_{i,j}$ takes $\mathcal{O}(\sum_{b<j} n_{i,b,s-1}) = \mathcal{O}(n^{2k-1})$ time, as we need to compute the leftmost point $l^{s-1}_{i,j-1}$. The space required to store $P^s_{i,j}$ as required by latter iterations and cells is in $\mathcal{O}(n^{2k})$. Computing $Q(N^s_{i,j} \cup V^s_{i,j} \cup D^s_{i,j})$ takes linear time in the complexity of $N^s_{i,j} \cup V^s_{i,j} \cup D^s_{i,j}$, i.e. $\mathcal{O}(n^{2k})$. As we do this for every cell in every round, the running time overall is $\mathcal{O}(kn^{2k+2} \log n)$, and the space is bounded by $\mathcal{O}(kn^{2k+2})$. ◀

Lemma 17 together with the claimed correctness then imply Theorem 5. The algorithm can also be used for the (unparameterized) shortcut Fréchet distance by choosing $k = n$, since there can be at most n proper tunnels. We obtain the following corollary.

▶ **Corollary 18.** *Let T and B be polygonal curves in the plane with overall complexity n and let $\delta > 0$. There exists an algorithm with running time in $\mathcal{O}\big(n^{2n+3} \log n\big)$ and space in $\mathcal{O}\big(n^{2n+3}\big)$ that decides whether the shortcut Fréchet distance of T and B is at most δ.*

4 Approximate decision algorithm

In this section we describe a $(3+\varepsilon)$-approximation algorithm for the decision problem of the k-shortcut Fréchet distance of two polygonal curves in the plane. The algorithm has running time near-linear in n for c-packed curves. For general curves the running time is still polynomial in n and linear in k.

4.1 The modified algorithm

We describe how to modify the algorithm of Section 3 to circumvent the exponential complexity of the reachable space and obtain a polynomial-time approximation algorithm.

Let two polygonal curves T and B be given, together with a distance threshhold δ and approximation parameter ε. As before, the algorithm iterates over the cells of the free-space diagram and computes sets $N^s_{i,j}$, $V^s_{i,j}$, and $D^s_{i,j}$ for each cell $C_{i,j}$. The main difference is

now that, instead of computing the exact set of points that can be reached by a diagonal tunnel, we want to use an approximation for this set. For this, we define an approximate diagonal tunnel procedure, see further below. This procedure is called with the rightmost point $r^{s-1}_{i-1,j-1}$ in $\bigcup_{a<i;b<j} P^{s-1}_{a,b}$, ε and $\delta' = 3\delta$. Crucially, the set resulting from one call to the procedure has constant complexity and is sufficient to approximate the set $D^s_{i,j}$. We then compute $P^s_{i,j} = Q(N^s_{i,j} \cup D^s_{i,j} \cup V^s_{i,j}) \cap \mathcal{D}^{(i,j)}_\delta$, similarly to Section 3. From this we compute (i) the leftmost point $l^s_{i,j}$ in $\bigcup_{b\leq j} P^s_{i,b}$ based on $P^s_{i,j}$ and $l^s_{i,j-1}$, (ii) the rightmost point $r^s_{i,j}$ in $\bigcup_{a\leq i;b\leq j} P^s_{a,b}$ based on $P^s_{i,j}$, $r^s_{i-1,j}$ and $r^s_{i,j-1}$, and (iii) the outgoing reachability intervals of $P^s_{i,j}$. We store these variables to be used in the next round. Finally, after k rounds, we check if $(1,1)$ is contained in the computed set of reachable points.

Our approximate diagonal tunnel procedure makes use of a data structure by Driemel and Har-Peled, which is summarized in the following lemma. This data structure needs to be built once on T in the beginning and is then available throughout the algorithm.

▶ **Lemma 19** (distance oracle [21]). *Given a polygonal curve Z with n vertices in $\mathbb{R}^d$ and $\varepsilon > 0$, one can build a data structure $\mathcal{F}_\varepsilon$ in $\mathcal{O}\big(\chi^2 n \log^2 n\big)$ time, that uses $\mathcal{O}\big(n\chi^2\big)$ space such that given a query segment $\overline{p\,q}$ and any two points u and v on the curve, one can $(1+\varepsilon)$-approximate $d_{\mathcal{F}}(\overline{p\,q}, Z[u,v])$ in $\mathcal{O}\big(\varepsilon^{-2}\log n \log\log n\big)$ time, where $\chi = \varepsilon^{-d}\log\big(\varepsilon^{-1}\big)$.*

▶ **Definition 20** (Grid). *We define the scaled integer grid $\mathbb{G}_\delta = \big\{(\delta x, \delta y) \mid (x,y) \in \mathbb{Z}^2\big\}$.*

Approximate diagonal tunnel procedure. The procedure is described by Algorithm 1. The procedure is provided with parameters ε, δ, some $r' = (r_T, r_B)$ in cell $C_{a,b}$ and the edge e_j that is associated with a cell $C_{i,j}$. We want to compute a set of stabbers starting at $r = B(r_B)$ that contains every stabber through the disks $\mathrm{b}_\delta(v_{a+1}), \ldots, \mathrm{b}_\delta(v_i)$, and is contained in the set of all stabbers through disks of radius $(1+\varepsilon)^2\delta$ centered at the same vertices. We approximate this set of stabbers as follows.

We iterate over all grid points t of $\mathbb{G}_{\frac{\delta\varepsilon}{\sqrt{2}}}$ inside the disk $\mathrm{b}_{(1+\varepsilon)\delta}(v_i)$, and make queries to $\mathcal{F}_\varepsilon$ to determine if the Fréchet distance of the query segment $\overline{r\,t}$ to the subcurve of T from $T(r_T)$ to v_i is sufficiently small. We mark t if the approximate distance returned by the data structure is at most $(1+\varepsilon)^2\delta$. We then compute the convex hull H of all marked grid points, and the two tangents t_1 and t_2 of H through $B(r_B)$. The true set of endpoints of stabbers is approximated by the set H' of points that lie inside and 'behind' the convex hull H, from the perspective of r. Figure 5 illustrates this. We then intersect H' with the edge e_j resulting in a single horizontal slab in $C_{i,j}$, which is then intersected with $\mathcal{D}^{(i,j)}_\delta$ and returned.

4.2 Correctness of the approximate diagonal tunnel procedure

We denote with $\{\mathrm{b}_\delta(v_i)\}_i$ a sequence of disks $\{\mathrm{b}_\delta(v_1), \ldots, \mathrm{b}_\delta(v_m)\}$ for some m.

▶ **Lemma 21.** *Let $a, b_1, b_2 \in \mathbb{R}^d$ together with a sequence of vertices $v_1, \ldots, v_n$ be given. If $\overline{a\,b_1}$ stabs through disks $\{\mathrm{b}_\delta(v_i)\}_i$, and $\overline{a\,b_2}$ stabs through $\{\mathrm{b}_\delta(v_i)\}_i$, then for any $t \in [0,1]$ the line segment $\overline{a\,b(t)}$ stabs through $\{\mathrm{b}_\delta(v_i)\}_i$, where $b(t) = (1-t)b_1 + tb_2$.*

Proof. Refer to Figure 6. We can interpret the setting as a triangle with sides $(b_1 - a)$, $(b_2 - a)$, $(b_1 - b_2)$, where the first two sides correspond to the original stabbers and the last side to $b(t)$. Note that any line segment $\overline{a\,b(t)}$ lies completely within this triangle with $(b_1 - a)$ on the one and $(b_2 - a)$ on the other side. Hence, for every i and realising points p_i of $\overline{a\,b_1}$ and q_i of $\overline{a\,b_2}$, p_i lies on the one and q_i on the other side of $\overline{a\,b(t)}$. Since $\mathrm{b}_\delta(v_i)$ is convex and p_i and q_i are inside this disk, the intersection of $\overline{p_i\,q_i}$ and $\overline{a\,b(t)}$ is inside the disk

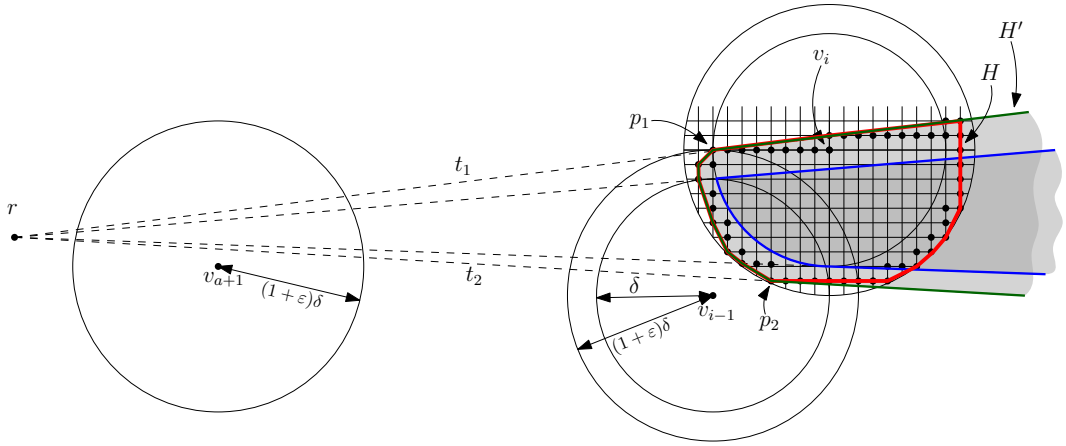


Figure 5 Illustration to the approximate diagonal tunnel procedure. The true line-stabbing wedge for disks with radius δ is shown in blue. The convex hull of eligible grid points is shown in red. The approximate line stabbing wedge is shown in green.

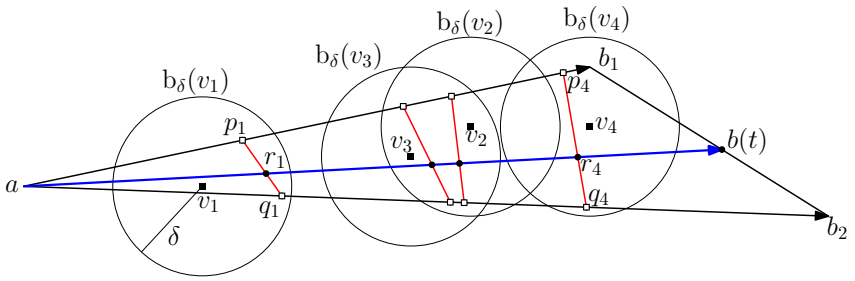


Figure 6 Linear interpolation between two δ-stabbers starting in a. Illustrations to the proof of Lemma 21. In blue $\overline{a\,b(t)}$, and in red $\overline{p_i\,q_i}$ is illustrated. Their intersections form the realising points r_i of $\overline{a\,b(t)}$.

as well. Call this intersection point r_i. The set $\{r_i\}_i$ are realising points for $\overline{a\,b(t)}$. This follows directly from the fact that $\{p_i\}_i$ and $\{q_i\}_i$ are ordered along their respective line segments, and thus $\overline{p_i\,q_i}$ never crosses another $\overline{p_j\,q_j}$. Thus for $i < j$, r_i appears before r_j along $\overline{a\,b(t)}$, implying the claim. ◀

▶ **Lemma 22.** *Let $a_1, a_2, b_1, b_2 \in \mathbb{R}^2$ together with a sequence of vertices $v_1, \ldots, v_n$ be given. If $\overline{a_1\,b_1}$ stabs through $\{\mathrm{b}_\delta(v_i)\}_i$, and $\|a_1 - b_1\| \leq \delta'$ and $\|a_2 - b_2\| \leq \delta'$, then $\overline{a_1\,b_1}$ stabs through $\{\mathrm{b}_{\delta+\delta'}(v_i)\}_i$.*

Proof. By Observation 11, $d_{\mathcal{F}}(\overline{a_1\,b_1}, \overline{a_2\,b_2}) \leq \delta'$, via the reparametrizaion (f, g) with $f(t) = (1-t)a_1 + ta_2$ and similarly $g(t) = (1-t)b_1 + tb_2$. As $p = \overline{a_1\,b_1}$ stabs through $\{\mathrm{b}_\delta(v_i)\}_i$, there exist realising points p_i along p, with p_i lying in the δ-disk centered at v_i. Then

$$\|g(f^{-1}(v_i)) - v_i\| \leq \|g(f^{-1}(v_i)) - p_i\| + \|p_i - v_i\| \leq \delta' + \delta.$$

Additionally, $q_i = g(f^{-1}(v_i))$ are ordered along $q = \overline{a_2\,b_2}$, proving the claim. ◀

▶ **Lemma 23.** *Given $r' = (r_T, r_B) \in C_{a,b}$, and edge e of B corresponding to the cell $C_{i,j}$, ε and δ like in the approximate diagonal tunnel procedure. Denote by S_δ the set of endpoints of all δ-stabbers (that is, stabbers through $\mathrm{b}_\delta(v_m)$ for $a+1 \leq m \leq i$) on the edge e_j starting at $r = B(r_B)$ and let C' be the point set computed by the algorithm. Then*

$$S_\delta \subseteq C' \subseteq S_{(1+\varepsilon)^2\delta}.$$

Algorithm 1 Approximate Diagonal Tunnel.

procedure apxDiagonalTunnel((r_T, r_B), (i, j), ε, δ)
  Let $r = B(r_B)$
  //r is the starting point of the shortcut
  for $t \in \left(\mathbb{G}_{\frac{\delta\varepsilon}{\sqrt{2}}} \cap \mathrm{b}_{3(1+\varepsilon)\delta}(v_i)\right)$ **do**
    Query $\mathcal{F}_\varepsilon$ for the distance $d_{\mathcal{F}_\varepsilon}(\overline{r\,t}, T[r_T, v_i])$ and store the answer in δ'
    if $\delta' \leq (1+\varepsilon)^2\delta$ **then**
      Mark t as eligible
      //t is an eligible endpoint of a shortcut
  Compute the convex hull H of eligible points
  if $r \in H$ **then**
    Return $C = \mathcal{D}_\delta^{(i,j)}$
  else
    Let U be the cone with apex r formed by tangents t_1 and t_2 from r to H
    Let $p_i \in H$ be a supporting point of the tangent t_i for $i \in \{1, 2\}$
    Let L be the subchain of ∂H with endpoints p_1 and p_2 which is facing r
    Let $H' \subset U$ be the set bounded by L and the rays supported by t_1 and t_2 facing away from r
    Let C' be the intersection of H' with e_j
    Return $C = (e_i \times C') \cap \mathcal{D}_\delta^{(i,j)}$

Proof. Let $y \in C'$. Then $q = B(y) \in H'$ where H' is set of points computed by the algorithm. Denote the intersection of $\overline{r\,q}$ and the boundary of H' by h. h is then a linear combination of at most two grid points whose stabbers from r have been marked as eligible i.e. who are $(1+\varepsilon)^2\delta$-stabber. Hence, Lemma 21 implies that $\overline{r\,q}$ is also a $(1+\varepsilon)^2\delta$-stabber, implying $C' \subseteq S_{(1+\varepsilon)^2\delta}$. Now let $q \in e$ be an arbitrary point such that $\overline{r\,q}$ is a δ-stabber. Let t be the last realising point of $\overline{r\,q}$. The line segment $\overline{r\,t}$ is a δ-stabber and t lies in $\mathrm{b}_\delta(v_i)$. We claim that t lies in H. Consider the set $G = \mathbb{G}_{\frac{\delta\varepsilon}{\sqrt{2}}} \cap \mathrm{b}_{\varepsilon\delta}(t)$. By the scale of the grid, t lies within the convex hull of G. Moreover $G \subset \mathrm{b}_{(1+\varepsilon)\delta}(v_i)$. Lemma 22 implies that $\overline{r\,t'}$ is a $((1+\varepsilon)\delta)$-stabber for any $t' \in G$. This in turn implies that for the first point s' of $\overline{r\,t'}$ inside $\mathrm{b}_{\delta(1+\varepsilon)}(v_a)$, $\overline{s'\,t'}$ is a $((1+\varepsilon)\delta)$-stabber, hence, t' would have been marked as an eligible endpoint of a $((1+\varepsilon)^2\delta)$-stabber. Since H is the convex hull of eligible points, it follows that $t \in \mathrm{conv}(G) \subset H$. Therefore $q \in H'$ and thus $q \in C'$. ◀

4.3 Result

We argue that the structure of $P_{i,j}^s$ as approximated by the algorithm is indeed as claimed. Namely for all i, j and s it holds that $\mathcal{R}_{\delta,s}^{(i,j)} \subset \bigcup_{0 \leq s' \leq s} P_{i,j}^{s'} \subset \mathcal{R}_{3(1+\varepsilon)^2\delta,s}^{(i,j)}$. The full proof can be found in [19]. We again consider any monotone path with s proper tunnels ending in some cell and show the set inclusion by induction. Indeed, it suffices to consider the tunnel starting in the rightmost reachable point in the lower left quadrant of the cell, if we call the approximate diagonal tunnel procedure with a distance threshold $\delta' = 3\delta$. This is implied by a lemma by Driemel and Har-Peled concerning the structure of prices of tunnels. The lemma states that if a feasible tunnel $\tau(r, q)$ costs more than 3δ then any feasible tunnel $\tau(p, q)$ with $x_p \leq x_r$ costs more than δ.

▶ **Lemma 24** (monotonicity of tunnels [21]). *Given a value $\delta > 0$ and two curves T_1 and T_2 such that T_2 is a subcurve of T_1, and given two line segments $\bar{B}_1$ and $\bar{B}_2$ such that $d_{\mathcal{F}}(T_1, \bar{B}_1) \leq \delta$ and the start (resp. end) point of T_2 is within distance δ to the start (resp. end) point of $\bar{B}_2$, then $d_{\mathcal{F}}(T_2, \bar{B}_2) \leq 3\delta$.*

The proof of the following theorem is very similar to the proof of Theorem 5. We invoke the algorithm of Section 4.1 with approximation parameter $\varepsilon' = \varepsilon/9$ and distance threshold δ, as $3(1+\varepsilon')^2 \leq (3+\varepsilon)\delta$. The main difference is the approximation of the set of points reachable by a diagonal tunnel.

▶ **Theorem 25.** *Let T and B be two polygonal curves in the plane with overall complexity n, together with values $0 < \varepsilon \leq 1$ and $\delta > 0$. There exists an algorithm with running time in $\mathcal{O}\big(kn^2\varepsilon^{-5}\log^2\big(n\varepsilon^{-1}\big)\big)$ and space in $\mathcal{O}\big(kn^2\varepsilon^{-4}\log^2\big(\varepsilon^{-1}\big)\big)$ which outputs one of the following: (i) $d_{\mathcal{S}}^k(T,B) \leq (3+\varepsilon)\delta$ or (ii) $d_{\mathcal{S}}^k(T,B) > \delta$. In any case, the output is correct.*

The running time of this algorithm can be improved even more for the special class of c-packed curves. For this, we use a known approximation scheme (see [22, 21, 7]) that uses μ-simplifications, which are a kind of approximation of the input curves. Crucially, the free-space diagram of two μ-simplifications of c-packed curves consists of only $\mathcal{O}(cn\varepsilon^{-1})$ many non-empty cells, if μ is chosen in the right way. These non-empty cells can be found using standard techniques in an output-sensitive manner.

Concretely, our algorithm first computes μ-simplifications of T and B, with $\mu = \varepsilon''\delta''$ where $\varepsilon'' = \varepsilon/20$ and $\delta'' = (1-2\varepsilon'')$, and then invokes the algorithm of Section 4.1 on the simplifications with distance threshold δ'' and approximation parameter ε''. From this, we obtain the following theorem. The detailed analysis can be found in [19].

▶ **Theorem 8.** *Let T and B be two c-packed polygonal curves in the plane with overall complexity n, together with values $0 < \varepsilon \leq 1$ and $\delta > 0$. There exists an algorithm with running time in $\mathcal{O}\big(kcn\varepsilon^{-5}\log^2\big(n\varepsilon^{-1}\big)\big)$ and space in $\mathcal{O}\big(kcn\varepsilon^{-4}\log^2\big(\varepsilon^{-1}\big)\big)$ which outputs one of the following: (i) $d_{\mathcal{S}}^k(T,B) \leq (3+\varepsilon)\delta$ or (ii) $d_{\mathcal{S}}^k(T,B) > \delta$. In any case, the output is correct.*

5 Hardness

We prove that the decision problem cannot be solved in $n^{o(k)}$ time, unless ETH fails. The full description can be found in [19]. We describe how to modify the NP-hardness reduction by Buchin, Driemel and Speckmann from [16] to suit our needs.

5.1 General idea

Our reduction from ETH works via the following intermediate problem. This link is facilitated by Pătraşcu and Williams [26].

▶ **Definition 26** (k-Table-SUM). *We are given k lists $S_1, \ldots, S_k$ of n non-negative integers $\{s_{i,1}, \ldots, s_{i,n}\}$ and a non-negative integer σ. We want to decide whether there are indices $\iota_1, \ldots, \iota_k$ such that $\sum_{i=1}^k s_{i,\iota_i} = \sigma$. We call $\sigma_j = \sum_{i=1}^j s_{i,\iota_i}$ the jth partial sum.*

Based on a k-Table-SUM instance we describe how to construct a $(4k+2)$-shortcut Fréchet distance instance consisting of the target curve T and the base curve B with the property, that they have a distance of 1 if and only if the underlying instance has a solution.

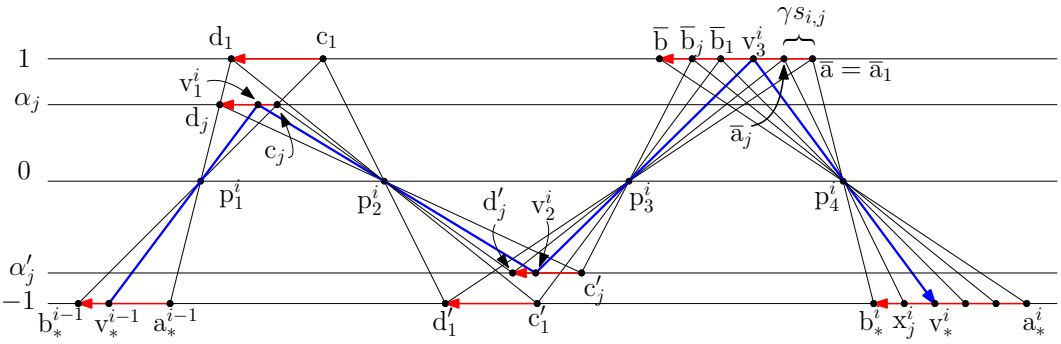


Figure 7 Schematic view of the path of a shortcut curve (in blue) through the gadget g_i in the case, where $s_{i,j}$ is selected from the ith list. Most top indices i are omitted. Furthermore, for presentation the mirror edges have horizontal overlap, while in the construction they do not.

The target curve T will lie on a horizontal line mostly going to the right. The only exceptions being so called *twists*. Twists force shortcuts traversing it to go through precisely one point, called its *focal point*. These twists are constructed by going a distance of 2 to the left, before continuing rightwards. Refer to Figure 8, points $p_1, \cdots, p_4$ and Figure 7, points $p_1^i, \cdots, p_4^i$.

The set of points in $\mathbb{R}^2$ which have a distance of at most 1 to the target curve we will call the *hippodrome*. The base curve will consist of several horizontal edges going to the left close to the boundary of the hippodrome. All other edges of the base curve will (essentially) lie outside the hippodrome. Exceptions have to be considered carefully. Any shortcut curve of B that has Fréchet distance of at most 1 to T we will call *feasible*. It is easy to see that any feasible shortcut curve must lie completely in the hippodrome. Since any edge of the base curve inside the hippodrome lies close to the boundary of it and is oriented in the opposite direction of the base curve, no feasible shortcut curve contains a large subcurve of the base curve. We will not place any edges of the base curve too close to twists, so that a shortcut must be taken to traverse these.

Intuitively we can think of the horizontal edges of the base curve as mirrors that disperse incoming light in all directions and focal points as a wall with a hole, like in a pinhole camera. A shortcut curve can be thought of as the path of a photon that tries to traverse this instance. It bounces from mirror to mirror, always passing through a focal point. A feasible shortcut curve exists if and only if it is possible to send a photon from the beginning of the base curve to the very end.

The instance as a whole can be segmented into gadgets, with a special gadget at the start and end, used to initialize and verify the solution (c.f. [16]). In between these, k gadgets will be placed, encoding one of the lists each from the k-Table-SUM instance.

5.2 Construction of the gadgets

In this section, we describe three gadgets: the encoding gadget g_i, the initialization gadget g_0 and the terminal gadget g_{k+1}. The base curve and the target curve pieces of these gadgets are then connected in the order of i. The encoding gadget is the heart-piece of our construction and constitutes the main difference to the NP-hardness reduction presented in [16].

Encoding gadget. The overall structure of a gadget g_i for some $1 \leq i \leq k$ is depicted in Figure 8. This gadget will encode the ith table $S_i = \{s_{i,1}, \ldots, s_{i,n}\}$ of the k-Table-SUM instance. As for the parameters, λ^i is the length of the entry edge, determined by the

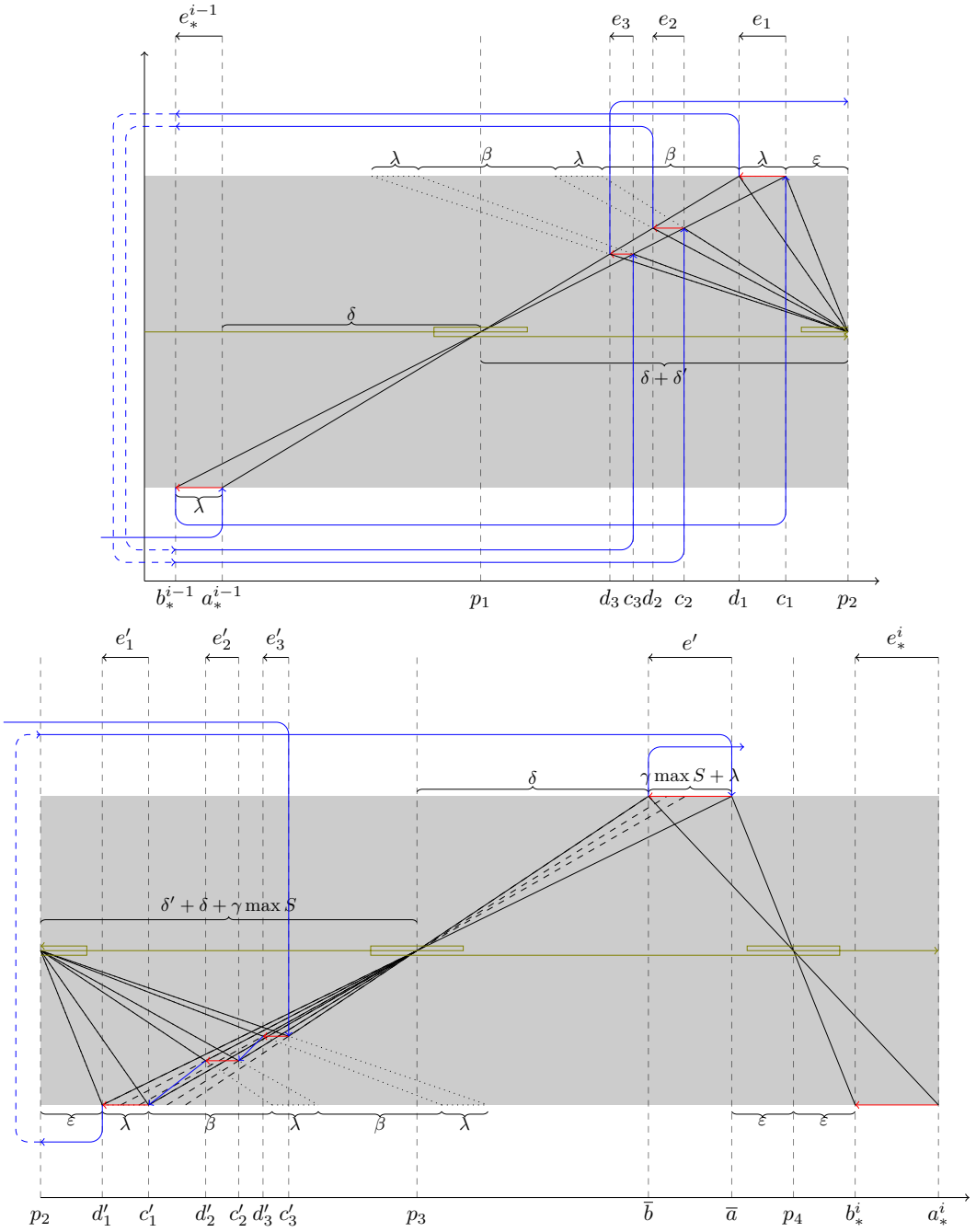


Figure 8 Detailed view of the construction of the encoding gadget. Mirror edges are red, connector edges blue and the target curve is green. Projection cones are black.

previous gadget g_{i-1}, and β is a global spacing parameter. The parameters δ^i and δ'^i are auxiliary parameters, with $\delta'^i = \lambda^i + \varepsilon$ and $\delta^i = \max(2\varepsilon + 1, (n-1)(\lambda^i + \beta) - \delta'^i)$ where ε is a globally fixed spacing parameter for twists. Excluding the entry edge of the base curve, B consists of $2n+2$ mirror edges and $\mathcal{O}(n)$ connector edges. For $1 \leq j \leq n$ the first n mirror edges e^i_j are defined by c^i_j and d^i_j, and the second n mirror edges e'^i_j are defined by c'^i_j and d'^i_j. The last two mirror edges are defined by $\bar{\mathrm{a}}^i$ and $\bar{\mathrm{b}}^i$, and a^i_* and b^i_*. The target curve T has four twists centred at $\mathrm{p}^i_1, \ldots, \mathrm{p}^i_4$. Since the index i will not change other than for the entry and exit edge, we will omit these indices in the construction of this gadget.

Intuition. The intuition behind the construction is as follows: We place the first projection point p_1 at a distance from the entry edge, such that n 'well-spaced' scaled copies of the entry edge fit 'behind' the projection point. These edges offer the choice, each edge corresponding to a single element in the table S_i. After this choice has been presented, we place another n copies of the entry edge behind the second projection point p_2. The two sets of copies correspond one-to-one, and any line starting at one copy going through p_2 only ever hits a single copy of the second set. The second set of copies is placed in such a way, such that passing through the third projection point p_3, any feasible shortcut curve receives an additional offset of $\gamma s_{i,j}$ on the edge e' corresponding to the choice of edge e_j. Lastly we define the entry edge to the next gadget.

How a solution is encoded. A shortcut curve traversing this gadget will look as follows. A shortcut curve reaches some point on the entry edge e^{i-1}_*. From here it takes a shortcut to some e^i_j, where the number j corresponds to a choice of edge to end on. The next shortcuts are forced to land on e'^i_j, then e'^i and finally e^i_*. The offset between the endpoint v^i_* of the last shortcut and b^i_* will be precisely the offset between v^{i-1}_* and b^i_* plus $\gamma s_{i,j}$ (refer to Figure 7), thus the choice of which edge to land on encodes picking an item. After k choices the offset will be approximately encoding the corresponding partial sum in the offset from b^k_*.

Initialization gadget. For the construction refer to Figure 9. Both the target curve T and the base curve B will start at x-coordinate 0 placing the start point for the base curve at $(0, 1)$, and the start point for the target curve at $(0, 0)$. The target curve will go rightwards, up to the first twist centred at $(\varepsilon + \gamma, 0)$ and continue rightwards after that. The base curve will immediately leave the hippodrome to the left and connect to the first mirror edge from $\mathrm{a}^0_* = (3\gamma + 2\varepsilon, -1)$ to $\mathrm{b}^0_* = (\gamma + 2\varepsilon, -1)$.

Terminal gadget. The terminal gadget g_{k+1} is the dual to the initialization gadget (refer to Figure 9). The entry edge from $(b^k_* + \lambda^k, -1)$ to $(b^k_*, -1)$ is defined by the previous gadget. The target curve T has a single twist at $(b^k_* + \lambda + \varepsilon, 0)$ and ends at $(b^k_* + 2\lambda + 2\varepsilon - \gamma(\sigma+1), 0)$. The base curve B connects the entry edge to $(b^k_* + 2\lambda + 2\varepsilon - \gamma(\sigma+1), 1)$ from outside the hippodrome. The final vertex $B(1)$ of the base curve is placed such that a shortcut from the entry edge e^k_* has to start precisely at x-coordinate $b^k_* + \gamma(\sigma+1)$ to hit the vertex.

5.3 Result

In the full version [19] of this paper we show that the curves can be constructed in $\mathcal{O}(kn)$ time. Furthermore, if we choose $\varepsilon = \frac{5}{2}$, $\beta = \max(32 + 4\lambda^k, \lambda^{k+1}) + 1$ and $\gamma = 16k + 6$, then the maximum numerical value of the coordinates constructed is in $\mathcal{O}(k^2 n \sum_{i=0}^{k} \max S_i)$. We then analyse the structure of a feasible solution under this choice of parameters and show that it properly encodes the partial sums, such that we can retain the solution to the k-Table-SUM instance. From this analysis, we obtain the following theorem.

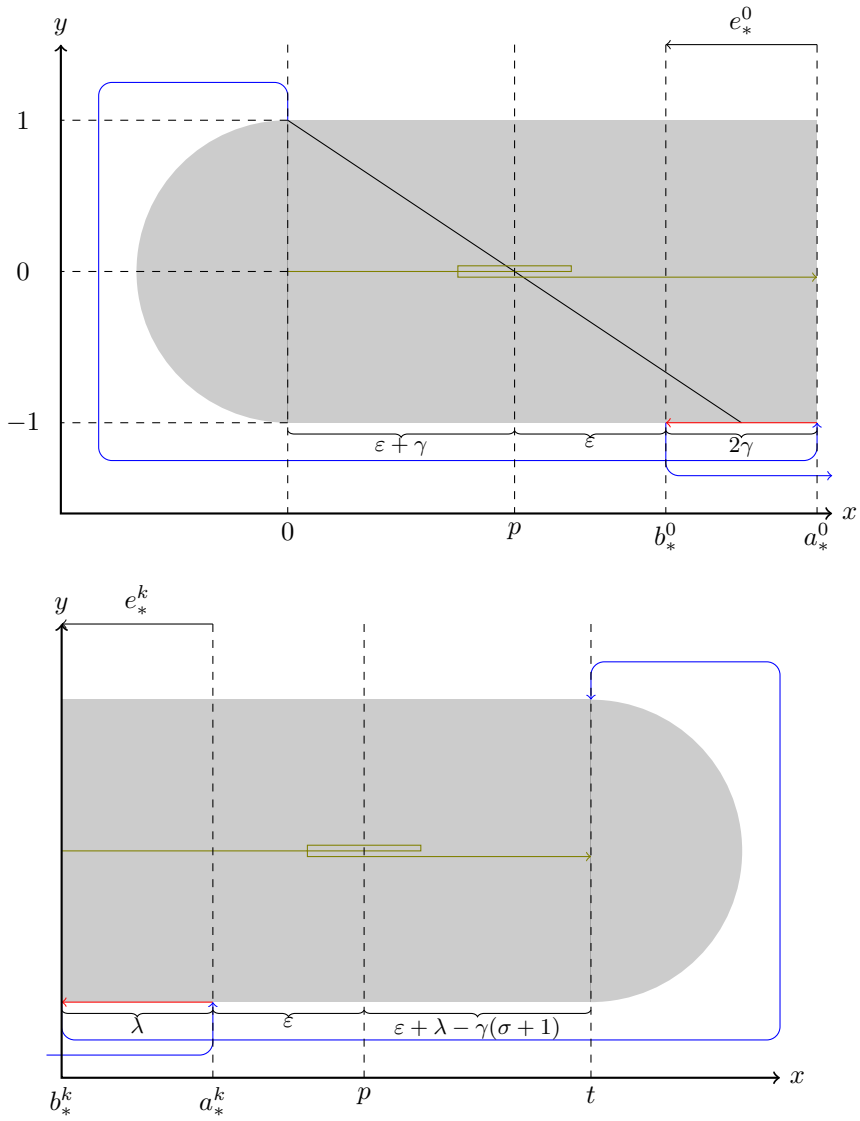


Figure 9 Construction of the Initialization and Terminal gadget. The first forced shortcut in the Initialization gadget is drawn in black. Mirror edges are red, connector edges are blue, and the target curve is green.

▶ **Theorem 6.** *Unless ETH fails, there is no algorithm for the k-shortcut Fréchet distance decision problem in $\mathbb{R}^d$ for $d \geq 2$, with running time $n^{o(k)}$.*

References

1 Amir Abboud and Kevin Lewi. Exact Weight Subgraphs and the k-Sum Conjecture. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming – Volume Part I*, pages 1–12. Springer-Verlag, 2013. doi:10.1007/978-3-642-39206-1_1.

2 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the Discrete Fréchet Distance in Subquadratic Time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 156–167. Society for Industrial and Applied Mathematics, 2013. doi:10.1137/1.9781611973105.12.

3 Hugo Alves Akitaya, Maike Buchin, Leonie Ryvkin, and Jérôme Urhausen. The k-Fréchet Distance: How to Walk Your Dog While Teleporting. In *30th International Symposium on Algorithms and Computation*, volume 149, pages 50:1–50:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.50.

4 Helmut Alt, Bernd Behrends, and Johannes Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13:251–265, 1995. doi:10.1007/BF01530830.

5 Helmut Alt and Michael Godau. Computing the Fréchet Distance between Two Polygonal Curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

6 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of Distance Measures for Planar Curves. *Algorithmica*, 38:45–58, 2003. doi:10.1007/s00453-003-1042-5.

7 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet Distance for Curves, Revisited. In *Proceedings of the 14th Conference on Annual European Symposium - Volume 14*, pages 52–63. Springer-Verlag, 2006. doi:10.1007/11841036_8.

8 Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The Discrete and Semicontinuous Fréchet Distance with Shortcuts via Approximate Distance Counting and Selection. *ACM Transactions on Algorithms*, 11(4):29:1–29:29, 2015. doi:10.1145/2700222.

9 Karl Bringmann. Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. In *55th IEEE Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

10 Karl Bringmann and Marvin Künnemann. Improved Approximation for Fréchet Distance on c-Packed Curves Matching Conditional Lower Bounds. *International Journal of Computational Geometry & Applications*, 27(1-2):85–120, 2017. doi:10.1142/S0218195917600056.

11 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016. doi:10.20382/jocg.v7i2a4.

12 Alexander M Bronstein, Michael M Bronstein, Alfred M Bruckstein, and Ron Kimmel. Partial Similarity of Objects, or How to Compare a Centaur to a Horse. *International Journal of Computer Vision*, 84(2):163, 2009. doi:10.1007/s11263-008-0147-3.

13 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets Walk the Dog: Improved Bounds for Computing the Fréchet Distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017. doi:10.1007/s00454-017-9878-7.

14 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 645–654. Society for Industrial and Applied Mathematics, 2009. doi:10.1137/1.9781611973068.71.

15 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH Says: Weak Fréchet Distance is Faster, but only if it is Continuous and in One Dimension. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2887–2901, 2019. doi:10.1137/1.9781611975482.179.

16 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, pages 367–376. Association for Computing Machinery, 2014. doi:10.1145/2582112.2582144.

17 Timothy M. Chan and Zahed Rahmati. An improved approximation algorithm for the discrete Fréchet distance. *Information Processing Letters*, 138:72–74, 2018. doi:10.1016/j.ipl.2018.06.011.

18 Connor Colombe and Kyle Fox. Approximating the (Continuous) Fréchet Distance. In *37th International Symposium on Computational Geometry*, volume 189, pages 26:1–26:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.SoCG.2021.26.

19 Jacobus Conradi and Anne Driemel. On Computing the k-Shortcut Fréchet Distance, 2022. doi:10.48550/ARXIV.2202.11534.

20 Jean-Lou De Carufel, Amin Gheibi, Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Similarity of polygonal curves in the presence of outliers. *Computational Geometry*, 47(5):625–641, 2014. doi:10.1016/j.comgeo.2014.01.002.

21 Anne Driemel and Sariel Har-Peled. Jaywalking Your Dog: Computing the Fréchet Distance with Shortcuts. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 318–337. Society for Industrial and Applied Mathematics, 2012. doi:10.1137/1.9781611973099.30.

22 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet Distance for Realistic Curves in near Linear Time. In *Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry*, pages 365–374. Association for Computing Machinery, 2010. doi:10.1145/1810959.1811019.

23 Leonidas Guibas, John Hershberger, Joseph Mitchell, and Jack Snoeyink. Approximating Polygons and Subdivisions with Minimum-Link Paths. In *International Journal of Computational Geometry & Applications*, volume 3, pages 151–162, 1994. doi:10.1007/3-540-54945-5_59.

24 Russell Impagliazzo and Ramamohan Paturi. The complexity of k-SAT. In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, pages 237–240. IEEE Computer Society, 1999. doi:10.1109/CCC.1999.766282.

25 David Jacobs, Daphna Weinshall, and Yoram Gdalyahu. Classification with Nonmetric Distances: Image Retrieval and Class Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):583–600, 2000. doi:10.1109/34.862197.

26 Mihai Pătraşcu and Ryan Williams. On the Possibility of Faster SAT Algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1065–1075. Society for Industrial and Applied Mathematics, 2010. doi:10.1137/1.9781611973075.86.

27 Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29(1):3–32, 2020. doi:10.1007/s00778-019-00574-9.

28 R.C. Veltkamp. Shape matching: similarity measures and algorithms. In *Proceedings International Conference on Shape Modeling and Applications*, pages 188–197, 2001. doi:10.1109/SMA.2001.923389.

Streaming Algorithms for Geometric Steiner Forest

Artur Czumaj
University of Warwick, Coventry, UK

Shaofeng H.-C. Jiang
Peking University, Beijing, China

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel

Pavel Veselý
Charles University, Prague, Czech Republic

Abstract

We consider an important generalization of the Steiner tree problem, the *Steiner forest problem*, in the Euclidean plane: the input is a multiset $X \subseteq \mathbb{R}^2$, partitioned into k color classes $C_1, C_2, \ldots, C_k \subseteq X$. The goal is to find a minimum-cost Euclidean graph G such that every color class C_i is connected in G. We study this Steiner forest problem in the streaming setting, where the stream consists of insertions and deletions of points to X. Each input point $x \in X$ arrives with its color $\mathsf{color}(x) \in [k]$, and as usual for dynamic geometric streams, the input is restricted to the discrete grid $\{0, \ldots, \Delta\}^2$.

We design a single-pass streaming algorithm that uses $\mathrm{poly}(k \cdot \log \Delta)$ space and time, and estimates the cost of an optimal Steiner forest solution within ratio arbitrarily close to the famous Euclidean Steiner ratio α_2 (currently $1.1547 \leq \alpha_2 \leq 1.214$). This approximation guarantee matches the state of the art bound for streaming Steiner tree, i.e., when $k = 1$. Our approach relies on a novel combination of streaming techniques, like sampling and linear sketching, with the classical Arora-style dynamic-programming framework for geometric optimization problems, which usually requires large memory and has so far not been applied in the streaming setting.

We complement our streaming algorithm for the Steiner forest problem with simple arguments showing that any finite approximation requires $\Omega(k)$ bits of space.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Theory of computation → Streaming models; Theory of computation → Computational geometry

Keywords and phrases Steiner forest, streaming, sublinear algorithms, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.47

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.04324`

Funding Work partially supported by the National key R & D program of China No. 2021YFA1000900.
Artur Czumaj: Research partially supported by the Centre for Discrete Mathematics and its Applications (DIMAP), by a Weizmann-UK Making Connections Grant, by an IBM Faculty Award, and by EPSRC award EP/V01305X/1.
Shaofeng H.-C. Jiang: Part of this work was done when the author was at the Weizmann Institute of Science and Aalto University. Partially suppported by a startup fund from Peking University, and the Advanced Institute of Information Technology, Peking University.
Robert Krauthgamer: Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, a Weizmann-UK Making Connections Grant, the Weizmann Data Science Research Center, and a Minerva Foundation grant.
Pavel Veselý: Part of this work was done when the author was at the University of Warwick. Partially supported by European Research Council grant ERC-2014-CoG 647557, by a Weizmann-UK Making Connections Grant, by GA ČR project 19-27871X, and by Charles University project UNCE/SCI/004.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 47; pp. 47:1–47:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We study combinatorial optimization problems in dynamic geometric streams, in the classical framework introduced by Indyk [36]. In this setting, focusing on low dimension $d = 2$, the input point set is presented as a stream of insertions and deletions of points restricted to the discrete grid $[\Delta]^2 := \{0, \ldots, \Delta\}^2$. Geometric data is very common in applications and has been a central object of algorithmic study, from different computational paradigms (like data streams, property testing and distributed/parallel computing) to different application domains (like sensor networks and scientific computing). Research on geometric streaming algorithms has been very fruitful, and in particular, streaming algorithms achieving $(1 + \varepsilon)$-factor estimation (i.e., approximation of the optimal value) have been obtained for fundamental geometric problems, such as k-clustering [15, 28, 35], facility location [24, 43], and minimum spanning tree (MST) [27].

Despite this significant progress, some similarly looking problems are still largely open. Specifically, for the TSP and Steiner tree problems, which are the cornerstone of combinatorial optimization, it is a major outstanding question (see, e.g., [50]) whether a streaming algorithm can match the $(1+\varepsilon)$-approximation known for the offline setting [7, 46]. In fact, the currently best streaming algorithms known for TSP and Steiner tree only achieve $O(1)$-approximation, and follow by a trivial application of the MST streaming algorithm.

While MST is closely related to TSP and Steiner tree – their optimal values are within a constant factor of each other – it seems unlikely that techniques built around MST could achieve $(1 + \varepsilon)$-approximation for either problem. Indeed, even in the offline setting, the only approach known to achieve $(1 + \varepsilon)$-approximation for TSP and/or Steiner tree relies on a framework devised independently by Arora [7] and Mitchell [46], that combines geometric decomposition (e.g., a randomly shifted quad-tree) and dynamic programming. These two techniques have been used *separately* in the streaming setting in the past: quad-tree decomposition in [3, 4, 20, 24, 27, 28, 37, 43] and dynamic programming, mainly for string processing problems, in [13, 16, 22, 25, 31, 48, 51]. However, we are not aware of any successful application of the Arora/Mitchell framework, which combines these two approaches, for any geometric optimization problem whatsoever.

We make an important step towards better understanding of these challenges by developing new techniques that *successfully adapt the Arora/Mitchell framework to streaming*. To this end, we consider a generalization of Steiner tree, the classical **Steiner Forest Problem (SFP)**. In this problem (also called *Generalized Steiner tree*, see, e.g., [7]), the input is a multiset of n *terminal* points $X \subseteq [\Delta]^2$, partitioned into k *color classes* $X = C_1 \sqcup \cdots \sqcup C_k$, presented as a dynamic stream. In addition, apart from the coordinates of the point $x \in X$, its color $\mathsf{color}(x) \in [k]$ is also revealed upon its arrival in the stream[1]. The goal is to find a minimum-cost Euclidean graph G such that every color class C_i is connected in G. Observe that the Steiner tree problem is a special case of SFP in which all terminal points should be connected (i.e., $k = 1$). Similar to the Steiner tree problem, a solution to SFP may use points other than X; those points are called *Steiner points*.

▶ Remark. In the literature, the term SFP sometimes refers to the *special case* where *each color class contains only a pair of points*, i.e., each $C_i = \{s_i, t_i\}$ [11, 12, 14, 17, 33]. It is not difficult to see (see [49]) that one can reduce one problem into another in the standard setting of *offline algorithms*. The special case of pairs is often simpler to present and does

[1] The points are arriving and leaving in an arbitrary order; there is no requirement that each color arrives in a batch, i.e., that its points are inserted/deleted consecutively in the stream.

not restrict algorithmic generality for offline algorithms, even though the setting considered here is more natural for applications (see [45]). Nevertheless, the definition used here allows for better parameterization over the number of colors k.

Background. While one might hope for a streaming algorithm for SFP with $o(k)$ space, we observe that *this task is impossible*, even in the one-dimensional case. In Theorem 4.1, we present a reduction that creates instances of SFP in $\mathbb{R}$ such that every streaming algorithm achieving any finite approximation ratio for SFP must use $\Omega(k)$ bits of space. This holds even for insertion-only algorithms, and even if all color classes are of size at most 2.

Even for $k = 1$, which is the famous Steiner tree problem, the only known streaming algorithm is to estimate the cost of a minimum spanning tree (MST) and report it as an estimate for SFP. It is useful to recall here the *Steiner ratio* α_d, defined as the supremum over all point sets $X \subseteq \mathbb{R}^d$, of the ratio between the cost of an MST and that of an optimal Steiner tree. The famous Steiner ratio Gilbert-Pollak Conjecture [29] speculates that $\alpha_2 = \frac{2}{\sqrt{3}} \approx 1.1547$, but the best upper bound to date is only that $\alpha_2 \leq 1.214$ [23]. It follows that employing the streaming algorithm of Frahling, Indyk, and Sohler [27], which $(1+\varepsilon)$-approximates the MST cost using space $\mathrm{poly}(\varepsilon^{-1} \log \Delta)$, immediately yields a streaming algorithm that $(\alpha_2 + \varepsilon)$-approximates the Steiner tree cost, with the same space bound.

1.1 Our Contribution

Our main result is a *space and time* efficient, *single-pass* streaming algorithm that estimates the optimal *cost* OPT for SFP within $(\alpha_2 + \varepsilon)$ factor. Our space bound is nearly optimal in terms of the dependence in k, since any finite approximation for SFP requires space $\Omega(k)$ (Theorem 4.1), and our ratio matches the state of the art for Steiner tree (i.e., $k = 1$).

▶ **Theorem 1.1** (Informal Version of Theorem 3.1). *For any integers $k, \Delta \geq 1$ and any fixed $\varepsilon > 0$, one can with high probability $(\alpha_2 + \varepsilon)$-approximate the SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic stream, using space and query/update times bounded by* $\mathrm{poly}(k \cdot \log \Delta)$.

We notice that while the algorithm in Theorem 1.1 returns only an approximate cost of the optimal solution and it cannot return the entire approximate solution (since the output is of size $\Omega(n)$), an additional desirable feature of our algorithm in Theorem 1.1 is that it can return information about the colors in the trees in an approximate solution. That is, our algorithm can maintain a partition of the colors used in X into $I_1, \ldots, I_r \subseteq \{1, \ldots, k\}$, so that the sum of the costs of the minimum-cost Steiner trees for sets $\bigcup_{i \in I_j} C_i$ is an $(\alpha_2 + \varepsilon)$-approximation of SFP. It is worth noting that in estimating the optimal cost, our algorithm does use Steiner points. This means that the MST costs for sets $\bigcup_{i \in I_j} C_i$ of the aforementioned partition may be by an $O(1)$ factor larger than the estimate of the algorithm.

Comparison to a Simple Exponential-time Approach. As we shall discuss in Section 1.2, a simple brute force enumeration combined with linear sketching techniques yields a streaming algorithm also with near-optimal space, but significantly worse running time that is *exponential* in k. Technically, while this approach demonstrates the amazing power of linear sketching, its core is exhaustive search rather than an algorithmic insight, and thus it is quite limited, offering no path for improvements or extensions. Furthermore, the $\mathrm{poly}(k)$ running time in Theorem 1.1 is exponentially better than the exhaustive search, which seems to be a limit of what linear sketching could possibly achieve. Therefore even though the primary focus of streaming algorithms is on their space complexity, the improvement of the running

time is critical in terms of pursuing efficient algorithms and making our techniques broadly applicable. Indeed, similar exponential improvements of running time have been of key importance in the advances of various other fundamental streaming problems, for instance, for moment estimation the query time was improved from $\text{poly}(\varepsilon^{-1})$ to $\text{poly}\log(\varepsilon^{-1})$ [40], and for heavy hitters from $\text{poly}(n)$ to $\text{poly}\log(n)$ [44].

1.1.1 Technical Contribution: Adapting Arora's Framework to Streaming

We introduce a method for efficient streaming implementation of an offline Arora-style [7] dynamic-programming framework based on the quad-tree decomposition. This method, which is probably the first of its kind for geometric streams, is our main technical contribution.

In the offline setting, Borradaile, Klein, and Mathieu [14] and then Bateni and Hajiaghayi [11] extended the Arora's approach to obtain a polynomial-time approximation scheme (PTAS) for SFP. The key insight of these works is that one can tweak the optimal solution so that its cost remains nearly optimal, but it satisfies certain structural properties that allow for designing a suitable dynamic program. In Section 2, we review the structural theorem and the dynamic-programming approach for SFP from [11, 14] in more detail.

The main difficulty of using the Arora-style approach in low-space streaming is that in general, such approach requires access to all input points, that is, $\Omega(n)$ space to store $\Omega(n)$ leaves at the bottom of the quad-tree input decomposition that have to be considered as basic subproblems. In order to ensure a low-space implementation of the Arora-style framework in the streaming setting, we will use only $O(k \log \Delta)$ non-uniform leaf nodes of the quad-tree, each corresponding to a square. The definition of these leaf nodes is one of the novel ideas needed to make the dynamic-programming approach work in the streaming setting. Moreover, since each internal node in the quad-tree has degree 4, the total number of quad-tree squares to consider is thus $O(k \cdot \log \Delta)$.

The next challenge is that for the dynamic program to run, we need to find an $(\alpha_2 + \varepsilon)$-approximate estimation for each new leaf and each dynamic-programming subproblem associated with it. The definition of leaf squares will enable us to reduce it to estimating the MST cost for a certain subset of points inside the square. It would then be natural to just employ the MST sketch designed in [27] to estimate the MST cost, in a black box manner. However, the leaf squares are not known in advance as we can only find them after processing the stream and thus, it is impossible to build the MST sketch for each leaf square and each subproblem associated with it. To overcome this, we observe that in essence, the MST sketch consists of uniformly sampled points (with suitably rounded coordinates). We thus obtain the MST sketch for each color separately and only use the sampled points that are relevant for the subproblem to estimate the MST cost for the subproblem, in a way similar to [27].

However, due to restricting the attention to a single subproblem, the original analysis of the MST sketch in [27] has to be modified to deal with additional technical challenges. For instance, we may not sample any point relevant to a leaf square in case there are relatively few points in it. We need to account for the error arising from this case in a global way, by observing that then the MST cost inside the leaf square is a small fraction of the overall cost.

Further, to be able to accurately enumerate the subproblems for a leaf square, we need to know the set of color classes that intersect every leaf square, but unfortunately doing so exactly is impossible in the streaming setting. To this end, we employ a δ-net for a small-enough δ, so that the intersection test can be approximately done by only looking at the nearby net points. We show that this only introduces a small error for SFP, and that this δ-net can be constructed in a dynamic stream, using space by only a factor of $\text{poly}\log(\Delta)$ larger than the net. Finally, we apply the dynamic program using our leaf nodes as basic subproblems to obtain the estimation.

1.2 Could Other Approaches Work?

A Simple Exponential-time Streaming Algorithm Based on Linear Sketching. An obvious challenge in solving SFP is to determine the connected components of an optimal (or approximate) solution. Each color class must be connected, hence the crucial information is which *colors* are connected together (even though they do not have to be). Suppose momentarily that the algorithm receives an advice with this information, which can be represented as a partition of the color set $[k] = P_1 \sqcup \cdots \sqcup P_l$. Then a straightforward approach for SFP is to solve the Steiner tree problem separately on each part P_j (i.e., the union of some color classes), and report their total cost. In our streaming model, we could apply the aforementioned MST-based algorithm [27], using space $\mathrm{poly}(\varepsilon^{-1} \log \Delta)$, to achieve $(\alpha_2 + \varepsilon)$-approximation, and we would need $l \le k$ parallel executions of it (one for each P_j). An algorithm can bypass having such an advice by enumeration, i.e., by trying in parallel all the k^k partitions of $[k]$ and reporting the minimum of all their outcomes. This would still achieve $(\alpha_2 + \varepsilon)$-approximation, because each possible partition gives rise to a feasible SFP solution (in fact, this algorithm optimizes the sum-of-MST objective). However, this naive enumeration increases the space and time complexities by a factor of $O(k^k)$. We can drastically improve the space complexity by the powerful fact that the MST algorithm of [27] is based on a *linear sketch*, i.e., its memory contents is obtained by applying a (randomized) linear mapping to the input X. The huge advantage is that linear sketches of several point sets are mergeable. In our context, one can compute a linear sketch for each color class C_i, and then obtain a sketch for the union of some color classes, say some P_j, by simply adding up their linear sketches. These sketches are randomized, and hence, one has to make sure they use the same random coins (same linear mapping), and also to amplify the success probability of the sketches so as to withstand a union bound over all 2^k subsets $P_j \subseteq [k]$. This technique improves the space complexity and update time to be basically $\mathrm{poly}(k\varepsilon^{-1} \log \Delta)$, however the query time is still *exponential* in k. We state this result as follows, and its formal proof can be found in the full version.

▶ **Theorem 1.2.** *For any integers $k, \Delta \ge 1$ and any $0 < \varepsilon < 1/2$, one can with high probability $(\alpha_2 + \varepsilon)$-approximate SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic geometric stream, using space and update time of $O(k^2 \cdot \mathrm{poly}(\varepsilon^{-1} \cdot \log \Delta))$ and with query time $O(k^k) \cdot \mathrm{poly}(\varepsilon^{-1} \cdot \log \Delta)$.*

Tree Embedding. Indyk [36] incorporated the low-distortion tree embedding approach of Bartal [9] to obtain dynamic streaming algorithms with $O(\log \Delta)$ ratio for several geometric problems. This technique can be easily applied to SFP as well, but the approximation ratio is $O(\log \Delta)$ which is far from what we are aiming at.

Other $O(1)$-Approximate Offline Approaches. In the regime of $O(1)$-approximation, SFP has been extensively studied using various other techniques, not only dynamic programming. For example, in the offline setting there are several 2-approximation algorithms for SFP using the primal-dual approach and linear programming relaxations [1, 30, 38], and there is also a combinatorial (greedy-type) constant-factor algorithm called *gluttonous* [33]. Both of these approaches work in the general metric setting. While there are no known methods to turn the LP approach into low-space streaming algorithms, the gluttonous algorithm of [33] might seem amenable to streaming. Indeed, it works similarly to Kruskal's MST algorithm as it also builds components by considering edges in the sorted order by length, and the MST cost estimation in [27] is similar in flavor to Kruskal's algorithm. However, a crucial

difference is that the gluttonous algorithm stops growing a component once all terminals inside the component are satisfied, i.e., for each color i, the component either contains all points of C_i, or no point from C_i. This creates a difficulty that the algorithm must know for each component whether or not it is "active" (i.e., not satisfied), and there are up to n components, requiring overall $\Omega(n)$ bits of space. This information is crucial because "inactive" components do not have to be connected to anything else, but they may help to connect two still "active" components in a much cheaper way than by connecting them directly. Furthermore, we have a simple one-dimensional example showing that the approximation ratio of the gluttonous algorithm cannot be better than 2 (moreover, its approximation guarantee in [33] is significantly larger than 2). In comparison, our dynamic-programming approach gives a substantially better ratio of $\alpha_2 + \varepsilon$. Nevertheless, an interesting open question is whether the gluttonous algorithm admits a low-space streaming implementation.

1.3 Related Work

SFP has been extensively studied in operations research and algorithmic communities for several decades. This problem has been also frequently considered as a part of a more general network design problem (see, e.g., [1, 30, 38, 45]), where one could require for some subsets of vertices to maintain some higher inter-connectivity.

In the classical, offline setting, it is known that the Steiner tree problem is APX-hard in general graphs and in high-dimensional Euclidean spaces, and the same thus holds for SFP as it is more general. In general graphs, a 2-approximation algorithm is known due to Agrawal et al. [1] (see also [30, 38]). These 2-approximation algorithms rely on linear-programming relaxations, and the only two combinatorial constant-factor approximations for SFP were recently devised by Gupta and Kumar [33] and by Groß et al. [32]. For low-dimensional Euclidean space, which is the main focus of our paper, Borradaile et al. [14] and then Bateni and Hajiaghayi [11] obtained a $(1+\varepsilon)$-approximation by applying dynamic programming and geometric space decomposition, significantly extending the approach of Arora [7]. Further extensions of the dynamic-programming approach have led to a PTAS for metrics of bounded doubling dimension [17], planar graphs, and graphs of bounded treewidth [12].

There has been also extensive work for geometric optimization problems in the dynamic (turnstile) streaming setting, with low space. Indyk [36] designed $O(\log \Delta)$-approximate algorithms for several basic problems, like MST and matching. Follow-up papers presented a number of streaming algorithms achieving approximation ratio of $1+\varepsilon$ or $O(1)$ to the cost of Euclidean MST [27], various clustering problems [28, 34], geometric facility location [24, 43], earth-mover distance [3, 36], and various geometric primitives (see, e.g., [5, 18, 19, 26]). Some papers have studied geometric problems with superlogarithmic but still sublinear space and in the multipass setting (see, e.g., [6]). We are not aware of prior results for the (Euclidean) Steiner tree problem nor SFP in the streaming context, although $(1+\varepsilon)$-approximation of the MST cost [27] immediately gives a $(\alpha_2+\varepsilon)$-approximation of the Euclidean Steiner tree.

2 Preliminaries

For $x, y \in \mathbb{R}^2$, let $\operatorname{dist}(x,y) := \|x-y\|_2$. For $S, T \subseteq \mathbb{R}^2$, let $\operatorname{dist}(S,T) := \min_{x\in S, y\in T} \operatorname{dist}(x,y)$. For $S \subseteq \mathbb{R}^2$, let $\operatorname{diam}(S) := \max_{x,y\in S} \operatorname{dist}(x,y)$. A *$\rho$-packing* $S \subseteq \mathbb{R}^2$ is a point set such that $\forall x, y \in S$, $\operatorname{dist}(x,y) \geq \rho$. A *$\rho$-covering* of X is a subset $S \subseteq \mathbb{R}^2$ such that $\forall x \in X$, $\exists y \in S$, $\operatorname{dist}(x,y) \leq \rho$. We call $S \subseteq \mathbb{R}^2$ a *ρ-net* for X if it is both a ρ-packing and a ρ-covering for X.

▶ **Fact 2.1** (Packing Property, cf. [47]). *A ρ-packing $S \subseteq \mathbb{R}^d$ has size $|S| \leq \left(\frac{3\,\mathrm{diam}(S)}{\rho}\right)^d$.*

Metric Graphs. We call a weighted undirected graph $G = (X, E, w)$ a *metric graph* if for every edge $\{u, v\} \in E$, $w(u, v) = \mathrm{dist}(u, v)$, and we let $w(G)$ to be the sum of the weights of edges in G. A solution F of SFP may be interpreted as a metric graph. For a set of points S (e.g., S can be a square), let $F|_S$ be the subgraph of F formed by edges whose both endpoints belong to S. Note that we think of F as a *continuous* graph in which every point of an edge is itself a vertex, so $F|_S$ may be interpreted as a geometric intersection of F and S.

Randomly-Shifted Quad-trees [7]. Without loss of generality, let Δ be a power of 2, and let $L := 2\Delta$. A quad-tree sub-division is constructed on $[L]^2$. In the quad-tree, each node u corresponds to a square R_u and if it's not a leaf, it has four children, whose squares partition R_u. The squares in the quad-tree are of side-lengths that are powers of 2, and we say a square R is of level i if its side-length is 2^i (this is also the level of its corresponding node in the quad-tree, where leaves have level 0 and the root is at level $\log_2 L$). The whole quad-tree is shifted by a random vector in $[-\Delta, 0]^2$. Throughout, we assume a randomly-shifted quad-tree has been sampled from the very beginning. When we talk about a quad-tree square R, we interpret it as the point set that consists of both the boundary and the internal points. For $i = 0, \ldots \log_2 L$, let 2^i-*grid* $\mathcal{G}_i \subset \mathbb{R}^2$ be the set of centers of all level-i squares in the quad-tree.

2.1 Review of Dynamic Programming (DP) [11, 14]

The PTAS for geometric SFP in the offline setting [11, 14] is based on the quad-tree sub-division framework of Arora [7], with modifications tailored to SFP. For each square R in the (randomly-shifted) quad-tree,

- $O(\varepsilon^{-1} \log L)$ equally-spaced points on the boundary edges are designated as *portals*; and
- the $\gamma \times \gamma$ sub-squares of R are designated as *cells* of R, denoted $\mathsf{cell}(R)$, where $\gamma = \Theta(\varepsilon^{-1})$ is a power of 2.

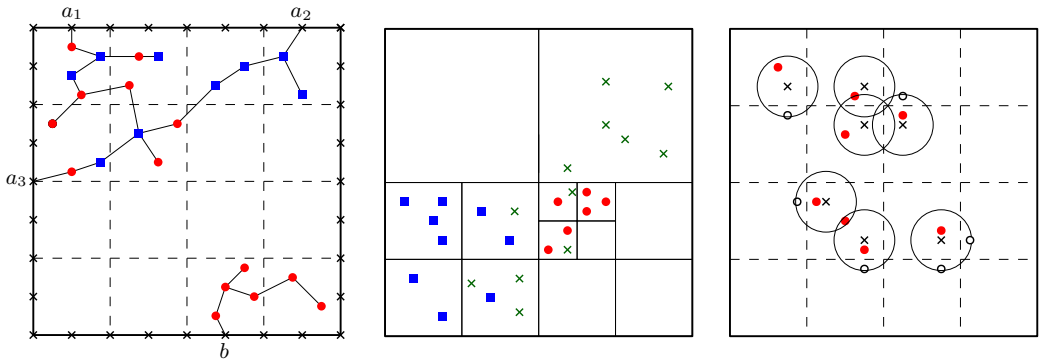
For each square R in the quad-tree, let ∂R be the boundary of R (which consists of four segments). The following is the main structural theorem from [11], and an illustration of it can be found in Figure 1a.

▶ **Theorem 2.2** ([11]). *For an optimal solution F of SFP, there is a solution F' (defined with respect to the randomly-shifted quad-tree), such that*

1. *$w(F') \leq (1+O(\varepsilon)) \cdot w(F)$ with constant probability (over the randomness of the quad-tree);*
2. *For each quad-tree square R, $F'|_{\partial R}$ has at most $O(\varepsilon^{-1})$ components, and each component of $F'|_{\partial R}$ contains a* portal *of R;*
3. *For each quad-tree square R and each cell P of R, if two points $x_1, x_2 \in X \cap P$ are connected to ∂R via F', then they are connected in $F'|_R$; this is called the* cell property.

It suffices to find the optimal solution that satisfies the structure defined in Theorem 2.2. This is implemented using dynamic programming (DP), where a subproblem of the DP is identified as a tuple (R, A, f, Π), specified as follows:

- R is a quad-tree square;
- A is a set of $O(\varepsilon^{-1})$ active portals through which the local solution enters/exits R;
- $f : \mathsf{cell}(R) \to 2^A$ s.t. for $S \in \mathsf{cell}(R)$, $f(S)$ represents the subset of A that S connects to;
- Π is a partition of A, where active portals in each part of Π have to be connected outside of R (in a larger subproblem).



(a) structural property. **(b)** simple squares. **(c)** compatibility checking.

Figure 1 Illustrations of the structural properties of Theorem 2.2 (Figure 1a), construction of simple squares by Algorithm 1 (Figure 1b) and the approximate compatibility checking idea in Section 3.2.2 (Figure 1c). Figure 1a shows a square R with portals (crosses) on ∂R, the 4×4 cells of R, and the part of solution $F'|_R$, such that $F'|_R$ passes ∂R through four portals a_1, a_2, a_3, b on the sides, and in each cell, points that are connected by F' to ∂R in the cell are connected in R. Figure 1b demonstrates the 13 simple squares constructed by Algorithm 1 for the three colors (noting that the 5 empty squares are also included as simple squares). In Figure 1c, red points are data points, cross points are the net points constructed from the data, and the black hollowed points are the added points for cells that are close-enough to a net point (for simplicity, not shown for cells containing a data point).

The use of R and A is immediate, and f is used to capture the connectivity between cells and portals (this suffices because we have the "cell property" in Theorem 2.2). Finally, Π is used to ensure feasibility, since a global connected component may be broken into several components in square R, and it is crucial to record whether or not these components still need to be connected from outside of R. An optimal solution for subproblem (R, A, f, Π) is defined as a minimum weight metric graph in R that satisfies the constraints A, f, Π. Standard combinatorial bounds show that the number of subproblems associated with each square is bounded by $(\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$ (see [11]).

▶ Remark 2.3. Strictly speaking, we use a simplified definition of DP subproblems, compared to [11]. Namely, one can additionally require that for any two cells $S, S' \in \mathsf{cell}(R)$, either $f(S) = f(S')$ or $f(S) \cap f(S') = \emptyset$ and that any active portal in A appears in $f(S)$ for some cell S. Then, f defines a partition of $\mathsf{cell}(R)$ and of A into local components inside R (taking into account only components connected to ∂R), and Π should encode which local components need be connected from the outside of R, implying that Π should be a partition of local components (instead of A). Thus, Π can also be thought of as a partition of the partition of A induced by f. We chose to give a more relaxed definition of DP subproblems as it is sufficient for describing how to implement the DP approach in the streaming setting.

3 Streaming Dynamic Programming: k^3-time-and-space Algorithm

In this section, we prove our main result, Theorem 1.1, restated with more precise bounds. Formally, we call the time for processing inserting/deleting one point as *update time*, and for reporting the estimate of OPT the *query time*.

▶ **Theorem 3.1.** *For any integers $k, \Delta \geq 1$ and any $0 < \varepsilon < 1/2$, one can with high probability $(\alpha_2 + \varepsilon)$-approximate the SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic geometric stream, using space and update time of $k^3 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \cdot \log \Delta)$ and with query time bounded by $k^3 \cdot \mathrm{poly}(\log k) \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$.*

Overview. Our approach for the streaming algorithm relies on a novel modification of the known PTAS for SFP in the offline setting [11, 14], which is based on dynamic programming (DP). One important reason why the DP requires $\Omega(n)$ space is that $\Omega(n)$ leaves in the quad-tree have to be considered as basic subproblems which correspond to singletons. To make the DP use only $\widetilde{O}(\mathrm{poly}(k))$ space, we will use only $\widetilde{O}(\mathrm{poly}(k))$ leaf nodes. Then, since each internal node in the quad-tree has degree 4, the total number of squares to consider is $\widetilde{O}(\mathrm{poly}(k))$. Furthermore, we design an algorithm that runs in time and space $\widetilde{O}(\mathrm{poly}(k))$ and finds an $(\alpha_2 + \varepsilon)$-approximate estimation for each new leaf and each DP subproblem associated with it. Finally, we apply the DP using such leaves as basic subproblems to obtain the estimation. We start in Section 3.1 with a description of this approach in the offline setting, and we make it streaming in Section 3.2. The proof of Theorem 3.1 is in Section 3.3.

3.1 Offline Algorithm

New Definition of Basic Subproblems. Each of our new leaves in the DP will be a *simple square* defined below. The idea behind the definition is also simple: If no color class is contained in R, then all points inside R must be connected to ∂R, so we can make better use of the cell property in Theorem 2.2.

▶ **Definition 3.2** (Simple Squares). *We call a square R* simple *if for every $1 \le i \le k$, $C_i \cap R \neq C_i$. In other words, there is no color class totally contained in R.*

We note that the number of all possible simple squares can still be large (in particular, any empty square is simple as well as any square containing a single point of color C_i with $|C_i| \ge 2$), and we use Lemma 3.3 below to show the existence of a small subset of simple squares that covers the whole instance and can be found efficiently. Our new leaves are naturally defined using such subset of squares.

▶ **Lemma 3.3.** *There is a subset $\mathcal{R}$ of disjoint simple squares, such that the union of the squares in $\mathcal{R}$ covers X, and $|\mathcal{R}| = O(k \cdot \log \Delta)$.*

Proof. Consider the recursive procedure specified in Algorithm 1 that takes as input a square R and returns a set of disjoint simple squares $\mathcal{R}$ that covers R; see Figure 1b for an illustration of the outcome of the procedure. For our proof, we apply the procedure with R being the root square covering the whole instance. Suppose the procedure returns $\mathcal{R}$.

Algorithm 1 Algorithm for finding simple squares.

```
1: procedure SIMP-SQUARE(R)
2:     if R is simple then return {R}
3:     else
4:         let {R_i}_i be the child squares of R in the quad-tree
5:         return ⋃_i SIMP-SQUARE(R_i)
```

We call a square R intermediate square if it is a square visited in the execution of the algorithm and it is not simple (i.e., R contains a color class). We observe that $|\mathcal{R}|$ is $O(1)$ times the number of intermediate squares. On the other hand, each color C_i can be totally contained in at most $O(\log \Delta)$ intermediate squares. Therefore, $|\mathcal{R}| = O(k \cdot \log \Delta)$. ◀

Approximation Algorithm for Subproblems on Simple Squares. Fix some simple square R. We now describe how each DP subproblem (R, A, f, Π) associated with R can be solved directly using an α_2-approximate algorithm that is amenable to the streaming setting.

Since R is a simple square, every point in R has to be connected to the outside of R, as otherwise the color connectivity constraint is violated. Hence, by the cell property of Theorem 2.2, for every cell $R' \in \mathsf{cell}(R)$, all points in R' are connected in R. Therefore, we enumerate all possible partitions of $\mathsf{cell}(R)$ that is consistent with the f constraint. For each partition, we further check whether it satisfies the constraint defined by Π. To do so, for each cell $R' \in \mathsf{cell}(R)$, we scan through all colors, and record the set of colors $\mathcal{C}_{R'} \subseteq \mathcal{C}$ that intersects R'. The $\mathcal{C}_{R'}$'s combined with the f constraint as well as the enumerated connectivity between cells suffice for checking the Π constraint.

Observe that every feasible solution of the subproblem corresponds to the above-mentioned partition of cells. Therefore, to evaluate the cost of the subproblem, we evaluate the sum of the MST costs of the parts in each partition and return the minimum one. The time complexity for evaluating each subproblem is bounded since $|A| = O(\varepsilon^{-1})$ and $|\mathsf{cell}(R)| = O(\varepsilon^{-2})$. The approximation ratio is α_2 because we use MST instead of Steiner tree for evaluating the cost. Using MST will enable us to implement this algorithm in the streaming setting.

3.2 Building Blocks for Streaming Algorithm

3.2.1 Constructing Simple Squares in the Streaming Setting

The first step is to construct a set of simple squares, as in Lemma 3.3, and an offline construction is outlined in Algorithm 1. For the streaming construction of simple squares, we observe that the key component of Algorithm 1 is a subroutine that tests whether a given square is simple or not. To implement the subroutine, we use a streaming algorithm to compute the bounding square for each color, and we test whether a given square contains any bounding square as a sub-square.

▶ **Lemma 3.4.** *Algorithm 1 can be implemented in the streaming setting, using space* $O(k \operatorname{poly}\log\Delta)$ *and in time* $O(k \operatorname{poly}\log\Delta)$ *per stream update, with success probability at least* $1 - \operatorname{poly}(\Delta^{-1})$.

Proof. The proof is ommited due to the space limit and can be found in the full version. ◀

3.2.2 Approximate Compatibility Checking

Suppose we applied Lemma 3.4 to obtain a set of simple squares $\mathcal{R}$. We proceed to evaluate the cost of subproblems associated with each simple square. Fix a simple square $R \in \mathcal{R}$. We next describe how to evaluate the cost for every subproblem associated with R, in streaming.

Suppose we are to evaluate the cost of a subproblem (R, A, f, Π). Since R is known, we have access to $\mathsf{cell}(R)$, and hence, we can enumerate the connectivity between the cells, which is a partition of $\mathsf{cell}(R)$, on-the-fly without maintaining other information about the input. Similarly, we can check the compatibility of the partition of cells with the f constraint, since the constraint only concerns the information about A and the partition. Then, when we check the compatibility of the partition of cells with Π, in the offline setting we need to compute the set of colors $\mathcal{C}_{R'} \subseteq \mathcal{C}$ that a cell R' intersects.

However, computing this set $\mathcal{C}_{R'}$ is difficult in the streaming setting, even if there is only one color C. Indeed, testing whether color C has an intersection with cell R' can be immediately reduced to the INDEX problem (see e.g. [42] for the definition), which implies an $\Omega(n)$ space bound, where n is the number of points of color C. Therefore, we need to modify the offline algorithm, and only test the intersection approximately.

To implement the approximate testing, for every color $C \in \mathcal{C}$, we impose a $\delta \cdot \operatorname{diam}(C)$-net N_C for C (see Section 2), where $\delta := O\left(\varepsilon^3 (k \log\Delta)^{-1}\right)$. A streaming algorithm in Lemma 3.5 is presented to compute this net. To be exact, the streaming algorithm in Lemma 3.5 returns

a set of points N_C such that for any point $x \in N_C$ at least one point in C is within distance $\delta \cdot \mathrm{diam}(C)$ from x (so N_C does not contain net points that are far away from C). Hence, take $D_C := \mathrm{diam}(N_C)$, and we have $D_C \in (1 \pm \delta) \cdot \mathrm{diam}(C)$. Then, for each cell $R' \in \mathsf{cell}(R)$ of each simple square R, we examine each point in N_C, and if $\mathrm{dist}(R', N_C) \le \delta \cdot D_C$, we add a new point $x \in R'$ such that $\mathrm{dist}(x, N_C) \le \delta \cdot D_C$ to the stream, and assign it color C. Furthermore, we declare C intersects R'. This idea is visually demonstrated in Figure 1c.

▶ **Lemma 3.5.** *There is an algorithm that for every $0 < \rho \le 1$ and every point set $S \subset \mathbb{R}^2$ provided as a dynamic geometric stream, computes a subset $N_S \subset \mathbb{R}^2$ that is a $\rho \cdot \mathrm{diam}(S)$-net for S such that for every $x \in N_S$ there exists $y \in S$ with $\mathrm{dist}(x, y) \le \rho \cdot \mathrm{diam}(S)$, with probability at least $1 - \mathrm{poly}(\Delta^{-1})$, using space $O(\rho)^{-2} \cdot \mathrm{poly}\log\Delta$, and running in time $O(\rho)^{-2} \cdot \mathrm{poly}\log\Delta$ per stream update.*

Proof. The proof is ommited due to the space limit and can be found in the full version. ◀

In fact, such procedure of adding points is oblivious to the subproblem, and should be done only once as a *pre-processing step* before evaluating any subproblems. Therefore, the subproblems are actually evaluated on a new instance $(X', \mathcal{C}')$ after the pre-processing. Since we apply Lemma 3.5 for every color i, and by the choice of δ, the space complexity for the pre-processing step is $O\left(k^3 \cdot \mathrm{poly}(\varepsilon^{-1}\log\Delta)\right)$, and the time complexity per update is bounded by this quantity. Next, we upper bound the error introduced by the new instance.

▶ **Lemma 3.6.** *Let* OPT *be the optimal SFP solution for the original instance $(X, \mathcal{C})$, and let* OPT′ *be that for $(X', \mathcal{C}')$. Then $w(\mathrm{OPT}) \le w(\mathrm{OPT}') \le (1+\varepsilon) \cdot w(\mathrm{OPT})$.*

Proof. Since OPT′ is a feasible solution for $(X, \mathcal{C})$, we obtain $w(\mathrm{OPT}) \le w(\mathrm{OPT}')$ by the optimality of OPT. It remains to prove the other side of the inequality.

Recall that for every color C, we use Lemma 3.5 to obtain a $\delta \cdot \mathrm{diam}(C)$-net N_C and estimate $\mathrm{diam}(C)$ using $D_C := \mathrm{diam}(N_C)$. Now, for every cell R' of every simple square, if $\mathrm{dist}(R', N_C) \le \delta \cdot D_C$ for some color C we add a point x to C satisfying $d(x, N_C) \le \delta \cdot D_C$, and for any other color $C' \ne C$ with $\mathrm{dist}(R', N_{C'}) \le \delta \cdot D_{C'}$, we add the same point x to C'. Note that we add at most one distinct point for each cell. Let $z \in N_C$ satisfy $d(x, z) \le \delta \cdot D_C$, then adding x increases OPT by at most $2\delta \cdot D_C \le 3\delta \cdot \mathrm{diam}(C)$, since one can connect x to $y \in C$ such that $\mathrm{dist}(y, z) \le \delta \cdot \mathrm{diam}(C)$ (such y must exist due to Lemma 3.5).

Since there are in total at most $O(k \log\Delta \cdot \varepsilon^{-2})$ cells in all simple squares by Lemma 3.3, the total increase of the cost is at most $O(\delta \cdot k \log\Delta \cdot \varepsilon^{-2}) \cdot \max_{C \in \mathcal{C}} \mathrm{diam}(C) \le \varepsilon \max_{C \in \mathcal{C}} \mathrm{diam}(C) \le \varepsilon w(\mathrm{OPT})$, using the definition of δ and $w(\mathrm{OPT}) \ge \max_{C \in \mathcal{C}}(\mathrm{diam}(C))$. We conclude that $w(\mathrm{OPT}') \le (1+\varepsilon) \cdot w(\mathrm{OPT})$. ◀

3.2.3 Evaluating Basic Subproblems in the Streaming Setting

After we obtain the new instance $(X', \mathcal{C}')$, we evaluate the cost for every subproblem (R, A, f, Π). Because of the modification of the instance, we know for sure the subset of colors $\mathcal{C}_{R'}$ for each cell R'. To evaluate the subproblem, recall that we start with enumerating a partition of $\mathsf{cell}(R)$ that is compatible with the subproblem, which can be tested efficiently using $\mathcal{C}_{R'}$'s. Suppose now $\{P_i := R_i \cup A_i\}_{i=1}^t$ is a partition of $\mathsf{cell}(R) \cup A$ that we enumerated (recalling that A is the set of active portals, which needs to be connected to cells in a way that is compatible to the constraint f). Then, as in the offline algorithm, we evaluate $\mathrm{MST}(P_i)$ of each part P_i, and compute the sum of them, i.e. $\sum_{i=1}^t \mathrm{MST}(P_i)$, however, we need to show how to do this in the streaming setting.

Frahling et al. [27] designed an algorithm that reports a $(1+\varepsilon)$-approximation for the value of the MST of a point set presented in a dynamic stream, using space $O(\varepsilon^{-1}\log\Delta)^{O(1)}$. Furthermore, as noted in Section 1, their algorithm maintains a linear sketch. Now, a natural idea is to apply this MST sketch, that is, create an MST sketch for each color, which only takes $k\cdot O(\varepsilon^{-1}\log\Delta)^{O(1)}$ space. Then, for each $P_i = R_i \cup A_i$, we compute the set of intersecting colors, and we create a new MST sketch $\mathcal{K}$ by first adding up the MST sketches of these colors (recalling that they are linear sketches), and then adding the active portals connected to P_i to the sketch. We wish to query the sketch $\mathcal{K}$ for the cost of $\mathrm{MST}(P_i)$.

However, this idea cannot directly work, since the algorithm by [27] only gives the MST value for *all* points represented by $\mathcal{K}$, instead of the MST value for a subset P_i. Therefore, we modify the MST sketch to answer the MST cost of a subset of points of interest.

Brief Review of the MST Sketch. We give a brief overview of the algorithm of [27] before we explain how we modify it. The first observation (already from [21]) is that the MST cost can be written as a weighted sum of the number of connected components in metric threshold graphs, which are obtained from the complete metric graph of the point set by removing edges of length larger than a threshold τ. Essentially, the idea is to count the number of MST edges of length larger than τ.

To estimate the number of components in a threshold graph, we round the points to a suitable grid and sample a small number of rounded points uniformly, using ℓ_0-samplers. An ℓ_0-sampler is a data structure that processes a dynamic stream (possibly containing duplicate items), succeeds with high probability, and conditioned on it succeeding, it returns a random item from the stream such that any item in the stream is chosen with the same probability $1/n$, where n is the ℓ_0 norm of the resulting frequency vector, i.e., the number of distinct items in the stream (see Lemma 3.7 for a more precise statement). For each sampled (rounded) point y, the algorithm in [27] runs a stochastic-stopping BFS from y and in particular, it checks if it explores the whole component of y within a random number of steps. We note that this requires an extended ℓ_0-sampler that also returns the neighboring points for each sampled point, as presented in [27] and stated in Lemma 3.7. The MST cost is estimated by a weighted sum of the number of completed BFS's, summed over all levels.

▶ **Lemma 3.7** (ℓ_0-Sampler with Neighborhood Information [27, Corollary 3]). *There is an algorithm that for $\delta > 0$, integer $\rho, \Delta \geq 1$, every set of points $S \subseteq [\Delta]^2$ presented as a dynamic geometric stream, succeeds with probability at least $1-\delta$ and, conditioned on it succeeding, returns a point $p \in S$ such that for every $s \in S$ it holds that $\Pr[p=s] = 1/|S|$. Moreover, if the algorithm succeeds, it also returns all points from $s \in S$ such that $\mathrm{dist}(p,s) \leq \rho$. The algorithm has space and both update and query times bounded by $\mathrm{poly}(\rho\cdot\varepsilon^{-1}\cdot\log\Delta\cdot\log\delta^{-1})$, and its memory contents is a linear sketch of S.*

Generalizing the MST Algorithm to Handle Subset Queries. Fix some part P_i. Recall that the P_i's always consist of at most $O(\varepsilon^{-2})$ cells (which are quad-tree squares), plus $O(\varepsilon^{-2})$ active portal points. Hence, a natural first attempt is to make the ℓ_0-samplers to sample only on these clipping squares defined by P_i. Unfortunately, this approach would not work, since the squares are not known in advance and may be very small (i.e., degenerate to a point), so sampling a point from them essentially solves the INDEX problem.

Therefore, to estimate $\mathrm{MST}(P_i)$, we still use the original ℓ_0-samplers, and we employ a careful sampling and estimation step. We sample from the whole point set maintained by the sketch $\mathcal{K}$ but we only keep the sampled points contained in P_i. We execute the stochastic BFS from these points that are kept, restricting the BFS to the points contained in P_i.

One outstanding problem of this sampling method is that if the number of points in P_i, or to be exact, the number of non-zero entries of level-i ℓ_0-samplers, is only a tiny portion of that of the full sketch, then with high probability, we do not sample any point from P_i at all. Hence, in this case, no stochastic BFS can be performed, and we inevitably answer 0 for the number of successful BFS's. This eventually leads to an additive error. We summarize the additive error and the whole idea of the above discussions in Lemma 3.8.

▶ **Lemma 3.8.** *There is an algorithm that for every $0 < \varepsilon < 1$, integer $k, \Delta \geq 1$, and every set of points $S \subseteq [\Delta]^2$ presented as a dynamic geometric stream, maintains a linear sketch of size $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$. For every query $(R, \{R_j\}_{j=1}^t, A)$ (provided after the stream ends) satisfying*

1. *R is a simple square, A is a subset of portals of R, and*
2. *$\{R_j\}_{j=1}^t \subseteq \mathsf{cell}(R)$,*

the algorithm computes from the linear sketch a real number E such that with probability at least $1 - \exp(-\log k \cdot \mathrm{poly}(\varepsilon^{-1} \log \Delta))$,

$$\mathrm{MST}(P) \leq E \leq (1+\varepsilon) \cdot \mathrm{MST}(P) + O\left(\frac{\mathrm{poly}(\varepsilon)}{k \log \Delta}\right) \cdot \mathrm{MST}(S)\,,$$

where $P = \left(\bigcup_{j=1}^t R_j\right) \cup A$. The algorithm runs in time $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$ per update and the query time is also $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$.

This lemma constitutes the main algorithm for the evaluation of the subproblem. Note that we only need to prove it for one point set S, since the sketch is linear. Indeed, when applying Lemma 3.8, we obtain the sketch for each color separately from the stream, and for every query, we first merge the sketches of colors relevant to the query and add query portals to the resulting sketch. By linearity, this is the same as if we obtain the sketch for all these colors and portals at once. Due to the space limit, the proof of Lemma 3.8 can be found in the full version.

3.3 Proof of Theorem 3.1

▶ **Theorem 3.1.** *For any integers $k, \Delta \geq 1$ and any $0 < \varepsilon < 1/2$, one can with high probability $(\alpha_2 + \varepsilon)$-approximate the SFP cost of an input $X \subseteq [\Delta]^2$ presented as a dynamic geometric stream, using space and update time of $k^3 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \cdot \log \Delta)$ and with query time bounded by $k^3 \cdot \mathrm{poly}(\log k) \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$.*

We combine the above building blocks to prove Theorem 3.1. See Algorithm 2 for a description of the complete algorithm. The space and update time follow immediately from Algorithm 2, Theorem 2.2 and Lemmas 3.4, 3.5, and 3.8.

The query time is bounded by $O(k \cdot \log \Delta) \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})} \cdot \varepsilon^{-O(\varepsilon^{-1})} \cdot k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta) \leq k^3 \cdot \mathrm{poly}(\log k) \cdot (\varepsilon^{-1} \log \Delta)^{O(\varepsilon^{-2})}$, where $O(k \cdot \log \Delta)$ is the number of simple squares (and thus, up to an $O(1)$ factor, the number of quad-tree squares for which we evaluate DP subproblems), $(\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$ is the number of subproblems associated with each square (see Section 2.1), $\varepsilon^{-O(\varepsilon^{-1})}$ is the number of MST queries evaluated for each subproblem, and each MST query takes $k^2 \cdot \mathrm{poly}(\log k \cdot \varepsilon^{-1} \log \Delta)$ time by Lemma 3.8.

[2] We need to use the same randomness for sketches $\{\mathcal{K}_C^{(3)}\}$ among all colors C so that they can be combined later.

Algorithm 2 Main streaming algorithm.

procedure SFPInitialization($\mathcal{C}$) ▷ $\mathcal{C}$ is the set of colors
  initialize a sketch $\mathcal{K}^{(1)}$ of Lemma 3.4, a set of sketches of Lemma 3.5 $\{\mathcal{K}^{(2)}_C\}_{C\in\mathcal{C}}$ for every color $C \in \mathcal{C}$ with parameter $\delta := \text{poly}(\varepsilon)(k \log \Delta)^{-1}$, and a set of (linear) sketches[2] of Lemma 3.8 $\{\mathcal{K}^{(3)}_C\}_{C\in\mathcal{C}}$ for every color $C \in \mathcal{C}$

procedure SFPUpdate(x, C, `insert/delete`) ▷ insert/delete point x of color C
  insert/delete point x in sketches $\mathcal{K}^{(1)}, \mathcal{K}^{(2)}_C, \mathcal{K}^{(3)}_C$

procedure SFPQuery ▷ the stream terminates
  use sketch $\mathcal{K}^{(1)}$ to compute a set of simple squares $\mathcal{R}$ ▷ see Section 3.2.1
  for each color $C \in \mathcal{C}$, use sketch $\mathcal{K}^{(2)}_C$ to compute a set of net points N_C, and let $D_C := \text{diam}(N_C)$ ▷ D_C is a $(1 \pm \varepsilon)$-approximation for $\text{diam}(C)$
  initialize a Boolean list $\mathcal{I}$ records whether a cell of a simple square and a color intersects ▷ This uses space at most $O(k \cdot \log \Delta \cdot \text{poly}(\varepsilon^{-1}))$
  for every $R \in \mathcal{R}$, $R' \in \mathsf{cell}(R)$ **do**
    if $\text{dist}(N_C, R') \le \rho \cdot D_C$ for some color C **then**
      let $x \in R'$ be a point such that $\text{dist}(x, N_C) \le \rho \cdot D_C$
      for every color C' with $\text{dist}(N_{C'}, R') \le \rho \cdot D_{C'}$ **do**
        add x to $\mathcal{K}^{(3)}_{C'}$ and record in $\mathcal{I}$ that R' intersects color C' ▷ see Section 3.2.2
  for each simple square R and an associated subproblem (R, A, f, Π) **do**
    for each partition of $\mathsf{cell}(R)$ **do**
      if the partition is compatible with the subproblem **then** ▷ see Section 3.2.2
        for each part R_j in the partition **do**
          let $A_j \subseteq A$ be the set of active portals that R_j connects to
          create linear sketch $\mathcal{K}'$, by adding up $\mathcal{K}^{(3)}_C$ for every C intersecting a cell in R_j ▷ the intersection information is recorded in $\mathcal{I}$
          add points in A_j to sketch $\mathcal{K}'$
          query sketch $\mathcal{K}'$ for the value of the MST of the part R_j and portals A_j (as in Lemma 3.8) ▷ see Section 3.2.3
        store the sum of the queried values of $\text{MST}(R_j, A_j)$ as the estimated cost for the subproblem
  invoke the DP (as in [11]) using the values of basic subproblem estimated as above
  return the DP value (for the root square with no active portals)

To bound the failure probability, we use a union bound over the failure probabilities of all applications and queries of the streaming algorithms as well as the error bound in Theorem 2.2. We observe that Theorem 2.2 incurs an $O(1)$ failure probability, and every other steps, except for the use of Lemma 3.8, have a failure probability of $\text{poly}(\Delta^{-1})$. Since we have $k \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})}$ basic subproblems (see Section 2), and for each basic subproblem we need to evaluate at most $\varepsilon^{-O(\varepsilon^{-1})}$ MST queries, the total failure probability of evaluating the subproblems is at most

$$k \cdot (\varepsilon^{-1} \cdot \log \Delta)^{O(\varepsilon^{-2})} \cdot \varepsilon^{-O(\varepsilon^{-1})} \cdot \exp(-\log k \cdot \text{poly}(\varepsilon^{-1} \log \Delta)) \le \text{poly}(\Delta^{-1}),$$

by the guarantee of Lemma 3.8. Therefore, we conclude that the failure probability is then at most $\frac{2}{3}$. It remains to analyze the error.

Error Analysis. For the remaining part of the analysis, we condition on no failure of the sketches used in Algorithm 2 and on that the error bound in Theorem 2.2 holds. By Lemma 3.6, for the part of evaluating the basic subproblems (lines 14-22 of Algorithm 2), the actual instance that the linear sketches work on is $(1+O(\varepsilon))$-approximate. Hence, it suffices to show the DP value is accurate to that instance.

Our estimation is never an underestimate, by Lemma 3.8 and since all partitions that we enumerated are compatible with the subproblems; see Section 3.2.2. Hence, it remains to upper bound the estimation. Consider an optimal DP solution F, which we interpret as a metric graph (see Section 2). Then we create a new solution F' from F by modifying F using the following procedure. For each simple square R, we consider $F|_R$ which is the portion of F that is totally inside of R (see Section 2). For each component $S \subset R$ in $F|_R$, let S' be the point set formed by removing all Steiner points from S, except for portals of R (note that we remove portals of subsquares of R if they appear in S). Then, for each component S, we replace the subtree in F that spans S with the MST on S'. It is immediate that after the replacement, the new solution has the same connectivity of portals and terminal points as before. We define F' as the solution after doing this replacement for all simple squares.

F' is still a feasible solution. Furthermore, for every simple square R, if F is compatible with a subproblem (R, A, f, Π), then so does F'. By the construction of F', the definition of Steiner ratio α_2, and Theorem 2.2, we know that

$$w(F') \le \alpha_2 \cdot w(F) \le (1+O(\varepsilon)) \cdot \alpha_2 \cdot \mathrm{OPT}, \tag{1}$$

where the last inequality holds as we condition on that the error bound in Theorem 2.2 holds.

Now we relate the algorithm's cost to $w(F')$. Fix a simple square R, and suppose (R, A, f, Π) is the subproblem that is compatible with $F'|_R$. Then, the components in $F'|_R$ can be described by a partition of the cells plus their connectivity to active portals. Such a subproblem, together with the partition, must be examined by Algorithm 2 (in lines 14-22), and the MST value for each part is estimated in Algorithm 2. Since the algorithm runs a DP using the estimated values, the final DP value is no worse than the DP value that is only evaluated from the subproblems that are compatible to F'. Recall that our estimation for each subproblem not only has a multiplicative error of $(1+\varepsilon)$ but also an additive error by Lemma 3.8. Therefore, by the fact that F' always uses MST to connect points in components of basic subproblems, it suffices to bound the *total* additive error for the estimation of the MST cost of the components of F'.

Fix a connected (global) component Q of F', and let $\mathcal{C}_Q \subseteq \mathcal{C}$ be the subset of colors that belongs to Q. By Lemma 3.8, for every basic subproblem (R, f, A, Π) that is compatible with F', and every component P of $Q|_R$, the additive error is at most $O\left(\frac{\mathrm{poly}(\varepsilon)}{k \log \Delta}\right) \cdot \mathrm{MST}(S)$, where S is the union of color classes that intersect P plus the active portals A. Observe that $\mathcal{C}_S \subseteq \mathcal{C}_Q$ (where $\mathcal{C}_S$ is the set of colors used in S), so S is a subset of the point set of Q (note that Q contains all portals in A as F' is a portal-respecting solution and the subproblem is compatible with F') and thus $\mathrm{MST}(S) \le \mathrm{MST}(Q)$, which implies

$$O\left(\frac{\mathrm{poly}(\varepsilon)}{k \log \Delta}\right) \cdot \mathrm{MST}(S) \le O\left(\frac{\mathrm{poly}(\varepsilon)}{k \log \Delta}\right) \cdot \mathrm{MST}(Q) \le O\left(\frac{\mathrm{poly}(\varepsilon)}{k \log \Delta}\right) \cdot w(Q).$$

Observe that for each simple square R, $Q|_R$ has at most $O(\varepsilon^{-2})$ local components, hence, summing over all local components of $Q|_R$ and all simple squares R, the total additive error is bounded by $\mathrm{poly}(\varepsilon^{-1}) \cdot O(k \log \Delta) \cdot O\left(\mathrm{poly}(\varepsilon)/(k \log \Delta)\right) \cdot w(Q) \le \varepsilon \cdot w(Q)$, where use that there are at most $O(k \log \Delta)$ simple squares by Lemma 3.3. Finally, summing over all components Q of F', we conclude that the total additive error is $\varepsilon \cdot w(F')$. Combining with Equation (1), we conclude the error guarantee. This finishes the proof of Theorem 3.1.

4 Lower Bound: $\Omega(k)$ Bits are Necessary

In this section we demonstrate that any streaming algorithm for SFP achieving any finite approximation ratio for SFP requires $\Omega(k)$ bits of space.

▶ **Theorem 4.1.** *For every $k > 0$, every randomized streaming algorithm achieving a finite approximation ratio for SFP with k color classes of size at most 2 must require $\Omega(k)$ bits of space. This holds even for insertion-only algorithms and even when points are from the one-dimensional line $\mathbb{R}$.*

Proof. The proof is a reduction from the INDEX problem on k bits, where Alice holds a binary string $x \in \{0,1\}^k$, and Bob has an index $i \in [k]$. The goal of Bob is to compute the bit x_i in the one-way communication model, where only Alice can send a message to Bob and not vice versa. It is well-known that Alice needs to send $\Omega(k)$ bits for Bob to succeed with constant probability [41] (see also [42, 39]). Our reduction is from INDEX to SFP on the (discretized) one-dimensional line $[2k]$. Consider a randomized streaming algorithm ALG for SFP that approximates the optimal cost and in particular can distinguish whether the optimal cost is 0 or 1 with constant probability. We show that it can be used to solve the INDEX problem, implying that ALG needs to use $\Omega(k)$ bits of space.

Indeed, Alice applies ALG on the following stream: For each bit x_j, she adds to the stream a point of color j at location $2j + x_j$. So far $\mathrm{OPT} = 0$. She now sends the internal state of ALG to Bob. Then, Bob continues the execution of ALG (using the same random coins) by adding one more point to the stream: Given his index $i \in [k]$, he adds a point of color i at location $2i$. After that, $\mathrm{OPT} = 0 + x_i$, which is either 0 or 1. It follows that if ALG achieves a finite approximation with constant probability, then Bob can discover x_i and solve INDEX. ◀

5 Conclusions and Future Directions

Our paper makes a progress in the understanding of geometric streaming algorithms and of applicability of Arora's framework for low-space streaming algorithms for geometric optimization problems. Still, our work leaves a number of very interesting open problems.

Our approximation ratio $\alpha_2 + \varepsilon$ matches the current approximation ratio for the Steiner tree problem in geometric streams. Hence, any improvement to our approximation ratio would require to first improve the approximation for Steiner tree, even in insertion-only streams. This naturally leads to the main open problem of obtaining a $(1+\varepsilon)$-approximation for Steiner tree in geometric streams using only $\mathrm{poly}(\varepsilon^{-1} \log \Delta)$ space.

Our naïve algorithm for the Steiner forest problem given in Theorem 1.2 is also an $(\alpha_2 + \varepsilon)$-approximation with $\mathrm{poly}(k\varepsilon^{-1} \log \Delta)$ space, but its running time is exponential in k because it queries an (approximate) MST-value oracle on all possible subsets of color classes to find the minimum. We do not know if a smaller number of queries suffices here, but it is known that in a similar setup for coverage problems any oracle-based $O(1)$-approximation requires exponentially many queries to an approximate oracle [10]. Thus, it would not be surprising if a similar lower bound holds for our problem.

Our Theorem 4.1 shows that for SFP with color classes of size at most 2 one cannot achieve any bounded approximation ratio using space that is sublinear in $n \le 2k$. This strongly suggests that SFP with pairs of terminals (i.e., $C_i = \{s_i, t_i\}$) does not admit a constant-factor approximation in the streaming setting, although our lower bound construction does not extend to this case (as it requires having some size-1 color classes). We leave it as an open problem whether a constant-factor approximation in sublinear (in $n = 2k$) space is possible

for this version. We notice however that for the case where both points of each terminal pair are inserted/deleted together, it is possible to get an $O(\log n)$-approximation using the metric embedding technique of Indyk [36].

The main focus of this paper is on the study of SFP for the Euclidean plane, but in principle, our entire analysis can be extended to the Euclidean space $\mathbb{R}^d$, for any fixed $d \geq 2$. However, this would require extending the arguments of [11, 14], namely, the structural result restated in Theorem 2.2, and these details were not written explicitly in these papers.

The techniques developed in this paper seem to be general enough to be applicable to other problems/objectives with *connectivity constraints*, where the connectivity is specified by the colors and a solution is feasible if the points of the same color are connected. One such closely related problem is the sum-of-MST objective, i.e., the problem of minimizing the sum of the costs of trees such that points of the same color are in the same tree (see also [2, 53] for related problems). We hope that the approach developed in our paper can lead to a $(1+\varepsilon)$-approximation of the geometric version of this problem, using $\mathrm{poly}(k\varepsilon^{-1}\log\Delta)$ space and time (for space only, one can use similar techniques as in Theorem 1.2). Moreover, it may be possible to apply our approach to solve the connectivity-constrained variants of other classical problems, especially those where dynamic programming has been employed successfully, like r-MST and TSP [7]. For example, the TSP variant could be to find a collection of cycles of minimum total length with points of the same color in the same cycle.

At a higher level, the connectivity constraints may be more generally interpreted as grouping constraints. For instance, in the context of clustering, our color constraints may be viewed as *must-link* constraints, where points of the same color have to be placed in the same cluster. Such constrained clustering framework is of significant interest in data analysis (see, e.g., [52]). Our framework, combined with coreset techniques [28] and Arora's quad-tree methods (see [8]), may be used to design streaming algorithms for such clustering problems.

Finally, we believe that the framework of optimization problems with connectivity and grouping constraints is interesting on its own, going beyond the streaming setup. Such problems may be studied also in the setting of standard (offline) algorithms, as well as of online algorithms, approximation algorithms, fixed-parameter tractability, and heuristics.

References

1 Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

2 Mattias Andersson, Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Balanced partition of minimum spanning trees. *International Journal of Computational Geometry and Applications*, 13(4):303–316, 2003.

3 Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David P. Woodruff. Efficient sketches for earth-mover distance, with applications. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 324–330, 2009.

4 Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 343–352, 2008.

5 Alexandr Andoni and Huy L. Nguyen. Width of points in the streaming model. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 447–452, 2012.

6 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 574–583, 2014.

7 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

8 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 106–113, 1998.

9 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

10 MohammadHossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni. Almost optimal streaming algorithms for coverage problems. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 13–23, 2017.

11 MohammadHossein Bateni and MohammadTaghi Hajiaghayi. Euclidean prize-collecting Steiner forest. *Algorithmica*, 62(3-4):906–929, 2012.

12 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM*, 58(5):21:1–21:37, 2011.

13 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 51–60, 2016.

14 Glencora Borradaile, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for Euclidean Steiner forest. *ACM Transactions of Algorithms*, 11(3):19:1–19:20, 2015.

15 Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 576–585, 2017.

16 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 712–725, 2016.

17 T.-H. Hubert Chan, Shuguang Hu, and Shaofeng H.-C. Jiang. A PTAS for the Steiner forest problem in doubling metrics. *SIAM Journal on Computing*, 47(4):1705–1734, 2018.

18 Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computation Geometry*, 35(1-2):20–35, 2006.

19 Timothy M. Chan. Dynamic streaming algorithms for ε-kernels. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG)*, pages 27:1–27:11, 2016.

20 Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002.

21 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005.

22 Kuan Cheng, Alireza Farhadi, MohammadTaghi Hajiaghayi, Zhengzhong Jin, Xin Li, Aviad Rubinstein, Saeed Seddighin, and Yu Zheng. Streaming and small space approximation algorithms for edit distance and longest common subsequence. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 54:1–54:20, 2021.

23 F. R. K. Chung and R. L. Graham. A new bound for Euclidean Steiner minimal trees. *Annals of the New York Academy of Sciences*, 440(1):328–346, 1985.

24 Artur Czumaj, Christiane Lammersen, Morteza Monemizadeh, and Christian Sohler. $(1+\varepsilon)$-approximation for facility location in data streams. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1710–1728, 2013.

25 Funda Ergün and Hossein Jowhari. On the monotonicity of a data stream. *Combinatorica*, 35(6):641–653, 2015.

26 Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2005.

27 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry and Applications*, 18(1/2):3–28, 2008.

28 Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–217, 2005.

29 Edgar N. Gilbert and Henry O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

30 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

31 Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007.

32 Martin Groß, Anupam Gupta, Amit Kumar, Jannik Matuschke, Daniel R. Schmidt, Melanie Schmidt, and José Verschae. A local-search algorithm for Steiner forest. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, pages 31:1–31:17, 2018.

33 Anupam Gupta and Amit Kumar. Greedy algorithms for Steiner forest. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 871–878, 2015.

34 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2004.

35 Wei Hu, Zhao Song, Lin F. Yang, and Peilin Zhong. Nearly optimal dynamic k-means clustering for high-dimensional data, 2019. `arXiv:1802.00459`.

36 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 373–380, 2004.

37 Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *Proceedings of the 3rd International Workshop on Statistical and Computational Theories of Vision (SCTV)*, 2003. URL: `https://people.csail.mit.edu/indyk/emd.pdf`.

38 Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

39 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of Hamming distance. *Theory of Computing*, 4(6):129–135, 2008.

40 Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 745–754, 2011.

41 Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

42 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

43 Christiane Lammersen and Christian Sohler. Facility location in dynamic geometric data streams. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA)*, pages 660–671, 2008.

44 Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. *Communications of the ACM*, 62(8):95–100, 2019.

45 Thomas L. Magnanti and Laurence A. Wolsey. Chapter 9: Optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503–615. Elsevier, 1995.

46 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

47 David Pollard. *Empirical Processes: Theory and Applications*, chapter 4: Packing and Covering in Euclidean Spaces, pages 14–20. IMS, 1990.

48 Michael E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1698–1709, 2013.

49 Guido Schäfer. Steiner Forest. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 2099–2102. Springer, New York, NY, 2016.

50 Christian Sohler. Problem 52: TSP in the streaming model. `https://sublinear.info/52`, 2012.

51 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007.

52 Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 577–584, 2001.

53 Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming, Series A*, 102(1):167–183, 2005.

Improved Reconstruction of Random Geometric Graphs

Varsha Dani
Department of Computer Science, Rochester Institute of Technology, Rochester, NY, USA

Josep Díaz
Department of Computer Science, Polytechnic University of Catalonia, Barcelona, Spain

Thomas P. Hayes
Department of Computer Science, University of New Mexico, Albuquerque, NM, USA

Cristopher Moore
Santa Fe Institute, NM, USA

Abstract

Embedding graphs in a geographical or latent space, i.e. inferring locations for vertices in Euclidean space or on a smooth manifold or submanifold, is a common task in network analysis, statistical inference, and graph visualization. We consider the classic model of random geometric graphs where n points are scattered uniformly in a square of area n, and two points have an edge between them if and only if their Euclidean distance is less than r. The reconstruction problem then consists of inferring the vertex positions, up to the symmetries of the square, given only the adjacency matrix of the resulting graph. We give an algorithm that, if $r = n^\alpha$ for $\alpha > 0$, with high probability reconstructs the vertex positions with a maximum error of $O(n^\beta)$ where $\beta = 1/2 - (4/3)\alpha$, until $\alpha \geq 3/8$ where $\beta = 0$ and the error becomes $O(\sqrt{\log n})$. This improves over earlier results, which were unable to reconstruct with error less than r. Our method estimates Euclidean distances using a hybrid of graph distances and short-range estimates based on the number of common neighbors. We extend our results to the surface of the sphere in $\mathbb{R}^3$ and to hypercubes in any constant dimension.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Reconstruction algorithm, distances in RGG, d-dimensional hypercube, 3 dimensional sphere

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.48

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2107.14323`

Funding *Josep Díaz*: Partially supported by PID-2020-112581GB-C21 (MOTION).
Thomas P. Hayes: Partially supported by NSF grant CCF-1150281.
Cristopher Moore: Partially supported by NSF grant IIS-1838251.

1 Introduction

Graph embedding is the art of assigning a position in some smooth space to each vertex, so that the graph's structure corresponds in some way to the metric structure of that space. If vertices with edges between them are geometrically close, this embedding can help us predict new or unobserved links, devise efficient routing strategies, and cluster vertices by similarity – not to mention (if the embedding is in two dimensions) give us a picture of the graph that we can look at and perhaps interpret. In social networks, this space might correspond literally to geography, or it might be a "latent space" whose coordinates measure

ideologies, affinities between individuals, or other continuous demographic variables (e.g. [15]). In some applications the underlying space is known; in others we wish to infer it, including the number of dimensions, whether it is flat or hyperbolic, and so on.

The literature on graph embedding is vast, and we apologize to the many authors who we will fail to cite. However, despite the broad utility of graph embedding in practice (see [27] for a recent experimental review) many popular heuristics lack rigorous guarantees. Here we pursue algorithms that reconstruct the position of every vertex with high accuracy, up to a symmetry of the underlying space.

Many versions of the reconstruction problem, including recognizing whether a graph has a realization as a geometric graph, are NP-complete [5, 8, 9] in the worst case. Thus we turn to distributions of random instances, and design algorithms that succeed with high probability in the instance. For many inference problems, there is a natural generative model where a ground truth structure is "planted," and the instance is then chosen from a simple distribution conditioned on its planted structure. For community detection a.k.a. the planted partition problem, for instance, we can consider graphs produced by the stochastic block model, a generative model where each vertex has a ground-truth label, and each edge (u, v) exists with a probability that depends on the labels of u and v. Reconstructing these labels from the adjacency matrix then becomes a well-defined problem in statistical inference, which may or may not be solvable depending on the parameters of the model (e.g. [1,19,20]). In the same spirit, a series of papers has asked to what extent we can reconstruct vertex positions from the adjacency matrix in random geometric graphs, where vertex positions are chosen independently from a simple distribution.

Random geometric graphs

Let n be an integer and let $r > 0$ be real. Let $V = \{v_i\}_{i=1}^n$ be a set of n points chosen uniformly at random in the square $[0, \sqrt{n}]^2$. The *random geometric graph* $G \in \mathcal{G}(n, r)$ has vertex set V and edge set $E = \{(u, v) : \|u - v\| \leq r\}$ where $\|u - v\|$ denotes the Euclidean distance. (We will often abuse notation by identifying a vertex with its position.)

This is a rescaling of the *unit disk model* where $r = 1$. We follow previous authors in varying the average degree of the graph by varying r rather than varying the density of points in the plane. Since the square has area n, the density is always 1: that is, the expected number of points in any measurable subset is equal to its area.

It is also natural to consider a Poisson model, where the points are generated by a Poisson point process with intensity 1. In that case the number of vertices fluctuates but is concentrated around n, and the local properties of the two models are asymptotically the same. The number of points in a region of area A is binomially distributed in the uniform model, and Poisson distributed with mean A in the Poisson model. In both cases, the probability that such a region of area is empty is at most e^{-A}; this is exact in the Poisson model, and is an upper bound on the probability $(1 - A/n)^n$ in the uniform model.

Random geometric graphs (RGGs) were first introduced by Gilbert in the early 1960s to model communications between radio stations [14]. Since then, RGGs have been widely used as models for wireless communication, in particular for wireless sensor networks. RGGs have also been extensively studied as mathematical objects, and much is known about their asymptotic properties [23, 26]. One well-known result is that $r_c = \sqrt{\log n/\pi}$ is a *sharp threshold* for connectivity for $G \in \mathcal{G}(n, r)$ in the square in both the uniform and Poisson models: that is, for any $\varepsilon > 0$, with high probability G is connected if $r > (1 + \varepsilon)r_c$ and disconnected if $r < (1 - \varepsilon)r_c$.

More generally, we can define RGGs on any compact Riemannian submanifold, by scattering n points uniformly according to the surface area or volume. We then define the edges as $E = \{(u, v) : \|u - v\|_g \leq r\}$ where $\|\cdot\|_g$ is the geodesic distance, i.e. the arc length of the shortest geodesic between u and v. On the sphere in particular this includes the cosine distance, since $\|u - v\|_g$ is a monotonic function of the angle between u and v.

The reconstruction problem

Given the adjacency matrix A of a random geometric graph defined on a smooth submanifold M, we want to find an embedding $\phi : V \to M$ which is as close as possible to the true positions of the vertices. As a measure of accuracy, we focus on the max distance $\max_v \|\phi(v) - v\|$ where we identify each vertex v with its true position.

However, if we are only given A, the most we can ask is for ϕ to be accurate up to M's symmetries. In the square, for instance, applying a rotation or reflection to the true positions results in exactly the same adjacency matrix. Thus we define the *distortion* $d^*(\phi)$ as the minimum of the maximum error achieved by composing ϕ with some element of the symmetry group $\mathrm{Sym}(M)$,

$$d^*(\phi) = \min_{\sigma \in \mathrm{Sym}(M)} \max_{v \in V} \|(\sigma \circ \phi)(v) - v\| . \tag{1}$$

We will sometimes refer to the distortion of a subset of the vertices or of a single vertex.

As in previous work, our strategy is to estimate the distances between pairs of vertices, and then use geometry to find points with those pairwise distances. We focus on the case where $M = [0, \sqrt{n}]^2$ and $\|\cdot\|$ is the Euclidean distance. However, many of our results apply more generally, both in higher dimensions and on curved manifolds.

Our contribution

An intuitive way to estimate the Euclidean distance $\|u - v\|$ in a random geometric graph is to relate it to the graph distance $d_G(u, v)$, i.e. the number of edges in a topologically shortest path from u to v. The upper bound $\|u - v\| \leq r d_G(u, v)$ is obvious. Moreover, if the graph is dense enough, then shortest paths are fairly straight geometrically and most of their edges have Euclidean length almost r, and this upper bound is not too far from the truth [4, 7, 13, 21].

As far as we know, the best upper and lower bounds relating Euclidean distances to graph distances in RGGs are given in [12]. In [11] these bounds were used to reconstruct with distortion $(1 + o(1))r$ when r is sufficiently large, namely if $r = n^\alpha$ for some $\alpha > 3/14$.

However, since the graph distance d_G is an integer, the bound $\|u - v\| \leq r d_G(u, v)$ cannot distinguish Euclidean distances that are between two multiples of r. Thus, as discussed after the statement of Theorem 4 below, the methods of [11] cannot avoid a distortion that grows as $\Omega(r)$. Intuitively, the opposite should hold: as r grows the graph gets denser, neighborhoods get smoother, and more precise reconstructions should be possible.

We break this $\Omega(r)$ barrier by using a hybrid distance estimate. First we note that $r d_G(u, v)$ is a rather good estimate of $\|u - v\|$ if $\|u - v\|$ is just below a multiple of r, and we improve the bounds of [12] using a greedy routing analysis. We then combine $r d_G$ with a more precise short-range estimate based on the number of neighbors that u and v have in common. In essence, we use a quantitative version of the popular heuristic that two vertices are close if they have a large Jaccard coefficient (see e.g. [24] for link prediction, and [2] for a related approach to small-world graphs).

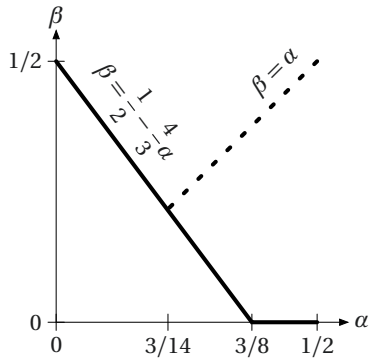


Figure 1 Our results (solid) compared to those of [11] (dotted). If $r = n^\alpha$, our reconstruction has distortion $O(n^\beta)$ where $\beta = 1/2 - (4/3)\alpha$, except for $\alpha > 3/8$ where the distortion is $O(\sqrt{\log n})$. The algorithm of [11] applies when $\alpha > 3/14$ and gives $\beta = \alpha$, i.e. distortion $\Theta(r)$. Our results apply for any constant $0 < \alpha < 1/2$ and give lower distortion than [11] when $\alpha > 3/14$.

As a result, we obtain a distortion d^* that decreases with r. Namely, if $r = n^\alpha$ for $\alpha > 0$, then $d^* = O(n^\beta)$ where $\beta = 1/2 - (4/3)\alpha$, until for $\alpha \geq 3/8$ where $d^* = O(\sqrt{\log n})$. (Note that any $\alpha > 0$ puts us well above the connectivity threshold.) Since it uses graph distances, the running time of our algorithm is essentially the same as that of All-Pairs Shortest Paths. To our knowledge, this is the smallest distortion achieved by any known polynomial-time algorithm. We compare our results with those of [11] in Figure 1.

We show that our results extend to higher dimensions and to some curved manifolds as well. With small modifications, our algorithm works in the m-dimensional hypercube for any fixed m (the distortion depends on m, but the running time does not). We also sketch a proof that it works on the surface of the sphere, using spherical rather than Euclidean geometry, solving an open problem posed in [11]. Our techniques are designed to be easy to apply on a variety of curved manifolds and submanifolds, although we leave the fullest generalizations to future work.

We use $N(u) = \{w : (u, w) \in E\}$ to denote the topological neighborhood of a vertex u, and $B(u, r)$ to denote the geometrical ball around it. Our results, as well as many of the cited results, hold *with high probability* (w.h.p.) in the random instance $G \in \mathcal{G}(n, r)$, i.e. with probability tending to 1 as $n \to \infty$. When we consider randomized algorithms, the probability is over both $\mathcal{G}(n, r)$ and the randomness of the algorithm.

Other related work

In the statistics community there are a number of consistency results for maximum-likelihood methods (e.g. [25]) but it is not clear how the accuracy of these methods scales with the size or density of the graph, or how to find the maximum-likelihood estimator efficiently. There are also results on the convergence of spectral methods, using relationships between the graph Laplacian and the Laplace-Beltrami operator on the underlying manifold (e.g. [3]). This approach yields bounded distortion for random dot-product graphs in certain regimes.

We assume that parameters of the model are known, including the underlying space and its metric structure (in particular, its curvature and the number of dimensions). Thus we avoid questions of model selection or hypothesis testing, for which some lovely techniques have been proposed (e.g. [10, 18, 22]). We also assume that the parameter r is known, since this is easy to estimate from the typical degree.

Organization of the extended abstract

In Section 2 we define the concept of a *deep* vertex. Intuitively, a vertex is deep if it is more than r from the boundary of the square, so that its ball of potential neighbors is entirely in the interior. However, since we are only given the adjacency matrix, we base our definition on the number of vertices two steps away from v in the graph, and show that these topological and geometric properties are closely related.

Section 3 shows that we can closely approximate Euclidean distances $\|u - v\|$ given the adjacency matrix whenever v is deep. We do this in two steps: we give a precise short-range estimate of $\|u - v\|$ when $d_G(u, v) \leq 2$, and a long-range estimate that uses the existence of a greedy path. By "hybridizing" these two distance estimates, switching from long to short range at a carefully chosen intermediate point, we obtain a significantly better estimate of $\|u - v\|$ than was given in [12]. We believe these distance estimation techniques may be of interest in themselves.

In Section 4, we use this new estimate of Euclidean distant to reconstruct the vertex positions up to a symmetry of the square, by starting with a few deep "landmarks" and then triangulating to the other vertices. This gives smaller distortion than the algorithm in [11], achieving the scaling shown in Figure 1.

Finally, in Section 5 we extend our method to random geometric graphs in the m-dimensional hypercube and on the surface of the sphere.

Due to space limitation, in this extended abstract, we limit ourselves to sketching the proofs and their main ideas, deferring the complete proofs to the full version of the paper, available on ArXiv: `https://arxiv.org/abs/2107.14323`

2 Deep vertices

Let G be a random geometric graph defined in the two-dimensional square $[0, \sqrt{n}]^2$. Because some of our arguments will break down for vertices near the boundary and corners of $[0, \sqrt{n}]^2$, it will be useful to have an easy way to tell these vertices apart from the rest. To this end, we introduce the notion of deep vertices.

▶ **Definition 1.** Let r be fixed. We say that a vertex $v \in V$ is *deep* if at least $11r^2$ vertices have graphical distance 2 or less from v.

Note that being deep is a topological property of the graph, rather than its embedding in the plane. We need such a definition since our reconstruction algorithm is only given access to the adjacency matrix. However, in the long version we show that with high probability all vertices that are deep in this topological sense are at least r from the boundary of the square. Moreover, with high probability there are many deep vertices.

3 Estimating Euclidean distances: Breaking the $\Omega(r)$ barrier

3.1 Estimating short-range distances

In this section we show how to estimate the Euclidean distance $\|u - v\|$ between two vertices that are topologically close, namely when $d_G(u, v) \leq 2$.

We first assume that $d_G(u, v) = 1$, i.e., that $\|u - v\| = x$ where $0 \leq x \leq r$. Then $\{N(v) \setminus N(u)\}$ consists of the points in the lune $L = B(v, r) \setminus B(u, r)$ shown in Figure 2. If v is deep, then $B(v, r)$ and therefore L lies in the interior of the square $[0, \sqrt{n}]^2$. Thus in expectation $|\{N(v) \setminus N(u)\}|$ is the area of L, which we denote $F(x)$. This suggests inverting F, estimating x as

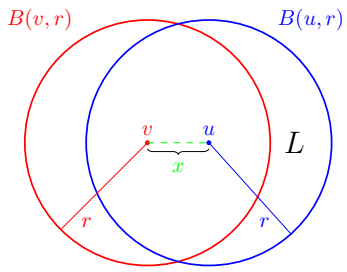


Figure 2 We can estimate the Euclidean distance $\|u-v\| = x$ of two vertices with $d_G(u,v) \le 2$ using the area of the lune $L = B(u,r) \cap B(v,r) \neq \emptyset$. Denoting this area $F(x)$, we can estimate x by appling the inverse F^{-1} to the number of points in $N(v) \setminus N(u)$.

$$\tilde{d}(u,v) = F^{-1}\left(|N(v) \setminus N(u)|\right) . \tag{2}$$

Since F is monotonic and is given explicitly as $F(x) = \pi r^2 - 2r^2 \arccos \frac{x}{2r} + \frac{x}{2}\sqrt{4r^2 - x^2}$, we can compute this inverse using binary search.

This estimate is w.h.p. an accurate estimate of $\|u-v\|$ for two reasons. First, $|\{N(v) \setminus N(u)\}|$ is concentrated around its expectation $F(x)$. In both the uniform and the Poisson models, with high probability we have

$$\Big||\{N(v) \setminus N(u)\}| - F(x)\Big| \le \sqrt{F(x)\log n}\,.$$

Second, the derivative of F is large, so the derivative of F^{-1} is small. Specifically, since $F(x)$ satisfies the differential equation $F'(x) = \sqrt{4r^2 - x^2}$, we have $F'(x) \ge r\sqrt{3} = \Omega(r)$ for $0 < x < r$. Noting also that $F(x) = \Theta(xr)$, we obtain

$$\left|x - \tilde{d}(u,v)\right| \le \frac{\sqrt{F(x)\log n}}{F'(x)} = O\left(\sqrt{\frac{x\log n}{r}}\right) . \tag{3}$$

If $d_G(u,v) = 2$ in which case $r < x < 2r$, we switch from the difference in the two neighborhoods to their intersection $N(u) \cap N(v)$, namely the points in the lens-shaped region $B(u,r) \cap B(v,r)$ in Figure 2 which has area $\pi r^2 - F(x)$. As $x \to 2r$ the area of this region tends to zero, but so does $F'(x)$. Specifically, if $x = 2r - \varepsilon$ then

$$\pi r^2 - F(x) = \Theta(r^{1/2}\varepsilon^{3/2}) \quad \text{and} \quad F'(x) = \Theta(r^{1/2}\varepsilon^{1/2})\,,$$

so (3) becomes

$$\left|x - \tilde{d}(u,v)\right| = O\left(\frac{\sqrt{r^{1/2}\,\varepsilon^{3/2}\log n}}{r^{1/2}\,\varepsilon^{1/2}}\right) = O\left(\left(\frac{\varepsilon}{r}\right)^{1/4}\sqrt{\log n}\right) . \tag{4}$$

Putting this all together gives the main theorem of the section,

▶ **Theorem 2.** *Given a $G \in \mathcal{G}(n,r)$, where $r \ge 100\sqrt{\log n}$. With probability at least $1 - 2/n^2$ we have, for all vertices $v \neq w$ such that $d_G(v,w) \le 2$ and v is deep,*

$$\left|\|v-w\| - \tilde{d}(v,w)\right| \le 100\eta(\|v-w\|)\sqrt{\log n}\,, \tag{5}$$

where $\eta : [0, 2r] \to [0, 1]$ *is defined by*

$$\eta(x) = \begin{cases} \frac{\sqrt{\log n}}{r} & \text{for } 0 \le x \le \frac{\log n}{r}, \\ \sqrt{\frac{x}{r}} & \text{for } \frac{\log n}{r} \le x \le r, \\ \left(\frac{2r-x}{r}\right)^{1/4} & \text{for } r \le x \le 2r - \frac{(\log n)^{2/3}}{r^{1/3}}, \\ \frac{(\log n)^{1/6}}{r^{1/3}} & \text{for } 2r - \frac{(\log n)^{2/3}}{r^{1/3}} \le x \le 2r. \end{cases}$$

3.2 Estimating long-range distances

Next we show a fairly tight relationship between geometric and topological distance for all pairs of vertices, including distant ones. This is a slightly sharper version of [12, Thm 1.1]. The main difference is that, where before, a short path between two given vertices is found by finding vertices close to a straight line between the endpoints, our proof instead analyses a greedy algorithm generating a path that may deviate further from the straight line.

We start with the following geometrical lemma

▶ **Lemma 3.** *Let* $B_1(v, r_1)$ *and* $B_1(u, r_2)$ *be overlapping balls in* $\mathbb{R}^2$*, and let* $d = \|u - v\|$*. Consider the lens* $L = B_1 \cap B_2$*. Let* δ *denote the width of* L*, i.e.,* $\delta = r_1 + r_2 - d$*. Then the area* A *of* L *satisfies*

$$A = \Theta\left(\delta^{3/2} \min\{r_1, r_2\}^{1/2}\right).$$

The main result of this section is the following theorem,

▶ **Theorem 4.** *Let* $G \in \mathcal{G}(n, r)$*. There exist absolute constants* C_1, C_2, C_3 *such that, for all* $n \ge 1$ *and all* $r \ge C_1\sqrt{\log n}$*, with probability at least* $1 - C_2/n^2$*, all pairs of vertices* u, v *satisfy*

$$\left\lceil \frac{\|u - v\|}{r} \right\rceil \le d_G(u, v) \le \left\lceil \frac{\|u - v\| + \kappa}{r} \right\rceil, \tag{6}$$

where

$$\kappa = \kappa(\|u - v\|) = C_3 \left(\frac{\|u - v\|}{r^{4/3}} + \frac{\log n}{r^{1/3}} \right). \tag{7}$$

The lower bound of (6) is trivial. The gist of the upper bound is to show the existence of a short path $u \rightsquigarrow v$ using a greedy routing algorithm that moves as close as possible, in Euclidean distance, to v at each step. (Note that this is only for the purpose of analysis, since our reconstruction algorithm is only given the adjacency matrix!) Start from $x_0 = u$. Then for each $i \ge 0$, let x_{i+1} be the neighbor of x_i that minimizes $\|x_i - v\|$ as shown in Fig. 3 (note that x_{i+1} is unique with probability 1). The algorithm terminates if no neighbor of x_i is closer to v than x_i is. If $x_i = v$, we have found our path and the algorithm succeeds. Otherwise, the algorithm has gotten stuck in a local minimum, and never reaches v.

Then in the full version we prove that the algorithm succeeds with probability $1 - O(n^{-3})$. Moreover, with the help of Lemma 3, we can show that each step gets about $r - O(r^{-1/3})$ closer to v. This yields the upper bound of (6), and taking a union bound over all pairs u, v completes the proof. ⌟

Let us discuss how we will use Theorems 2 and 4 to break the $\Omega(r)$ barrier in distance estimation, and thus in reconstruction. Suppose $r = n^\alpha$ where $0 < \alpha < 1/2$ is a constant. Then since $\|u - v\| = O(n^{1/2})$, we have from (7)

$$\kappa = O\left(\max\left(n^{\frac{1}{2} - \frac{4}{3}\alpha}, n^{-\frac{1}{3}\alpha} \log n\right)\right), \tag{8}$$

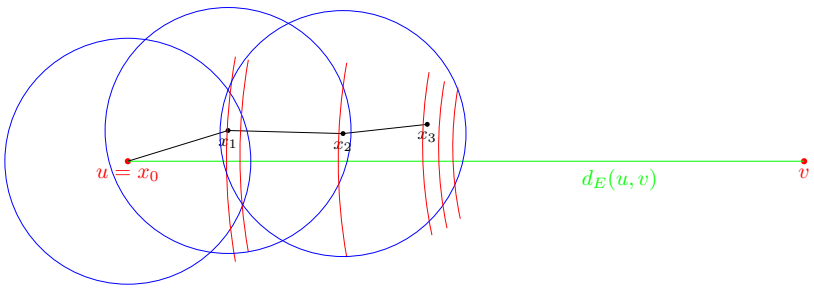


Figure 3 The greedy routing analysis of Theorem 4. At each step we go from x_i to the neighbor x_{i+1} closest to v. In the analysis, we consider the intersections of x_i's neighborhood with balls centered at v, with the radii of the latter chosen so that these intersections have area $\ln 2$, $2\ln 2$, $3\ln 2$, and so on. Each of these intersections contains a point with constant probability, so that most steps make significant progress towards v.

and since $\frac{1}{2} - \frac{4}{3}\alpha > -\frac{1}{3}\alpha$ we have

$$\kappa = O(n^\beta)\,, \quad \text{where} \quad \beta = \frac{1}{2} - \frac{4}{3}\,\alpha\,. \tag{9}$$

If $\alpha > 3/14$, then $\beta < \alpha$ and $\kappa = o(r)$. In this case the upper and lower bounds on $d_G(u,v)$ differ by at most 1, and moreover are equal for most pairs of vertices, making $d_G(u,v)$ a nearly-deterministic function of $\|u-v\|$. Using $\lceil x \rceil \le x+1$ and multiplying through by r gives the bounds

$$d_G(u,v)r - (r+\kappa) \le \|u-v\| \le d_G(u,v)r\,,$$

so that $d_G(u,v)r$ is an estimate of $\|u-v\|$ with error $r+\kappa = (1+o(1))r$. Previous work [11,12] used essentially this bound to reconstruct the graph with a distortion of $(1+\varepsilon)r$ for arbitrarily small constant ε. This gives the performance shown by the dotted line in Figure 1.

But in fact $d_G(u,v)r$ is a much more accurate estimate of $\|u-v\|$ for certain pairs of vertices. If $\|u-v\|$ is just below a multiple of r, then rounding up the left and right sides of (6) doesn't change either very much. We state this with in the following corollary,

▶ **Corollary 5.** *With $\kappa = \kappa(\|u-v\|)$ defined as in (7), suppose that for some $0 \le \delta < r$ and some integer $t \ge 0$ we have $tr - (\kappa+\delta) < \|u-v\| < tr - \kappa$. Then*

$$d_G(u,v)r - (\kappa+\delta) \le \|u-v\| \le d_G(u,v)r\,. \tag{10}$$

Thus, if $\|u-v\|$ is in one of these intervals, Theorem 4 lets us estimate $\|u-v\|$ from the adjacency matrix with error $\delta+\kappa$ instead of $r+\kappa$. Below we will combine this with the more precise estimate of short-range distances from Theorem 2 to achieve this error for all pairs u, v where v is deep, not just those for which $\|u-v\|$ is almost a multiple of r.

As a result, the error in our distance estimates and the distortion of our reconstruction is $O(r^\beta)$ where β decreases from 1 to 0 as α increases as shown by the solid line in Figure 1. Specifically, we obtain a nontrivial result for any $\alpha > 0$ and a more accurate reconstruction than in [11] in the range $\alpha > 3/14$ where their theorem applies. At $\alpha = 3/8$ where $\beta = 0$ another source of error takes over, leaving us with $O(\sqrt{\log n})$ distortion.

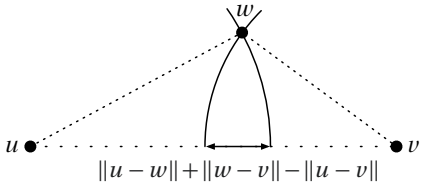


Figure 4 For any intermediate point w, the hybrid distance estimate $d_1(u,w) + d_2(w,v)$ is an upper bound on $||u - v||$ with error bounded by Lemma 8.

3.3 Hybrid distance estimates

In this subsection we combine the long-range estimates of Theorem 4 with the short-range estimates in Theorem 2, to estimate Euclidean distances with an error of $o(n)$. We start with the following definition:

▶ **Definition 6.** *Let $V \subset \mathbb{R}^2$ and $d : V^2 \to [0,\infty)$ and $\varepsilon : \mathbb{R} \to [0,\infty)$ be two functions satisfying, for all $u, v \in V$, $d(u,v) - \varepsilon(u,v) \le \|u - v\| \le d(u,v)$. Then we say d is an upper bound on Euclidean distance with error function ε.*

The basic tool for combining distance estimates is the following lemma.

▶ **Lemma 7.** *If d_1 and d_2 are upper bounds on Euclidean distance with error functions $\varepsilon_1, \varepsilon_2$ respectively, then $\min\{d_1, d_2\}$ is an upper bound on Euclidean distance with error $\min\{\varepsilon_1, \varepsilon_2\}$.*

The next lemma shows another way to combine two upper bounds on $\|u - v\|$. We choose a vertex w between u and v and use the triangle inequality, using d_1 to bound $\|u - w\|$ and d_2 to bound $\|w - v\|$. Finally, we minimize over all intermediate vertices w. This hybrid is especially useful when, as with our long-range and short-range estimates, d_1 and d_2 have different ranges of $\|u - v\|$ in which they achieve small error.

▶ **Lemma 8.** *Suppose d_1 and d_2 are upper bounds on Euclidean distance with error functions ε_1 and ε_2 respectively. Define the hybrid distance estimate by*

$$\widehat{d}(u,v) = \min_w \left(d_1(u,w) + d_2(w,v) \right). \tag{11}$$

Then $\widehat{d}$ is an upper bound on Euclidean distance with error

$$\widehat{\varepsilon}(u,v) \le \min_w \Big[\varepsilon_1(u,w) + \varepsilon_2(w,v) + \|u - w\| + \|w - v\| - \|u - v\| \Big].$$

For an intuition of the proof, see Figure 4. ⌟

The next lemma uses the fact that if a lens is sufficiently large to contain at least one point w with high probability, then this gives an upper bound on the minimum in Lemma 8.

▶ **Lemma 9.** *Let $G \in \mathcal{G}(n,r)$ and suppose that with high probability d_1 and d_2 are upper bounds on Euclidean distance with errors $\varepsilon_1(u,v) = \varepsilon_1(\|u - v\|)$ and $\varepsilon_2(u,v) = \varepsilon_2(\|u - v\|)$. Define $\widehat{d}$ as in Lemma 8. Then there is a constant C such that, with high probability, $\widehat{d}$ is also an upper bound on Euclidean distance, with error $\widehat{\varepsilon}(u,v) = \widehat{\varepsilon}(\|u - v\|)$ where*

$$\widehat{\varepsilon}(\|u - v\|) \le \min_{0 < x < \|u-v\|} \max_{0 \le \delta_1, \delta_2 \le \delta(x)} \left[\varepsilon_1(x + \delta_1) + \varepsilon_2(\|u - v\| - x - \delta_2) + \delta(x) \right], \tag{12}$$

with $\delta(x) = C(\log n)^{2/3} \left(\min\{x, \|u - v\| - x\}\right)^{-1/3}$.

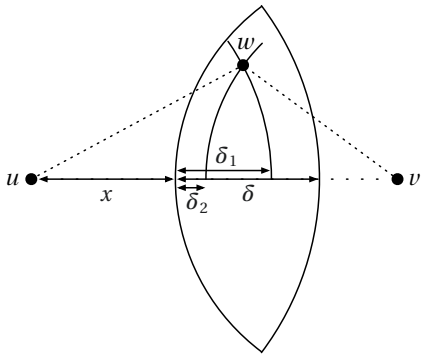


Figure 5 The lens $L(x)$ of Lemma 9. If δ is large enough, this lens is nonempty with high probability, in which case we can use any point w in it as an intermediate point for Lemma 8.

Proof sketch. Fix u, v and consider the lens $L(x) = B(u, x+\delta) \cap B(u, x+\|u-v\|-x)$ of width δ as shown in Fig. 5. By Lemma 3, the area of $L(x)$ is proportional to $C^{3/2}\log n$. Since the probability a region of area A is empty is at most e^{-A}, w.h.p. $L(x)$ contains at least one vertex w. For C sufficiently large, Lemma 8 then yields (12). ◀

Now we use the previous lemma to break the $\Omega(r)$ barrier for the error in estimating Euclidean distances in $G \in \mathcal{G}(n,r)$.

Assume v is deep. First define $d_1 = r d_G(u,v)$, i.e., the upper bound of Corollary 5. Now define d_2 using the precise short-range estimate $\tilde{d}$ from Theorem 2, with a small increment to make it an upper bound on Euclidean distance with high probability. Specifically, for a sufficiently large constant C_2, let

$$d_2(u,v) = \begin{cases} \tilde{d}(u,v) + C_2\sqrt{\log n} & \text{if } d_G(u,v) \le 2\,, \\ +\infty & \text{otherwise}\,. \end{cases} \tag{13}$$

▶ Remark 10. Given this choice of d_1 and d_2, the hybrid estimate $\widehat{d}(u,v)$ is the graph distance from u to v in a weighted graph G_v where each edge (w,v) with $d_G(w,v) \le 2$ has weight $\tilde{d}(w,v) + C_2\sqrt{\log n}$ and all other edges have weight r. Thus, for any fixed v, we can compute $\widehat{d}(u,v)$ for all u in with an application of Dijkstra's algorithm.

Now let us bound the error functions ε_1 and ε_2 of d_1 and d_2. As discussed above, for most values of $\|u-v\|$ we have $\varepsilon_1(\|u-v\|) = \Theta(r)$. However, we will choose the lens in Lemma 9 so that $\|u-w\|$ is almost a multiple of r, in which case Corollary 5 shows that $\varepsilon_1(\|u-w\|)$ is much smaller.

To bound ε_2, Theorem 2 implies that, for some absolute constant C_4, w.h.p.

$$\varepsilon_2(\|u-v\|) \le \begin{cases} C_4\sqrt{\log n} & \text{if } \|u-v\| \le 2r - C_4 r^{-1/3}\log n, \\ +\infty & \text{otherwise.} \end{cases} \tag{14}$$

Having gathered these facts, we will apply Lemma 9 to d_1 and d_2 with a judicious choice of lens $L(x)$. First note that, since $d_2(w,v) = +\infty$ if $d_G(w,v) > 2$, we can write

$$\widehat{d}(u,v) = \min_{w : d_G(w,v) \le 2} \{d_1(u,w) + d_2(w,v)\}. \tag{15}$$

▶ **Theorem 11.** *Let $r = n^\alpha$ for a constant $0 < \alpha < 1/2$. For all pairs u, v where v is deep, define $\widehat{d}(u,v)$ as in eq. 15. Then w.h.p., $\widehat{d}$ is an upper bound on the Euclidean distance $\|u - v\|$ with error*

$$\widehat{\varepsilon}(u,v) \le C' \begin{cases} n^{\frac{1}{2}-\frac{4}{3}\alpha} & \alpha < 3/8, \\ \sqrt{\log n} & 3/8 \le \alpha < 1/2, \end{cases} \tag{16}$$

for some absolute constant C'. That is, $\widehat{d}(u,v) - \widehat{\varepsilon}(u,v) \le \|u - v\| \le \widehat{d}(u,v)$.

Proof sketch. We choose x and the lens $L(x)$ in Lemma 3 such that $\|u - w\|$ is almost an integer multiple of r. We use this choice of x to upper bound (12), bounding the two terms inside the minimum separately. Using the definition of κ in Theorem 4, Corollary 5 tells us that $d_1(u,w)$ has error at most $\varepsilon_1 \le \kappa + \delta$. This implies that for all $0 \le \delta_1 \le \delta$, the first term of (12) is at most $\varepsilon_1(x + \delta_1) \le \kappa + \delta$.

To bound the second term of (12) we first prove that w.h.p. $d_G(w,v) \le 2$. We then use (14) to get $\varepsilon_2(w,v) \le C_4\sqrt{\log n}$, which implies $\varepsilon_2(\|u - v\| - x - \delta_2) \le C_4\sqrt{\log n}$. ◀

4 The Reconstruction Algorithm

In this section we use our distance estimates to reconstruct the positions of the points up to a symmetry of the square. Our global strategy is similar to [11]: we first fix a small number of "landmark" vertices v whose positions can be estimated accurately up to a symmetry of the plane. Then for each vertex u we use the estimated distances $\widehat{d}(u,v)$ to reconstruct u's position by triangulation. In [11], the landmarks are vertices close to the corners of the square. Here they will instead be a set of three deep vertices that are far from collinear, forming a triangle which is acute and sufficiently large.

▶ **Definition 12.** *We say a triple of deep vertices x, y, z is* good *if they form an acute triangle with all three side lengths at least $0.1\sqrt{n}$.*

▶ Remark 13. The bounds of Theorem 4 imply that if x, y, z are deep and have pairwise graph distances in the interval $[0.1\sqrt{n}/r, 0.14\sqrt{n}/r]$, then they are a good triple; the triangle is acute since $0.14 < 0.1\sqrt{2}$.

Once we have found a good triple, we perform triangulation using the following lemma.

▶ **Lemma 14.** *Let x, y, z, u be four points in the plane. Suppose x, y, z form an acute triangle with minimum side length at least ℓ. Then, if we know the positions of x, y, z with error at most η, and we have upper bounds $\widehat{d}(u,v)$ on the Euclidean distances $\|u - v\|$ for all $v \in \{x, y, z\}$ with error $\widehat{\varepsilon}$, and all of these distances are at most D, we can determine the position of u relative to x, y, z with error at most*

$$C_5 \frac{D(\widehat{\varepsilon} + \eta)}{\ell}, \tag{17}$$

for an absolute constant C_5.

▶ Remark 15. In our application, (x, y, z) is a good triple, so $\ell = \Omega(\sqrt{n})$. Since we also have $D \le \sqrt{2n}$ and $\eta = O(\widehat{\varepsilon})$, we can reconstruct u's position relative to x, y, z with error $O(\widehat{\varepsilon})$.

Proof. First, let us assume fixed positions for x, y, z within η of their estimated positions (which we can always do so that they form an acute triangle). By the triangle inequality, this changes the distances $\|u - v\|$ for $v \in \{x, y, z\}$ by at most $\pm\eta$. Thus u is in the intersection U of three annuli,

$$U = \bigcap_{v \in \{x,y,z\}} B(v, \widehat{d}(u,v) + \eta) \setminus B(v, \widehat{d}(u,v) - \eta - \widehat{\varepsilon})\,. \tag{18}$$

Any point u' in U gives an approximation of u's position with error at most the Euclidean diameter of U, namely $\max_{u,u' \in U} \|u - u'\|$. We will show this diameter is bounded by (17).

We use some basic vector algebra. Let $\varepsilon' = \widehat{\varepsilon} + 2\eta \le 2(\widehat{\varepsilon} + \eta)$. For any $u, u' \in U$ we have, for all $v \in \{x, y, z\}$,

$$-\varepsilon' \le \|u - v\| - \|u' - v\| \le \varepsilon'\,.$$

Since the triangle x, y, z is acute, at least one of its sides makes an angle φ with the vector $u - u'$ where $0 \le \varphi \le \pi/4$. Taking this side to be (x, y) we have, without loss of generality,

$$(y - x) \cdot (u - u') = \|y - x\|\,\|u - u'\| \cos\varphi \ge \|y - x\|\,\|u - u'\| \sqrt{\frac{1}{2}}.$$

Next, we rewrite this dot product as follows,

$$\begin{aligned} 2(y - x) \cdot (u - u') &= \|x - u\|^2 - \|x - u'\|^2 - \|y - u\|^2 + \|y - u'\|^2 \\ &= (\|x - u\| - \|x - u'\|)(\|x - u\| + \|x - u'\|) \\ &\quad - (\|y - u\| - \|y - u'\|)(\|y - u\| + \|y - u'\|) \\ &\le \varepsilon'(\|x - u\| + \|x - u'\| + \|y - u\| + \|y - u'\|)\,, \end{aligned}$$

where the first line is a classical polarization identity. Putting these together, we have

$$\|u - u'\| \le \frac{\sqrt{2}}{\|y - x\|}\,\varepsilon'\big(\|x - u\| + \|x - u'\| + \|y - u\| + \|y - u'\|\big) \le \frac{4\sqrt{2}R\varepsilon'}{\ell}\,,$$

completing the proof with $C_5 = 8\sqrt{2}$. ◀

Finally we state our main theorem.

▶ **Theorem 16.** *Let $r = n^\alpha$ for a constant $0 < \alpha < 1/2$. There is an algorithm with running time $O(n^2)$ that w.h.p. reconstructs the vertex positions of a random geometric graph $G \in \mathcal{G}(n, r)$, modulo symmetries of the square, with distortion d^* an absolute constant times times $\widehat{\varepsilon}$ as defined in* (16)*. That is, for some constant C'',*

$$d^* = C'' \begin{cases} n^{\frac{1}{2} - \frac{4}{3}\alpha} & \text{if } \alpha < 3/8\,, \\ \sqrt{\log n} & \text{if } 3/8 \le \alpha < 1/2\,. \end{cases}$$

Proof sketch. We use the fact, proved in [11], that w.h.p. the true positions of the lowest-degree vertices are within $\sqrt{\log n}$ of the corners of the square. Call these vertices a, b, c, d.

1. Find a good triple x, y, z. One way to do this is to find a vertex x near the center of the square, for instance one such that $d_G(x, t) \ge 0.65\sqrt{n}/r$ for all $t \in \{a, b, c, d\}$. Then find a y with $d_G(x, y) \in [0.1\sqrt{n}/r, 0.14\sqrt{n}/r]$, and then find a z such that $d_G(x, z), d_G(y, z) \in [0.1\sqrt{n}/r, 0.14\sqrt{n}/r]$. At each stage of this process, such a vertex exists with high probability, and all three are deep.

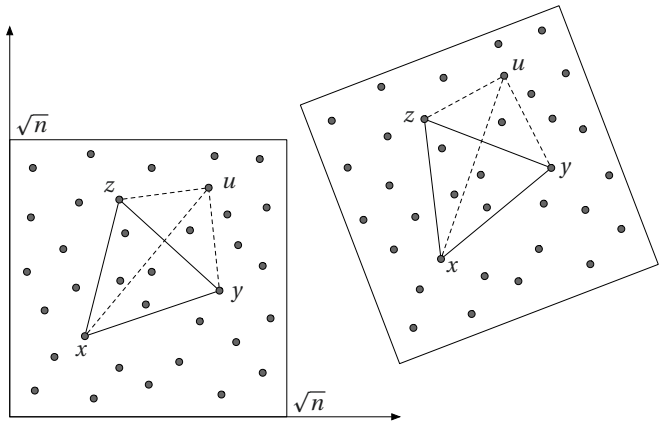


Figure 6 Our reconstruction is built around a triangle x, y, z of deep vertices. It may be translated, rotated, or reflected in $\mathbb{R}^2$ by an isometry, but it can then be shifted to the square $[0, \sqrt{n}]^2$. Then it will be a good reconstruction up to a rotation or reflection of the square.

2. Construct a triangle $x, y, z \in \mathbb{R}^2$ which is congruent to the true positions of the vertices x, y, z within error $\eta = O(\widehat{\varepsilon})$.
3. For each $v \in \{x, y, z\}$, compute the hybrid distance estimate $\widehat{d}(u, v)$ for all u as follows. First, for each w such that $d_G(w, v) \leq 2$, compute $|N(w) \cap N(v)|$ and thus the short-range distance estimates $\tilde{d}(w, v)$. Then compute $\widehat{d}(u, v)$ for all u using Dijkstra's algorithm on the weighted graph G_v described in Remark 10.
4. Use Lemma 14 to reconstruct the position of each vertex u relative to triangle x, y, z with error $O(\widehat{\varepsilon})$. This gives us a reconstruction up to an isometry of $\mathbb{R}^2$ as shown in Figure 6.
5. Finally, rotate and translate this reconstruction to the square $[0, \sqrt{n}]^2$. We choose a mapping of a, b, c, d to the corners of the square arbitrarily, using distance estimates to deduce which pairs are diagonally opposite, and then translate and rotate them as close as possible to $\{0, \sqrt{n}\}^2$. Since our definition of distortion allows rotations and reflections of the square, this gives a reconstruction with distortion $d^* = O(\widehat{\varepsilon} + \sqrt{\log n}) = O(\widehat{\varepsilon})$.

Step 1 can be done by breadth-first search, first from a, b, c, d and then from x and y, and thus takes $O(n)$ time. Steps 2, 3, 4, and 5 require $O(n)$ calculations of finite precision using standard functions, for which $O(\log n)$ bits of accuracy suffices. Thus the running time is dominated by the three uses of Dijkstra's algorithm, one for each $v \in \{x, y, z\}$, giving a running time of $O(n^2)$. ◀

▶ Remark 17. Since the typical degree in the graph is $\pi r^2 = O(n^{2\alpha})$ where $\alpha < 1/2$, and since Dijkstra's algorithm in a graph with n vertices and m edges runs in time $O(m + n \log n)$, the running time is w.h.p. $O(n^{2\alpha} + 1) = o(n^2)$.

▶ Remark 18. Once we reconstruct the positions of all vertices, we can get a good estimate of $||u - v||$ by direct computation from their approximate coordinates for all pairs u, v, including those where neither u nor v is deep.

5 Extensions to Other Domains

Our results can be generalized from the square to a number of alternative domains for random geometric graphs, including higher-dimensional Euclidean spaces and some curved manifolds. Here we sketch extensions of our algorithm to the m-dimensional hypercube and to the sphere, solving an open problem posed in [11].

5.1 Reconstruction in higher-dimensional Euclidean space

The simplest generalization is where the underlying domain is $[0, n^{1/m}]^m \subset \mathbb{R}^m$, i.e., an m-dimensional hypercube with volume n. We assume that m is a constant that does not vary with n. As before, n points are scattered uniformly in the hypercube, pairs u, v are adjacent if they are within Euclidean distance r, and our goal is to reconstruct the points' positions based on the adjacency matrix of the graph.

The following lemma generalizes Lemma 3 to $\mathbb{R}^m$, giving the m-dimensional volume of a lens-shaped intersection of two balls.

▶ **Lemma 19.** *Let $B_1(x, r_1)$ and $B_2(y, r_2)$ be two overlapping balls in $\mathbb{R}^m$ with $r_1 \le r_2$. Consider the lens $L = B_1 \cap B_2$. Let δ be the width of L, i.e., $\delta = \min\{r_1 + r_2 - d, 2r_1\}$ where $d = \|x - y\|$. Then the volume V of L satisfies $V = \Theta\left(\delta^{\frac{m+1}{2}} r_1^{\frac{m-1}{2}}\right)$, where the constant in Θ depends only on m.*

Given this relation between the width and volume of the lens, analogously to Section 3, we can compute both short- and long-range estimates of the distance, and combine them into a hybrid estimate. The error in the hybrid estimate is given by the following theorem.

▶ **Theorem 20.** *Let $r = n^\alpha$ for a constant $0 < \alpha < 1/m$. For all pairs u, v where v is deep, define $\widehat{d}(u, v)$ be the hybrid estimate of the distance. Then with high probability, $\widehat{d}$ is an upper bound on the Euclidean distance $\|u - v\|$ with error*

$$\widehat{\varepsilon}(u, v) \le C_m \begin{cases} n^{\frac{1}{m} - \frac{2m}{m+1}\alpha} & \alpha < \frac{m+1}{2m^2}, \\ \sqrt{\log n} & \frac{m+1}{2m^2} \le \alpha < \frac{1}{m}, \end{cases} \tag{19}$$

for some dimension-dependent constant C_m.

We omit the details of the proof since it closely follows the steps in Section 3.

In order to use the hybrid estimates for reconstruction, we need to find an appropriate number of deep landmarks. Using linear algebra, it suffices to have $m + 1$ landmarks that form a non-degenerate simplex. As in Theorem 16, we find an approximately equilateral m-simplex, namely a set of $m + 1$ points whose graph distances are all roughly the same constant times the diameter $n^{1/m}/r$. We again triangulate the positions of the other points based on their distance estimates, giving a reconstruction up to an isometry of $\mathbb{R}^m$. It is then easy to compute an isometry that shifts the reconstructed hypercube to $[0, n^{1/m}]^m$ by identifying low-degree vertices with the 2^m corners.

Putting this all together gives the following reconstruction theorem for random geometric graphs in $[0, n^{1/m}]^m$. We omit further details of the proof.

▶ **Theorem 21.** *Let $r = n^\alpha$ for a constant $0 < \alpha < 1/m$. There is an algorithm with running time $O(n^2)$ that w.h.p. reconstructs the vertex positions of a random geometric graph, modulo symmetries of the hypercube, with distortion*

$$d^* \le C_m \begin{cases} n^{\frac{1}{m} - \frac{2m}{m+1}\alpha} & \text{for } \alpha < \frac{m+1}{2m^2}, \\ \sqrt{\log n} & \text{for } \frac{m+1}{2m^2} \le \alpha < \frac{1}{m}, \end{cases}$$

for some dimension-dependent constant C_m.

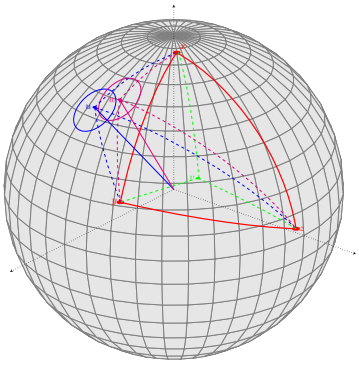


Figure 7 Using three landmarks x, y, z on the sphere to triangulate to other points in Theorem 22.

5.2 Reconstruction on the sphere

Finally, we argue that our algorithm also works on some curved manifolds and submanifolds where the geometric graph is defined in terms of geodesic distance. In particular we claim this for the m-dimensional spherical (hyper)surface S_m of a ball in $\mathbb{R}^{m+1}$. Here we sketch the proof for the two-dimensional surface of a sphere in $\mathbb{R}^3$. Note that the distortion is now defined by minimizing over the sphere's continuous symmetry group, i.e., over all rotations and reflections of the sphere.

In previous work, the authors of [10] gave a procedure to distinguish random geometric graphs on S^m from Erdős-Rényi random graphs. In addition, [3] gave a spectral method for reconstructing random graphs generated by a sparsified graphon model on the sphere, but since this model connects distant pairs of vertices with nonzero probability, it does not include the geodesic disk model we study here.

To define random geometric graphs on the sphere we scale the sphere so that its surface area is n, setting its radius to $R = \sqrt{n/(4\pi)}$. We scatter n points uniformly at random on it, or generate them with a Poisson point process with intensity 1, so that the expected number of points in a region is equal to its surface area. We define the graph as $(u, v) \in E$ if and only if $\|u - v\|_g \leq r$ where $\|u - v\|_g$ is the geodesic distance, i.e., the length of the shorter arc of a great circle that connects u and v. If we associate each point u with a unit vector $\vec{u} \in \mathbb{R}^3$ that points toward it from the center of the sphere, $\|u - v\|_g$ is R times the angle between $\vec{u}$ and $\vec{v}$.

▶ **Theorem 22.** *Let $r = n^\alpha$ for a constant $0 < \alpha < 1/2$. There is an algorithm with running time $O(n^2)$ that with high probability reconstructs the vertex positions of a random geometric graph, modulo a rotation or reflection of the sphere, with distortion an absolute constant times $n^{\frac{1}{2}-\frac{4}{3}\alpha}$ if $\alpha < 3/8$ and $\sqrt{\log n}$ if $\alpha \geq 3/8$.*

The algorithm is similar to that described in Theorem 16. The main difference is that our initial landmarks consist of three points x, y, z which approximately form a right spherical triangle, i.e., such that the vectors $\vec{x}, \vec{y}, \vec{z}$ have angles of about $\pi/2$ between them: see Fig 7.

6 Conclusion and Future Work

We have shown how a combination of geometric ideas can be used to reconstruct random geometric graphs with lower distortion than in previous work [11], achieving a distortion of $o(r)$ whenever $r = n^\alpha$ for $\alpha > 3/14$. Here we pose several questions for further work.

First, let us call a reconstruction ϕ *consistent* if its distances are consistent with the graph: that is, if $(u, v) \in E$ if and only if $\|\phi(u) - \phi(v)\| \leq r$. Even if ϕ has small distortion d^*, it might not be consistent: some edges $(u, v) \in E$ might have $\|\phi(u) - \phi(v)\|$ between r and $r + 2d^*$, and similarly some non-neighboring pairs might have $\|\phi(u) - \phi(v)\|$ between $r - 2d^*$ and r. To the best of our knowledge, even finding a single consistent embedding for random geometric graphs is an open question. It might be possible to refine our embedding to make it consistent, by using "forces" to move neighbors slightly closer together, and push non-neighbors farther away.

Second, a natural question is whether we can prove a significant lower bound on the distortion. An information-theoretic approach to this question would be to show that even the Bayesian algorithm, which chooses from the uniform measure on all consistent embeddings, has a typical distortion. We have been unable to prove this. However, here we sketch an argument that there *exist* consistent embeddings with a certain distortion by applying a continuous function f to the square $[0, \sqrt{n}]^2$ that "warps" the true embedding. If f's derivatives are at most δ in absolute value, then for each v, points close to the edge of v's neighborhood may move $O(\delta r)$ closer or farther away. However, a typical v has some $\varepsilon = O(1/r)$ for which there are no points whose distance is between $r - \varepsilon$ and $r + \varepsilon$, since the area of the corresponding annulus is $O(1)$. This suggests that if $\delta = O(\varepsilon/r) = O(1/r^2)$, the warped embedding is still consistent (except for a few vertices where we need to be more careful). On other other hand, even if f does not change the distance between nearby vertices very much, it can still move some vertices $\delta\sqrt{n}$ from their true positions, giving a distortion $d^* = \Omega(\sqrt{n}/r^2)$. If $r = n^\alpha$ this gives $\Omega(n^{1/2-2\alpha})$.

Even if this lower bound can be made rigorous, and even if it applies to typical consistent embeddings rather than just a few, there is a large gap between it and our upper bounds. Thus it is tempting to think that our algorithm can be improved, reducing the distortion still further. One approach would be to try to extend the geometry of overlapping disks in Theorem 2 to larger graph distances. Another would be to combine them with the spectral ideas of e.g. [3].

Finally, we would like to see how far these techniques can be extended to curved manifolds and submanifolds with boundary. In Theorem 22 we took advantage of the fact that the 2-sphere has a convenient embedding in $\mathbb{R}^3$. A more general approach, which we claim applies to any compact Riemannian submanifold with bounded curvature, would be to work entirely within the manifold itself, building a sufficiently dense mesh of landmarks and then triangulating within mesh cells. In particular, in the popular model of hyperbolic embeddings (e.g. [6, 16, 17]) where the submanifold is a ball of radius ℓ in a negatively curved space with radius of curvature R, we believe similar algorithms will work as long as $\ell/R = O(1)$. We leave this for future work.

References

1 Emmanuel Abbe. Community detection and stochastic block models: Recent developments. *Journal of Machine Learning Research*, 18(177):1–86, 2018. URL: `http://jmlr.org/papers/v18/16-480.html`.

2 Ittai Abraham, Shiri Chechik, David Kempe, and Aleksandrs Slivkins. Low-distortion inference of latent similarities from a multiplex social network. *SIAM J. Comput.*, 44(3):617–668, 2015.

3 Ernesto Araya Valdivia and De Castro Yohann. Latent distance estimation for random geometric graphs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/c4414e538a5475ec0244673b7f2f7dbb-Paper.pdf`.

4 Ery Arias-Castro, Antoine Channarond, Bruno Pelletier, and Nicolas Verzelen. On the estimation of latent distances using graph distances. *Electronic Journal of Statistics*, 15(1):722–747, 2021.

5 J. Aspnes, D.K. Goldenberg, and Y.R. Yang. On the computational complexity of sensor network localization. In *Proc. ALGOSENSORS-04*, pages 32–44, 2004.

6 Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient embedding of scale-free graphs in the hyperbolic plane. *IEEE/ACM Transactions on Networking*, 26(2):920–933, 2018.

7 M. Bradonjic, T. Elsasser, T. Friedrich, T. Sauerwald, and A. Stauffer. Efficient broadcast on random geometric graphs. In *21st. ACM-SIAM SODA*, pages 1412–1421, 2010.

8 H. Breu and D.G. Kirkpatrick. Unit disk graph recognition in np-hard. *Compt. Geometry Theory Appl.*, 9(1-2):3–24, 1998.

9 J. Bruck, J. Gao, and J. Jiang. On the computational complexity of sensor network localization. In *ACM MOBIHOC-05*, pages 181–192, 2005.

10 J. Bubeck, J. Ding, R. Eldan, and M. Racz. Testing for high-dimensional geometry in random graphs. *Random Structures and Algorithms*, 49(3):503–532, 2016.

11 Josep Diaz, Colin McDiarmid, and Dieter Mitsche. Learning random points from geometric graphs or orderings. *Random Structures & Algorithms*, 2019.

12 Josep Diaz, Dieter Mitsche, Guillem Perarnau, and Xavier Pérez-Giménez. On the relation between graph distance and Euclidean distance in random geometric graphs. *Advances in Applied Probability*, 48(3):848–864, 2016.

13 R. Ellis, J.L. Martin, and C. Yan. Random geometric graph diameter in the unit ball. *Algorithmica*, 47(4):421–438, 2007.

14 Edward. E. Gilbert. Random plane networks. *J. Soc. Industrial Applied Mathematics*, 9(5):533–543, 1961.

15 Peter D. Hoff, Adrian E. Raftery, and Mark S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97:1090+, December 2002.

16 Maksim Kitsak, Ivan Voitalov, and Dmitri Krioukov. Link prediction with hyperbolic geometry. *Phys. Rev. Research*, 2:043113, 2020.

17 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, September 2010. doi:10.1103/PhysRevE.82.036106.

18 Shane Lubold, Arun G. Chandrasekhar, and Tyler H. McCormick. Identifying the latent space geometry of network models through analysis of curvature, 2021. arXiv:2012.10559.

19 Cristopher Moore. The computer science and physics of community detection: Landscapes, phase transitions, and hardness. *Bull. EATCS*, 121, 2017. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/480.

20 Elchanan Mossel, Joe Neeman, and Allan Sly. Reconstruction and estimation in the planted partition model. *Probability Theory and Related Fields*, 162(3-4):431–461, 2015.

21 M. Muthukrishnan and G. Pandurangan. The bin-covering technique for thresholding random geometric graph properties,. In *16th. ACM-SIAM SODA*, pages 989–998, 2005.

22 S. Parthasarathy, D. Sivakoff, M. Tian, and Y. Wang. A quest to unravel the metric structure behind perturbed networks. In *Proc of the 33rd SoCG*, pages 53:1–53:16, 2017.

23 Mathew Penrose. *Random Geometric Graphs*. Oxford University Press, 2003.

24 Purnamrita Sarkar, Deepayan Chakrabarti, and Andrew W Moore. Theoretical justification of popular link prediction heuristics. In *Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 2722–2727, 2011.

25 Cosma Rohilla Shalizi and Dena Asta. Consistency of maximum likelihood for continuous-space network models, 2019. arXiv:1711.02123.

26 M. Walters. Random geometric graphs. *Surveys in Combinatorics*, pages 365–402, 2011.

27 Yi-Jiao Zhang, Kai-Cheng Yang, and F. Radicchi. Systematic comparison of graph embedding methods in practical tasks, 2021. arXiv:2106.10198.

Improved Approximation Algorithms for Dyck Edit Distance and RNA Folding

Debarati Das
Pennsylvania State University, University Park, PA, USA

Tomasz Kociumaka
Max Planck Institute for Informatics, Saarbrücken, Germany

Barna Saha
University of California, San Diego, CA, USA

Abstract

The Dyck language, which consists of well-balanced sequences of parentheses, is one of the most fundamental context-free languages. The Dyck edit distance quantifies the number of edits (character insertions, deletions, and substitutions) required to make a given length-n parenthesis sequence well-balanced. RNA Folding involves a similar problem, where a closing parenthesis can match an opening parenthesis of the same type irrespective of their ordering. For example, in RNA Folding, both `()` and `)(` are valid matches, whereas the Dyck language only allows `()` as a match. Both of these problems have been studied extensively in the literature. Using fast matrix multiplication, it is possible to compute their exact solutions in time $O(n^{2.687})$ (Chi, Duan, Xie, Zhang, STOC'22), and a $(1+\epsilon)$-multiplicative approximation is known with a running time of $\Omega(n^{2.372})$.

The impracticality of fast matrix multiplication often makes combinatorial algorithms much more desirable. Unfortunately, it is known that the problems of (exactly) computing the Dyck edit distance and the folding distance are at least as hard as Boolean matrix multiplication. Thereby, they are unlikely to admit truly subcubic-time combinatorial algorithms. In terms of fast approximation algorithms that are combinatorial in nature, the state of the art for Dyck edit distance is an $O(\log n)$-factor approximation algorithm that runs in near-linear time (Saha, FOCS'14), whereas for RNA Folding only an ϵn-additive approximation in $\tilde{O}(\frac{n^2}{\epsilon})$ time (Saha, FOCS'17) is known.

In this paper, we make substantial improvements to the state of the art for Dyck edit distance (with any number of parenthesis types). We design a constant-factor approximation algorithm that runs in $\tilde{O}(n^{1.971})$ time (the first constant-factor approximation in subquadratic time). Moreover, we develop a $(1+\epsilon)$-factor approximation algorithm running in $\tilde{O}(\frac{n^2}{\epsilon})$ time, which improves upon the earlier additive approximation. Finally, we design a $(3+\epsilon)$-approximation that takes $\tilde{O}(\frac{nd}{\epsilon})$ time, where $d \geq 1$ is an upper bound on the sought distance.

As for RNA folding, for any $s \geq 1$, we design a factor-s approximation algorithm that runs in $O(n + (\frac{n}{s})^3)$ time. To the best of our knowledge, this is the first nontrivial approximation algorithm for RNA Folding that can go below the n^2 barrier. All our algorithms are combinatorial in nature.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Dyck Edit Distance, RNA Folding, String Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.49

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2112.05866` [18]

Funding *Tomasz Kociumaka*: Work done while at University of California, Berkeley, supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.
Barna Saha: Partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 49; pp. 49:1–49:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The Dyck language is a well-known context-free language consisting of well-balanced sequences of parentheses. Ranging from programming syntaxes to arithmetic and algebraic expressions, environments in LaTeX, and tags in HTML/XML documents – we observe instances of the Dyck language everywhere. For a comprehensive discussion on the Dyck language and context-free grammars; see [23, 26]. Given a sequence x of n parentheses (which may be unbalanced), the *Dyck edit distance problem* asks for the minimum number of edits (character insertions, deletions, and substitutions) needed to make x well-balanced. Interestingly, string edit distance, which is one of the fundamental string similarity measures, can be interpreted as a special case of Dyck edit distance.[1]

A simple dynamic programming computes Dyck edit distance in $O(n^3)$ time. In 2016, after nearly four decades, Bringmann, Grandoni, Saha, and Vassilevska Williams [13] gave the first truly subcubic-time exact algorithm for a more general problem of language edit distance [2]. Very recently, Chi, Duan, Xie, and Zhang [17] provided a faster implementation of the same algorithm. The algorithm uses not-so-practical fast Boolean matrix multiplication, arguably so because computing Dyck edit distance is at least as hard as Boolean matrix multiplication [1], and hence combinatorial truly subcubic-time algorithms are unlikely to exist.

A problem closely related to Dyck edit distance is RNA Folding [31]. Both in RNA Folding and Dyck edit distance, parentheses must match in an uncrossing way. However, in an RNA folding instance, a closing parenthesis can match an opening parenthesis of the same type irrespective of the order of their occurrences. For example, under the RNA Folding distance, both `()` and `)(` are valid matches, whereas the Dyck language only allows `()` as a match. In terms of exact computation, they exhibit the same time complexity [1, 13].[2]

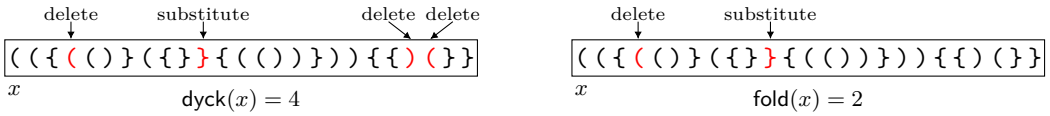


Figure 1 Example of Dyck and folding edit distance.

Can we design fast approximation algorithms for Dyck edit distance and RNA Folding? The first progress on this question for Dyck edit distance was made by Saha [34], who proposed a polylogarithmic-factor approximation algorithm that runs in near-linear time. It is also possible to provide an ϵn-additive approximation for any $\epsilon > 0$ in $\tilde{O}(\frac{n^2}{\epsilon})$ time [36]. However, unless the distance is $O(n)$ and we allow quadratic time, the above algorithm does not provide a constant-factor approximation to Dyck edit distance. This latter result on additive approximation applies to RNA Folding as well. Backurs and Onak [7] showed an exact algorithm for Dyck edit distance that runs in $O(n + d^{16})$ time, which was recently improved by Fried, Golan, Kociumaka, Kopelowitz, Porat, and Starikovskaya [21] to run in $O(n + d^5)$ time (and $\tilde{O}(n + d^{4.783})$ using fast matrix multiplication). Therefore, prior

[1] Given two strings s and t, form a sequence of parentheses by concatenating s, interpreted as a sequence of opening parentheses, and the reverse complement of t, obtained by reversing t and replacing each symbol with the corresponding closing parenthesis, not present in the original alphabet.

[2] In these problems, we are aiming to minimize the number of non-matched parentheses as opposed to maximizing the matched parentheses.

to this work, (i) there was no nontrivial multiplicative approximation for RNA Folding in subquadratic time, and (ii) there was no subquadratic-time constant-factor approximation for Dyck edit distance that would work for the entire distance regime.

Let us contrast this state of affairs with the progress on string edit distance approximation. As mentioned earlier, string edit distance is a special case of Dyck edit distance. Early work [6, 8, 9, 28] on approximating string edit distance resulted in the first near-linear-time polylogarithmic-factor approximation in 2010 by Andoni, Krauthgamer, and Onak [4]. It took another eight years to obtain the first constant-factor approximation of edit distance in subquadratic time [15] (see [11] for a quantum analog). Finally, Andoni and Nosatzki improved the running time to near-linear while maintaining a constant approximation ratio [5]. Using the best result in string edit distance approximation [5], it is possible to improve the approximation factor of [34] to $O(\log n)$. However, designing a constant-factor approximation for Dyck edit distance in subquadratic time remains wide open. RNA Folding, even though conceptually very similar to Dyck edit distance, is incompatible with the algorithm of [34].

Saha's work on Dyck edit distance approximation [34] developed a random walk technique which has later been used for edit distance embedding and document exchange [10, 16]. This random walk allows decomposing a parenthesis sequence into many instances of string edit distance problem. However, this decomposition loses a logarithmic factor in the approximation, raising the question of whether there exists an efficiently computable decomposition with a significantly smaller loss.

Contributions for Dyck Edit Distance.

- **Constant-factor approximation in subquadratic time.** *The main contribution of this paper is the first constant-factor approximation algorithm for Dyck edit distance that runs in truly subquadratic time*, namely $\tilde{O}(n^{1.971})$. (In the interest of simplicity, we did not optimize the exponent in the running time.) We employ and significantly extend the tools previously developed in connection with string edit distance, such as the windowing strategy, window-to-window computation, sparse and dense window decomposition, etc. [11, 15, 22]. These methods are tied to problems involving two or more strings (unlike the Dyck edit distance, which is a single-sequence problem). Given the universality of Dyck edit distance, the tools we developed may lead to further advancements for more generic problems like the language edit distance problem, etc. [13, 35, 36].
  Our main algorithm handles the cases of large and small Dyck edit distance separately.
- **Small Dyck edit distance.** When the Dyck edit distance d is small, we give a $(3+\epsilon)$-approximation algorithm that runs in $\tilde{O}(\frac{nd}{\epsilon})$ time. We can contrast this result with the time complexity of computing the Dyck edit distance exactly, which is $O(n+d^5)$ (combinatorially) and $\tilde{O}(n+d^{4.783})$ (using fast matrix multiplication), obtained in [21]. Nevertheless, even in a hypothetical bast-case scenario that a combinatorial $O(n+d^3)$-time algorithm exists, an $\tilde{O}(nd)$-time algorithm is still faster for all $d \gg \sqrt{n}$.
- **Quadratic-time PTAS.** We also give a $(1+\epsilon)$-approximation algorithm for Dyck edit distance that runs in $\tilde{O}(\frac{n^2}{\epsilon})$ time. This improves upon the previous result of [36] that gets such a result only when $d = \Theta(n)$. The prior $(1+\epsilon)$-approximation algorithm uses fast Boolean matrix multiplication and has super-quadratic running time [35].

Contribution for RNA Folding. For RNA Folding, we are aiming to minimize the number of non-matched characters; we henceforth call this value the *folding distance*. For any $s > 1$, we give a factor-s approximation of the folding distance in time $O(n + (\frac{n}{s})^3)$. This is the first result to our knowledge that goes below the quadratic running time (for $s = \omega(n^{1/3})$). We

remark here that the triangle inequality we proved for Dyck distance (Lemma 2.3) as well as the machinery developed in Section 5 apply to the RNA folding problem equally well. This yields a constant-factor approximation for RNA folding in $\tilde{O}(n^{1.971})$ time when the distance is larger than $n^{0.971}$.

Discussion and Open Problems. The resemblance between Dyck and string edit distance has already been studied in the literature. As mentioned earlier, the decomposition obtained by the random walk technique ensures only an $O(\log n)$ approximation [34]. In this work, instead of reducing the Dyck edit distance to string edit distance, we try to find a direct decomposition of the sequence x into different substrings, where for each substring there is a peer such that they are matched by some optimal alignment (with some error leading to a constant-factor approximation). However, unlike the string counterpart, Dyck edit distance does not have the structural property that if an optimal alignment matches the characters of a substring s_1 with the characters of a substring s_2, then the lengths of s_1 and s_2 are roughly the same (see Figure 2). Thus, in our decomposition, the substrings can have varied lengths. In fact, it turns out that if the Dyck edit distance is truly sublinear (i.e., $n^{1-\epsilon}$), then we need to consider roughly n^{ϵ} different lengths to ensure a constant-factor approximation. We remark that this is one of the barriers in further pushing down the running time from subquadratic to $O(n^{1.6+o(1)})$ (as in [22]) or near-linear. We also note that if an analog of our $\tilde{O}(nd)$-time algorithm can be provided for RNA Folding, then we would also get a constant-factor subquadratic algorithm for RNA folding for all distance regimes.

The Dyck recognition problem has been studied extensively in different models, including the streaming [14, 27, 30] and property testing [3, 19, 32] frameworks. However, neither Dyck edit distance nor RNA Folding admits sublinear-time approximation algorithms. Our algorithm for RNA Folding (which also applies to Dyck edit distance after straightforward adaptations) runs in $O(n + (\frac{n}{s})^3)$ time and requires a linear-time preprocessing step that eliminates pairs of matching adjacent characters, which leaves strongly structured instances. This preprocessing step is currently the main barrier to going in the sublinear-time setting.

1.1 Technical Overview

As input to the Dyck edit distance problem, we are given a string x of length n over an alphabet Σ that consists of two disjoint sets T and $\overline{T}$ of opening and closing parentheses respectively. The task is to compute the Dyck edit distance $\mathsf{dyck}(x)$, defined as the minimum number of parentheses insertions, deletions, and substitutions required to make x well-parenthesized.

Quadratic-time PTAS. The standard $O(n^3)$-time algorithm for Dyck edit distance is a dynamic-programming procedure that computes the distance of each substring of the input string. The bottleneck of this approach is that, to compute the distance of each substring $x(i\,..\,j]$, starting at index $i+1$ and ending at index j, one needs to iterate over decompositions of $x(i\,..\,j]$ into a prefix $x(i\,..\,k]$ and a suffix $x(k\,..\,j]$ for every possible intermediate index $k \in (i\,..\,j)$ (this corresponds to the fact that the concatenation of two well-parenthesized expression is a well-parenthesized expression).[3] We call the index k a *pivot* corresponding to a decomposition. The $\tilde{O}(\frac{n^2}{\epsilon})$-time ϵn additive approximation of [36] reduces the number of considered pivots to $\tilde{O}(\frac{1}{\epsilon})$; thus, $\tilde{O}(\frac{n}{\epsilon d})$ (where $d = \mathsf{dyck}(x)$) different pivots would be

[3] For $i, j \in \mathbb{Z}$, we denote $[i\,..\,j] = \{k \in \mathbb{Z} : i \le k \le j\}$, $[i\,..\,j) = \{k \in \mathbb{Z} : i \le k < j\}$, $(i\,..\,j] = \{k \in \mathbb{Z} : i < k \le j\}$, and $(i\,..\,j) = \{k \in \mathbb{Z} : i < k < j\}$.

necessary for a $(1+\epsilon)$-factor approximation (which is same as an ϵd-additive approximation). On the other hand, a simple $O(n^2 d)$-time algorithm recently developed in [21] is based on a combinatorial observation that $O(d)$ pivots are sufficient after the $O(n)$-time preprocessing from [7, 34]. We start with a brief overview of this algorithm. For any index $i \in [0\,..\,n]$, we define the height of i to be $h(i) = |\{j \in [1\,..\,i] : x[j] \in T\}| - |\{j \in [1\,..\,i] : x[j] \in \overline{T}\}|$, i.e., the difference between the number of opening and closing parentheses in prefix $x[1\,..\,i]$. An index v is called a valley if $h(v-1) > h(v) < h(v+1)$, i.e., $x[v]$ is a closing parenthesis whereas $x[v+1]$ is an opening parenthesis. Backurs and Onak [7] showed a linear-time preprocessing of x that generates another string x' such that $\mathsf{dyck}(x) = \mathsf{dyck}(x')$ and x' has at most $2d$ valleys. Fried, Golan, Kociumaka, Kopelowitz, Porat, and Starikovskaya [21] proved that, without loss of generality, it is enough to consider pivots that are at distance 0 or 1 from a valley (we henceforth denote the set of such pivots by K) plus $O(1)$ pivots next to the boundary of the considered range $(i\,..\,j)$; this observation yields a $O(n^2 d)$-time algorithm.

In Section 3, we provide an algorithm that further restricts the set of pivots K considered for each range $(i\,..\,j)$ and provides a $(1+\epsilon)$-approximation of $\mathsf{dyck}(x)$ in $\tilde{O}(\frac{n^2}{\epsilon})$ time (Theorem 3.2). This is inspired by how Saha [36] considered only $\tilde{O}(\frac{1}{\epsilon})$ pivots out of each range $(i\,..\,j)$. The original argument relies on two observations: that using pivot k' instead of k incurs at most $O(|k-k'|)$ extra edit operations, and that, for an ϵn-additive approximation, we can afford $O(\frac{\epsilon \min(k-i, j-k)}{\log n})$ extra operations when using pivot $k \in (i\,..\,j)$. In our multiplicative approximation, we refine the second observation by replacing $\min(k-i, j-k)$ with $\min(|K \cap (i\,..\,k)|, |K \cap (k\,..\,j)|)$. On the other hand, the first observation is not useful because the set $K \cap (i\,..\,j)$ is already relatively sparse. Thus, instead of restricting each range $(i\,..\,j)$ to use few pivots k, we restrict each pivot k to be used within few ranges $(i\,..\,j)$. This is feasible with respect to the approximation ratio because the costs for $x(i\,..\,j]$ and $x(i'\,..\,j']$ may only differ by $O(|i-i'| + |j-j'|)$, and because the $O(n^2 d)$-time algorithm still considers each pivot $k \in K$ for all ranges $(i\,..\,j)$ containing k (which leaves room for sparsification).

Constant-factor approximation in $\tilde{O}(nd)$ time. Overcoming the $O(n^2)$ barrier with a dynamic-programming approach poses significant challenges: there are $\Theta(n^2)$ substrings to consider and, for $d \geq \sqrt{n}$, this quantity does not decrease (in the worst case) even if we run the preprocessing of [7, 34] and exclude substrings with costs exceeding d. Thus, we artificially restrict the DP states to substrings whose all prefixes have at least as many opening parentheses as closing ones and whose all suffixes have at least as many closing parentheses as opening ones. Surprisingly, as shown in Section 4, this yields a 3-approximation of the original cost. Furthermore, if we additionally require that the number of opening parentheses and the number of closing parentheses across the entire substring are within $2d$ from each other (otherwise, the Dyck edit distance trivially exceeds the threshold), we end up with $O(nd)$ substrings. Reusing the pivot sparsification of Section 3, one can process them in $\tilde{O}(\frac{nd}{\epsilon})$ total time at the cost of increasing the approximation ratio from 3 to $3+\epsilon$.

Constant-factor approximation in $\tilde{O}(n^{1.971})$ time. In Section 5, we exhibit an $\tilde{O}(n^{1.971})$-time algorithm that provides a constant-factor approximation of Dyck edit distance. At a high level, the framework of our algorithm is similar to the three-step procedure of [15] that provides a constant-factor approximation of string edit distance in subquadratic time. Thus, we start with a brief recap of [15], pointing out the major bottlenecks for applying this framework directly to our problem. Given two strings x, y of length n, the algorithm of [15] starts by constructing a set of windows $\mathcal{W}_x$ for x and $\mathcal{W}_y$ for y, where each window is a length-s subinterval of $[1\,..\,n]$, representing a substring of x or y. The motivation behind

this construction is the following: given the edit distances between all pairs of windows from $\mathcal{W}_x$ and $\mathcal{W}_y$, one can compute a constant-factor approximation of the edit distance $\mathsf{ED}(x, y)$ using an $O(\frac{n^2}{s^2})$-time dynamic-programming procedure. The challenge here is that if the edit distances are computed using a trivial dynamic-programming algorithm for all pair of windows from $\mathcal{W}_x$ and $\mathcal{W}_y$, then the total running time becomes quadratic. The key insight of [15] is that, in some favorable situation, one can use random sampling to select a subset of window pairs from $\mathcal{W}_x \times \mathcal{W}_y$ such that evaluating the edit distances of the window pairs in the subset is enough to construct a nearly-optimal alignment of x, y. On the other extreme, instead of computing edit distance for each window pairs explicitly, one can use triangle inequality to get constant-factor approximation of the optimal costs.[4] Several other works have subsequently used this framework to solve related problems [12, 22, 33, 37].

As discussed above, much of the previous work on Dyck edit distance relies on similarities with edit distance, either via black-box reductions (such as the random walk of Saha [34]) or by transferring techniques (e.g., [7, 21] build on top of the $O(n + d^2)$-time algorithm [29] for edit distance). Hence, we try to adapt the framework of [15] to the Dyck setting.

The first challenge is that the Dyck edit distance is defined for a single string, so it is not immediately clear how to formulate the triangle inequality in this setting. However, the embedding of string edit distance into the Dyck edit distance hints a candidate for a metric: a function mapping strings $x, y \in \Sigma^*$ to $\mathsf{dyck}(x\overline{y})$, where $\overline{y}$ is the reverse complement of y (obtained by reversing y and flipping the direction of each parenthesis). This choice turns out to be a valid one: we show (in Lemma 2.3) that any three strings x, y, z satisfy $\mathsf{dyck}(x\overline{z}) \leq \mathsf{dyck}(x\overline{y}) + \mathsf{dyck}(y\overline{z})$, which we dub the triangle inequality for Dyck edit distance. Our proof is based on a subtle inductive argument that reduces the general case to that of $|y| \leq 1$ and $|x\overline{z}| \leq 2$. This base case, in turn, requires some case analysis.

A more serious issue is that the very first step of the algorithms of [11, 15], *window decomposition*, fails for our purposes, and, as discussed below, a workaround poses significant difficulties. To conclude the high-level discussion, we list our two main technical contributions leading to the subquadratic-time constant-factor approximation for Dyck edit distance:

1. We propose a new window decomposition strategy and show that any optimal alignment of x can be approximated by matching the window pairs generated by our strategy.
2. We establish the triangle inequality for Dyck edit distance.

Next, we discuss the limitations of the windowing strategy of [15] and explain how to overcome them. The algorithm of [15] partitions the input strings into fixed-length (overlapping) substrings, estimates the distances between relevant pairs of substrings, and then runs a dynamic-programming procedure to derive a global alignment. Regardless, following the strategy of dimension reduction, one idea could be to partition the input string x into windows $w_1, \ldots, w_\ell \subseteq [1 \,..\, n]$ of length s (where $s = n^{\Theta(1)}$) with the hope, that given the Dyck edit distances for all strings $x[w_i] \circ x[w_j]$ (here, $x[w_i]$ represents the substring of x restricted to the indices in w_i and $\circ$ denotes concatenation), one can use the cubic-time dynamic-programming algorithm to estimate $\mathsf{dyck}(x)$ in time $\tilde{O}(\frac{n^3}{s^3})$. However, this straightforward decomposition fails for the following reasons:

- In case of string edit distance, if an optimal alignment (with cost d) matches $x[i]$ with $y[j]$, then $x[i+1]$ can be matched only with a character of $y[j+1 \,..\, j+d+1]$. Thus, if we consider a window w_1 in x and a window w_2 in y such that $\mathsf{ED}(x[w_1], y[w_2])$ is

[4] The use of triangle inequality was first proposed in [11], where Grover search was used instead of random sampling, resulting in a quantum constant-factor approximation of edit distance in subquadratic time.

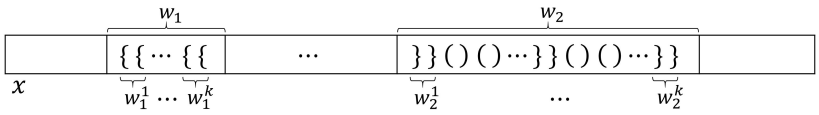


Figure 2 An example showing two very different length substrings can be matched with cost 0.

small, then we can assume $|w_1| \approx |w_2|$. This structural property completely breaks down for Dyck edit distance. For example, the two windows w_1, w_2 in Figure 2 satisfy $\mathsf{dyck}(x[w_1] \circ x[w_2]) = 0$ even though their lengths are very different. This indicates that partitioning x into single-length windows does not suffice.

- To overcome the aforementioned issue, let us assume that we allow variable-length windows. Note that an optimal alignment may match a window w_1 of length s with a window w_2 of length $\gg s$ ($|w_2|$ can be as large as $\Omega(n)$). However, if we allow windows of lengths $\gg s$, then estimating the costs of such large window pairs may be inefficient. One way out could be to subdivide both w_1 and w_2 into smaller windows $w_1^1, w_1^2, \ldots, w_1^k$ and $w_2^1, w_2^2, \ldots, w_2^k$, respectively, and separately compute the cost of each substring $x[w_1^i] \circ x[w_2^j]$. Here, as $|w_1^i|$ and $|w_2^j|$ are not too large, any individual cost can be approximated efficiently. However, since $|w_1^i|$ can be very small (as small as $n^{o(1)}$), the total number of subproblems (window pairs whose cost we evaluate) can explode, and hence the dynamic-programming procedure combining these subproblems may become inefficient.

Thus, for Dyck edit distance, the main challenge is to partition the input string x into variable-length windows which are neither too short (this ensures that the total number of subproblems, i.e., window pairs we evaluate, is not too large, and hence the DP combining them is efficient) nor too long (so that computing the costs of window pairs is efficient as well), and any optimal alignment of x can be approximated by matching these window pairs. Formally, we need to construct a set of windows $\mathcal{J}$, where each window has length at most s (we set this s to be a polynomial in n), $|\mathcal{J}| \approx \frac{n}{s}$, and there exists a subset $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{J}$ such that $\mathcal{S}$ is a consistent window decomposition of $[1\,..\,n]$ (i.e., the windows involved in $\mathcal{S}$ form a decomposition of $[1\,..\,n]$ and the window pairs in $\mathcal{S}$ do not cross) and there is a nearly-optimal alignment that aligns w with w' for each $(w, w') \in \mathcal{S}$. The latter condition is formalized as follows (the construction of $\mathcal{J}$ is parameterized by θ, chosen so that $\theta n \le \mathsf{dyck}(x)$):

▶ **Lemma 1.1.** *There exists a consistent window decomposition $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{J}$ of $[1\,..\,n]$ such that $\sum_{(w,w')\in\mathcal{S}} \mathsf{dyck}(x[w] \circ x[w']) \le \mathsf{dyck}(x) + 8\theta n$.*

The construction of an appropriate family $\mathcal{J}$ and the proof of Lemma 1.1 are among the novelties of our algorithm; this is where our approach significantly differs from [15].

Window Decomposition. Our proof of Lemma 1.1 (given in Section 5.1) follows a two-step strategy. In the first step, independent of the choice of $\mathcal{J}$, the decomposition $\mathcal{S}$ may contain arbitrary window pairs (w, w') with $|w|, |w'| \le s$, but we require $\sum_{(w,w')\in\mathcal{S}} \mathsf{dyck}(x[w] \circ x[w']) = \mathsf{dyck}(x)$ (no approximation allowed) and $|\mathcal{S}| = O(\frac{n}{s})$. In the second step, we locally perturb the endpoints of all windows in $\mathcal{S}$ so that the resulting windows belong to $\mathcal{J}$; this incurs an additive overhead of $O(\theta n)$ on the cost of the consistent window decomposition $\mathcal{S}$.

Our proof for the **first step** inductively constructs a consistent window decomposition of any window $(i_1\,..\,i_2] \subseteq [1\,..\,n]$. In the base case of $|(i_1\,..\,i_2]| \le 2s$, we build a single window pair composed of the two halves of $(i_1\,..\,i_2]$. In the main case, we identify an *outermost* window pair $((i_1\,..\,j_1], (j_2\,..\,i_2])$, with $j_1 \in [i_1\,..\,i_1 + s]$ and $j_2 \in [i_2 - s\,..\,i_2]$, and a *pivot* $p \in [i_1 + s\,..\,i_2 - s]$ so that

$$\mathsf{dyck}(x(i_1\,..\,i_2]) = \mathsf{dyck}(x(j_1\,..\,p]) + \mathsf{dyck}(x(p\,..\,j_2]) + \mathsf{dyck}(x(i_1\,..\,j_1] \circ x(j_2\,..\,i_2]).$$

The appropriate positions j_1, j_2, p can be derived from an optimal alignment of $x(i_1 \,..\, i_2]$:

- $(i_1 \,..\, j_1]$ can be defined as the shortest (possibly empty) prefix of $(i_1 \,..\, i_1 + s]$ containing all positions in $(i_1 \,..\, i_1 + s]$ matched with $(i_2 - s \,..\, i_2]$;
- $(j_2 \,..\, i_2]$ can be defined as the shortest (possibly empty) suffix of $(i_2 - s \,..\, i_2]$ containing all positions in $(i_2 - s \,..\, i_2]$ matched with $(i_1 \,..\, i_1 + s]$;
- $(i_1 + s \,..\, p]$ can be defined as the shortest (possibly empty) prefix of $(i_1 + s \,..\, i_2 - s]$ containing all positions in $(i_1 + s \,..\, i_2 - s]$ matched with $(i_1 \,..\, i_1 + s]$.

The sought decomposition of $(i_1 \,..\, i_2]$ is obtained by inserting $((i_1 \,..\, j_1], (j_2 \,..\, i_2])$ to the union of decompositions of $(j_1 \,..\, p]$ and $(p \,..\, j_2]$ (constructed recursively). It is not hard to prove that this construction satisfies the claims made above. The most subtle argument involves the size of the decomposition. This is because the bound $|\mathcal{S}| = O(\frac{n}{s})$ requires windows of average size $\Theta(s)$, but $(i_1 \,..\, j_1]$ and $(j_2 \,..\, i_2]$ can be arbitrarily short (even empty). Even worse, $(j_1 \,..\, p]$ and $(p \,..\, j_2]$ may also be arbitrarily short. However, we still have $|(i_1 \,..\, p]|, |(p \,..\, i_2]| \geq s$, and this suffices to inductively prove an upper bound of $\max(1, \frac{2(i_2 - i_1)}{s} - 1)$.

As for the **second step** of the proof of Lemma 1.1, we need to specify the choice of $\mathcal{J}$, which is parameterized by θ and s. We simply include in $\mathcal{J}$ all windows $w = (i_1 \,..\, i_2]$ of length at most s whose endpoints i_1, i_2 are both integer multiples of θs (in this overview, we assume for simplicity that $\frac{n}{s}$, θs, and $\frac{1}{\theta}$ are all integers). This way, $|\mathcal{J}| = O(\frac{n}{\theta^2 s})$ (there are $O(\frac{n}{\theta s})$ choices for the starting position and $O(\frac{s}{\theta s})$ choices for the length of a window in $\mathcal{J}$). Moreover, each window of length at most s can be transformed to a window in $\mathcal{J}$ by rounding both endpoints up to the nearest multiple of θs. When performed simultaneously on all windows in $\mathcal{S}$, this perturbation preserves the relative order of the windows, and thus $\mathcal{S}$ remains a consistent window decomposition of $[1 \,..\, n]$. Furthermore, for each window pair (w, w'), the value $\mathsf{dyck}(x[w] \circ x[w'])$ changes by at most $4\theta s$. Given that $|\mathcal{S}| \leq \frac{2n}{s}$, the overall additive overhead does not exceed $8\theta n$.

Two-Level Window Decomposition. If we could estimate the cost of each window pair in $\mathcal{J} \times \mathcal{J}$, this would provide a cost estimation for all window pairs in the unknown set $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{J}$ of Lemma 1.1. Thus, using a dynamic-programming procedure to optimize the cost over consistent decompositions $\tilde{\mathcal{S}} \subseteq \mathcal{J} \times \mathcal{J}$ of $[1 \,..\, n]$, we could approximate $\mathsf{dyck}(x)$.

However, similarly to [15], in order to estimate the cost of each window pair in $\mathcal{J} \times \mathcal{J}$, we further partition each large window into smaller windows and estimate the cost of these smaller window pairs. Thus, given another (smaller) window size parameter, we analogously construct a family $\mathcal{K}$ of variable-sized windows. Adapting the argument behind Lemma 1.1, we can show that, for each window pair $(w, w') \in \mathcal{J} \times \mathcal{J}$, the set $w \cup w'$ admits a consistent window decomposition $\mathcal{S}_{(w,w')} \subseteq \mathcal{K} \times \mathcal{K}$ such that $\sum_{(q,q') \in \mathcal{S}_{(w,w')}} \mathsf{dyck}(x[q] \circ x[q']) \leq \mathsf{dyck}(x[w] \circ x[w']) + O(\theta s)$ (Lemma 5.6 is formulated analogously to Lemma 1.1).

Certifying Window Pairs. With the two-level window decomposition at hand, adapting the remaining two phases of [15] is relatively easy. For this, we design a procedure `CertifyWindowPairs` that finds a cost estimation for selected window pairs in $\mathcal{J} \times \mathcal{J}$ and $\mathcal{K} \times \mathcal{K}$. The procedure shares a similar flavor with the Covering algorithm of [15] and relies on the triangle inequality (Lemma 2.3) discussed above. Its implementation and analysis is provided in the full version [18] only. The main guarantee of `CertifyWindowPairs` is that (with high probability) some of the certified window pairs can be combined to form a consistent window decomposition of $[1 \,..\, n]$ whose cost is $O(\mathsf{dyck}(x) + \theta n)$. Thus, a simple dynamic-programming algorithm (also provided in the full version [18] only) can be used to retrieve a constant-factor approximation of $\mathsf{dyck}(x)$ (recall that $\theta n \leq \mathsf{dyck}(x)$).

Folding Distance. The key difference between the *folding distance* (originating from the RNA folding problem) compared to the Dyck edit distance is that the alphabet is no longer partitioned into the set T of opening parentheses and the set $\overline{T}$ of closing parentheses. In other words, every character c can be matched with its complement $\overline{c}$ regardless of their order in the text. In particular, this means that the notions of heights and valleys are not meaningful anymore. Nevertheless, for any fixed alignment, one can distinguish the unmatched, opening (matched with a character to the right), and closing (matched with a character to the left) characters. Moreover, one can still greedily eliminate substrings of the form $c\overline{c}$ (similarly to the preprocessing of [7]). In any optimal alignment of an instance preprocessed this way, there must be an unmatched character between any two characters matched with each other. Although this reduction does not seem helpful in sparsifying the set of pivots to be considered, it does bring a strong structural property: there is a subset $[1\,..\,n]$ of size $n - O(d)$ (containing all characters matched without edits) which admits a consistent window decomposition into $O(d)$ window pairs (w, w') such that $x[w'] = \overline{x[w]}$ (and thus $\mathsf{fold}(x[w] \circ x[w']) = 0$). The strategy behind our $O(s)$-factor approximation is to sacrifice $O(s)$ boundary characters out of each window pair and, in exchange, make sure that the "closing windows" w' have both endpoints at positions divisible by s. We then use Internal Pattern Matching [24, 25] to efficiently search for "opening windows" w that could match our closing windows. Doing so, we cannot guarantee that the opening windows have their endpoints divisible by s, but we can sparsify the set of candidates so that they start at least s positions apart. This results in $O((\frac{n}{s})^3)$ window pairs to be considered and leads to the overall running time of $O(n + (\frac{n}{s})^3)$. The details are given in the full version [18].

2 Preliminaries

The alphabet Σ consists of two disjoint sets T and $\overline{T}$ of *opening* and *closing* parentheses, respectively, with a bijection $\overline{\cdot} : T \to \overline{T}$ mapping each opening parenthesis to the corresponding closing parenthesis. We extend this mapping to an involution $\overline{\cdot} : T \cup \overline{T} \to T \cup \overline{T}$ and then to an involution $\overline{\cdot} : \Sigma^* \to \Sigma^*$ mapping each string $x[1]x[2]\cdots x[n]$ to its reverse complement $\overline{x[n]}\cdots\overline{x[2]}\,\overline{x[1]}$. Given two strings x, y, we denote their concatenation by xy or $x \circ y$.

The *Dyck* language $\mathsf{Dyck}(\Sigma) \subseteq \Sigma^*$ consists of all well-parenthesized expression over Σ; formally, it can be defined using a context-free grammar whose only non-terminal S admits productions $S \to SS$, $S \to \varnothing$ (empty string), and $S \to aS\overline{a}$ for all $a \in T$.

▶ **Definition 2.1.** *The* Dyck edit distance $\mathsf{dyck}(x)$ *of a string $x \in \Sigma^*$ is the minimum number of character insertions, deletions, and substitutions required to transform x to a string in* $\mathsf{Dyck}(\Sigma)$.

We say that $M \subseteq \{(i,j) \subseteq \mathbb{Z}^2 : i < j\}$ is a *non-crossing matching* if any two distinct pairs $(i,j), (i',j') \in M$ satisfy $i < j < i' < j'$ or $i < i' < j' < j$. Such a matching can also be interpreted as a function $M : \mathbb{Z} \to \mathbb{Z} \cup \{\perp\}$ with $M(i) = j$ if $(i,j) \in M$ or $(j,i) \in M$ for some $j \in \mathbb{Z}$, and $M(i) = \perp$ otherwise.

For a string $x \in \Sigma^n$, the *cost* of a non-crossing matching $M \subseteq [n]^2$ on x (henceforth M is called an *alignment* of x) is defined as $\mathsf{cost}_M(x) = n - 2|M| + \sum_{(i,j)\in M} \mathsf{dyck}(x[i]x[j])$.

The following folklore fact (proved for completeness in the full version [18]), relates the Dyck edit distance with the optimum alignment cost.

▶ **Fact 2.2.** *For every string $x \in \Sigma^*$, the Dyck edit distance* $\mathsf{dyck}(x)$ *is the minimum cost* $\mathsf{cost}_M(x)$ *of an alignment M of x.*

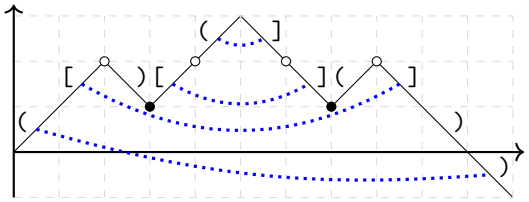


Figure 3 A plot of the height function h for $x = \texttt{([)[(]](])}$. The blue dotted lines represent an alignment $M = \{(1,11),(2,9),(4,7),(5,6)\}$ of cost 4. The valleys $\{3,7\}$ are marked as black circles. The set $K = \{2,3,4,6,7,8\}$ of Fact 2.6 also includes points marked as white circles.

We show that a function mapping $x, y \in \Sigma^*$ to $\mathsf{dyck}(x\overline{y})$ satisfies the triangle inequality.

▶ **Lemma 2.3.** *All strings $x, y, z \in \Sigma^*$ satisfy $\mathsf{dyck}(x\overline{z}) \le \mathsf{dyck}(x\overline{y}y\overline{z}) \le \mathsf{dyck}(x\overline{y}) + \mathsf{dyck}(y\overline{z})$.*

Proof. The second inequality follows from the fact that the Dyck language is closed under concatenations. As for the first inequality, we observe that it suffices to consider $|y| = 1$: the case of $|y| = 0$ is trivial, and the case of $|y| > 1$ can be derived from that of $|y| = 1$ by processing y letter by letter. Now, we proceed by induction on $2\mathsf{dyck}(x\overline{y}y\overline{z}) + |x| + |z|$. If any optimum alignment of $x\overline{y}y\overline{z}$ modifies a character in x or $\overline{z}$, we apply the inductive assumption for an instance (x', y, z') obtained from this modification: $\mathsf{dyck}(x\overline{z}) \le \mathsf{dyck}(x'\overline{z}') + 1 \le \mathsf{dyck}(x'\overline{y}y\overline{z}') + 1 = \mathsf{dyck}(x\overline{y}y\overline{z})$. If any optimum alignment of $x\overline{y}y\overline{z}$ matches any two adjacent characters of x, any two adjacent characters of $\overline{z}$, or the first character of x with the last character of $\overline{z}$, we apply the inductive assumption for an instance (x', y, z') obtained by removing these two characters: $\mathsf{dyck}(x\overline{z}) \le \mathsf{dyck}(x'\overline{z}') \le \mathsf{dyck}(x'\overline{y}y\overline{z}') = \mathsf{dyck}(x\overline{y}y\overline{z})$. In the remaining case, all characters of x and $\overline{z}$ must be matched to $\overline{y}$ or y, so $|x\overline{z}| \le 2$. If $|x\overline{z}| \le \mathsf{dyck}(x\overline{y}y\overline{z})$, then trivially $\mathsf{dyck}(x\overline{z}) \le |x\overline{z}| \le \mathsf{dyck}(x\overline{y}y\overline{z})$, so we may assume $\mathsf{dyck}(x\overline{y}y\overline{z}) < |x\overline{z}|$. The case of $\mathsf{dyck}(x\overline{y}y\overline{z}) = 0$ and $|x\overline{z}| = 1$ is impossible because only strings of even length belong to the Dyck language. Thus, we may assume that $|x\overline{z}| = 2$ and $\mathsf{dyck}(x\overline{y}y\overline{z}) \le 1$. If $|x| = 2$, then the optimum matching of $x\overline{y}y\overline{z}$ must be $\{(1,4),(2,3)\}$, and the sequence transforming $\mathsf{dyck}(x\overline{y}y\overline{z})$ to a word in $\mathsf{Dyck}(\Sigma)$ must include substituting $\overline{y}$ or y (whichever is an opening parenthesis). In particular, $x[1]$ must be an opening parenthesis, so $\mathsf{dyck}(x\overline{z}) = \mathsf{dyck}(x) \le 1 = \mathsf{dyck}(x\overline{y}y\overline{z})$. If $|\overline{z}| = 2$, then the optimum matching of $x\overline{y}y\overline{z}$ must be $\{(1,4),(2,3)\}$, and the sequence transforming $\mathsf{dyck}(x\overline{y}y\overline{z})$ to a word in $\mathsf{Dyck}(\Sigma)$ must include substituting $\overline{y}$ or y (whichever is a closing parenthesis). In particular, $z[1]$ must be an opening parenthesis, so $\mathsf{dyck}(x\overline{z}) = \mathsf{dyck}(\overline{z}) \le 1 = \mathsf{dyck}(x\overline{y}y\overline{z})$. Finally, if $|x| = |\overline{z}| = 1$, then the optimum matching of $x\overline{y}y\overline{z}$ must be $\{(1,2),(3,4)\}$. If $\mathsf{dyck}(x\overline{y}y\overline{z}) = 0$, then we must have $x = y = z \in T$, so $\mathsf{dyck}(x\overline{z}) = 0 \le \mathsf{dyck}(x\overline{y}y\overline{z})$. Otherwise, $x \in T$ or $z \in T$, so $\mathsf{dyck}(x\overline{z}) \le 1 = \mathsf{dyck}(x\overline{y}y\overline{z})$. ◀

In the remainder of this section, we recall several results from [7, 21] that we then use in our $\tilde{O}_\epsilon(n^2)$-time PTAS (Section 3) and $\tilde{O}_\epsilon(nd)$-time $(3+\epsilon)$-approximation (Section 4).

▶ **Definition 2.4** (Heights). *For a fixed string $x \in \Sigma^n$, the* height *function $h : [0\,..\,n] \to [-n\,..\,n]$ is defined so that $h(i) = |\{j \in [1\,..\,i] : x[j] \in T\}| - |\{j \in [1\,..\,i] : x[j] \in \overline{T}\}|$ for $i \in [0\,..\,n]$.*

▶ **Fact 2.5** ([7]). *There is a linear-time algorithm that, given a string $x \in \Sigma^n$, produces a string $x' \in \Sigma^{\le n}$ such that $\mathsf{dyck}(x) = \mathsf{dyck}(x')$ and x' has at most $2\mathsf{dyck}(x)$* valleys*, i.e., positions $v \in [1\,..\,n)$ such that $h(v-1) > h(v) < h(v+1)$.*

For a fixed string $x \in \Sigma^n$, let us define a function $\mathtt{D}$ such that $\mathtt{D}(i,j) = \mathsf{dyck}(x(i\,..\,j])$ for $i,j \in [0\,..\,n]$ with $i \le j$. Note that $\mathtt{D}(i,i) = 0$ for $i \in [0\,..\,n]$, $\mathtt{D}(i,i+1) = 1$ for $i \in [0\,..\,n)$, and $\mathtt{D}(i,j)$ satisfies the following recursion for $i,j \in [0\,..\,n]$ with $j - i \ge 2$:

$$\mathtt{D}(i,j) = \min \begin{cases} \mathtt{D}(i,k) + \mathtt{D}(k,j) & \text{for } k \in (i\,..\,j), \\ \mathtt{D}(i+1,j-1) + \mathsf{dyck}(x[i+1]x[j]). \end{cases} \tag{1}$$

This yields the classic $O(n^3)$-time algorithm computing $\mathtt{D}(0,n) = \mathsf{dyck}(x)$. The following result, combined with Fact 2.5, improves this time complexity to $O(n + n^2\mathsf{dyck}(x))$.

▶ **Fact 2.6** ([21, Lemma 2.1]). *For a string $x \in \Sigma^n$, let $K \subseteq [0\,..\,n]$ consist of all positions at distance 0 or 1 from a valley. For all $i,j \in [0\,..\,n]$ with $j - i \ge 2$, we have*

$$\mathtt{D}(i,j) = \min \begin{cases} \mathtt{D}(i,k) + \mathtt{D}(k,j) & \text{for } k \in (i\,..\,j) \cap (K \cup \{i+1, i+2, j-1, j-2\}), \\ \mathtt{D}(i+1,j-1) + \mathsf{dyck}(x[i+1]x[j]). \end{cases} \tag{2}$$

▶ **Observation 2.7** ([21, Fact 3.1]). *For all strings $x \in \Sigma^n$ and integers $0 \le i \le k \le j \le n$, we have $h(k) \ge \max(h(i), h(j)) - 2\mathtt{D}(i,j)$. In particular, $|h(i) - h(j)| \le 2\mathtt{D}(i,j)$.*

3 Quadratic-Time PTAS

In this section, we develop an $\tilde{O}(\epsilon^{-1}n^2)$-time algorithm that approximates $\mathsf{dyck}(x)$ within a $(1+\epsilon)$ factor. The starting point of our solution is the dynamic program derived from Fact 2.6. Instead of computing the exact value $\mathtt{D}(i,j) = \mathsf{dyck}(x(i\,..\,j])$, that depends on $\mathtt{D}(i,k) + \mathtt{D}(k,j)$ for all pivots $k \in (i\,..\,j) \cap (K \cup \{i+1, i+2, j-1, j-2\})$, we compute an approximation $\mathtt{AD}(i,j) \approx \mathsf{dyck}(x(i\,..\,j])$ in Algorithm 1 that depends only on $\mathtt{D}(i,k) + \mathtt{D}(k,j)$ for pivots $k \in (i\,..\,j) \cap (K_{i,j} \cup \{i+1, i+2, j-1, j-2\})$, where $K_{i,j}$ consists of $\tau_{i,j}$ leftmost and rightmost elements of $K \cap (i\,..\,j)$. Here, $\tau_{i,j}$ is proportional to the largest power of two dividing both i and j. Formally, we set $\tau_{i,j} := \tau \cdot 2^{\min(\nu(i),\nu(j))}$, where $\tau \ge 2$ is a parameter to be set later and $\nu : \mathbb{Z} \to \mathbb{Z}_{\ge 0} \cup \{\infty\}$ is a function that maps an integer $r \in \mathbb{Z}$ to $\nu(r) := \max\{k \in \mathbb{Z} : 2^k \text{ divides } r\}$, with the convention that $\nu(0) = \infty$.

In the following lemma, we inductively bound the quality of $\mathtt{AD}(i,j)$ as an additive approximation of $\mathtt{D}(i,j)$. In particular, we show that $\mathtt{D}(i,j) \le \mathtt{AD}(i,j) \le \mathtt{D}(i,j) + \frac{8}{\tau}|K|\log|K|$.

▶ **Lemma 3.1.** *If $\tau \ge 2$, then, for each $i,j \in [0\,..\,n]$ with $i \le j$, we have $\mathtt{D}(i,j) \le \mathtt{AD}(i,j) \le \mathtt{D}(i,j) + \frac{8}{\tau}c_{i,j}\log c_{i,j}$, where $c_{i,j} := |K \cap (i\,..\,j)|$, and we assume $0\log 0 = 0$.*

Algorithm 1 Recursive implementation of $\mathtt{AD}(i,j)$.

```
AD(i, j)
    if j = i then return 0;
    if j = i + 1 then return 1;
    c := AD(i + 1, j − 1) + dyck(x[i + 1]x[j]);
    τ_{i,j} := τ · 2^{min(ν(i),ν(j))};
    K_{i,j} := the set of τ_{i,j} smallest and τ_{i,j} largest elements of K ∩ (i .. j);
    foreach k ∈ K_{i,j} ∪ ({i + 1, i + 2, j − 2, j − 1} \ {i, j}) do
        c := min(c, AD(i, k) + AD(k, j));
    return c;
```

Proof. We proceed by induction on $j-i$. For $j-i \le 1$, we have $\mathtt{AD}(i,j) = \mathtt{D}(i,j)$. For $j-i \ge 2$, the lower bound $\mathtt{D}(i,j) \le \mathtt{AD}(i,j)$ follows directly from Fact 2.6. Unless $\mathtt{D}(i,j) = \mathtt{D}(i,k) + \mathtt{D}(k,j)$ for some $k \in (i\,..\,j) \cap K$, the upper bound also follows from Fact 2.6 since $c_{i,j}\log c_{i,j} \ge \max(c_{i+1,j-1}\log c_{i+1,j-1},\ c_{i,k}\log c_{i,k} + c_{k,j}\log c_{k,j})$. Let $r = \min(c_{i,k}, c_{k,j})$ and let i', j' be the smallest and the largest multiples of $2^{\lceil \log((r+1)/\tau)\rceil}$ within $[i\,..\,j]$.

Let us first prove that $k \in K_{i',j'}$. Note that $\tau(i'-i) < \tau 2^{\lceil\log((r+1)/\tau)\rceil} < \tau\cdot 2\cdot\frac{r+1}{\tau} = 2(r+1)$, so $\tau(i'-i) \le 2r$ (because both strict inequalities are between integers). A symmetric argument yields $\tau(j-j') \le 2r$. Due to $\tau \ge 2$, we thus have $i'-i \le r \le c_{i,k} < k-i$ and $j-j' \le r \le c_{j,k} < j-k$, so $k \in (i'\,..\,j')$. Moreover, $\tau_{i',j'} \ge \tau\cdot 2^{\lceil\log((r+1)/\tau)\rceil} \ge r+1 = \min(c_{i,k},c_{k,j})+1 \ge \min(c_{i',k},c_{k,j'})+1$, so $k \in K_{i',j'}$ holds as claimed.

Thus, due to $2r = 2\min(c_{i,k},c_{k,j}) \le c_{i,k}+c_{k,j} \le c_{i,j}$, we have

$$\begin{aligned}
\mathtt{AD}(i,j) &\le (i'-i) + \mathtt{AD}(i',j') + (j-j')\\
&\le \tfrac{2r}{\tau} + \mathtt{AD}(i',k) + \mathtt{AD}(k,j') + \tfrac{2r}{\tau}\\
&\le \mathtt{D}(i',k) + \tfrac{8}{\tau}c_{i',k}\log c_{i',k} + \mathtt{D}(k,j') + \tfrac{8}{\tau}c_{k,j'}\log c_{k,j'} + \tfrac{4r}{\tau}\\
&\le (i'-i) + \mathtt{D}(i,k) + \tfrac{8}{\tau}c_{i,k}\log c_{i,k} + \mathtt{D}(k,j) + (j-j') + \tfrac{8}{\tau}c_{k,j}\log c_{k,j} + \tfrac{4r}{\tau}\\
&\le \mathtt{D}(i,j) + \tfrac{8}{\tau}(c_{i,k}\log c_{i,k} + c_{k,j}\log c_{k,j} + r)\\
&= \mathtt{D}(i,j) + \tfrac{8}{\tau}(\max(c_{i,k},c_{k,j})\log\max(c_{i,k},c_{k,j}) + r\log(2r))\\
&\le \mathtt{D}(i,j) + \tfrac{8}{\tau}(\max(c_{i,k},c_{k,j})\log c_{i,j} + \min(c_{i,k},c_{k,j})\log c_{i,j})\\
&\le \mathtt{D}(i,j) + \tfrac{8}{\tau}c_{i,j}\log c_{i,j}.
\end{aligned}$$

◀

Our final solution simply uses Algorithm 1 with an appropriate choice of the parameter τ and the input string preprocessed using Fact 2.5 so that $|K| = O(\mathsf{dyck}(x))$.

▶ **Theorem 3.2.** *There is an algorithm Dyck-Approx that, given a string $x \in \Sigma^n$ and a parameter $\epsilon \in (0,1)$, in $\tilde{O}(\epsilon^{-1}n^2)$ time computes a value v such that $\mathsf{dyck}(x) \le v \le (1+\epsilon)\mathsf{dyck}(x)$.*

Proof. In the preprocessing, we use Fact 2.5 to guarantee that there are at most $2\mathsf{dyck}(x)$ valleys and thus $|K| \le 6\mathsf{dyck}(x)$. Next, we call $\mathtt{AD}(0,n)$ with $\tau = \lceil 48\epsilon^{-1}\log|K|\rceil$ and an array of size $(n+1)\times(n+1)$ memorizing the outputs of recursive calls. The resulting value satisfies $\mathsf{dyck}(x) \le \mathtt{AD}(0,n) \le \mathsf{dyck}(x) + \frac{8}{\tau}|K|\log|K| \le \mathsf{dyck}(x) + \frac{8}{48\epsilon^{-1}\log|K|}\cdot 6\mathsf{dyck}(x)\cdot\log|K| = (1+\epsilon)\mathsf{dyck}(x)$ by Lemma 3.1. The running time is proportional to

$$\begin{aligned}
n^2\sum_{i=0}^{n}\sum_{j=i+2}^{n}\tau_{i,j} &\le n^2 + \sum_{i=0}^{n}\sum_{j=i+2}^{n}\tau 2^{\nu(j)} \le n^2 + n\tau\sum_{j=2}^{n}2^{\nu(j)} \le n^2 + n\tau\sum_{\nu=0}^{\lfloor\log n\rfloor}\left\lfloor\frac{n}{2^\nu}\right\rfloor 2^\nu\\
&= O(n^2\tau\log n) = O(\epsilon^{-1}n^2\log^2 n) = \tilde{O}(\epsilon^{-1}n^2).
\end{aligned}$$

◀

4 Constant-Factor Approximation for Small Distances

In this section, we speed up the algorithm of Section 3 at the cost of increasing the approximation ratio from $1+\epsilon$ to $3+\epsilon$. The key idea behind our solution is to re-use the DP of Fact 2.6 and Algorithm 1 with an extra constraint that the transition from (i,j) to $(i+1,j-1)$ (which corresponds to adding $(i+1,j)$ to the alignment M, i.e., matching $x[i+1]$ with $x[j]$) is forbidden if there is a deep valley within $(i\,..\,j)$. This condition is expressed in terms of the following function:

▶ **Definition 4.1.** *For a fixed string $x \in \Sigma^n$ and $i,j \in [0\,..\,n]$ with $i \le j$, let $h(i,j) = \min_{k=i}^{j} h(k)$.*

Namely, we require that $h(i+1, j-1) > h(i,j)$ holds for all $(i+1, j) \in M$. For example, in the alignment M of Figure 3, $(2, 9) \in M$ violates this condition due to $h(1, 9) = 1 = h(2, 8)$, whereas the remaining pairs satisfy this condition. Formally, we transform the recursion of Fact 2.6 into the following one, specified through a function $\mathtt{GD} : [0\,..\,n]^2 \to [0\,..\,n]$ such that $\mathtt{GD}(i, i) = 0$ for $i \in [0\,..\,n]$, $\mathtt{GD}(i, i+1) = 1$ for $i \in [0\,..\,n)$, and, for all $i, j \in [0\,..\,n]$ with $j - i \geq 2$:

$$\mathtt{GD}(i,j) = \min \begin{cases} \mathtt{GD}(i,k) + \mathtt{GD}(k,j) & \text{for } k \in (i\,..\,j) \cap (K \cup \{i+1, i+2, j-2, j-1\}), \\ \mathtt{GD}(i+1, j-1) + \mathsf{dyck}(x[i+1]x[j]) & \text{if } h(i+1, j-1) > h(i,j). \end{cases}$$

Somewhat surprisingly, this significant limitation on the allowed alignments M incurs no more that a factor-3 loss in optimum alignment cost. Specifically, if we take an arbitrary alignment M of x and remove all pairs $(i+1, j)$ with $h(i+1, j-1) = h(i,j)$, the resulting alignment M' satisfies $\mathsf{cost}_{M'}(x) \leq 3\mathsf{cost}_M(x)$. This can be proved by induction on the structure of M using a potential function $h(i) + h(j) - 2h(i,j)$ as a "budget" for future deletions of matched pairs. Nevertheless, the proof of the following lemma operates directly on $\mathtt{D}$ and $\mathtt{GD}$.

▶ **Lemma 4.2.** *Let $x \in \Sigma^n$. For all $i, j \in [0\,..\,n]$ with $i \leq j$, we have $\mathtt{D}(i,j) \leq \mathtt{GD}(i,j) \leq 3\mathtt{D}(i,j) - h(i) - h(j) + 2h(i,j)$.*

Proof. We proceed by induction on $j - i$. The lower bound holds trivially. As for the upper bound, we consider several cases:

- $j = i$. In this case, $\mathtt{GD}(i,j) = 0 = 3 \cdot 0 - h(i) - h(i) + 2h(i) = \mathtt{D}(i,j) - h(i) - h(j) + 2h(i,j)$.
- $j = i + 1$. In this case, $\mathtt{GD}(i,j) = 1 < 2 = 3 \cdot 1 - h(i) - h(i+1) + 2\min(h(i), h(i+1)) = \mathtt{D}(i,j) - h(i) - h(j) + 2h(i,j)$.
- $\mathtt{D}(i,j) = \mathtt{D}(i,k) + \mathtt{D}(k,j)$ for some $k \in (i\,..\,j) \cap (K \cup \{i+1, i+2, j-2, j-1\})$. Then,

$$\begin{aligned} \mathtt{GD}(i,j) &\leq \mathtt{GD}(i,k) + \mathtt{GD}(k,j) \\ &\leq 3\mathtt{D}(i,k) - h(i) - h(k) + 2h(i,k) + 3\mathtt{D}(k,j) - h(k) - h(j) + 2h(k,j) \\ &= 3\mathtt{D}(i,j) - h(i) - h(j) - 2h(k) + 2\min(h(i,k), h(k,j)) + 2\max(h(i,k), h(k,j)) \\ &\leq 3\mathtt{D}(i,j) - h(i) - h(j) - 2h(k) + 2h(i,j) + 2h(k) \\ &= 3\mathtt{D}(i,j) - h(i) - h(j) + 2h(i,j). \end{aligned}$$

- $\mathtt{D}(i,j) = \mathtt{D}(i+1, j-1) + \mathsf{dyck}(x[i+1]x[j])$ and $h(i+1, j-1) = h(i,j) + 1$. Then,

$$\begin{aligned} \mathtt{GD}(i,j) &\leq \mathtt{GD}(i+1, j-1) + \mathsf{dyck}(x[i+1]x[j]) \\ &\leq 3\mathtt{D}(i+1, j-1) - h(i+1) - h(j-1) + 2h(i+1, j-1) + \mathsf{dyck}(x[i+1]x[j]) \\ &= 3\mathtt{D}(i,j) - h(i+1) - h(j-1) + 2h(i,j) + 2 - 2\mathsf{dyck}(x[i+1]x[j]) \\ &\leq 3\mathtt{D}(i,j) - h(i) - h(j) + 2h(i,j) \end{aligned}$$

  because $2\mathsf{dyck}(x[i+1]x[j]) \geq 2 + h(i) - h(i+1) + h(j) - h(j-1)$.
- $\mathtt{D}(i,j) = \mathtt{D}(i+1, j-1) + \mathsf{dyck}(x[i+1]x[j])$ and $h(i+1, j-1) = h(i,j)$. Then,

$$\begin{aligned} \mathtt{GD}(i,j) &\leq \mathtt{GD}(i, i+1) + \mathtt{GD}(i+1, j-1) + \mathtt{GD}(j-1, j) \\ &= \mathtt{GD}(i+1, j-1) + 2 \\ &\leq 3\mathtt{D}(i+1, j-1) - h(i+1) - h(j-1) + 2h(i+1, j-1) + 2 \\ &= 3\mathtt{D}(i,j) - h(i+1) - h(j+1) + 2h(i,j) - 3\mathsf{dyck}(x[i+1]x[j]) + 2 \\ &\leq 3\mathtt{D}(i,j) - h(i) - h(j) + 2h(i,j) \end{aligned}$$

  because $3\mathsf{dyck}(x[i+1]x[j]) \geq 2\mathsf{dyck}(x[i+1]x[j]) \geq 2 + h(i) - h(i+1) + h(j) - h(j-1)$. ◀

Next, we derive a property of $\texttt{GD}$ that allows for a speedup compared to $\texttt{D}$. Recall that $\texttt{GD}$ forbids the transition from (i,j) to $(i+1,j-1)$ if $h(i+1,j-1)=h(i,j)$. We further show that, in this case, it suffices to consider one specific pivot while computing $\texttt{GD}(i,j)$ (specifically, the pivot of minimum height, with ties resolved arbitrarily; see Fact 4.3). Later on (in the proof of Theorem 4.6), we argue that, after the preprocessing of Fact 2.5, there are only $O(nd)$ pairs (i,j) for which $h(i+1,j-1)>h(i,j)$ yet $\texttt{D}(i,j)\le d$.

▶ **Fact 4.3.** *Let $x\in\Sigma^n$ and let $i,j\in[0\,..\,n]$ with $j-i\ge 2$ and $h(i+1,j-1)=h(i,j)$. Then, every $k^*\in(i\,..\,j)$ with $h(k^*)=h(i,j)$ satisfies $\texttt{GD}(i,j)=\texttt{GD}(i,k^*)+\texttt{GD}(k^*,j)$.*

Proof. We proceed by induction on $j-i$. Fix $k\in(i\,..\,j)\cap(K\cup\{i+1,i+2,j-2,j-1\})$ such that $\texttt{GD}(i,j)=\texttt{GD}(i,k)+\texttt{GD}(k,j)$. If $k=k^*$, then the claim is trivial. Thus, by symmetry, we assume without loss of generality that $k^*\in(i\,..\,k)$. In particular, this means that $k-i\ge 2$ and $h(i+1,k-1)=h(k^*)=h(i,k)$, Consequently, by the inductive assumption, $\texttt{GD}(i,j)=\texttt{GD}(i,k)+\texttt{GD}(k,j)=\texttt{GD}(i,k^*)+\texttt{GD}(k^*,k)+\texttt{GD}(k,j)\ge\texttt{GD}(i,k^*)+\texttt{GD}(k^*,j)\ge\texttt{GD}(i,j)$, i.e., $\texttt{GD}(i,j)=\texttt{GD}(i,k^*)+\texttt{GD}(k^*,j)$ holds as claimed. Here, the first inequality holds because $k\in(k^*\,..\,j)\cap(K\cup\{k^*+1,k^*+2,j-2,j-1\})$, whereas the second one is due to $k^*\in K$ (because k^* is a valley). ◀

Our approximation algorithm (implemented as Algorithm 2) computes $\texttt{AGD}$ that approximates $\texttt{GD}$ in the same way $\texttt{AD}$ approximates $\texttt{D}$ in Algorithm 1. The only difference is that we use Observation 2.7 and Fact 4.3 (and the definition of $\texttt{GD}$) to prune some states and transitions. For each of the remaining states, the algorithm computes a value $\texttt{AGD}(i,j)\approx\texttt{GD}(i,j)$. If $h(i,j)=h(i+1,j-1)$, then Algorithm 2 relies on Fact 4.3 and considers the smallest index $k\in(i\,..\,j)$ with $h(k)=h(i,j)$ as the sole potential pivot, i.e, it returns $\texttt{AGD}(i,k)+\texttt{AGD}(k,j)$. If $h(i,j)<h(i+1,j-1)$, then Algorithm 2 mimics Algorithm 1.

The analysis of the approximation ratio of Algorithm 2 resembles that of Algorithm 1.

▶ **Lemma 4.4.** *If $\tau\ge 2$, then, for each $i,j\in[0\,..\,n]$ with $i\le j$, we have $\texttt{GD}(i,j)\le\texttt{AGD}(i,j)$ and, if $\texttt{GD}(i,j)\le d$, we further have $\texttt{AGD}(i,j)\le\texttt{GD}(i,j)+\frac{8}{\tau}c_{i,j}\log c_{i,j}$, where $c_{i,j}:=|K\cap(i\,..\,j)|$, and we assume $0\log 0=0$.*

Proof. As for the upper bound, we proceed by induction on $j-i$. For $j-i\le 1$, we have $\texttt{AGD}(i,j)=\texttt{GD}(i,j)$. For $j-i\ge 2$, the lower bound $\texttt{GD}(i,j)\le\texttt{AGD}(i,j)$ follows directly form the definitions of $\texttt{AGD}$ and $\texttt{GD}$. If $h(i,j)<\max(h(i),h(j))-2d$, then the upper bound follows

Algorithm 2 Recursive implementation of $\texttt{AGD}(i,j)$.

```
1  AGD(i, j)
2      if j = i then return 0;
3      if j = i + 1 then return 1;
4      if h(i, j) < max(h(i), h(j)) − 2d then return ∞;
5      if h(i, j) = h(i + 1, j − 1) then
6          Select the smallest k ∈ (i .. j) such that h(k) = h(i, j);
7          return AGD(i, k) + AGD(k, j);
8      c := AGD(i + 1, j − 1) + dyck(x[i + 1]x[j]);
9      τ_{i,j} := τ · 2^{min(ν(i),ν(j))};
10     K_{i,j} := the set of τ_{i,j} smallest and τ_{i,j} largest elements of K ∩ (i .. j);
11     foreach k ∈ K_{i,j} ∪ ({i + 1, i + 2, j − 2, j − 1} \ {i, j}) do
12         c := min(c, AGD(i, k) + AGD(k, j));
13     return c;
```

from Observation 2.7 and Lemma 4.2. If $h(i,j) = h(i+1,j-1)$, then the upper bound follows form Fact 4.3 because $c_{i,k}\log c_{i,k} + c_{k,j}\log c_{k,j} \le c_{i,j}\log c_{i,j}$. Otherwise, the upper bound follows directly from the definitions of $\texttt{AGD}$ and $\texttt{GD}$ unless $\texttt{GD}(i,j) = \texttt{GD}(i,k) + \texttt{GD}(k,j)$ for some $k \in (i\,..\,j) \cap K$. Let $r = \min(c_{i,k}, c_{k,j})$ and let i', j' be the smallest and the largest multiple of $2^{\lceil\log((r+1)/\tau)\rceil}$ within $[i\,..\,j]$.

Let us next prove that $k \in K_{i',j'}$. Note that $\tau(i'-i) < \tau 2^{\lceil\log((r+1)/\tau)\rceil} < \tau\cdot 2\cdot\frac{r+1}{\tau} = 2(r+1)$, so $\tau(i'-i) \le 2r$ (because both strict inequalities are between integers). A symmetric argument yields $\tau(j-j') \le 2r$. Due to $\tau \ge 2$, we thus have $i' - i \le r \le c_{i,k} < k - i$ and $j - j' \le r \le c_{j,k} < j - k$, so $k \in (i'\,..\,j')$. Moreover, $\tau_{i',j'} \ge \tau\cdot 2^{\lceil\log((r+1)/\tau)\rceil} \ge r+1 = \min(c_{i,k},c_{k,j})+1 \ge \min(c_{i',k},c_{k,j'})+1$, so $k \in K_{i',j'}$ holds as claimed.

Thus, due to $2r = 2\min(c_{i,k},c_{k,j}) \le c_{i,k}+c_{k,j} \le c_{i,j}$, we have

$$\begin{aligned}
\texttt{AGD}(i,j) &\le (i'-i) + \texttt{AGD}(i',j') + (j-j')\\
&\le \tfrac{2r}{\tau} + \texttt{AGD}(i',k) + \texttt{AGD}(k,j') + \tfrac{2r}{\tau}\\
&\le \texttt{GD}(i',k) + \tfrac{8}{\tau}c_{i',k}\log c_{i',k} + \texttt{GD}(k,j') + \tfrac{8}{\tau}c_{k,j'}\log c_{k,j'} + \tfrac{4r}{\tau}\\
&\le (i'-i) + \texttt{GD}(i,k) + \tfrac{8}{\tau}c_{i,k}\log c_{i,k} + \texttt{GD}(k,j) + (j-j') + \tfrac{8}{\tau}c_{k,j}\log c_{k,j} + \tfrac{4r}{\tau}\\
&\le \texttt{GD}(i,j) + \tfrac{8}{\tau}(c_{i,k}\log c_{i,k} + c_{k,j}\log c_{k,j} + r)\\
&= \texttt{GD}(i,j) + \tfrac{8}{\tau}(\max(c_{i,k},c_{k,j})\log\max(c_{i,k},c_{k,j}) + r\log(2r))\\
&\le \texttt{GD}(i,j) + \tfrac{8}{\tau}(\max(c_{i,k},c_{k,j})\log c_{i,j} + \min(c_{i,k},c_{k,j})\log c_{i,j})\\
&\le \texttt{GD}(i,j) + \tfrac{8}{\tau}c_{i,j}\log c_{i,j}.
\end{aligned}$$

◀

On the other hand, the complexity analysis is not as simple as in Section 3: it involves a charging argument bounding the number of states processed using the insight of Fact 4.3.

▶ **Proposition 4.5.** *There is an algorithm that, given a string $x \in \Sigma^n$, a threshold $d \in [1\,..\,n]$, and a parameter $\epsilon \in (0,1)$, in $\tilde{O}(\epsilon^{-1}nd)$ time reports that $\texttt{GD}(0,n) > d$ or outputs a value v such that $\texttt{GD}(0,n) \le v \le (1+\epsilon)\texttt{GD}(0,n)$.*

Proof. In the preprocessing, we use Fact 2.5 to guarantee that there are at most $2\mathsf{dyck}(x)$ valleys and thus $|K| \le 6\mathsf{dyck}(x)$. Then, we construct a data structure that, given $i,j \in [0\,..\,n]$, reports the smallest $k \in [i\,..\,j]$ such that $h(k) = h(i,j)$ [20]. Finally, we run $\texttt{AGD}(0,n)$ with $\tau = \lceil 48\epsilon^{-1}\log|K|\rceil$ and memoization of the results of recursive calls. By Lemma 4.4, the returned value satisfies $\texttt{GD}(0,n) \le \texttt{AGD}(0,n) \le \texttt{GD}(0,n) + \frac{8}{\tau}|K|\log|K| \le (1+\epsilon)\texttt{GD}(0,n)$.

The running time analysis is more complex than in the proof of Theorem 3.2. We say that a call $\texttt{AGD}(i,j)$ is *hard* if it reaches Line 8, *easy* if it terminates at Line 4 or Line 7, and *trivial* otherwise. Observe that the total cost of trivial calls is $\tilde{O}(n)$. Moreover, the cost of each easy call is $\tilde{O}(1)$ plus the cost of the two calls made in Line 7, but the call $\texttt{AGD}(i,k)$ is never easy (it can be hard or trivial). This is because the choice of k as the smallest index in $(i\,..\,j)$ with $h(k) = h(i,j)$ guarantees that either $k = i+1$ or $h(i+1,k-1) > h(k) = h(i,k) = h(i,j) \ge \max(h(i),h(j)) - 2d \ge h(i) - 2d = \max(h(i),h(k)) - 2d$. Consequently, the cost of each easy call can be charged to its parent or sibling (which is hard or trivial), and it suffices to bound the total running time of hard calls. By symmetry, we only bound the cost of hard calls $\texttt{AGD}(i,j)$ with $h(i) \ge h(j)$. We then observe that if $h(i) \ge h(j) = h(j')$ and $i > j > j'$, then $\texttt{AGD}(i,j')$ is easy. Consequently, there are at most $4d+1$ hard calls per i. The cost of each hard call $\texttt{AGD}(i,j)$ is $\tilde{O}(\tau_{i,j}) = \tilde{O}(\tau 2^{\nu(i)})$, for a total of $\tilde{O}(d\tau\sum_{i=0}^{n} 2^{\nu(i)}) = \tilde{O}(\epsilon^{-1}nd)$. ◀

▶ **Theorem 4.6.** *There is an algorithm that, given a string $x \in \Sigma^n$, a threshold $d \in [1\,..\,n]$, and a parameter $\epsilon \in (0,1)$, in $\tilde{O}(\epsilon^{-1}nd)$ time reports that $\mathsf{dyck}(x) > d$ or outputs a value v such that $\mathsf{dyck}(x) \le v \le (3+\epsilon)\mathsf{dyck}(x)$.*

Proof. We apply Proposition 4.5 with adjusted d (three times larger) and ϵ (three times smaller). The correctness follows from Lemma 4.2. ◀

5 Constant-Factor Approximation in Subquadratic Time

In this section, we provide an algorithm that, given a string $x \in \Sigma^*$, computes constant-factor approximation of $\mathsf{dyck}(x)$ in subquadratic time. Formally, we show the following:

▶ **Theorem 5.1.** *There exist a randomized algorithm that, given a string $x \in \Sigma^n$, in $\tilde{O}(n^{67/34}) = O(n^{1.971})$ time, outputs a value v such that $\mathsf{dyck}(x) \le v \le 41 \cdot \mathsf{dyck}(x)$ holds with probability at least $1 - n^{-9}$.*

Instead of directly proving the above theorem, we develop the following result:

▶ **Theorem 5.2.** *There exist a constant C and a randomized algorithm that, given a string $x \in \Sigma^n$, in time $\tilde{O}(n^{67/34})$, outputs a value v such that $\mathsf{dyck}(x) \le v \le 40 \cdot \mathsf{dyck}(x) + Cn^{33/34}$ holds with probability at least $1 - n^{-9}$.*

Proof of Theorem 5.1 from Theorem 5.2. First, run the algorithm of Theorem 4.6 with $\epsilon = 1$ and $d = \lceil Cn^{33/34} \rceil$, where C is the constant of Theorem 5.2, this procedure takes $\tilde{O}(n^{67/34})$ time and either outputs a 4-approximation of $\mathsf{dyck}(x)$ or reports that $\mathsf{dyck}(x) > d$. In the latter case, run the algorithm of Theorem 5.2 and return the resulting value v. Then, $\mathsf{dyck}(x) \le v \le 40 \cdot \mathsf{dyck}(x) + Cn^{33/34} \le 41 \cdot \mathsf{dyck}(x)$ holds with probability at least $1 - n^{-9}$. ◀

The rest of the section is devoted to prove Theorem 5.2.

5.1 Window Decomposition

Let us fix a string $x \in \Sigma^n$. For all integers $0 \le i_1 \le i_2 \le n$, we define a *window* $w := (i_1 \,..\, i_2]$ with *endpoints* $b(w) := i_1$, $e(w) := i_2$ and with *length* $|w| := i_2 - i_1$. We distinguish $n + 1$ distinct *empty* windows $(i \,..\, i]$ for $i \in [0 \,..\, n]$. For $w = (i_1 \,..\, i_2]$, we denote $x[w] := x(i_1 \,..\, i_2]$.

A *window pair* is a pair of windows (w, w'), and a weighed window pair is a triple (w, w', c) such that (w, w') is a window pair and $c \in \mathbb{R}_{\ge 0}$ is a weight. The *cost* of a window pair (w, w') is $\mathsf{dyck}(x[w] \circ x[w'])$, and a weighted window pair (w, w', c) is *certified* if $c \ge \mathsf{dyck}(x[w] \circ x[w'])$.

▶ **Definition 5.3.** *A set $\{(w_1, w_1'), \ldots, (w_\ell, w_\ell')\}$ of window pairs is a* consistent decomposition *of $\bigcup_{i=1}^{\ell}(w_i \cup w_i')$ if the 2ℓ windows are disjoint and $\{(b(w_i), b(w_i')) : i \in [1 \,..\, \ell]\}$ forms a non-crossing matching. We also lift this definition to sets of weighted window pairs.*

For a consistent decomposition $\mathcal{S}$, we write $\mathsf{dyck}(\mathcal{S}) := \sum_{(w,w') \in \mathcal{S}} \mathsf{dyck}(x[w] \circ x[w'])$ to denote the total cost of windows pairs in $\mathcal{S}$. Observe that if $\mathcal{S}$ is a consistent decomposition of $[1 \,..\, n]$, then $\mathsf{dyck}(\mathcal{S}) \ge \mathsf{dyck}(x)$.

Our first goal is to prove that, for every $s \in [1 \,..\, n]$, there exists a consistent decomposition $\mathcal{S}$ of $[1 \,..\, n]$ such that $\mathsf{dyck}(\mathcal{S}) = \mathsf{dyck}(x)$, $|\mathcal{S}| = O(\frac{n}{s})$, and each window in $\mathcal{S}$ is of length at most s. For this, we inductively construct a consistent window decomposition of an arbitrary interval $(i_1 \,..\, i_2] \subseteq [1 \,..\, n]$ specified as follows (recall that $\mathtt{D}(i_1, i_2)$ denotes $\mathsf{dyck}(x(i_1 \,..\, i_2])$):

▶ **Lemma 5.4.** *Let x be a string of length n and let $s \in [1 \,..\, n]$. For every interval $(i_1 \,..\, i_2] \subseteq [1 \,..\, n]$, there exists a consistent decomposition $\mathtt{Dec}(i_1, i_2)$ of $(i_1 \,..\, i_2]$ such that:*

1. *each window pair $(w, w') \in \mathtt{Dec}(i_1, i_2)$ satisfies $|w|, |w'| \le s$,*
2. *$\mathsf{dyck}(\mathtt{Dec}(i_1, i_2)) = \mathtt{D}(i_1, i_2)$, and*
3. *$|\mathtt{Dec}(i_1, i_2)| \le \max(1, \frac{2(i_2 - i_1)}{s} - 1)$.*

Proof. If $|(i_1 \,..\, i_2]| \le 2s$, we return $\mathtt{Dec}(i_1, i_2) := \{((i_1 \,..\, \lfloor \frac{i_1 + i_2}{2} \rfloor], (\lfloor \frac{i_1 + i_2}{2} \rfloor \,..\, i_2])\}$. In this simple base case, all the claims hold trivially.

In the main case, we grow the outermost window pair $((i_1\,..\,j_1],(j_2\,..\,i_2])$, starting with empty windows and maintaining two invariants: $|(i_1\,..\,j_1]|,|(j_2\,..\,i_2]| \le s$ and $\mathtt{D}(i_1,i_2) = \mathtt{D}(j_1,j_2) + \mathsf{dyck}(x(i_1\,..\,j_1]\circ x(j_2\,..\,i_2])$. Once there exists $p \in [i_1+s\,..\,i_2-s]$ with $\mathtt{D}(j_1,j_2) = \mathtt{D}(j_1,p)+\mathtt{D}(p,j_2)$ (in particular, this holds when $|(i_1\,..\,j_1]| = s$ or $|(j_2\,..\,i_2]| = s$), we terminate the process and return $\mathtt{Dec}(i_1,i_2) := \{((i_1\,..\,j_1],(j_2\,..\,i_2])\} \cup \mathtt{Dec}(j_1,p) \cup \mathtt{Dec}(p,j_2)$. By the inductive hypothesis, $\mathtt{Dec}(j_1,p), \mathtt{Dec}(p,j_2)$ satisfy all the claimed conditions. In particular, they form consistent decompositions of $(j_1\,..\,p]$ and $(p\,..\,j_2]$, respectively, and thus $\mathtt{Dec}(i_1,i_2)$ forms a consistent decomposition of $(i_1\,..\,i_2]$. By the first invariant, each window in $\mathtt{Dec}(i_1,i_2)$ is of length at most s. The second invariant and the definition of p yield

$$\begin{aligned}\mathtt{D}(i_1,i_2) &= \mathtt{D}(j_1,p) + \mathtt{D}(p,j_2) + \mathsf{dyck}(x(i_1\,..\,j_1]\circ x(j_2\,..\,i_2])\\ &= \mathsf{dyck}(\mathtt{Dec}(j_1,p)) + \mathsf{dyck}(\mathtt{Dec}(p,j_2)) + \mathsf{dyck}(x(i_1\,..\,j_1]\circ x(j_2\,..\,i_2])\\ &= \mathsf{dyck}(\mathtt{Dec}(i_1,i_2)).\end{aligned}$$

The choice of $p \in [i_1+s\,..\,i_2-s]$ further gives

$$\begin{aligned}&|\mathtt{Dec}(i_1,i_2)| \le 1+|\mathtt{Dec}(j_1,p)|+|\mathtt{Dec}(p,j_2)| \le 1+\max(1,\tfrac{2(p-j_1)}{s}-1)+\max(1,\tfrac{2(j_2-p)}{s}-1)\\ &\le 1+\max(1,\tfrac{2(p-i_1)}{s}-1)+\max(1,\tfrac{2(i_2-p)}{s}-1) = 1+\tfrac{2(p-i_1)}{s}-1+\tfrac{2(i_2-p)}{s}-1 = \tfrac{2(i_2-i_1)}{s}-1.\end{aligned}$$

Otherwise, we grow the outermost window pair using one of the following three cases. If there exists $p \in (j_1\,..\,i_1-s)$ such that $\mathtt{D}(j_1,j_2) = \mathtt{D}(j_1,p)+\mathtt{D}(p,j_2)$, we append $(j_1\,..\,p]$ to the window $(i_1\,..\,j_1]$. Then, the choice of p guarantees the first invariant, whereas the second invariant holds due to

$$\begin{aligned}\mathtt{D}(i_1,i_2) &= \mathtt{D}(j_1,p) + \mathtt{D}(p,j_2) + \mathsf{dyck}(x(i_1\,..\,j_1]\circ x(j_2\,..\,i_2])\\ &\ge \mathtt{D}(p,j_2) + \mathsf{dyck}(x(i_1\,..\,p]\circ x(j_2\,..\,i_2]) \ge \mathtt{D}(i_1,i_2).\end{aligned}$$

Symmetrically, if there exists $p \in (i_2-s\,..\,i_2)$ such that $\mathtt{D}(j_1,j_2) = \mathtt{D}(j_1,p)+\mathtt{D}(p,j_2)$, we prepend $(p\,..\,j_2]$ to the window $(j_2\,..\,i_2]$. By (1), the remaining case is when $\mathtt{D}(j_1,j_2) = \mathtt{D}(j_1+1,j_2-1)+\mathsf{dyck}(x[j_1+1]x[j_2])$, i.e., the optimum alignment matches $x[j_1+1]$ and $x[j_2]$. Then, we add both these characters to the outermost window pair. In this case, the first invariant holds due to $|(i_1\,..\,j_1]|,|(j_2\,..\,i_2]| < s$. As for the second invariant, we have

$$\begin{aligned}\mathtt{D}(i_1,i_2) &= \mathtt{D}(j_1+1,j_2-1) + \mathsf{dyck}(x[j_1+1]x[j_2]) + \mathsf{dyck}(x(i_1\,..\,j_1]\circ x(j_2\,..\,i_2])\\ &\ge \mathtt{D}(j_1+1,j_2-1) + \mathsf{dyck}(x(i_1\,..\,j_1+1]\circ x(j_2-1\,..\,i_2]) \ge \mathtt{D}(i_1,i_2).\end{aligned}$$

◀

Large and small windows. Let us fix an integer power of two $\theta \in [\frac{1}{n},1]$ (which will be set to $n^{-1/34}$ rounded down appropriately). For each power of two $s \in [1\,..\,\theta^{-1}]$, define a function $\mathtt{up}_s$ that maps each $i \in [0\,..\,n]$ to $\mathtt{up}_s(i) := \min(n,\theta s\lceil\frac{i}{\theta s}\rceil)$ and denote its image with N_s. Note that $N_s \subseteq [0\,..\,n]$ consists of n as well as all integer multiples of θs. Moreover, for each $i \in [0\,..\,n]$, the value $\mathtt{up}_s(i)$ is the successor of i in N_s.

We introduce the following family of variable-size windows:

$$\mathcal{I}_s := \{w \subseteq [1\,..\,n] : |w| \le s_1 \text{ and } b(w), e(w) \in N_s\}.$$

The following claim is a direct consequence of the construction.

▷ Claim 5.5. $|\mathcal{I}_s| = O(\frac{n}{\theta^2 s})$.

We pick two scales $s_1 \ge s_2$, denoting $\mathcal{J} := \mathcal{I}_{s_1}$ and $\mathcal{K} := \mathcal{I}_{s_2}$. For larger windows, we prove the following result:

▶ **Lemma 1.1.** *There exists a consistent window decomposition $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{J}$ of $[1\,..\,n]$ such that $\sum_{(w,w')\in\mathcal{S}} \mathsf{dyck}(x[w] \circ x[w']) \leq \mathsf{dyck}(x) + 8\theta n$.*

Proof. By Lemma 5.4 applied for $s := s_1$, there exists a consistent decomposition $\mathtt{Dec}(0,n)$ of $[1\,..\,n]$ such that $\mathsf{dyck}(\mathtt{Dec}(0,n)) = \mathsf{dyck}(x)$, $|\mathtt{Dec}(0,n)| \leq \frac{2n}{s_1}$, and each window in $\mathtt{Dec}(0,n)$ is of length at most s_1. In order to meet the condition $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{J}$, we round the window endpoints up using the $\mathtt{up}_{s_1}$ function. Formally, for each window pair $(w,w') \in \mathtt{Dec}(0,n)$, we create one window pair $(\tilde{w},\tilde{w}') \in \mathcal{S}$, where $b(\tilde{w}) = \mathtt{up}_{s_1}(b(w))$, $e(\tilde{w}) = \mathtt{up}_{s_1}(e(w))$, $b(\tilde{w}') = \mathtt{up}_{s_1}(b(w'))$, and $e(\tilde{w}') = \mathtt{up}_{s_1}(e(w'))$. The resulting family $\mathcal{S}$ satisfies $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{J}$ since $|w|, |w'| \leq s_1$ implies $|\tilde{w}|, |\tilde{w}'| \leq s_1$ (because s_1 is an integer multiple of θs_1). The relative order of windows involved in $\mathcal{S}$ is the same as in $\mathtt{Dec}(0,n)$, and thus $\mathcal{S}$ remains a consistent decomposition of $(0\,..\,n]$ (this is also because $\mathtt{up}_{s_1}(0) = 0$ and $\mathtt{up}_{s_1}(n) = n$). Moreover, a single edit may increase the Dyck edit distance by at most one, and thus

$$\begin{aligned}\mathsf{dyck}(\mathcal{S}) &= \sum_{(\tilde{w},\tilde{w}')\in\mathcal{S}} \mathsf{dyck}(x[\tilde{w}] \circ x[\tilde{w}']) \leq \sum_{(w,w')\in\mathtt{Dec}(0,n)} (\mathsf{dyck}(x[w] \circ x[w']) + 4\theta s_1) \\ &= \mathsf{dyck}(\mathtt{Dec}(0,n)) + 4\theta s_1 |\mathtt{Dec}(0,n)| \leq \mathsf{dyck}(x) + 8\theta n. \end{aligned}$$

◀

Our next objective is to estimate $\mathsf{dyck}(x[w] \circ x[w'])$ for each $w, w' \in \mathcal{J}$. For this, we utilize the smaller windows via the following result. Its proof, similar to that of Lemma 1.1, is left for the full version [18].

▶ **Lemma 5.6.** *For every $(w,w') \in \mathcal{J} \times \mathcal{J}$ with $e(w) \leq b(w')$, there exists a consistent window decomposition $\mathcal{S} \subseteq \mathcal{K} \times \mathcal{K}$ of $w \cup w'$ such that $\mathsf{dyck}(\mathcal{S}) \leq \mathsf{dyck}(x[w] \circ x[w']) + O(\theta|w \cup w'|)$.*

5.2 Outline of the Proof of Theorem 5.2

We set θ, s_1, s_2 to be the largest integer powers of two satisfying $s_1 \leq n^{21/34}$, $s_2 \leq n^{13/34}$, and $\theta \leq n^{-1/34}$, respectively. We first construct the families $\mathcal{J}$ and $\mathcal{K}$ of large and small windows, as defined in Section 5.1. Then, we run a procedure `CertifyWindowPairs` (described in the full version [18]), which certifies window pairs in $\mathcal{J} \times \mathcal{J}$ and $\mathcal{K} \times \mathcal{K}$; some pairs are certified directly, using Theorem 3.2 with $\epsilon = 1$ (which provides a 2-approximation), whereas others indirectly, using the triangle inequality (Lemma 2.3). The resulting family $\mathcal{W}$ of certified window pairs satisfies the following property with probability $1 - n^{-9}$: there exists a consistent decomposition $T \subseteq \mathcal{W}$ of $[1\,..\,n]$ such that $\sum_{(w,w',c)\in T} c \leq 40 \cdot \mathsf{dyck}(x) + O(\theta n)$.

Next, we use a simple dynamic-programming procedure (described in the full version [18]) to minimize the total cost $\sum_{(w,w',c)\in\tilde{T}} c$ among all consistent decompositions $\tilde{T} \subseteq \mathcal{W}$ of $[1\,..\,n]$. The resulting cost is at least $\mathsf{dyck}(x)$ because $\mathcal{W}$ contains certified window pairs only (that is, $c \geq \mathsf{dyck}(x[w] \circ x[w'])$ holds for each $(w,w',c) \in \mathcal{W}$). Moreover, the cost is at most $40 \cdot \mathsf{dyck}(x) + O(\theta n) \leq 40 \cdot \mathsf{dyck}(x) + O(n^{33/34})$ by the existence of T. The running time of the DP procedure is $\tilde{O}(|N_{s_2}|^3 + |\mathcal{W}|)$, which is $\tilde{O}(n^{67/34})$ by the choice of parameters. The details of the running-time analysis are left for the full version [18].

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.

2 Alfred V. Aho and Thomas G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4), 1972. doi:10.1137/0201022.

3 Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2001. doi:10.1137/s0097539700366528.

4 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *FOCS 2010*, pages 377–386. IEEE, 2010. doi:10.1109/focs.2010.43.

5 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *FOCS 2020*, pages 990–1001. IEEE, 2020. doi:10.1109/focs46700.2020.00096.

6 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM Journal on Computing*, 41(6):1635–1648, 2012. doi:10.1137/090767182.

7 Arturs Backurs and Krzysztof Onak. Fast algorithms for parsing sequences of parentheses with few errors. In *PODS 2016*, pages 477–488. ACM, 2016. doi:10.1145/2902251.2902304.

8 Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *FOCS 2004*, pages 550–559. IEEE, 2004. doi:10.1109/FOCS.2004.14.

9 Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *SODA 2006*, pages 792–801. ACM, 2006. doi:10.1145/1109557.1109644.

10 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *FOCS 2016*, pages 51–60. IEEE, 2016. doi:10.1109/FOCS.2016.15.

11 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. *Journal of the ACM*, 68(3):19:1–19:41, 2021. doi:10.1145/3456807.

12 Mahdi Boroujeni, Masoud Seddighin, and Saeed Seddighin. Improved algorithms for edit distance and LCS: Beyond worst case. In *SODA 2020*, pages 1601–1620. SIAM, 2020. doi:10.1137/1.9781611975994.99.

13 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48(2):481–512, 2019. doi:10.1137/17M112720X.

14 Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. *SIAM Journal on Computing*, 42(1):61–83, 2013. doi:10.1137/100816481.

15 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *Journal of the ACM*, 67(6):1–22, 2020. doi:10.1145/3422823.

16 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *STOC 2016*, pages 712–725. ACM, 2016. doi:10.1145/2897518.2897577.

17 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC 2022*. ACM, 2022. arXiv:2204.04500, doi:10.48550/arXiv.2204.04500.

18 Debarati Das, Tomasz Kociumaka, and Barna Saha. Improved approximation algorithms for dyck edit distance and RNA folding, 2021. arXiv:2112.05866.

19 Eldar Fischer, Frédéric Magniez, and Tatiana Starikovskaya. Improved bounds for testing Dyck languages. In *SODA 2018*, pages 1529–1544. SIAM, 2018. doi:10.1137/1.9781611975031.100.

20 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011. doi:10.1137/090779759.

21 Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. An improved algorithm for the k-Dyck edit distance problem. In *SODA 2022*. SIAM, 2022.

22 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *STOC 2020*, pages 657–670. ACM, 2020. doi:10.1145/3357713.3384300.

23 Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978.

24 Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018. URL: https://depotuw.ceon.pl/handle/item/3614.

25 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In *SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

26 Dexter C. Kozen. *Automata and Computability*. Springer New York, 1997. doi:10.1007/978-1-4612-1844-9.

27 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *MFCS 2011*, volume 6907 of *LNCS*, pages 412–423. Springer, 2011. doi:10.1007/978-3-642-22993-0_38.

28 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/S0097539794264810.

29 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

30 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

31 Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980. doi:10.1073/pnas.77.11.6309.

32 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Testing membership in parenthesis languages. *Random Structures & Algorithms*, 22(1), January 2003. doi:10.1002/rsa.10067.

33 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *FOCS 2019*, pages 1121–1145. IEEE, 2019. doi:10.1109/focs.2019.00071.

34 Barna Saha. The Dyck language edit distance problem in near-linear time. In *FOCS 2014*, pages 611–620. IEEE, 2014. doi:10.1109/focs.2014.71.

35 Barna Saha. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *FOCS 2015*, pages 118–135. IEEE, 2015. doi:10.1109/focs.2015.17.

36 Barna Saha. Fast & space-efficient approximations of language edit distance and RNA folding: An amnesic dynamic programming approach. In *FOCS 2017*, pages 295–306. IEEE, 2017. doi:10.1109/FOCS.2017.35.

37 Masoud Seddighin and Saeed Seddighin. $3+\epsilon$ approximation of tree edit distance in truly subquadratic time. In *ITCS 2022*, volume 215, pages 115:1–115:22, 2022. doi:10.4230/LIPIcs.ITCS.2022.115.

New Additive Approximations for Shortest Paths and Cycles

Mingyang Deng
Massachusetts Institute of Technology, Cambridge, MA, USA

Yael Kirkpatrick
Massachusetts Institute of Technology, Cambridge, MA, USA

Victor Rong
Massachusetts Institute of Technology, Cambridge, MA, USA

Virginia Vassilevska Williams
Massachusetts Institute of Technology, Cambridge, MA, USA

Ziqian Zhong
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

This paper considers additive approximation algorithms for All-Pairs Shortest Paths (APSP) and Shortest Cycle in undirected unweighted graphs. The results are as follows:

- We obtain the first $+2$-approximation algorithm for APSP in n-vertex graphs that improves upon Dor, Halperin and Zwick's (SICOMP'00) $\tilde{O}(n^{7/3})$ time algorithm. The new algorithm runs in $\tilde{O}(n^{2.29})$ time and is obtained via a reduction to Min-Plus product of bounded difference matrices.
- We obtain the first additive approximation scheme for Shortest Cycle, generalizing the approximation algorithms of Itai and Rodeh (SICOMP'78) and Roditty and Vassilevska W. (SODA'12). For every integer $r \geq 0$, we give an $\tilde{O}(n + n^{2+r}/m^r)$ time algorithm that returns a $+(2r+1)$-approximate shortest cycle in any n-vertex, m-edge graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Fine-grained Complexity, Additive Approximation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.50

Category Track A: Algorithms, Complexity and Games

Funding *Virginia Vassilevska Williams*: Partially funded by NSF awards 2129139 and 1909429, BSF award 2020356, and a Google Research Fellowship.

1 Introduction

The all-pairs shortest paths problem (APSP) is among the most fundamental problems in algorithms. The fastest algorithms for the problem in n vertex, m edge graphs with integer edge weights and no negative cycles, run in $n^3/2^{\Theta(\sqrt{\log n})}$ time [22] and in $O(mn+n^2 \log\log n)$ time [14]. These running times are believed to be optimal, up to $n^{o(1)}$ factors (see [19, 12]).

APSP in unweighted graphs has long been known to admit faster algorithms. In undirected graphs, Seidel [16] gave an $\tilde{O}(n^\omega)$ time algorithm, where $\omega < 2.373$ [2] is the exponent of square matrix multiplication. This running time is believed to be optimal, as APSP in undirected unweighted graphs is at least as hard as Boolean Matrix Multiplication (BMM). [1]

[1] In fact, later algorithms by Shoshan and Zwick [17] imply that undirected unweighted APSP is equivalent to BMM.

Due to the impracticalities of fast matrix multiplication and since it is unavoidable for the exact computation of APSP, it is natural to consider approximation algorithms, trying to get running times as close to n^2 as possible (n^2 is the size of the output). There is significant work on multiplicative approximations of APSP and distance oracles (e.g. [18, 6, 1]). However, for unweighted undirected graphs, more desirable fast *additive* approximations are also possible. In a $+C$-approximation to APSP, one returns estimates $d'(u,v)$ for the distance $d(u,v)$ between any pair of vertices u, v, so that $d(u,v) \leq d'(u,v) \leq d(u,v) + C$. We let $+C$-APSP denote the problem of computing a $+C$-approximation to APSP in an undirected unweighted graph.

Following work of Aingworth et al. [1], so far the best approximation-running time tradeoff for unweighted undirected APSP is achieved by Dor, Halperin and Zwick [7]: For every even integer $k \geq 2$, there is an $\tilde{O}(n^{2+2/(3k-2)})$ time algorithm for $+k$-APSP. (For sparser graphs, the improved running time $\tilde{O}(n^{2-2/(k+2)}m^{2/(k+2)})$ is also given.)

In particular, Dor, Halperin and Zwick provide a $+2$-approximation algorithm that runs in $\tilde{O}(n^{7/3})$ time. The algorithm is simple and "combinatorial", and notably is *faster* than the fastest exact algorithm for APSP by Seidel for the current bounds on ω. Aignworth et al. [1] showed that $+1$-approximating APSP is at least as hard as BMM, and thus likely requires $n^{\omega-o(1)}$ time, so beating n^ω is only possible if the additive approximation is at least 2.

There are no known conditional lower bounds for $+2$-approximations of APSP, and the Dor, Halperin and Zwick running time has remained unchallenged for over *three decades*.

Is $n^{7/3-o(1)}$ time necessary for $+2$-APSP? Or, can one do better?

Our first result is the first *improvement* over the $+2$-approximation algorithm of [7].

▶ **Theorem 1.** *There is an $O(n^{2.2867})$ time algorithm that returns a $+2$-approximation to APSP in undirected unweighted n vertex graphs.*

We obtain the new tradeoff by reducing the $+2$-APSP problem to Min-Plus product of (rectangular) Bounded Difference Matrices. While Min-Plus product for $n \times n$ matrices is believed to require $n^{3-o(1)}$ time (see e.g. [19]), a string of recent papers has developed faster and faster truly subcubic time algorithms for Min-Plus product for Bounded Difference Matrices [4, 5, 8, 21]. Our reduction is completely black-box, so that if there is an improved algorithm for the Min-Plus product of an integer matrix with a matrix with bounded difference columns or rows, then this improvement would immediately translate into an improved algorithm for $+2$-APSP.

Vassilevska W. and Williams [20] showed that APSP in weighted graphs is fine-grained equivalent to the Girth problem, which asks to compute the length of the shortest cycle in a given undirected graph. The same paper [20] shows that the Girth problem in *undirected unweighted* graphs is at least as hard as triangle detection, and is known to be subcubically equivalent to undirected unweighted APSP. As triangle detection in n vertex graphs is believed to require $n^{\omega-o(1)}$ time (e.g. [19]), so is Girth, and thus faster approximation algorithms for the Girth are also well-motivated and well-studied. Multiplicative approximation schemes are well-understood (e.g. [10, 13, 15]). Meanwhile, unlike for APSP, there is *no known tradeoff* of additive approximation algorithms for the girth of undirected unweighted graphs.

There are only two known additive approximation results for Girth in undirected unweighted graphs. First, Itai and Rodeh [9] showed that a $+1$-approximation to the girth can be obtained in $O(n^2)$ time. Then, Roditty and Vassilevska W. [15] showed that a $+3$-approximation to the girth in n vertex, m edge graphs, can be computed in $\tilde{O}(n^3/m)$ time. *Can one generalize these two algorithms to a scheme?*

Our second result is to obtain the first additive approximation scheme for the girth.

▶ **Theorem 2.** *Let $G = (V, E)$ be an unweighted, undirected graph with $|V| = n, |E| = m$. Let r be an integer and denote the (unknown) girth of G by g. There is an $\tilde{O}\left(n + \frac{n^{2+r}}{m^r}\right)$ time algorithm that returns with high probability a cycle of length $\hat{g}$ that satisfies $g \leq \hat{g} \leq g+(2r+1)$.*

Thus, for instance, there is an $\tilde{O}(n + n^4/m^2)$ time $+5$-approximation algorithm. This running time is always better than the previously known additive approximations as long as $m \geq \Omega(n^{1+\varepsilon})$ and $m \leq O(n^{1.5-\varepsilon})$ for some $\varepsilon > 0$. More generally, as r grows, each $+(2r+1)$ approximation is always faster than the approximations for smaller r for the sparsity range $m \in [\Omega(n^{1+\varepsilon}), O(n^{1+1/r-\varepsilon})]$ for any arbitrarily small $\varepsilon > 0$.

2 Additive Approximation Algorithm for APSP

We first formally define $+C$-approximation of APSP and $(\min, +)$ matrix product.

▶ **Definition 3** (APSP $+C$-approximation). *For an undirected unweighted simple graph $G = (V, E)$, output $\hat{d} : V \times V \to \mathbb{N}$ such that $d(u, v) \leq \hat{d}(u, v) \leq d(u, v) + C$, where $d(u, v)$ is the distance from u to v in graph G.*

▶ **Definition 4** ($(\min, +)$ matrix product). *$(\min, +)$ matrix product between matrix A and B is defined as $C = A \star B$ where $C[i, j] = \min_k\{A[i, k] + B[k, j]\}$.*

While in general, no sub-cubic algorithm has been found for $(\min, +)$ matrix product, many special cases have been addressed (e.g. [4] [21] [5]). Specifically, we consider the case between column bounded-difference and row bounded-difference matrix.

▶ **Definition 5** (Column bounded-difference, Row bounded-difference). *A matrix A is column bounded-difference if there exists some constant C so that $|A[i, j] - A[i + 1, j]| \leq C$ for all valid (i, j)'s. Symmetrically, a matrix A is row bounded-difference if there exists some constant C so that $|A[i, j] - A[i, j + 1]| \leq C$ for all valid (i, j)'s.*

▶ **Definition 6** ($(\min, +)$ matrix product between column bounded-difference matrix and row bounded-difference matrix). *Given column bounded-difference matrix A of size $n \times m$ and row bounded-difference matrix B of size $m \times n$, calculate their $(\min, +)$ matrix product $A \star B$. Call the time complexity for such a problem $MPCRBD(n, m)$.*

A similar case was previously addressed by Bringmann et al. [4] and an algorithm of runtime $O(n^{2.9217})$ is given (Theorem 1.3, [4]). By adapting the method of Chi, Duan and Xie [5], we can get the following bound.

We want to point out that since our reduction is black-box, the use of the following lemma is not necessary and applying the algorithm in Bringmann et al. [4] also gives a $O(n^{7/3-\Omega(1)})$ algorithm. Therefore, we defer the proof of this lemma to Appendix A.

▶ **Lemma 7** (Appendix A). *Let $M(n, u, n)$ be the time to multiply $n \times u$ and $u \times n$ matrices. For parameters $\alpha, \beta, \gamma > 0$, $MPCRBD(n, m) = \tilde{O}(n^2 m/\alpha^2 + n^2\beta + M(n, \gamma, n)\alpha m/\beta + n^2 m^2/\gamma)$. Specifically, $MPCRBD(n, n) = O(n^{2.811})$.*

We start with Dor, Halperin and Zwick's original algorithm, used as a black-box.

▶ **Lemma 8** (Theorem 3.1, [7]). *There is an algorithm for APSP $+2$-approximation that runs in $\tilde{O}(|V|^{3/2}|E|^{1/2})$ time on input graph $G = (V, E)$.*

For a graph $G = (V, E)$, define $N(v) = \{u \mid (u, v) \in E\}$ as the set of adjacent vertices of v for $v \in V$. We also use the following lemma.

▶ **Lemma 9** (e.g. Theorem 1, [1]). *For a graph $G = (V, E)$ and a parameter s, let $V_s = \{v \mid v \in V,\ |N(v)| \geq s\}$, we can deterministically compute a hitting set $D \subseteq V$ where $|D| = O(|V| \log |V|/s)$ and $N(v) \cap D \neq \emptyset$ for all $v \in V_s$ in $O(|V|^2)$ time.*

Crucial to our algorithm is to consider an Euler tour of a spanning tree.

▶ **Definition 10** (Euler Tour). *For a spanning tree $T \subseteq E$ of a connected graph $G = (V, E)$, an Euler tour of T is a sequence of vertices $v_1, v_2, \cdots, v_{2|V|-1}$, where each vertex of G appears at least once and the edges $(v_i, v_{i+1}) \in T$ for all $1 \leq i \leq 2|V| - 2$.*

Given a tree T, an Euler tour of T could be easily found by running depth-first search on T in $O(|T|)$ time.

We first give a high-level overview of our improvement. In Theorem 3.2, [7], a hitting set D_1 is computed to update the distance between all pairs of nodes $(u, v) \in V^2$. That is, for all $u, v \in V$, we update $d(u, v)$ with $\min_{t \in D_1}(d(u, t) + d(t, v))$. Notice that if we consider V, D_1, V as three dimensions in a matrix multiplication, the updating process is essentially a min-plus matrix product. Now if we arrange u, v in the order of an Euler tour, the matrices would then be (row/column) bounded difference. Thus, we can gain a speedup by aforementioned algorithms. To calculate distances between D_1 and V efficiently, we partition nodes in V by their degree.

For a graph $G = (V, E)$, let $n = |V|,\ m = |E|$. We set a parameter $l = O(\log n)$ to be determined, and define $s_i = n/2^{i-1}$ for $i \in [1, l]$. Specifically, let $s_0 = n + 1$.

By Lemma 9, we can find hitting sets D_i of size $\tilde{O}(n/s_i)$ for all vertices with degree $\geq s_i$ in G in $\tilde{O}(n^2)$ time. Let $f_i(u)$ be any element in $D_i \cap N(u)$ for $\deg u \geq s_i$, let F_i be the set of edges $(u, f_i(u))$ where $\deg u \geq s_i$ ($\deg v$ stands for the degree of vertex v), and $F = \cup_{i=0}^{l} F_i$. Thus $|F_i| = O(n), |F| = \tilde{O}(n)$.

We define a series of graphs $G_0, G_1, G_2, \cdots, G_l$. Define $G_0 = G$, and $G_i = (V, E_i)$ where $E_i = F \cup \{(u, v) \in E \mid \deg u \leq s_i \text{ or } \deg v \leq s_i\}$ for $1 \leq i \leq l$. Let $d_i(u, v)$ be the distance from u to v on graph G_i.

We then run breadth-first search on graph G_{i-1} from each vertex in set D_i to obtain the distances $d_{i-1}(t, v)$ for all pairs $(t, v) \in D_i \times V$. Define $w_i(u, v) = \min_{t \in D_i}\{d_{i-1}(t, u) + d_{i-1}(t, v)\}$ Let $g(u, v)$ to be the output of algorithm in Lemma 8 on input G_l. Let $h(u, v) = \min(\min_{i=1}^{l}\{w_i(u, v)\}, g(u, v))$.

We now argue that h is the desired approximation to distances, i.e. $d(u, v) \leq h(u, v) \leq d(u, v) + 2$ for all $u, v \in V$. Note that $w_i(u, v), g(u, v) \geq d(u, v)$ since their values all arise from valid paths, thus $h(u, v) \geq d(u, v)$.

Now we show that $d(u, v) + 2 \geq h(u, v)$. Consider the shortest path p from u to v in graph G. Let r be the node with maximum degree on p. If $\deg r \geq s_l$, assume that $s_a \leq \deg r < s_{a-1}$ for some $1 \leq a \leq l$. Since all nodes on p have degree $< s_{a-1}$, $d(u, v) = d_{a-1}(u, v)$. In G_{a-1}, consider $g = f_a(r)$, we have $h(u, v) \leq w_a(u, v) \leq d_{a-1}(u, g) + d_{a-1}(g, v) \leq d_{a-1}(u, r) + 1 + d_{a-1}(r, v) + 1 = d_{a-1}(u, v) + 2 \leq d(u, v) + 2$. Otherwise (if $\deg r < s_l$), p is preserved in the graph G_l and $g(u, v) \leq d(u, v) + 2$ by Lemma 8. Therefore, $h(u, v) \leq d(u, v) + 2$.

We now show how to compute w_i for a given $1 \leq i \leq l$. Without loss of generality, assume G_{i-1} is connected (otherwise we can treat each connected component separately), and let T be any spanning tree of G_{i-1}. Let $t_1, t_2, t_3, \cdots, t_{2n-1}$ be an arbitrary Euler tour of T and suppose $D_i = \{x_1, x_2, \cdots, x_{|D_i|}\}$. Define A to be a matrix of size $(2n - 1) \times |D_i|$, where $A(i, j) = d_{i-1}(t_i, x_j)$. Let $B = A \star A^T$, then by definition, $w_i(u, v) = B[pos(u), pos(v)]$ where $t_{pos(a)} = a$ (any suffice).

Note that $|A[a, j] - A[a+1, j]| \leq 1$ since $|d_{i-1}(t_a, d_j) - d_{i-1}(t_{a+1}, d_j)| \leq d_{i-1}(t_a, t_{a+1}) = 1$, so A is column bounded-difference, A^T is row bounded-difference. The time complexity to compute $f(u, v)$ for all $u, v \in V$ is then $O(n^2 + \text{MPCRBD}(n, |D_i|))$.

The total runtime of the above algorithm consists of three parts.

1. The computation of $\{D_i\}$, $\{G_i\}$, $\{F_i\}$ takes $\tilde{O}(n^2)$ time.
2. Compute w_i for all $1 \le i \le l$. Constructing the depth-first search tree and Euler tour takes $\tilde{O}(n^2)$ time. Since $|E_i| \le s_i n$, the breadth-first search for each D_i in G_{i-1} takes $O(|D_i|s_i n) = \tilde{O}(n^2)$ time, so this part also takes $\tilde{O}(n^2)$ time. The $(\min, +)$ matrix products takes time $\tilde{O}(n^2 + \text{MPCRBD}(n, |D_l|))$ since $|D_l| \ge |D_i|$ for any $1 \le i \le l$.
3. The computation of g takes time $N^{3/2}|E_l|^{1/2} = \tilde{O}(n^2 s_l^{1/2})$.

Let s_l be the power of 2 closest to t, we obtain the following result:

▶ **Theorem 11.** *For a parameter $t \ge 1$, there is an algorithm for APSP +2-approximation on input graph $G = (V, E)$ that runs in $\tilde{O}(n^2 t^{1/2} + MPCRBD(n, n/t))$ time, where $n = |V|$, $m = |E|$.*

By Lemma 7, we could obtain an upper bound for $\text{MPCRBD}(n, m)$.

▶ **Theorem 12.** *There is an algorithm for APSP +2-approximation on input graph $G = (V, E)$ that runs in $O(n^{2.2867})$ time where n stands for $|V|$.*

Proof. In Theorem 11, set $t = n^{0.57339}$, and apply Lemma 7 with $\alpha = n^{0.07006}, \beta = n^{0.28662}, \gamma = n^{0.56688}$. $M(n, \gamma, n) = M(n, n^{0.56688}, n) = O(n^{2.076433})$ by [11]. The claimed bound then follows from a direct computation. ◀

We also provide the following simpler bound using square MPCRBD. Plugging in Theorem 1.3 in Bringmann et al. [4] gives an algorithm of $O(n^{2.32416})$.

▶ **Corollary 13.** *Suppose $MPCRBD(n, n) = O(n^{2+\alpha})$ for constant α, there is an algorithm for APSP +2-approximation on input graph $G = (V, E)$ that runs in $\tilde{O}(|V|^{2+\frac{\alpha}{1+2\alpha}})$ time.*

Proof. Let $n = |V|$. $\text{MPCRBD}(n, n/t) = O(t^2 \text{MPCRBD}(n/t, n/t)) = O(n^{2+\alpha}/t^\alpha)$. Set $t = n^{\alpha/(1/2+\alpha)}$, the total runtime would then be $\tilde{O}(n^{2+\frac{\alpha}{1+2\alpha}})$. ◀

3 Additive Approximation Algorithm for Girth

The first additive approximation algorithm for the girth of a graph was presented by Itai and Rodeh in 1978 [9]. In their paper they showed an $O(n^2)$ time algorithm that, given a graph of girth g, returns a cycle of length at most $g + 1$.

The next improvement, an additive approximation algorithm running in subquadratic time, was presented in 2012 by Roditty and Vassilevska W. [15]. Their algorithm provided a $+3$ approximation of the girth in $\tilde{O}(n^3/m)$ time.

Both algorithms use a subroutine which we will call BFS-cycle. This subroutine runs BFS from a given vertex s until it reaches some vertex v for a second time. When v is reached for the second time, BFS-cycle returns the cycle enclosed by the two paths between s and v. We will use the following result regarding this algorithm:

▶ **Lemma 14** (e.g. [9, 13])**.** *BFS-cycle(v) runs in $O(n)$ time. If a vertex v is at distance ℓ from a vertex on a simple cycle of length k, then BFS-cycle(v) reports a cycle of length at most $k + 2\ell + 1$. If k is even, BFS-cycle(v) reports a cycle of length at most $k + 2\ell$.*

Another tool used in Roditty and Vassilevska W.'s algorithm is a result in extremal graph theory proved by Bondy and Simonovits [3], regarding even cycles in a graph:

▶ **Theorem 15** ([3])**.** *Let $k \ge 2$ be an integer. If an n-node graph G has at least $100kn^{1+1/k}$ edges then G contains a $2k$-cycle.*

As a result of this theorem, for any graph with at least $200kn^{1+1/k}$ edges, at least half of the edges are part of some $2k$-cycle. Therefore, if we uniformly sample an edge, it is part of a $2k$-cycle with probability $\geq \frac{1}{2}$. By running BFS-cycle from each sampled edge and taking the lowest result, we obtain a randomized algorithm with efficient runtime:

▶ **Lemma 16** (e.g. [23]). *If an n-node graph G has at least $200kn^{1+1/k}$ edges, then there exists an $O(n \log n)$ time algorithm that finds a cycle of length at most $2k$ with high probability.*

Using these methods, we provide an algorithm for any odd additive approximation. In the following theorem we generalize the algorithm of Roditty and Vassilevska W. [15] to provide an additive $+(2r+1)$ approximation to the girth for any integer r.

▶ **Theorem 17.** *Let $G(V,E)$ be an unweighted, undirected graph with $|V| = n, |E| = m$. Let r be an integer and denote the girth of G by g. There is an $\tilde{O}\left(n + \frac{n^{2+r}}{m^r}\right)$ time algorithm that returns with high probability a cycle of length $\hat{g}$ that satisfies $g \leq \hat{g} \leq g + 2r + 1$.*

Proof. First we consider the case where $m < 200Ln^{1+1/L}$ for $L = \left\lceil \frac{\log n}{\log \log n} \right\rceil$. This implies that $m \leq \tilde{O}(n)$. We can use the $O(n^2)$ time algorithm of Itai and Rodeh to obtain an additive $+1$ approximation of the girth of G. Notably, in this case $O(n^2) \leq \tilde{O}\left(\frac{n^{2+r}}{m^r}\right)$.

Suppose now that $m \geq 200Ln^{1+1/L}$. Then there exists an integer $k \leq L$ such that

$$200(k+1)n^{1+\frac{1}{1+k}} < m \leq 200kn^{1+\frac{1}{k}}.$$

It follows from Lemma 16 that in $O(n \log n)$ time we can find a cycle of length at most $2k+2$ with high probability. If $g > 2k - 2r$, this cycle is an additive $+(2r+1)$ approximation of the girth.

We are left to handle the case when $g \leq 2k - 2r$. For any non-negative integer p, denote by $T_p(v) := \{u \in V : d(u,v) \leq p\}$ the vertices in the graph of distance $\leq p$ from v. Let Δ be a degree parameter; we will refer to vertices with $|T_r(v)| \leq \Delta$ as low r-degree vertices and vertices with $|T_r(v)| > \Delta$ as high r-degree vertices.

We sample a set S of $O\left(\frac{n}{\Delta} \log n\right)$ vertices uniformly at random. With high probability, any high degree vertex v satisfies $T_r(v) \cap S \neq \emptyset$. We now run BFS-cycle from each vertex of S. If the shortest cycle in the graph contains some node v such that $T_r(v)$ intersects S, Lemma 14 implies that the shortest cycle the algorithm finds is of length at most $g + 2r + 1$. The running time of this is $O\left(\frac{n^2 \log n}{\Delta}\right)$.

Now we only need to handle the case where the shortest cycle contains only vertices v for which $T_r(v) \cap S = \emptyset$, or equivalently $v \notin T_r(s)\ \forall s \in S$. To do so, we remove from the graph all vertices in $T_r(s)$ for any $s \in S$. With high probability the remaining vertices are all of low r-degree.

For each remaining vertex v, consider performing BFS from v up to ℓ levels for some $k - r \leq \ell < k$, while keeping track of the cumulative size of the layers. We break at the first $\ell \geq k - r$ for which $|T_\ell(v)| \leq \Delta^{(k-r)/r} |T_{\ell-(k-r)}(v)|$. Since we are assuming $g \leq 2k - 2r$, if there is a vertex $u \in T_{\ell-(k-r)}(v)$ which is part of the shortest cycle, then the BFS from v up to ℓ levels must have found a $+2(\ell - (k-r)) + 1$ approximation of this. In Lemma 18, we justify why we break before $\ell = k$. Then, we discard all vertices in $T_{\ell-(k-r)}(v)$ from further consideration. We do this for all v, and the total time complexity is $O\left(n\Delta^{(k-r)/r}\right)$.

The proof that this algorithm is valid is deferred to Lemma 18. So if the shortest cycle in G is comprised of only low r-degree vertices, we will have found a $+2r+1$ approximation of it. The full algorithm for this is described in Algorithm 1.

Algorithm 1 All Low BFS-Cycle.

```
for s ∈ V do
    L_0 ← {s}
    S_0 ← 1
    for 0 ≤ i < k do
        for u ∈ L_i do
            for (u, v) ∈ E do
                if d_v ≠ 0 then
                    Cycle ← cycle formed by backtracking ancestors until LCA of u, v
                    return Cycle
                end if
                L_{i+1} ← L_{i+1} ∪ {v}
            end for
        end for
        S_{i+1} ← S_i + |L_{i+1}|
        if i ≥ k − r and S_i ≤ ΔS_{i−k+r} then
            for 0 ≤ j ≤ i − k + r do
                for u ∈ L_j do
                    for (u, v) ∈ E do
                        E ← E − {(u, v)}
                    end for
                end for
            end for
        end if
    end for
end for
```

To minimize the runtime, we set $\Delta = n^{r/k}$ and obtain a running time of $\tilde{O}\left(n^{2-r/k}\right)$. Since $m = \Theta\left(n^{1+1/k}\right)$,

$$\tilde{O}\left(n^{2-r/k}\right) = \tilde{O}\left(\frac{n^{2+r}}{n^{(1+1/k)\cdot r}}\right) = \tilde{O}\left(\frac{n^{2+r}}{m^r}\right).$$

This gives us the desired runtime. ◀

▶ **Lemma 18.** *The time complexity of Algorithm 1 is $O\left(n\Delta^{(k-r)/r}\right)$ and it is guaranteed to return a cycle of length $\leq 2k$ if $g \leq 2k - 2r$ and if there is a shortest cycle where all vertices have low r-degree.*

Proof. Firstly, we show that for any low r-degree vertex v, we will find an $\ell \geq k - r$ such that $|T_\ell(v)| \leq \Delta^{(k-r)/r}|T_{\ell-(k-r)}|$ and $\ell < k$. Assume the contrary, that $|T_\ell(v)| > \Delta^{(k-r)/r}|T_{\ell-(k-r)}|$ for all $k - r \leq \ell < k$. Multiplying these inequalities for all ℓ in this range gives

$$\Delta^{k-r}\prod_{j=0}^{r-1}|T_j(v)| < \prod_{i=k-r}^{k-1}|T_i(v)|.$$

For any non-negative integer t, denote $\bar{t}$ to be the unique integer such that $t \equiv \bar{t} \pmod r$ and $0 \leq \bar{t} < r$. Since all vertices w in this subgraph have the property that $|T_r(w)| \leq \Delta$, for any non-negative integer i and any vertex v we have

$$|T_{\bar{i}}(v)|\Delta^{(i-\bar{i})/r} \geq |T_i(v)|.$$

Now note that $\{\overline{k-r}, \overline{k-r+1}, \ldots, \overline{k-1}\}$ is exactly $\{0, 1, 2, \ldots, r-1\}$ as these are r consecutive integers. Thus we can bound the right-hand side of the inequality like so:

$$\begin{aligned}
\Delta^{k-r} \prod_{j=0}^{r-1} |T_j(v)| &< \prod_{i=k-r}^{k-1} |T_i(v)| \\
&\leq \prod_{i=k-r}^{k-1} \Delta^{(i-\overline{i})/r} |T_{\overline{i}}(v)| \\
&\leq \Delta^{\frac{\left(\sum_{i=k-r}^{k-1} i\right) - \left(\sum_{j=0}^{r-1} j\right)}{r}} \prod_{j=0}^{r-1} |T_j(v)| \\
&\leq \Delta^{\frac{\sum_{j=0}^{r-1} (k-r+j)-j}{r}} \prod_{j=0}^{r-1} |T_j(v)| \\
&= \Delta^{k-r} \prod_{j=0}^{r-1} |T_j(v)|.
\end{aligned}$$

So we reach a contradiction as desired and we conclude that there is some $\ell < k$ and so $\ell - (k-r) < r$. If $T_{\ell-(k-r)}(v)$ contains a vertex in the shortest cycle, we indeed obtain a $+(2r-1)$ approximation. Furthermore, we now no longer need to BFS from any vertex in $T_{\ell-(k-r)}(v)$ so we can discard all of these. So although we had to do $O(|T_\ell(v)|)$ work for BFSing from this v, we were able to discard $|T_{\ell-(k-r)}(v)|$ vertices. Thus, the amortized time complexity per vertex is $O(\Delta^{(k-r)/r})$ and so the total time complexity is $O(n\Delta^{(k-r)/r})$.

Note that we only guarantee returning a cycle of length $\leq 2k$ rather than of length $\leq g + 2r + 1$ as it is possible we find a longer cycle first, in which case we must terminate immediately to guarantee that the BFS-Cycle subroutine step is still linear only in terms of the number of vertices and not edges. This is fine as we now have a shorter cycle and we can repeat the algorithm with smaller k. To make this efficient, we can binary search, incurring only an additional $\log k$ factor. ◀

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.

3 John A. Bondy and Miklós Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory*, pages 16:97–16:105, 1974.

4 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM J. Comput.*, 48(2):481–512, 2019.

5 Shucheng Chi, Ran Duan, and Tianle Xie. Faster algorithms for bounded-difference min-plus product. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1435–1447, 2022.

6 Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *J. Algorithms*, 38(2):335–353, 2001.

7 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.

8 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 75:1–75:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

9 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

10 Avi Kadria, Liam Roditty, Aaron Sidford, Virginia Vassilevska Williams, and Uri Zwick. Algorithmic trade-offs for girth approximation in undirected graphs. In *Proc. SODA 2022*, pages 1471–1492, 2022.

11 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, Philadelphia, PA, 2018.

12 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018.

13 Andrzej Lingas and Eva-Marta Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Inf. Process. Lett.*, 109(10):493–498, 2009.

14 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.

15 Liam Roditty and Virginia Vassilevska Williams. Subquadratic time approximation algorithms for the girth. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 833–845. SIAM, 2012.

16 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

17 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 605–615. IEEE Computer Society, 1999.

18 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

19 Virginia Vassilevska Williams. On Some Fine-Grained Questions in Algorithms and Complexity. In *Proceedings of the International Congress of Mathematicians ICM 2018*, volume 3, pages 3447–3487, 2019.

20 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

21 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 12–29. SIAM, 2020.

22 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018.

23 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discret. Math.*, 10(2):209–222, 1997.

A Fast $(\min, +)$ Product between Column and Row Bounded-Difference Matrices

In this section, we adapt the method of Chi, Duan and Xie [5] for bounded difference min-plus matrix product for our case, thus proving Lemma 7. The main difference in our method is that we can no longer divide matrices into square blocks since differences are only bounded

in one direction. Instead, we divide them into rectangular blocks. Most terminologies and analysis in their work can be adapted to our case, so we only sketch the needed modifications here.

Suppose the matrices to multiply are A, B_0 with size $n \times m$ and $m \times n$. For simplicity, we transpose B_0 to be $B = B_0^T$. Our assumption is that for some constant Δ, $|A_{i,j} - A_{i+1,j}| \leq \Delta$, $|B_{i,j} - B_{i+1,j}| \leq \Delta$ for valid indexes. We want to compute $C_{i,j} = \min_k\{A_{i,k} + B_{j,k}\}$.

We divide the rows of A and B into blocks of size α. The main difference from [5] is that we can no longer divide columns into blocks. For each pair of blocks (one block of columns in A and one block of columns in B), we pick any i and j from each block and compute $C_{i,j}$. This step takes $O(n^2m/\alpha^2)$ time.

By locality, we know for any (i', j') in these two blocks, $|C_{i',j'} - C_{i,j}| \leq 2\alpha\Delta$ (Since $C_{i',j'}$ equals to $A_{i',k} + B_{j',k}$ for some k, and $|A_{i',k} + B_{j',k} - A_{i,k} - B_{j,k}| \leq 2\alpha\Delta$ for all k's by the property of bounded-difference within columns). We call k so that $|A_{i,k} + B_{j,k} - C_{i,j}| \leq 4\alpha\Delta$ candidates, and only these k's could contribute to (i', j')'s in these two blocks.

For block pairs with no more than β candidates, we simply enumerate through these k's for every (i', j'), taking $O(n^2\beta)$ time.

For block pairs with more than β candidates, we use the method in Section 2, [5]. Sample a set of columns S with size $\Omega(m \log n/\beta)$, then reduce via these columns, mapping resulting segments to γ columns. Computing the bounded min-plus matrix product after mapping would take time $\tilde{O}(M(n, \gamma, n)\alpha m/\beta)$ where $M(n, \gamma, n)$ is the complexity of multiplying $n \times \gamma$ and $\gamma \times n$ matrices. For each of the n^2/α^2 blocks in C, each of the m^2 column pairs has probability $1/\gamma$ to collide (mapped to the same column), and subtracting each collision takes $O(\alpha^2)$ time. Thus, subtracting the contribution of all collisions would take $O(n^2m^2/\gamma)$ time.

The total time complexity is $\tilde{O}(n^2m/\alpha^2 + n^2\beta + M(n, \gamma, n)\alpha m/\beta + n^2m^2/\gamma)$.

Particularly when $m = n$, let $\alpha = n^{0.094513}$, $\beta = n^{0.810974}$, $\gamma = n^{1.189026}$, $M(n, \gamma, n) = O(n^{2.527435})$ [11], we can get the complexity of $O(n^{2.811})$.

One-Pass Additive-Error Subset Selection for ℓ_p Subspace Approximation

Amit Deshpande
Microsoft Research, Bengaluru, India

Rameshwar Pratap
Indian Institute of Technology, Mandi, H.P., India

Abstract

We consider the problem of subset selection for ℓ_p subspace approximation, that is, to efficiently find a *small* subset of data points such that solving the problem optimally for this subset gives a good approximation to solving the problem optimally for the original input. Previously known subset selection algorithms based on volume sampling and adaptive sampling [16], for the general case of $p \in [1, \infty)$, require multiple passes over the data. In this paper, we give a one-pass subset selection with an additive approximation guarantee for ℓ_p subspace approximation, for any $p \in [1, \infty)$. Earlier subset selection algorithms that give a one-pass multiplicative $(1 + \epsilon)$ approximation work under the special cases. Cohen et al. [11] gives a one-pass subset section that offers multiplicative $(1 + \epsilon)$ approximation guarantee for the special case of ℓ_2 subspace approximation. Mahabadi et al. [31] gives a one-pass *noisy* subset selection with $(1+\epsilon)$ approximation guarantee for ℓ_p subspace approximation when $p \in \{1, 2\}$. Our subset selection algorithm gives a weaker, additive approximation guarantee, but it works for any $p \in [1, \infty)$.

2012 ACM Subject Classification Theory of computation → Streaming models; Mathematics of computing → Dimensionality reduction; Computing methodologies → Dimensionality reduction and manifold learning; Theory of computation → Sketching and sampling

Keywords and phrases Subspace approximation, streaming algorithms, low-rank approximation, adaptive sampling, volume sampling, subset selection

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.51

Category Track A: Algorithms, Complexity and Games

Acknowledgements We would like to thank anonymous reviewers for their careful reading of our manuscript and their insightful comments and suggestions.

1 Introduction

In *subset selection* problems, the objective is to pick a small subset of the given data such that solving a problem optimally on this subset gives a good approximation to solving it optimally on the entire data. Many coreset constructions in computational geometry and clustering [22], sampling-based algorithms for large matrices [24], algorithms for submodular optimization and active learning [37] essentially perform subset selection. The main advantage of subset selection lies in its interpretability, for example, in gene expression analysis, we would like to find a representative subset of genes from gene expression data rather than just fitting a subspace to the data [20, 33, 36, 32, 29]. In several machine learning applications such as document classification, face recognition etc., it is desirable to go beyond dimension reduction alone, and pick a subset of representative items or features [28, 33]. Subset selection has been well studied for many fundamental problems such as k-means clustering [2, 14], low-rank approximation [24, 17, 15, 28] and regression [13], to name a few. In low-rank and subspace approximation, the subset selection approach leads to more interpretable solutions than using SVD or random projections-based results. Therefore, subset selection has been a separate and well-studied problem even within the low-rank approximation and subspace approximation literature [28, 12].

In the following, we formally state the ℓ_p subspace approximation problem for $p \in [1, \infty)$.

ℓ_p subspace approximation: In this problem, given a dataset $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ of n points in $\mathbb{R}^d$, a positive integer $1 \leq k \leq d$ and a real number $p \in [1, \infty)$, the objective is to find a linear subspace V in $\mathbb{R}^d$ of dimension at most k that minimizes the sum of p-th powers of the Euclidean distances of all the points to the subspace V, that is, to minimize

$$\mathrm{err}_p(\mathcal{X}, V) := \sum_{i=1}^{n} d(x_i, V)^p. \tag{1}$$

Throughout this paper, we use V^* to denote the optimal subspace for ℓ_p subspace approximation. The optimal solutions are different for different values of p but we do not include that in the notation to keep the presentation simple, as our results hold for any $p \in [1, \infty)$.

Before stating our results, we first explain what a *small* subset and a *good* approximation means in the context of subset selection for ℓ_p subspace approximation.

For ℓ_p subspace approximation, we consider n and d to be large, $k \ll n, d$, and p to be a small constant. Thus, a *small* subset of $\mathcal{X}$ desired in subset selection has size independent of n and d, and is bounded by $\mathrm{poly}(k/\epsilon)$, where ϵ is a parameter that controls the approximation guarantee (as explained later). Note that the trivial solution $V = 0$ gives $\mathrm{err}_p(\mathcal{X}, V) = \sum_{i=1}^{n} \|x_i\|^p$. Using the standard terminology from previous work [24, 15, 16], an additive approximation guarantee means outputting V such that $\mathrm{err}_p(\mathcal{X}, V) \leq \mathrm{err}_p(\mathcal{X}, V^*) + \epsilon \sum_{i=1}^{n} \|x_i\|^p$, whereas a multiplicative approximation guarantee means $\mathrm{err}_p(\mathcal{X}, V) \leq (1 + \epsilon)\ \mathrm{err}_p(\mathcal{X}, V^*)$. Most subset selection algorithms for ℓ_p subspace approximation select a $\mathrm{poly}(k/\epsilon)$-sized subset of $\mathcal{X}$ such that its span contains a subspace V of dimension at most k that is close enough to V^* to obtain the above approximation guarantees.

Our objective in this paper is to propose an efficient, one-pass sampling algorithm that performs subset selection for ℓ_p subspace approximation for $p \in [1, \infty)$ defined as above. We note that the problem of one-pass subset selection for ℓ_p subspace approximation has been studied for special values of p, for example, Cohen et al. [11] gives one-pass subset selection for $p = 2$, Mahabadi et al. [31] suggest one-pass *noisy* subset selection for $p = \{1, 2\}$. To the best of our knowledge this problem has not been studied in generality for $p \in [1, \infty)$. In this work, we consider studying this problem. We state our results as follows.

1.1 Our results

Our main technical contribution is a one-pass MCMC-based sampling algorithm that can approximately simulate multiple rounds of adaptive sampling. As a direct application of the above, we get the following results for the ℓ_p subspace approximation problem: For $p \in [1, \infty)$, our algorithm makes only one pass over the given data and outputs a subset of $\mathrm{poly}(k/\epsilon)^p$ points whose span contains a k dimensional subspace with an additive approximation guarantee for ℓ_p subspace approximation. This generalizes the well-known squared-length sampling algorithm of Frieze et al. [24] that gives additive approximation guarantee for ℓ_2 subspace approximation (or low-rank approximation under the Frobenium norm). Even though stronger multiplicative $(1+\epsilon)$ approximation algorithms for ℓ_p subspace approximation are known in the previous work, either they cannot do subset selection, or they are not one-pass, or they do not work for all $p \in [1, \infty)$.

Organization of the paper. In Section 2, we compare and contrast our result with the state-of-the-art algorithms, and explain the key technical challenges, and workarounds. In Section 3, we state our MCMC based subset selection algorithm for subset selection for ℓ_p

subspace approximation. In Section 4, we give theoretical bounds on the sample size and approximation guarantee. Finally, in Section 5, we conclude our discussion and state some potential open questions of the paper.

2 Related work

In this section, we discuss related work on sampling and sketching algorithms for ℓ_p subspace approximation, and do a thorough comparison of our results with the state of the art.

2.1 Sampling-based ℓ_p subspace approximation

Frieze et al. [24] show that selecting a subset of $O(k/\epsilon)$ data points as an *i.i.d.* sample from $x_1, x_2, \ldots, x_n$ picked by squared-length sampling, i.e., x_i is picked with probability proportional to $\|x_i\|_2^2$, gives an additive approximation for ℓ_2 subspace approximation (also known as low-rank approximation under the Frobenius norm). Squared-length sampling can be implemented in one pass over $\mathcal{X}$ using reservoir sampling [35, 21]. It is known how to improve the additive approximation guarantee to a multiplicative approximation by combining two generalizations of squared-length sampling, namely, adaptive sampling and volume sampling [15, 16] but it requires $O(k \log k)$ passes over the data. In adaptive sampling, we pick points with probability proportional to the distance from the span of previously picked points, and in volume sampling, we pick a subset of points with probability proportional to the squared volume of the parallelepiped formed by them. Volume sampling a subset of size k can itself be simulated with an approximation factor $k!$ in k rounds of adaptive sampling [15]. For $p = 2$, it is also known that picking a subset of $O(k/\epsilon)$ points by volume sampling gives a bi-criteria $(1+\epsilon)$ approximation for ℓ_2 subspace approximation [28]. For general $p \in [1, \infty)$, it is known that subset selection based on adaptive sampling and volume sampling can be generalized to get a $(1+\epsilon)$ multiplicative approximation for ℓ_p subspace approximation, for any $p \in [1, \infty)$, where the subset is of size $O\left((k/\epsilon)^p\right)$ and it is picked in $O(k \log k)$ passes over the data [16]. The main bottleneck for implementing this in one pass is the inability to simulate multiple rounds of adaptive sampling in a single pass.

The only known workarounds to get one-pass subset selection for ℓ_p subspace approximation are known for the special cases $p = 1$ and $p = 2$. Cohen et al. [11] give a one-pass subset selection algorithm with a multiplicative $(1+\epsilon)$ approximation guarantee for ℓ_2 subspace approximation based on ridge leverage score sampling. Their one-pass implementation crucially uses deterministic matrix sketching [25] to approximate the SVD and ridge leverage scores, and works only for $p = 2$, to the best of our knowledge. Braverman et al. [6] give online algorithms for ℓ_2 subspace approximation (or low-rank approximation) via subset selection but their subset size $O(\frac{k}{\epsilon} \log n \log \kappa)$ is not independent on n and depends logarithmically on the number of points n and the condition number κ. Recent work by Mahabadi et al. [31] gives a one-pass algorithm with a multiplicative $(1+\epsilon)$ approximation guarantee for ℓ_p subspace approximation. However, their algorithm works only in the special cases $p \in \{1, 2\}$ and it outputs a subset of noisy data points instead of the actual data points.

A different objective for ℓ_p subspace approximation has also been studied in literature [5, 9], namely, minimizing the entry-wise ℓ_p-norm low-rank approximation error. To state it formally, given an input matrix $A \in \mathbb{R}^{n \times d}$ and a real number $p \in [0, \infty)$, their objective is to find a matrix B of rank at most k that minimizes $\sum_{i,j} |A_{i,j} - B_{i,j}|^p$.

2.2 Sketching-based ℓ_p subspace approximation

Sketching-based algorithms compute a sketch of a given data in a single pass, using which one can compute an approximately optimal solution to a given problem on the original data. The problem of ℓ_p subspace approximation has been well-studied in previous work on sketching algorithms. However, a limitation of these results is that they do not directly perform subset selection. We mention a few notable results as follows: For $p = 2$, extending deterministic matrix sketching of Liberty [30], Ghashami et al. [27, 26] give a deterministic one-pass sketching algorithm that gives a multiplicative $(1 + \epsilon)$ approximation guarantee for ℓ_2 subspace approximation (or low-rank approximation under the Frobenius norm). Cormode et al. [19] extend the above deterministic sketching idea to $p \neq 2$ and give a poly(k) approximation for entry-wise ℓ_1-norm low-rank approximation and an additive $\epsilon \ \|b\|_\infty$ approximation for ℓ_∞ regression. There is another line of work based on sketching algorithms using random projection. Random projection gives a multiplicative $(1+\epsilon)$ approximation for ℓ_2 subspace approximation in running time $O(\text{nnz}(X) \cdot \text{poly}(k/\epsilon))$, subsequently improved to a running time of $O(\text{nnz}(X) + (n + d) \cdot \text{poly}(k/\epsilon))$ by Clarkson and Woodruff [10]. Feldman et al. [23] also give a one-pass algorithm for multiplicative $(1 + \epsilon)$ approximation for ℓ_p subspace approximation, for $p \in [1, 2]$. However, these results do not provide a one-pass subset selection.

2.3 Comparison with other MCMC-based sampling results

Theorem 4 of Anari et al. [1] gives a MCMC based sampling algorithm to approximate volume sampling distribution. However, their algorithm requires a greedy algorithm to pick the initial subset that requires k passes over the input.

The MCMC sampling has also been explored in the context of k-means clustering. The D^2-sampling proposed by Arthur and Vassilvitskii [2] adaptively samples k points – one point in each passes over the input, and the sampled points give $O(\log k)$ approximation to the optimal clustering solution. The results due to [4, 3] suggest generating MCMC sampling distribution by taking only one pass over the input that closely approximates the underlying D^2 sampling distribution, and offer close to the optimal clustering solution. Building on these MCMC based sampling techniques, Pratap et al. [34] gives one pass subset section for *spherical* k-means clustering [18].

3 MCMC sampling algorithm

In this section, we state our MCMC based sampling algorithm for subset selection for ℓ_p subspace approximation. We first recall the adaptive sampling algorithm[15, 16] for ℓ_p subspace approximation.

Adaptive sampling [15, 16] *w.r.t.* a subset $S \subseteq \mathcal{X}$ is defined as picking points from $\mathcal{X}$ such that the probability of picking any point $x \in \mathcal{X}$ is proportional to $d(x, \text{span}\,(S))^p$. We denote this probability by

$$p_S(x) = \frac{d(x, \text{span}\,(S))^p}{\text{err}_p(\mathcal{X}, S)}, \quad \text{for } x \in \mathcal{X}. \tag{2}$$

For any subset S whose $\text{err}_p(\mathcal{X}, S)$ is not too small, we show that adaptive sampling *w.r.t.* S can be approximately simulated by an MCMC sampling algorithm that only has access to *i.i.d.* samples of points $x \in \mathcal{X}$ picked from the following easier distribution:

$$q(x) = \frac{d(x, \text{span}\left(\tilde{S}\right))^p}{2\ \text{err}_p(\mathcal{X}, \tilde{S})} + \frac{1}{2\,|\mathcal{X}|}, \tag{3}$$

for some initial subset $\tilde{S}$. We give the above definition of $q(x)$ using an arbitrary initial or *pivot* subset $\tilde{S}$ because it will be useful in our analysis of multiple rounds of adaptive sampling. However, our final algorithm uses a fixed subset $\tilde{S} = \emptyset$ such that

$$q(x) = \frac{\|x\|_2^p}{2\sum_{x\in\mathcal{X}} \|x\|_2^p} + \frac{1}{2\,|\mathcal{X}|}. \tag{4}$$

Note that sampling from this easier distribution, namely, picking $x \in \mathcal{X}$ with probability $q(x)$ (mentioned in Equation (4)), can be done in only one pass over $\mathcal{X}$ using weighted reservoir sampling [8]. Weighted reservoir sampling keeps a reservoir of finite items, and for every new item, calculates its relative weight to randomly decide if the item should be added to the reservoir. If the new item is selected, then one of the existing items from the reservoir is picked uniformly and replaced with the new item. Further, given any non-negative weights w_x, for each point $x \in \mathcal{X}$, weighted reservoir sampling can pick an *i.i.d.* sample of points, where x is picked with probability proportional to its weight w_x. Note that this does not require the knowledge of $\sum_{x\in\mathcal{X}} w_x$. Thus, we can run two reservoir sampling algorithms in parallel to maintain two samples, one that picks points with probability proportional to $||x||_2^p$, and another that picks points with uniform probability. Our actual sampling with probability proportional $q(x) = \frac{\|x\|_2^p}{2\sum_{x\in\mathcal{X}}\|x\|_2^p} + \frac{1}{2|\mathcal{X}|}$ picks from one of the two reservoirs with probability $1/2$ each. Therefore, our MCMC algorithm uses a single pass of $\mathcal{X}$ to pick a small sample of *i.i.d.* random points from the probability distribution $q(\cdot)$, in advance. Note that $q(\cdot)$ is an easier and fixed distribution compared to $p_S(\cdot)$. The latter one depends on S and could change over multiple rounds of adaptive sampling.

Let $x \in \mathcal{X}$ be a random point sampled with probability $q(x)$. Consider a random walk whose single step is defined as follows: sample another point $y \in \mathcal{X}$ independently with probability $q(y)$ and sample a real number r *u.a.r.* from the interval $(0, 1)$, and if

$$\frac{d(y, \text{span}\,(S))^p\ q(x)}{d(x, \text{span}\,(S))^p\ q(y)} = \frac{p_S(y)\ q(x)}{p_S(x)\ q(y)} > r,$$

then move from x to y, else, stay at x. Essentially, this does rejection sampling using a simpler distribution $q(\cdot)$. Observe that the stationary distribution of the above random walk is the adaptive sampling distribution $p_S(\cdot)$. We use $\tilde{P}_m^{(1)}(\cdot \mid S)$ to denote the resulting distribution on $\mathcal{X}$ after m steps of the above random walk. Note that m steps of the above random walk can be simulated by sampling m *i.i.d.* points from the distribution $q(\cdot)$ in advance, and representing them implicitly as m-dimensional points.

Lemma 1 below shows that for any subsets $\tilde{S} \subseteq S \subseteq \mathcal{X}$ (where $\tilde{S}$ is the initial subset, and S is the current subset), either $\text{err}_p(\mathcal{X}, S)$ is small compared to $\text{err}_p(\mathcal{X}, \tilde{S})$, or our MCMC sampling distribution closely approximates the adaptive sampling distribution $p_S(\cdot)$ in total variation distance. Proof of Lemma 1 relies on Corollary 1 of Cai [7] that gives an upper bound on the TV distance between these two distributions in terms of: 1) length of the Markov chain, and 2) upper bound on the ratio between these two distributions for any input point.

▶ **Lemma 1.** *Let $\epsilon_1, \epsilon_2 \in (0, 1)$ and $\tilde{S} \subseteq S \subseteq \mathcal{X}$. Let $P^{(1)}(\cdot \mid S)$ denote the distribution over an i.i.d. sample of t points picked from adaptive sampling w.r.t. S, and let $\tilde{P}_m^{(1)}(\cdot \mid \tilde{S})$ denote the distribution over t points picked by t independent random walks of length m each in our one-pass adaptive sampling algorithm; see step 3(a). Then for $m \geq 1 + \frac{2}{\epsilon_1}\log\frac{1}{\epsilon_2}$, either $\text{err}_p(\mathcal{X}, S) \leq \epsilon_1\ \text{err}_p(\mathcal{X}, \tilde{S})$ or $\left\|P^{(1)}(\cdot \mid S) - \tilde{P}_m^{(1)}(\cdot \mid S)\right\|_{TV} \leq \epsilon_2 t$.*

One-pass (approximate MCMC) adaptive sampling algorithm:
Input: a discrete subset $\mathcal{X} \subseteq \mathbb{R}^d$ and integer parameters $t, l, m \in \mathbf{Z}_{\geq 0}$.
Ouput: a subset $S \subseteq \mathcal{X}$.

1. Pick an *i.i.d.* sample $\mathcal{Y}$ of size $|\mathcal{Y}| = ltm$ from $\mathcal{X}$, without replacement, where the probability of picking $x \in \mathcal{X}$ is

$$q(x) = \frac{d(x, \operatorname{span}(\tilde{S}))^p}{2 \operatorname{err}_p(\mathcal{X}, \tilde{S})} + \frac{1}{2|\mathcal{X}|}.$$

   We use the *pivot* subset $\tilde{S} = \emptyset$ so the corresponding distribution is

$$q(x) = \frac{1}{2} \frac{\|x\|_2^p}{\sum_{x \in \mathcal{X}} \|x\|_2^p} + \frac{1}{2|\mathcal{X}|}.$$

   %% This can be implemented in one pass over $\mathcal{X}$ using weighted reservoir sampling [8]. Weighted reservoir sampling is a weighted version of the classical reservoir sampling where the probability of inclusion of an item in the sample is proportional to the weight associated with the item.
2. Initialize $S \leftarrow \emptyset$.
3. For $i = 1, 2, \ldots, l$ do:
   a. Pick an *i.i.d.* sample A_i of size $|A_i| = t$ from $\mathcal{X}$ as follows. Each point in A_i is sampled by taking m steps of the following random walk starting from a point x picked with probability $q(x)$. In each step of the random walk, we pick another point y from $\mathcal{X}$ with probability $q(y)$ and pick a real number r uniformly at random from the interval $(0, 1)$. If $\frac{d(y, \operatorname{span}(S))^p \; q(x)}{d(x, \operatorname{span}(S))^p \; q(y)} > r$ then move to y, else, stay at the current point.
   %% Note that we add only the final point obtained after the m-step random walk in the subset S.
   %% We note that the steps 1-3 of the algorithm can be simulated by taking only one pass over the input as discussed below. Suppose we have a single-pass Algorithm A for sampling from a particular distribution, we can design another Algorithm B that runs in parallel to Algorithm A and post-processes its sample. In our setting, once we know how to get an *i.i.d.* sample of points, where point x is picked with probability $q(x)$, we can run another parallel thread that simulates a random walk whose each step requires a point picked with probability $q(x)$ and performs Step 3.
   b. $S \leftarrow S \cup A_i$.
4. Output S.

Proof. First, consider the $l = 1, t = 1$ case of the one-pass adaptive sampling algorithm described above. In this case, the procedure outputs only one element of $\mathcal{X}$. This random element is picked by m steps of the following random walk starting from an x picked with probability $q(x)$. In each step, we pick another point y with probability $q(y)$ and sample a real

One-pass MCMC ℓ_p subspace approximation algorithm:
Input: a discrete subset $\mathcal{X} \subseteq \mathbb{R}^d$, an integer parameter $k \in \mathbf{Z}_{\geq 0}$ and an error parameter $\delta \in \mathbb{R}_{\geq 0}$.
Output: a subset $\mathcal{S} \subseteq \mathcal{X}$ of $\tilde{O}\left((k/\epsilon)^{p+1}\right)$ points.
1. Repeat the following $O(k \log \frac{1}{\epsilon})$ times in parallel and pick the best sample, $\mathcal{S}$ that minimizes $\sum_{x \in \mathcal{X}} d(x, \text{span}\,(\mathcal{S}))^p$.
   a. Call **One-pass (approximate MCMC) adaptive sampling algorithm** with $t = \tilde{O}((k/\epsilon)^{p+1})$, $l = k$ and $m = 1 + \frac{2}{\epsilon_1} \log \frac{1}{\epsilon_2}$.
2. Output $\mathcal{S}$.

number r *u.a.r.* from the interval $(0, 1)$, and if $p_S(y)q(x)/p_S(x)q(y) > r$, then we move from x to y, else, we stay at x. Observe that the stationary distribution of the above random walk is the adaptive sampling distribution *w.r.t.* S given by $p_S(x) = d(x, \text{span}\,(S))^p / \text{err}_p(\mathcal{X}, S)$. Using Corollary 1 of [7], the total variation distance after m steps of the random walk is bounded by

$$\left(1 - \frac{1}{\gamma}\right)^{m-1} \leq e^{-(m-1)/\gamma} \leq \epsilon_2, \text{ where } \gamma = \max_{x \in \mathcal{X}} \frac{p_S(x)}{q(x)}.$$

The above bound is at most ϵ_2 if we choose to run the random walk for $m \geq 1 + \gamma \log \frac{1}{\epsilon_2}$ steps. Now suppose $\text{err}_p(\mathcal{X}, S) > \epsilon_1 \text{ err}_p(\mathcal{X}, \tilde{S})$. Then, for any $x \in \mathcal{X}$

$$\begin{aligned}\frac{p_S(x)}{q(x)} &= \frac{\frac{d(x, \text{span}\,(S))^p}{\text{err}_p(\mathcal{X}, S)}}{\frac{1}{2}\frac{d(x, \text{span}\,(\tilde{S}))^p}{\text{err}_p(\mathcal{X}, \tilde{S})} + \frac{1}{2|\mathcal{X}|}} \\ &\leq \frac{2\, d(x, \text{span}\,(S))^p \text{ err}_p(\mathcal{X}, \tilde{S})}{d(x, \text{span}\,(\tilde{S}))^p \text{ err}_p(\mathcal{X}, S)} \leq \frac{2}{\epsilon_1},\end{aligned}$$

using $d(x, \text{span}\,(S))^p \leq d(x, \text{span}\,(\tilde{S}))^p$ because $\tilde{S} \subseteq S$, and the above assumption $\text{err}_p(\mathcal{X}, S) > \epsilon_1 \text{ err}_p(\mathcal{X}, \tilde{S})$. Therefore, $m > \frac{2}{\epsilon_1} \log \frac{1}{\epsilon_2}$ ensures that m steps of the random walk gives a distribution within total variation distance ϵ_2 from the adaptive sampling distribution for picking a single point.

Note that for $t > 1$ both the adaptive sampling and the MCMC sampling procedure pick an *i.i.d.* sample of t points, so the total variation distance is additive in t, which means

$$\left\| P^{(1)}(\cdot \mid S) - \tilde{P}_m^{(1)}(\cdot \mid S) \right\|_{TV} \leq \epsilon_2 t,$$

assuming $\text{err}_p(\mathcal{X}, S) > \epsilon_1 \text{ err}_p(\mathcal{X}, \tilde{S})$. This completes a proof of the lemma. ◀

4 ℓ_p subspace approximation

In this section, we give our result for one pass subset selection for ℓ_p subspace approximation. We first show (in Lemma 2) that the true adaptive sampling can be well approximated by one pass (approximate) MCMC based sampling algorithm. Building on this result, in Proposition 3 and Theorem 4, we show bounds on the number of steps taken by the Markov chain, and on the sample size that gives an additive approximation for the ℓ_p subspace approximation. Our MCMC-based sampling ensures that our problem statement's single-pass subset selection criteria are satisfied.

First, let's set up the notation required to analyze the true adaptive sampling as well as our one-pass (approximate MCMC) adaptive sampling algorithm. For any fixed subset $S \subseteq \mathcal{X}$, we define

$$\mathrm{err}_p(\mathcal{X}, S) = \sum_{x \in \mathcal{X}} d(x, \mathrm{span}\,(S))^p, \tag{5}$$

$$P^{(1)}(T|S) = \prod_{x \in T} \frac{d(x, \mathrm{span}\,(S))^p}{\mathrm{err}_p(\mathcal{X}, S)}, \tag{6}$$

$$\text{for any subset } T \text{ of size } t,$$

$$\mathop{\mathrm{E}}_{T}\left[\mathrm{err}_p(\mathcal{X}, S \cup T)\right] = \sum_{T\,:\,|T|=t} P^{(1)}(T \mid S)\ \mathrm{err}_p(\mathcal{X}, S \cup T). \tag{7}$$

Given a subset $S \subseteq \mathcal{X}$, $P^{(1)}(T \mid S)$ denotes the probability of picking a subset $T \subseteq \mathcal{X}$ of *i.i.d.* t points by adaptive sampling *w.r.t.* S. We use $P^{(l)}(T_{1:l}|S)$ to denote the probability of picking a subset $T_{1:l} = B_1 \cup B_2 \cup \ldots \cup B_l \subseteq \mathcal{X}$ of tl points by l iterative rounds of adaptive sampling, where in the first round we sample a subset B_1 consisting of *i.i.d.* t points *w.r.t.* S, in the second round we sample a subset B_2 consisting of *i.i.d.* t points *w.r.t.* $S \cup B_1$, and so on to pick $T_{1:l} = B_1 \cup B_2 \cup \ldots \cup B_l$ over l iterations. Similarly, in the context of adaptive sampling, we use $T_{2:l}$ to denote $B_2 \cup \ldots \cup B_l$. We abuse the notation $\mathop{\mathrm{E}}_{T_{1:l} \mid S}[\cdot]$ to denote the expectation over $T_{1:l}$ picked in l iterative rounds of adaptive sampling starting from S.

Given a *pivot* subset $\tilde{S} \subseteq \mathcal{X}$ and another subset $S \subseteq \mathcal{X}$ such that $\tilde{S} \subseteq S$, consider the following MCMC sampling with parameters l, t, m that picks l subsets $A_1, A_2, \ldots, A_l$ of t points each, where m denotes the number of steps of a random walk used to pick these points. This sampling can be implemented in a single pass over $\mathcal{X}$, for any l, t, m, and any given subsets $\tilde{S} \subseteq S$. For $T_{1:l} = A_1 \cup A_2 \cup \ldots \cup A_l$. We use $\tilde{P}^{(l)}_m(T_{1:l} \mid S)$ to denote the probability of picking $T_{1:l}$ as the output of the following MCMC sampling procedure. Similarly, in the context of MCMC sampling, we use $T_{2:l}$ to denote $A_2 \cup \ldots \cup A_l$. We abuse the notation $\mathop{\tilde{\mathrm{E}}}_{T_{1:l} \mid S}[\cdot]$ to denote the expectation over $T_{1:l}$ picked using the MCMC sampling procedure starting from S with a pivot subset $\tilde{S} \subseteq S$.

We require the following additional notation in our analysis of the above MCMC sampling. We use $\tilde{P}^{(1)}_m(T \mid S)$ to denote the resulting distribution over subsets T of size t, when we use the above sampling procedure with $l = 1$. We define the following expressions:

$$\mathrm{ind}_p(\mathcal{X}, S) = \mathbb{1}\left(\mathrm{err}_p(\mathcal{X}, S) \leq \epsilon_1\ \mathrm{err}_p(\mathcal{X}, \tilde{S})\right), \tag{8}$$

$$\mathop{\tilde{\mathrm{E}}}_{T}\left[\mathrm{err}_p(\mathcal{X}, S \cup T)\right] = \sum_{T\,:\,|T|=t} \tilde{P}^{(1)}_m(T \mid S)\ \mathrm{err}_p(\mathcal{X}, S \cup T), \tag{9}$$

$$\mathop{\tilde{\mathrm{E}}}_{T}\left[\mathrm{ind}_p(\mathcal{X}, S \cup T)\right] = \sum_{T\,:\,|T|=t} \tilde{P}^{(1)}_m(T \mid S)\ \mathrm{ind}_p(\mathcal{X}, S \cup T). \tag{10}$$

The expression $\mathrm{ind}_p(\mathcal{X}, S)$ (in Equation (8)) denotes an indicator random variable that takes value 1 if error *w.r.t.* subset S is smaller than ϵ_1 times error *w.r.t.* subset $\tilde{S}$, and 0 otherwise. The expression $\mathop{\tilde{\mathrm{E}}}_{T}\left[\mathrm{err}_p(\mathcal{X}, S \cup T)\right]$ (in Equation (9)) denotes the expected error over the subset T picked using the MCMC sampling procedure starting from the set S such that the initial subset $\tilde{S} \subseteq S$.

Now Lemma 2 analyzes the effect of starting with an initial subset S_0 and using the same S_0 as a pivot subset for doing the MCMC sampling for l subsequent iterations of adaptive sampling, where we pick t *i.i.d.* points in each iteration using t independent random walks

of m steps. Lemma 2 shows that the expected error for subspace approximation after doing the l iterations of adaptive sampling is not too far from the expected error for subspace approximation after replacing the l iterations with MCMC sampling.

▶ **Lemma 2.** *For any subset $S_0 \subseteq \mathcal{X}$, any $\epsilon_1, \epsilon_2 \in (0,1)$ and any positive integers t, l, m with $m \geq 1 + \frac{2}{\epsilon_1} \log \frac{1}{\epsilon_2}$,*

$$\tilde{\mathrm{E}}_{T_{1:l} \mid S_0} \left[\mathrm{err}_p(\mathcal{X}, S_0 \cup T_{1:l})\right] \leq \mathrm{E}_{T_{1:l} \mid S_0} \left[\mathrm{err}_p(\mathcal{X}, S_0 \cup T_{1:l})\right] + (\epsilon_1 + \epsilon_2 tl)\, \mathrm{err}_p(\mathcal{X}, S_0).$$

Proof. We show a slightly stronger inequality than the one given above, i.e., for any S_0 such that $\tilde{S} \subseteq S_0$,

$$\tilde{\mathrm{E}}_{T_{1:l} \mid S_0} \left[\mathrm{err}_p(\mathcal{X}, S_0 \cup T_{1:l})\right] \leq \mathrm{E}_{T_{1:l} \mid S_0} \left[\mathrm{err}_p(\mathcal{X}, S_0 \cup T_{1:l})\right] + \left(\epsilon_1 \tilde{\mathrm{E}}_{T_{1:l} \mid S_0} \left[\mathrm{ind}_p(\mathcal{X}, S_0 \cup T_{1:l})\right] + \epsilon_2 tl\right) \mathrm{err}_p(\mathcal{X}, \tilde{S}).$$

The special case $S_0 = \tilde{S}$ gives the lemma. We prove the above-mentioned stronger statement by induction on l. For $l = 0$, the above inequality holds trivially. Now assuming induction hypothesis, the above holds true for $l-1$ iterations (instead of l) starting with any subset $S_1 = S_0 \cup A \subseteq \mathcal{X}$ because $\tilde{S} \subseteq S_0 \subseteq S_1$.

$$\begin{aligned}
&\tilde{\mathrm{E}}_{T_{1:l} \mid S_0} \left[\mathrm{err}_p(\mathcal{X}, S_0 \cup T_{1:l})\right] \\
&= \tilde{\mathrm{E}}_{S_1 \mid S_0} \left[\tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right]\right] \\
&= \sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 1} \tilde{P}_m^{(1)}(S_1 \mid S_0) \tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right] \\
&\quad + \sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right]. \qquad (11)
\end{aligned}$$

If $\mathrm{ind}_p(\mathcal{X}, S_1) = 1$ then $\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l}) \leq \mathrm{err}_p(\mathcal{X}, S_1) \leq \epsilon_1\, \mathrm{err}_p(\mathcal{X}, S_0)$, so the first part of the above sum can be bounded as follows.

$$\begin{aligned}
&\sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 1} \tilde{P}_m^{(1)}(S_1 \mid S_0) \tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right] \\
&\leq \epsilon_1\, \mathrm{err}_p(\mathcal{X}, S_0) \cdot \sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 1} \tilde{P}_m^{(1)}(S_1 \mid S_0) \tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{ind}_p(\mathcal{X}, S_1 \cup T_{2:l})\right]. \qquad (12)
\end{aligned}$$

We give an upper bound on the second part as follows.

$$\begin{aligned}
&\sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right] \\
&= \sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right]. \\
&\leq \sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \cdot \\
&\quad \left(\mathrm{E}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right] + (\epsilon_1 \tilde{\mathrm{E}}_{T_{2:l} \mid S_1} \left[\mathrm{ind}_p(\mathcal{X}, S_1 \cup T_{2:l})\right] + \epsilon_2 t(l-1))\, \mathrm{err}_p(\mathcal{X}, \tilde{S})\right). \qquad (13) \\
&\qquad \text{(by applying the induction hypothesis to } (l-1) \text{ iterations starting from } S_1.) \\
&\leq \sum_{S_1 \,:\, \mathrm{ind}_p(\mathcal{X}, S_1) = 0} P^{(1)}(S_1 \mid S_0) \mathrm{E}_{T_{2:l} \mid S_1} \left[\mathrm{err}_p(\mathcal{X}, S_1 \cup T_{2:l})\right]
\end{aligned}$$

$$
\begin{aligned}
&+ \epsilon_1 \operatorname{err}_p(\mathcal{X}, \tilde{S}) \cdot \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \mathop{\tilde{\mathrm{E}}}_{T_{2:l} \mid S_1} [\operatorname{ind}_p(\mathcal{X}, S_1 \cup T_{2:l})] \\
&+ \epsilon_2 t(l-1) \operatorname{err}_p(\mathcal{X}, \tilde{S}) \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \\
&+ \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \left| \tilde{P}_m^{(1)}(S_1 \mid S_0) - P^{(1)}(S_1 \mid S_0) \right| \cdot \mathop{\mathrm{E}}_{T_{2:l} \mid S_1} [\operatorname{err}_p(\mathcal{X}, S_1 \cup T_{2:l})] .
\end{aligned}
$$

$$
\left(\text{by adding and subtracting the term } \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} P^{(1)}(S_1 \mid S_0) \mathop{\mathrm{E}}_{T_{2:l} \mid S_1} [\operatorname{err}_p(\mathcal{X}, S_1 \cup T_{2:l})] \text{ in Eq. (13).} \right)
$$

$$
\begin{aligned}
&\leq \sum_{S_1} P^{(1)}(S_1 \mid S_0) \mathop{\mathrm{E}}_{T_{2:l} \mid S_1} [\operatorname{err}_p(\mathcal{X}, S_1 \cup T_{2:l})] \\
&\quad + \epsilon_1 \operatorname{err}_p(\mathcal{X}, \tilde{S}) \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \cdot \mathop{\tilde{\mathrm{E}}}_{T_{2:l} \mid S_1} [\operatorname{ind}_p(\mathcal{X}, S_1 \cup T_{2:l})] \\
&\quad + \epsilon_2 t(l-1) \operatorname{err}_p(\mathcal{X}, \tilde{S}) + \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \left| \tilde{P}_m^{(1)}(S_1 \mid S_0) - P^{(1)}(S_1 \mid S_0) \right| \cdot \operatorname{err}_p(\mathcal{X}, \tilde{S}).
\end{aligned}
$$

$$
\left(\text{by upper bounding the probability expression } \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \text{ by 1.} \right)
$$

$$
\begin{aligned}
&\leq \mathop{\mathrm{E}}_{T_{1:l} \mid S_0} [\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})] \\
&\quad + \epsilon_1 \operatorname{err}_p(\mathcal{X}, \tilde{S}) \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \cdot \mathop{\tilde{\mathrm{E}}}_{T_{2:l} \mid S_1} [\operatorname{ind}_p(\mathcal{X}, S_1 \cup T_{2:l})] \\
&\quad + \epsilon_2 t(l-1) \operatorname{err}_p(\mathcal{X}, \tilde{S}) + \left\| \tilde{P}^{(1)}(\cdot \mid S_0) - P^{(1)}(\cdot \mid S_0) \right\|_{TV} \operatorname{err}_p(\mathcal{X}, \tilde{S}).
\end{aligned}
$$

$$
\left(\text{as } \mathop{\mathrm{E}}_{T_{1:l} \mid S_0} [\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})] = \sum_{S_1} P^{(1)}(S_1 \mid S_0) \mathop{\mathrm{E}}_{T_{2:l} \mid S_1} [\operatorname{err}_p(\mathcal{X}, S_1 \cup T_{2:l})] \text{ by Eq. (7).} \right)
$$

$$
\begin{aligned}
&\leq \mathop{\mathrm{E}}_{T_{1:l} \mid S_0} [\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})] \\
&\quad + \epsilon_1 \operatorname{err}_p(\mathcal{X}, \tilde{S}) \sum_{S_1 \,:\, \operatorname{ind}_p(\mathcal{X}, S_1)=0} \tilde{P}_m^{(1)}(S_1 \mid S_0) \cdot \mathop{\tilde{\mathrm{E}}}_{T_{2:l} \mid S_1} [\operatorname{ind}_p(\mathcal{X}, S_1 \cup T_{2:l})] \\
&\quad + \epsilon_2 t(l-1) \operatorname{err}_p(\mathcal{X}, \tilde{S}) + \epsilon_2 t \operatorname{err}_p(\mathcal{X}, \tilde{S}). \qquad (14)
\end{aligned}
$$

Finally, Equation (14) holds using Lemma 1 about the total variation distance between $P^{(1)}$ and $\tilde{P}^{(1)}$ distributions. Plugging the bounds (12) and (14) into (11), we get

$$
\begin{aligned}
&\mathop{\tilde{\mathrm{E}}}_{T_{1:l} \mid S_0} [\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})] \\
&\leq \mathop{\mathrm{E}}_{T_{1:l} \mid S_0} [\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})] + \epsilon_1 \operatorname{err}_p(\mathcal{X}, \tilde{S}) \sum_{S_1} \tilde{P}_m^{(1)}(S_1 \mid S_0) \cdot \mathop{\tilde{\mathrm{E}}}_{T_{2:l} \mid S_1} [\operatorname{ind}_p(\mathcal{X}, S_1 \cup T_{2:l})] \\
&\quad + \epsilon_2 t(l-1) \operatorname{err}_p(\mathcal{X}, \tilde{S}) + \epsilon_2 t \operatorname{err}_p(\mathcal{X}, \tilde{S}). \\
&= \mathop{\mathrm{E}}_{T_{1:l} \mid S_0} [\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})] + \left(\epsilon_1 \mathop{\tilde{\mathrm{E}}}_{T_{1:l} \mid S_0} [\operatorname{ind}_p(\mathcal{X}, S_0 \cup T_{1:l})] + \epsilon_2 t l \right) \operatorname{err}_p(\mathcal{X}, \tilde{S}). \\
&\leq \mathop{\mathrm{E}}_{T_{1:l} \mid S_0} [\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})] + (\epsilon_1 + \epsilon_2 t l) \operatorname{err}_p(\mathcal{X}, \tilde{S}),
\end{aligned}
$$

which completes the proof of Lemma 2. ◀

Theorem 5 from [16] shows that in $l = k$ rounds of adaptive sampling, where in each round we pick $t = \tilde{O}\left((k/\epsilon)^{p+1}\right)$ points and take their union, gives an additive approximation guarantee for ℓ_p subspace approximation with probability at least $1/2k$. Repeating it multiple times and taking the best can boost the probability further. We restate the main part of this theorem below.

▶ **Proposition 3** (Theorem 5, [16]). *Let k be any positive integer, let $\epsilon \in (0,1)$ and $S_0 = \emptyset$. Let $l = k$ and $t = \tilde{O}\left((k/\epsilon)^{p+1}\right)$. If $S_l = S_0 \cup T_{1:l}$ is obtained by starting from S_0 and doing adaptive sampling according to the p-th power of distances in l iterations, and in each iteration we add t points from $\mathcal{X}$, then we have $|S_l| = tl = \tilde{O}(k \cdot (k/\epsilon)^{p+1})$ such that*

$$\operatorname{err}_p(\mathcal{X}, S_0 \cup T_{1:l})^{1/p} \leq \operatorname{err}_p(\mathcal{X}, V^*)^{1/p} + \epsilon \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p},$$

with probability at least $1/2k$, and where V^* minimizes $\operatorname{err}_p(\mathcal{X}, V)$ over all linear subspaces V of dimension at most k. If we repeat this $O(k \log \frac{1}{\epsilon})$ times then the probability of success can be boosted to $1 - \epsilon$.

Combining Lemma 2 and Proposition 3 we get the following Theorem.

▶ **Theorem 4.** *For any positive integer k, any $p \in [1, \infty)$, and any $\delta \in \mathbb{R}_{\geq 0}$, starting from $S_0 = \emptyset$ and setting the following parameters in one-pass MCMC ℓ_p subspace approximation algorithm (see Section 3)*

$$\begin{aligned}
\epsilon &= \delta/4, \\
\epsilon_1 &= \delta^p/2^{p+1}, \\
\epsilon_2 &= \delta^p/2^{p+1}tl, \\
m &= 1 + \frac{2}{\delta^p} \log \frac{k}{\delta^p}, \\
t &= \tilde{O}((k/\epsilon)^{p+1}), \\
l &= k,
\end{aligned}$$

we get a subset $\mathcal{S}$ of size $\tilde{O}(k \cdot (k/\delta)^{p+1})$ with an additive approximation guarantee on its expected error as $\operatorname{err}_p(\mathcal{X}, V^*)^{1/p} + \delta \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p}$. Further, the running time of the algorithm is $nd + k \cdot \tilde{O}\left(\left(\frac{k}{\delta}\right)^{p+1}\right)$.

Proof. From Lemma 2 we know that

$$\tilde{\mathrm{E}}_{T_{1:l} \mid \emptyset} \left[\operatorname{err}_p(\mathcal{X}, T_{1:l})\right] \leq \mathrm{E}_{T_{1:l} \mid \emptyset} \left[\operatorname{err}_p(\mathcal{X}, T_{1:l})\right] + (\epsilon_1 + \epsilon_2 tl) \operatorname{err}_p(\mathcal{X}, \emptyset).$$

Thus, for $p \in [1, \infty)$ we have

$$\begin{aligned}
\tilde{\mathrm{E}}_{T_{1:l} \mid \emptyset} \left[\operatorname{err}_p(\mathcal{X}, T_{1:l})\right]^{1/p} &\leq \mathrm{E}_{T_{1:l} \mid \emptyset} \left[\operatorname{err}_p(\mathcal{X}, T_{1:l})\right]^{1/p} + (\epsilon_1 + \epsilon_2 tl)^{1/p} \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p}. \\
&\leq (1 - \epsilon) \left(\operatorname{err}_p(\mathcal{X}, V^*)^{1/p} + \epsilon \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p}\right) + \epsilon \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p} \\
&\qquad + (\epsilon_1 + \epsilon_2 tl)^{1/p} \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p}. \\
&\qquad \text{(using Proposition 3.)} \\
&\leq \operatorname{err}_p(\mathcal{X}, V^*)^{1/p} + \left(2\epsilon + (\epsilon_1 + \epsilon_2 tl)^{1/p}\right) \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p}. \\
&\leq \operatorname{err}_p(\mathcal{X}, V^*)^{1/p} + \delta \operatorname{err}_p(\mathcal{X}, \emptyset)^{1/p},
\end{aligned}$$

using $\epsilon = \delta/4$, $\epsilon_1 = \delta^p/2^{p+1}$ and $\epsilon_2 = \delta^p/2^{p+1}tl$.

We now give a bound on the running time of our algorithm. We require nd time to generate the probability distribution $q(x)$, for $x \in \mathcal{X}$. Further, the running time of MCMC sampling step is $t \cdot m \cdot l = k \cdot \tilde{O}\left(\left(\frac{k}{\delta}\right)^{p+1}\right)$. Therefore, the overall running time of the algorithm is $nd + k \cdot \tilde{O}\left(\left(\frac{k}{\delta}\right)^{p+1}\right)$. ◀

5 Conclusion and open questions

In this work, we give an efficient one-pass MCMC algorithm that does subset selection with additive approximation guarantee for ℓ_p subspace approximation, for any $p \in [1, \infty)$. Previously this was only known for the special case of $p = 2$ [11]. For general case $p \in [1, \infty)$, adaptive sampling algorithm due to [16] requires taking multiple passes over the input. Coming up with a one-pass subset selection algorithm that offers stronger multiplicative guarantees for $p \in [1, \infty)$ remains an interesting open problem.

References

1 Nima Anari, Shayan Oveis Gharan, and Alireza Rezaei. Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and determinantal point processes. In *29th Annual Conference on Learning Theory (COLT)*, volume 49, pages 103–115. PMLR, 2016. URL: `http://proceedings.mlr.press/v49/anari16.html`.

2 David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035. SIAM, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283494`.

3 Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 1459–1467. AAAI Press, 2016.

4 Olivier Bachem, Mario Lucic, Seyed Hamed Hassani, and Andreas Krause. Fast and provably good seedings for k-means. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 55–63, 2016. URL: `https://proceedings.neurips.cc/paper/2016/hash/d67d8ab4f4c10bf22aa353e27879133c-Abstract.html`.

5 Frank Ban, Vijay Bhattiprolu, Karl Bringmann, Pavel Kolev, Euiwoong Lee, and David P. Woodruff. A PTAS for p-low rank approximation. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 747–766. SIAM, 2019. `doi:10.1137/1.9781611975482.47`.

6 Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 517–528. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00055`.

7 Haiyan Cai. Exact bound for the convergence of metropolis chains. *Stochastic Analysis and Applications*, 18(1):63–71, 2000. `doi:10.1080/07362990008809654`.

8 M. T. CHAO. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, December 1982. `doi:10.1093/biomet/69.3.653`.

9 Flavio Chierichetti, Sreenivas Gollapudi, Ravi Kumar, Silvio Lattanzi, Rina Panigrahy, and David P. Woodruff. Algorithms for ℓ_p low-rank approximation. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 806–814. PMLR, 06–11 August 2017. URL: `https://proceedings.mlr.press/v70/chierichetti17a.html`.

10 Kenneth L. Clarkson and David P. Woodruff. Low-rank approximation and regression in input sparsity time. *J. ACM*, 63(6), January 2017. doi:10.1145/3019134.

11 Michael B. Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1758–1777. SIAM, 2017. doi:10.1137/1.9781611974782.115.

12 Chen Dan, Hong Wang, Hongyang Zhang, Yuchen Zhou, and Pradeep K Ravikumar. Optimal analysis of subset-selection based l_p low-rank approximation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/80a8155eb153025ea1d513d0b2c4b675-Paper.pdf.

13 Michal Derezinski and Manfred K. Warmuth. Unbiased estimates for linear regression via volume sampling. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3084–3093, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/54e36c5ff5f6a1802925ca009f3ebb68-Abstract.html.

14 Amit Deshpande, Praneeth Kacham, and Rameshwar Pratap. Robust k-means++. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2020, virtual online, August 3-6, 2020*, volume 124 of *Proceedings of Machine Learning Research*, pages 799–808. AUAI Press, 2020. URL: http://proceedings.mlr.press/v124/deshpande20a.html.

15 Amit Deshpande, Luis Rademacher, Santosh S. Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. *Theory Comput.*, 2(12):225–247, 2006. doi:10.4086/toc.2006.v002a012.

16 Amit Deshpande and Kasturi Varadarajan. Sampling-based dimension reduction for subspace approximation. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 641–650, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250884.

17 Amit Deshpande and Santosh S. Vempala. Adaptive sampling and fast low-rank matrix approximation. In Josep Díaz, Klaus Jansen, José D. P. Rolim, and Uri Zwick, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28-30 2006, Proceedings*, volume 4110 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 2006. doi:10.1007/11830924_28.

18 Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Mach. Learn.*, 42(1/2):143–175, 2001. doi:10.1023/A:1007612920971.

19 Charlie Dickens, Graham Cormode, and David Woodruff. Leveraging well-conditioned bases: Streaming and distributed summaries in Minkowski p-norms. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1243–1251. PMLR, 10–15 July 2018. URL: https://proceedings.mlr.press/v80/dickens18a.html.

20 Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM J. Matrix Anal. Appl.*, 30(2):844–881, 2008. doi:10.1137/07070471X.

21 Pavlos S. Efraimidis and Paul (Pavlos) Spirakis. Weighted random sampling. In *Encyclopedia of Algorithms*, pages 2365–2367. Springer, 2016. doi:10.1007/978-1-4939-2864-4_478.

22 Dan Feldman. Introduction to core-sets: an updated survey, 2020. arXiv:2011.09384.

23 Dan Feldman, Morteza Monemizadeh, Christian Sohler, and David P. Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 630–649. SIAM, 2010. doi:10.1137/1.9781611973075.53.

24 Alan M. Frieze, Ravi Kannan, and Santosh S. Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *J. ACM*, 51(6):1025–1041, 2004. doi:10.1145/1039488.1039494.

25 Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. Frequent directions : Simple and deterministic matrix sketching. *CoRR*, abs/1501.01711, 2015. arXiv:1501.01711.

26 Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM J. Comput.*, 45(5):1762–1792, 2016. doi:10.1137/15M1009718.

27 Mina Ghashami and Jeff M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 707–717, USA, 2014. Society for Industrial and Applied Mathematics.

28 Venkatesan Guruswami and Ali Kemal Sinop. Optimal column-based low-rank matrix reconstruction. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1207–1214. SIAM, 2012. doi:10.1137/1.9781611973099.95.

29 Yasutoshi Ida, Sekitoshi Kanai, Yasuhiro Fujiwara, Tomoharu Iwata, Koh Takeuchi, and Hisashi Kashima. Fast deterministic CUR matrix decomposition with accuracy assurance. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4594–4603. PMLR, 13–18 July 2020. URL: http://proceedings.mlr.press/v119/ida20a.html.

30 Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 581–588, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2487575.2487623.

31 Sepideh Mahabadi, Ilya P. Razenshteyn, David P. Woodruff, and Samson Zhou. Non-adaptive adaptive sampling on turnstile streams. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1251–1264. ACM, 2020. doi:10.1145/3357713.3384331.

32 Michael W Mahoney. Randomized algorithms for matrices and data. *arXiv preprint*, 2011. arXiv:1104.5557.

33 Michael W. Mahoney and Petros Drineas. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci. USA*, 106(3):697–702, 2009. doi:10.1073/pnas.0803205106.

34 Rameshwar Pratap, Anup Anand Deshmukh, Pratheeksha Nair, and Tarun Dutt. A faster sampling algorithm for spherical k-means. In Jun Zhu and Ichiro Takeuchi, editors, *Proceedings of The 10th Asian Conference on Machine Learning, ACML 2018, Beijing, China, November 14-16, 2018*, volume 95 of *Proceedings of Machine Learning Research*, pages 343–358. PMLR, 2018. URL: http://proceedings.mlr.press/v95/pratap18a.html.

35 Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. doi:10.1145/3147.3165.

36 Shusen Wang and Zhihua Zhang. Improving CUR matrix decomposition and the nyström approximation via adaptive sampling. *J. Mach. Learn. Res.*, 14(1):2729–2769, 2013. URL: http://dl.acm.org/citation.cfm?id=2567748.

37 Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1954–1963, 2015.

Set Membership with Two Classical and Quantum Bit Probes

Shyam S. Dhamapurkar
Southern University of Science and Technology, Shenzhen, China

Shubham Vivek Pawar
Hubli, Karnataka, India

Jaikumar Radhakrishnan
Tata Institute of Fundamental Research, Mumbai, India

Abstract

We study the classical and quantum bit-probe versions of the static set membership problem : Given a subset, S ($|S| \leq n$) of a universe, $\mathcal{U}$ ($|\mathcal{U}| = m \gg n$), represent it as a binary string in memory so that the query "Is x in S?" ($x \in \mathcal{U}$) can be answered by making at most t probes into the string. Let $s_A(m, n, t)$ denote the minimum length of the bit string in any scheme that solves this static set membership problem. We show that for $n \geq 4$

$$s_A(m, n, t = 2) = \begin{cases} \mathcal{O}(m^{1-1/(n-1)}) & \text{if } n = 0 \pmod 3; \\ \mathcal{O}(m^{1-1/n}) & \text{if } n = 1, 2 \pmod 3; \\ \mathcal{O}(m^{6/7}) & \text{if } n = 8, 9. \end{cases}$$

These bounds are shown using a common scheme that is based on a graph-theoretic observation on orienting the edges of a graph of high girth. For all $n \geq 4$, these bounds substantially improve on the previous best bounds known for this problem, some of which required elaborate constructions [4]. Our schemes are explicit. A lower bound of the form $s_A(m, n, 2) = \Omega(m^{1-\frac{1}{\lfloor n/4 \rfloor}})$ was known for this problem. We show an improved lower bound of $s_A(m, n, 2) = \Omega(m^{1-\frac{2}{n+3}})$; this bound was previously known only for $n = 3, 5$ [5, 6, 2, 7, 4].

We consider the quantum version of the problem, where access to the bit-string $b \in \{0, 1\}^s$ is provided in the form of a quantum oracle that performs the transformation $\mathcal{O}_b : |i\rangle \mapsto (-1)^{b_i} |i\rangle$. Let $s_Q(m, n, 2)$ denote the minimum length of the bit string that solves the above set membership problem in the quantum model (with adaptive queries but no error). We show that for all $n \leq m^{1/8}$, we have $s_{QA}(m, n, 2) = \mathcal{O}(m^{7/8})$. This upper bound makes crucial use of Nash-William's theorem [10] for decomposing a graph into forests. This result is significant because, prior to this work, it was not known if quantum schemes yield any advantage over classical schemes. We also consider schemes that make a small number of quantum *non-adaptive* probes. In particular, we show that the space required in this case, $s_{QN}(m, n = 2, t = 2) = O(\sqrt{m})$ and $s_{QN}(m, n = 2, t = 3) = O(m^{1/3})$; in contrast, it is known that two non-adaptive classical probes yield no savings. Our quantum schemes are simple and use only the fact that the XOR of two bits of memory can be computed using just one quantum query to the oracle.

2012 ACM Subject Classification Theory of computation; Theory of computation → Computational complexity and cryptography

Keywords and phrases set membership problem, bit probe complexity, graphs with high girth, quantum data structure

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.52

Category Track A: Algorithms, Complexity and Games

Related Version *Previous Version*: https://arxiv.org/abs/2112.14954

Funding *Shubham Vivek Pawar*: Partial work was done while interning at TCS Innovation Lab, Mumbai.

Jaikumar Radhakrishnan: Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 52; pp. 52:1–52:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We consider the problem of representing small subsets S of a universe $[m] = \{1, 2, \ldots, m\}$ in memory as a bit string so that membership queries of the form "Is $x \in S$?" can be answered with a small number of bit probes to the memory [9]. This is a fundamental question that asks how much a very sparse string can be compressed if we want to extract the bits of the original string efficiently from its compressed version. One natural way of representing sets in memory is the characteristic vector, which uses m bits of memory and answers membership queries using a single bit probe. Since there are only $O(\binom{m}{n})$ sets of size at most n (assume $n \ll m$) one might hope to represent them using $O(\log_2 \binom{m}{n}) \approx O(n \log m)$ bits of memory. However, compression to near the information-theoretic limits comes with a cost: membership can no longer be determined by reading just a small number of bits. To describe the trade-offs between efficiency of compression and the effort for extraction (measured as the number of bit probes), we will use the following notation [11]. Let $s_N(m, n, t)$ denote the minimum number of bits in a scheme that can represent sets of size at most n and answer membership queries by probing at most t bits of the memory non-adaptively (that is, the probes are made in parallel). We write $s_A(m, n, t)$ if the scheme is adaptive; we use the subscript Q if the scheme makes quantum queries (zero-error), which can be adaptive or non-adaptive [18], and write $s_{QA}(m, n, t)$ and $s_{QN}(m, n, t)$. Clearly,

$$s_N(m, n, t) \geq \left\{ \begin{array}{l} s_A(m, n, t) \\ s_{QN}(m, n, t) \end{array} \right\} \geq s_{QA}(m, n, t)$$

Radhakrishnan, Sen and Venkatesh [18] obtained lower bounds, which for the range of parameters of interest to us ($n \leq \sqrt{m}$, t constant) implies the following.

$$s_{QA}(m, n, t) = \Omega(m^{1/t} n^{1-1/t}).$$

(A similar lower bound in the classical setting was shown by Buhrman *et al.* [9].) Note that this bound shows that if the space is compressed to $O(n \log m)$, then $t = \Omega(\log m)$. Furthermore, if $t = 1$ no compression is possible even for $n = 1$; it also shows that $s_Q(m, 1, 2) = \Omega(\sqrt{m})$, for which there is a matching upper bound $s_N(m, n = 1, 2) = O(\sqrt{m})$. The first, non-trivial case is $n = 2$ and $t = 2$, where the above bound implies that $s_{QA}(m, 2, 2) = \Omega(\sqrt{m})$. It is known that this bound is not tight for classical schemes; better lower bound are known: $s_N(m, 2, 2) = m$ and $s_A(m, 2, 2) = \Omega(m^{4/7})$ [9, 19]. Remarkably, it is known that $s_A(m, 3, 2) = \Theta(m^{2/3})$ [13, 5]. Two-probe classical schemes have been constructed for representing small sets in several works starting with Alon and Feige [1] (see, e.g., [4, 2, 7, 12, 13, 15, 17], where sets of specific sizes are considered); the following upper and lower bounds was obtained by Garg and Radhakrishnan [12].

$$\Omega(m^{1-\frac{1}{\lfloor n/4 \rfloor}}) \leq s_A(m, n, t = 2) \leq \mathcal{O}(m^{1-\frac{1}{4n+1}}). \tag{1}$$

which roughly characterizes the space requirement for the problem, and, in particular, establishes that no savings over the standard characteristic vector representation can be expected if $n \geq \log m$. We show the following.

▶ **Theorem 1** (Result 1). *For $n \geq 4$,*

$$s_A(m, n, t = 2) = \begin{cases} \mathcal{O}(m^{1-1/(n-1)}) & \text{if } n = 0 \pmod 3; \\ \mathcal{O}(m^{1-1/n}) & \text{if } n = 1, 2 \pmod 3; \\ \mathcal{O}(m^{6/7}) & \text{if } n = 8, 9. \end{cases}$$

The above bounds improve the bounds for all other values of n (see Figure 1) for a comparison. Unlike the previous works, where different constructions (some of which quite intricate) were invented for different set sizes, our result is obtained using a unified approach based on graphs of high girth. (For $n = 2, 3$, the construction matches the currently best bounds.) To obtain a scheme for a set of a given size, one just plugs in the best available result for high-girth graph and obtains the bound claimed above. More importantly for us, the method used here inspires a quantum scheme that yields Result 3 below. We also obtain improved lower bounds.

▶ **Theorem 2** (Result 2). *For all odd n $(3 \leq n \leq \log m)$, $s_A(m, n, t = 2) = \Omega(m^{1-2/(n+3)})$.*

This bound matches the best bound known earlier for $n = 3$ (see [13]) and $n = 5$ (see [3]), and improves the current best lower bound (see Equation (1) above) for all larger values of n. We follow the approach of [12], who consider the graph underlying a two-probe scheme, and show that if it is dense then it must contain a forbidden configuration. We make better use of the structure of the underlying graph to force the existence of forbidden configurations.

We now describe our results in the quantum model. As stated above, our classical approach helps develop a new quantum schemes.

▶ **Theorem 3** (Result 3). $s_{QA}(m, n = m^{1/8}, t = 2) = \mathcal{O}(m^{7/8})$.

This result is especially significant because it shows that, unlike in the classical case, two probes give substantial savings over the characteristic vector representation for sets substantially larger than $\log m$ (see the remark above following Equation (1)). Before this work, quantum schemes were not known to provide significant savings over classical schemes. Our quantum scheme is also based on dense graphs that are locally sparse, this time we do not make use of high girth. Instead, we invoke a result of Nash-Williams [10] on covering the edges of a graph with two forests. After this, our construction uses only the following basic fact from quantum computation (Deutsch's algorithm [16]): the parity of two bits of memory can be computed using just one quantum probe. In fact, only the second probe in our scheme is truly quantum. This result opens the possibility that the lower bounds of $\sqrt{mn}$ cited above is perhaps achievable for quantum schemes. We show in fact that for $n = 2$, the lower bound can be matched using *non-adaptive* constructions.

▶ **Theorem 4** (Result4).

$$s_{QN}(m, n = 2, t = 2) = \mathcal{O}(\sqrt{m});$$
$$s_{QN}(m, n = 2, t = 3) = \mathcal{O}(m^{1/3}).$$

The query scheme is simple. The query scheme for $(n = 2, t = 2)$ on input x computes for locations $\ell_1(x), \ell_2(x), \ell_3(x), \ell_4(x)$, and returns "Yes" iff the bits at the first two locations are different and the bits at the last two locations are different, that is, we use an AND of two inequality computations, each of which requires just one quantum probe. A similar query scheme that uses an AND of three inequality computations gives the three-probe non-adaptive quantum scheme. We also obtain non-adaptive two-probe schemes with sublinear space for storing sets with more elements (see Appendix D). These bounds are interesting because no non-adaptive two-probe classical scheme exists with sublinear space [9].

2 Classical two-probe adaptive schemes

In this section we establish Theorem 1. Our two-probe adaptive schemes are based on dense graphs of high girth. We first specify the storage scheme and the query schemes based on an underlying graph. Then, to complete the proof, we will show the following: (i) if the

underlying graph has high girth, then there is an assignment of values to the memory such that all queries are answered correctly; (ii) the available explicit constructions of dense graphs of high girth in the literature yield the claimed bounds.

▶ **Definition 5** (Classical (G, K)-scheme)**.** *Let G be a directed graph with N vertices and M edges; let K be a positive integer. We refer to the following as a (G, K)-scheme. The storage consists of two bit arrays, A and B. To answer a membership query the decision tree will make the first probe to array A and the second probe to array B.*

Edge array: *An array $A : E(G) \to \{0, 1\}$, indexed by edges of G.*

Vertex array: *A two dimensional array $B : V \times [K] \to \{0, 1\}$.*

Elements: *We identify our universe of elements $[m]$ with a subset of $E(G) \times [K]$ (so we must ensure that the graph has at least m/K edges); thus, each element $x \in [m]$ will be referred to as (e, i).*

Query: *We represent an edge of G as an ordered pair of the form $e = (v_0, v_1)$ with the convention that $v_0 < v_1$. To process the query for the element $x = (e, i)$, we read $A[e]$ (first probe); then we return the value $B[v_{A[e]}, i]$ (by making the second probe into B). In other words, we may think of $A[e]$ as a bit that orients the edge e towards either its smaller vertex or the larger vertex; depending on this bit, the second probe is made into the array B corresponding to the vertex towards which the edge points. Note that this scheme is adaptive: the second probe depends on the first.*

Space: *We will ensure that $MK \geq m$. The space used by this scheme is then $NK + M$ bits (NK for the N vertex array and K for the edge array). By choosing the graph G and the parameter K appropriately we will show that our schemes use small space.*

The following lemma provides the connection between dense graphs of high girth and efficient two-probe adaptive schemes.

▶ **Lemma 6.** *Let G be a graph with N vertices M edges and girth g such that $n \leq \lfloor \frac{3g}{4} \rfloor$ and $M \leq m$. Then, $s_A(m, n, 2) \leq M + N\lceil m/M \rceil$.*

Before we present the proof of this lemma formally, let us derive from it the bounds claimed in Theorem 1. Since every graph has a bipartite subgraph that includes at least half the edges, it is sufficient to restrict attention to bipartite graphs, and hence to graphs whose girth is even. The smallest even number g such that $n \leq \lfloor \frac{3g}{4} \rfloor$ is given by

$$g(n) = \begin{cases} 4\lceil n/3 \rceil & n = -1, 0 \pmod 3; \\ 4\lceil n/3 \rceil - 2 & n = 1 \pmod 3. \end{cases} \tag{2}$$

Now, suppose that for a girth g, there are constant $c(g)$ $d(g)$ and $\tau(g)$, such that for all large L, there is a graph with at most $c(g)L$ vertices, girth g and $d(g)L^{1+\tau(g)}$ edges. Then, taking a graph with $N = \Theta(m^{1/(1+2\tau(g(n)))})$ vertices and $M = \Theta(m^{(1+\tau)/(1+2\tau)})$ edges in Lemma 6, we obtain the following.

▶ **Corollary 7.** $s_A(m, n, 2) = \mathcal{O}(m^{(1+\tau(g(n)))/(1+2\tau(g(n)))})$.

In particular, by using the current best constructions of dense graphs with large girth we obtain the following bounds for $s_A(m, n, 2)$. For example, we may take $\tau(6) = 1/2$, $\tau(8) = 1/3$ and $\tau(12) = 1/5$ based on graphs described Wenger [20]. (In Appendix A, we explain in what sense these constructions, and hence the resulting schemes, are explicit.)

Proof of Lemma 6. Fix a graph G with N vertices and M edges as in the statement of the theorem. Consider the (G, K)-scheme with $K = \lceil m/M \rceil$. Clearly the space used by the scheme is $N + KM = N + M\lceil m/M \rceil$. It remains to show that there is an assignment to the edge and vertex arrays of this scheme so that every query is answered correctly. Fix a set S

n	girth $g(n)$	$\tau(g(n))$	Our bound ($m^{(1+\tau)/(1+2\tau)}$)	Previous best bound
2,3	4	1	$\mathcal{O}(m^{2/3})$	$\mathcal{O}(m^{2/3})$ [13]
4	6	1/2	$\mathcal{O}(m^{3/4})$ (using [20])	$\mathcal{O}(m^{5/6})$ [6, 4]
5,6	8	1/3	$\mathcal{O}(m^{4/5})$ (using [20])	$\mathcal{O}(m^{5/6})$ (for $n = 5$ [4])
7	10	1/5	$\mathcal{O}(m^{6/7})$ (using [8])	↓
8,9	12	1/5	$\mathcal{O}(m^{6/7})$ (using [8])	↓
$n = 3r - 2$	$4r - 2$	$1/(3r - 4)$	$\mathcal{O}(m^{1-\frac{1}{n}})$ (using [14])	$\mathcal{O}(m^{1-1/(4n+1)})$ [12]
$n = 3r - 1, 3r$	$4r$	$1/(3r - 3)$	$\mathcal{O}(m^{1-\frac{1}{n}})$, $\mathcal{O}(m^{1-\frac{1}{n-1}})$ (using [14])	$\mathcal{O}(m^{1-1/(4n+1)})$ [12]

Figure 1 Our upper bounds use explicit constructions of graphs of large girth available in the literature (see Appendix A).

of at most n elements; recall that the elements of the universe have the form (e, i), where e is an edge of the graph and $i \in [K]$. We will assign values to the two arrays in two steps. First the edge array A will be assigned values. Recall that this assignment corresponds to assigning directions to the edges. We will show below how this is to be done. Assuming this we show how the array B is initialized. To start with, we initialize array B with zeros. Now for each element $(e, i) \in S$ (say $e = (v_0, v_1)$ where $v_0 < v_1$), if $A[e] = 0$ we set $B[v_0, i] = 1$, otherwise we set $B[v_1, i] = 1$. This assignment ensures that the query scheme described above will answer correctly for each element in S, so there are no *false negatives*, no matter what initial orientation of the edges is chosen. The key idea is to choose an orientation that avoids *false positives*; we must ensure that the value in the array A are set in such a way that an element not in S and an element in S do not make second probes to the same location in array B. Definition 8 below formally describes such a *safe orientation*. Here edges e such that $(e, i) \in S$ for some i are colored GREEN and the other edges are colored RED. Thus, there are at most n GREEN edges. Our choice of colors RED and GREEN are based on the following consideration. Some edges support elements in the set, some others do not support any such element. We chose to regard edges with elements in the set as GREEN, because the eventual answer to the query in that case is 'Yes'. In our definition of safe orientation, RED edges and GREEN edges are not allowed to point to a common vertex. Two GREEN edges are not allowed to point to the same vertex either but two RED edges are allowed to. We warn the reader that our choice of colors might be confusing, because GREEN edges are more restrictive/dangerous than RED edges! Then, Lemma 9 below shows that the graph G above has a safe orientation. It follows that our query scheme answers all the queries correctly. ◀

▶ **Definition 8** (Safe orientation). *Suppose H is a graph whose edges are colored RED and GREEN. We say that an orientation of edges is* safe *if every vertex with an incoming GREEN edge has only one edge coming into it.*

▶ **Lemma 9.** *Suppose H is a graph with even girth $g \geq 4$ and $n \leq \lfloor 3g/4 \rfloor$ GREEN edges. Then, G has a safe orientation. (This claim should have a simple proof, but we have not been able to find one that covers all cases succinctly.)*

Preliminaries: In the following, suppose H is a graph with even girth g and $n \leq \lfloor 3g/4 \rfloor$ GREEN edges. To find the necessary orientation, we will proceed by induction on the size of H (its total number of edges plus vertices). For the base case, note that a graph with no edges clearly has a safe orientation. For the induction step, we will identify an initial set of vertices V' such that all edges that have at least one end point in V' can be safely oriented towards a vertex in V'. We then delete V' and the edges incident on it, and use induction to extend this orientation to the rest of H. To identify the set V', we will use a *breadth first*

search procedure formally described below. This procedure produces a *breadth first search tree* or a *breadth first search forest* as usual, but we need to impose the following condition on it.

> If a vertex w in the tree is connected to its parent by a RED edge, then all of w's children are connected to w using GREEN edges; thus, in any root to leaf path in the tree, RED edges do not appear consecutively.

To enforce this, when a RED edge is added to the tree, we will mark the vertex it leads to RED; then when we visit this vertex, we only explore vertices connected to it by GREEN edges. If a GREEN edge is added to the tree, we mark the new vertex GREEN; then when we visit this vertex, we explore all its edges, whether GREEN or RED. The formal code is presented in Algorithm 1. (This is reminiscent of the breadth first search procedure employed by certain matching algorithms to discover augmenting paths; there one alternately explores either only the matched edge or all edges). As a first attempt we might want to orient the

Algorithm 1 Breadth-First Search (BFS).

```
   Input  : A non-empty subset Z ⊆ V(H)
   Output : A BFS forest rooted at the vertices in Z
 1 Q = empty queue ;
 2 push all elements of Z into Q and mark them GREEN;
 3 while Q is non-empty do
 4 |   v = pop(Q);
 5 |   if v is marked GREEN then
 6 |   |   push all unmarked neighbors w of v into Q;
   |   |   // now assign them colors as follows
 7 |   |   if {v, w} is GREEN then
 8 |   |   |   add {v, w} to the forest, and mark w GREEN
 9 |   |   end
10 |   |   else
11 |   |   |   mark w RED
12 |   |   end
13 |   end
14 |   if v is marked RED then
15 |   |   push all unmarked neighbors w of v with {v, w} GREEN into Q;
16 |   |   add {v, w} to the forest and mark w GREEN;
17 |   end
18 end
```

edges of the forest away from the roots and hope to extend this orientation to the other edges that have at least one end point in the forest. If this gives a valid orientation for these edges, we let V' be the vertex set of the forest, and proceed as above. Unfortunately, this straightforward method may run into problems; this motivates the following definition.

▶ **Definition 10** (Blocking edge, see Figure 2)**.** *In the forest constructed by BFS, we say that a non-tree edge is a blocking edge if (i) it is a non-tree* GREEN *edge both of whose end points are visited during BFS, or (ii) it is a non-tree* RED *edge with both end points marked* GREEN*.*

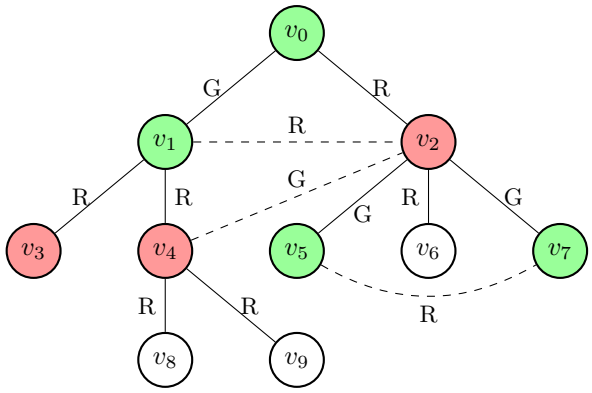


Figure 2 The BFS tree: (v_1, v_2) is not a blocking edge, (v_2, v_4) and (v_5, v_7) are blocking edges.

We will see that if there are no blocking edges, then the above strategy will work; otherwise, H has a cycle with many GREEN edges, and we will be able to exploit that.

▶ **Definition 11** (GREEN-dominated cycle, see Figure 3)**.** *We say that a cycle in H is* GREEN-dominated *if all but (perhaps) one of its RED edges are followed by a* GREEN edge.

We will establish the following two lemmas below.

▶ **Lemma 12.** *Suppose H has no* GREEN*-dominated cycle. Suppose V' is the set of vertices of H visited by BFS starting at a vertex v_0. Then there is a safe orientation of edges of H incident on V'.*

▶ **Lemma 13.** *Suppose H has a* GREEN*-dominated cycle C. Let H' be the graph obtained by deleting from H all edges of C. Let V' be the vertices visited by BFS in H' starting with the vetex set $V(C)$ of the cycle C. Then, there is a safe orientation of edges of H incident on V'.*

Let us use these lemmas to complete the proof of Lemma 9.

Proof of the Lemma 9. If H has no GREEN-dominated cycle, then by Lemma 12, we obtain an initial set of vertices V' and an orientation of all edges incident on it. If H has a GREEN-dominated cycle, then by Lemma 13 we obtain an initial set of vertices V' and an orientation of all edges incident on it. We extend this orientation to the remaining edges of the graph by deleting V' and all edges incident on it, and applying induction to the resulting subgraph induced by the vertex set $V \setminus V'$. ◀

We now return to the unproved lemmas.

Proof of Lemma 12. Let v_0 be an arbitrary vertex. Consider the tree produced by BFS starting with $Z = \{v_0\}$. We claim that there is no blocking edge for the resulting tree. For suppose $e = \{a, b\}$ is a blocking edge. Let v be the least common ancestor of a and b, and recall that in the paths that connect v to a and v to b, no RED edge is followed by a RED edge. Let C be the cycle formed by taking the path from v to a followed by e and then the path from b back to v. If e is GREEN, then this cycle is GREEN-dominated, contradicting our assumption. If e is RED, then by the definition of blocking edge, both a and b are marked GREEN, that is, the tree edges connecting them to their parents are GREEN (note that e is not a back edge because both its vertices are GREEN). Again the

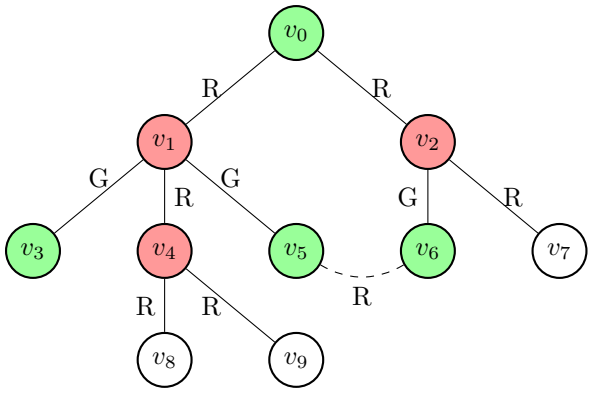


Figure 3 The edge is (v_5, v_6) is a blocking edge and $(v_0, v_1, v_5, v_6, v_2, v_0)$ is the resulting GREEN-dominated cycle, even though it has more RED edges than GREEN edges.

cycle C is GREEN-dominated, contradicting our assumption. Thus, the tree has no blocking edges. Let V' be the vertices visited by BFS. Orient all tree edges away from the root v_0. The remaining edges incident on V' (which cannot be GREEN) have at least one vertex marked RED, because they are not blocking. Orient them towards that RED vertex. It can be verified that the GREEN edges that received an orientation are all tree edges, and are oriented towards distinct GREEN vertices. The RED edges are oriented towards RED vertices. So all edges incident on V' can be oriented safely. ◀

Proof of Lemma 13. Fix a GREEN-dominated cycle C in H. Suppose it has ℓ_1 edges (for some $\ell_1 \geq g$) of which say n_1 are GREEN. Then,

$$n_1 \geq \lceil (\ell_1 - 1)/2 \rceil \geq g/2, \tag{3}$$

because g is even. First, suppose the resulting BFS forest has no blocking edges, then let V' be the vertices of this BFS forest. We orient the edges in C so that it becomes a directed cycle (we may choose either of the two ways to do this). Then, we orient all tree edges away from the roots in the BFS forest. Note that all other edges incident on V' must necessarily be RED; each such edge has at least one RED vertex in V'. We orient each such edge towards a RED vertex, and obtain the desired safe orientation.

Next, suppose there is a blocking edge $e = \{a, b\}$. If a and b belong to the same tree of the forest, then e and the paths from a and b to their least common ancestor form a GREEN-dominated cycle, consisting of say $\ell_2 \geq g$ edges of which n_2 are GREEN. Then,

$$n_2 \geq \lceil (\ell_2 - 1)/2 \rceil \geq g/2. \tag{4}$$

From Equation (3) and Equation (4), we obtain, $n \geq n_1 + n_2 \geq g$, contradicting our assumption that $n \leq 3g/4$. So, we may assume that a and b belong to different trees of the forest. Then, travelling from the root r_1 of a's tree to a, crossing over along e to b, and then travelling to the root r_2 of the tree of b, we obtain a path P^*, where no RED edge is followed by a RED edge; in particular, every RED edge except perhaps the last, is followed by a GREEN edge. Suppose this path has ℓ_3 edges of which n_3 are GREEN. We have the following.

$$\ell_3 \geq g - \lfloor \ell_1/2 \rfloor; \qquad (G \text{ has girth } g) \tag{5}$$

$$2n_3 \geq \ell_3 - 1. \tag{6}$$

From Equation (3), Equation (5) and Equation (6), we obtain

$$\begin{aligned} 2(n_1+n_3) &\geq g + 2\lceil(\ell_1-1)/2\rceil - \lfloor \ell_1/2 \rfloor - 1 && (7)\\ &= g + \lceil(\ell_1-1)/2\rceil - 1 && (8)\\ &\geq 3g/2 - 1. && (9) \end{aligned}$$

If $n > n_1 + n_3$ (that is, there is some GREEN edge outside $C \cup P^*$), then we obtain $n \geq n_1 + n_3 + 1 \geq 3g/4 + 1/2$, contradicting our assumption $n \leq 3g/4$. So, we may assume that all GREEN edges in the graph are contained in $C \cup P^*$. We think of $C \cup P^*$ as a set of three edge disjoint paths, P_1, P_2 and P_3, connecting r_1 to r_2, where $P_1 \cup P_2 = C$ and $P_3 = P^*$. Let V^* be the set of vertices in $P_1 \cup P_2 \cup P^*$. We will show that the graph induced by V^* can be safely oriented. Then, we will orient all other (necessarily RED) edges of G towards a vertex not in V^* to obtain a safe orientation of the *entire* graph H, and conclude that the lemma holds with $V' = V(H)$.

First, we show that we may assume that each of the three cycles $C = P_1 \cup P_2$, $P_1 \cup P_3$ and $P_2 \cup P_3$ is chordless. Since C is GREEN-dominated, it has at most $2n+1$ edges. If it had a chord, we would get a cycle with at most $n \leq 3g/4 < g$ edges, a contradiction. Thus, the cycle $C = P_1 \cup P_2$ has no chord. Next, using a similar argument we show that the other two cycles are chordless. We first observe that both P_1 and P_2 have GREEN edges. If P_1 has no GREEN edges, then it can have at most two edges, and the at least $g/2$ GREEN edges of C all lie in P_2. Also, P_3 has at least $g-2$ edges. Then, the number of GREEN edges in P_2 is at least $\max\{|P_2|, g\}/2$ (because C is GREEN-dominated), and similarly the number of GREEN edges in P_3 is at least $\lceil(|P_3|-1)/2\rceil \geq g/2 - 1$. Thus,

$$3/4g \geq n \geq \max\{|P_2|, g\}/2 + \lceil(|P_3|-1)/2\rceil \geq g - 1; \quad (10)$$

that is, $g = 4$, $|P_2| \leq 4$ and $|P_3| \leq 3$. Thus, $P_2 \cup P_3$ is a cycle with at most 7 vertices and it cannot have a chord because $g = 4$. Thus, we may assume that P_1 has at least one GREEN edge, that is, $P_2 \cup P_3$ has at most $n-1$ GREEN edges. Let k_2 be the number of GREEN edges in P_2 and k_3 the number of GREEN edges in P_3. Since every RED edge in P_3, except perhaps one is followed by a GREEN edges, the number of edges in P_3 is at most $2k_2 + 2$. Then, the total number of edges in $P_2 \cup P_3 \leq (2k_2+2) + (2k_3+1)$ (the second term comes from Equation (6)), that is, at most $2n+1$ edges in all. If $P_2 \cup P_3$ has a chord, then we have a cycle of length at most n, which, as we saw earlier, is not possible. Similarly, $P_2 \cup P_3$ has not chord.

So the graph induced by V^* consists of three disjoint paths, with no chords across them. If any path has two consecutive RED edges, then we may orient them towards each other and be left with a graph consisting of a cycle with two dangling paths, which can be oriented safely. Similarly, if some two paths start with RED edges or end with RED edges, then these edges can be oriented towards each other, and the remaining edges (which form a tree) can be oriented safely. We are left with the case where on all paths a RED edge is followed by a GREEN edge, and at both ends (r_1 and r_2), two of the paths start with GREEN edges. We will show that this is impossible. For otherwise, there is path (say, P_3), which has GREEN edges at both ends, so $|P_3|$ has at least $(|P_3|+1)/2$ GREEN edges. For the remaining paths, either some path has both ends GREEN, or both paths have one end GREEN. In either case, they together have at least $(|P_1|+|P_2|)/2$ GREEN edges. Note that $2(|P_1|+|P_2|+|P_3|) \geq 3g$, because H has girth at least g. Thus, the total number of GREEN edges is $n \geq (|P_1|+|P_2|+|P_3|+1)/2 \geq (3g/2+1)/2 > 3g/4$, contradicting our assumption that $n \leq 3g/4$. ◀

3 Quantum adaptive schemes

In this section, we establish Theorem 3. Our quantum scheme is based on a graph and is similar in some respects to the classical scheme described above. The main difference is in the second probe, which now computes the XOR of two bits of memory.

▶ **Definition 14** (Quantum (G, K)-scheme). *Let G be a directed graph with N vertices and M edges; let K be a positive integer. We refer to the following as a quantum (G, K)-scheme. The storage consists of three bit arrays, A, B_0 and B_1. To answer a membership query, the quantum decision tree first probes array A (this probe is classical) and then computes the XOR of two bits in either B_0 or B_1, using just one quantum probe.*

Edge array: *An array $A : E(G) \to \{0, 1\}$, indexed by edges of G.*

Vertex arrays: *Two two-dimensional arrays $B_0, B_1 : V \times [K] \to \{0, 1\}$, indexed by elements of the form (v, i).*

Elements: *As before, we identify our universe of elements $[m]$ with a subset of $E(G) \times [K]$; thus, each element $x \in [m]$ will be referred to as (e, i).*

Query: *Let the query be "Is x in S?", where $x = (e, i)$; suppose $e = \{x_0, x_1\}$. To process this query for we read $A[e]$ (first probe); then, based on the value of $A[e]$, we return either $B_0[(v_0, i)] + B_0[(v_1, i)] \pmod 2$ or $B_1[(v_0, i)] + B_1[(v_1, i)] \pmod 2$. In other words, the first probe directs us to either array B_0 or B_1; we then return the XOR of the bits in the i-th location in the rows corresponding to the two vertices of e.*

Space: *We will ensure that $MK \geq m$, to accommodate all elements of the universe. The space used by this scheme is then $2NK + M$ bits. By choosing the graph G and the parameter K appropriately we will show that our schemes uses small space.*

The main idea is the following. To store the set S in the data structure, we partition the edges of G using the 0-1 assignment to the array A. Let G_0 be the graph induced by the edges that are assigned 0 in A, and let G_1 be the graph induced by the edges assigned 1. Now, the bits of the arrays B_0 and B_1 must be assigned in such a way that certain XORs of two bits evaluate to 1 and others evaluate to 0. This leads to a system of linear equations in the bits of the arrays B_0 and B_1. To ensure that this system has a solution, we ensure that if $A[e] = 0$, then e is not in any cycle in G_0, and similarly, if $A[e] = 1$, then e is not in any cycle in G_1. It is then easy to see that the required assignment to the arrays B_0 and B_1 exists. To ensure that the edges of G can be partitioned in G_0 and G_1 to satisfy the requirements imposed by the set S, we will start with the graph G that is dense but locally sparse in the following sense, and use a theorem of Nash-Williams.

▶ **Definition 15** (Locally sparse graph). *A graph G is (k, α)-locally sparse if for every subsets $V' \subseteq V$ with $4 \leq |V'| \leq k$ vertices, the induced subgraph on V' has at most $\alpha|V'|$ edges.*

▶ **Lemma 16.** *If G has N vertices, M edges and is $(4n, 5/4)$-locally sparse, then*

$$s_{QA}(m, n, t = 2) \leq M + 2N\lceil \frac{m}{M} \rceil.$$

Before we present the proof of this lemma, let us see how this leads to Theorem 3. We will need a family of dense locally sparse graphs, whose existence we establish in Appendix C using a routine probabilistic argument.

▶ **Lemma 17.** *For all large N there is a $(4N^{1/6}, 5/4)$-locally sparse graph with N vertices and $\Omega(N^{7/6})$ edges.*

We set $N = m^{3/4}$, and plugging in the graph promised by Lemma 17 in Lemma 16, obtain $s_{QA}(m, m^{1/8}, 2) = O(m^{7/8})$, as claimed in Theorem 3. It remains to establish Lemma 16.

Proof of Lemma 16. Fix G with the given parameters. We now describe how the three arrays in our quantum scheme are assigned values. Recall that we view elements of $[m]$ as pairs (e, i). Edges of G for which there is an element of the form $(e, i) \in S$ will be called GREEN; the other edges of G will be called RED. Say, there are $\ell \leq n$ GREEN edges. We will construct a sets of vertices $D_0, D_1, D_2, \ldots$ by adding one vertex at a time. Let $D_0 \subseteq V(G)$ be the union of the GREEN edges; thus D_0 has at most 2ℓ vertices. To obtain D_{i+1} from D_i, add to D_i a new vertex that has at least two edges leading into D_i; if no such vertex exists, stop. We claim that this process stops before $2n$ vertices are added, for otherwise, the graph induced by D_{2n}, a set of size at most $2n + 2\ell \leq 4n$ vertices, has at least $4n + \ell$ edges. Since G is $(4n, 5/4)$-locally sparse, we have $(5/4)(2n + 2\ell) > 4n + \ell$, implying $\ell > n$, a contradiction. Let D be the set of vertices when the above process stops.

▷ Claim 18. The subgraph induced by D can be split into two disjoint forests.

We will justify this claim below (using Nash-Williams theorem). Let us assume it and complete the proof. Let the two forests guaranteed by this claim be F_1 and F_2. We set $A[e] = 0$ for all edges $e \in F_1$ and all edges that connect D to $V \setminus D$. Let G_0 be the subgraph of G with vertex set $V(G)$ that consist of edges e such that $A[e] = 0$. Let G_1 be the subgraph with vertex set $V(G)$ and all edges not included in G_0. Note that the connected components of G_1 are either in the forest F_2 or consist of RED edges with both end points in $V \setminus D$. Now, we are ready to describe the assignment to arrays B_0 and B_1. As stated above the constraints imposed by the GREEN and RED edges give a system of equational constraints; since G_0 has no cycle, it is easy to see that these constraints can all be satisfied by assigning B_0 appropriately. In G_1 again, the edges corresponding to F_2 do not induce a cycle, so the constraints imposed by them can be satisfied by assigning appropriate values to the rows of B_2 corresponding to vertices in D. The remaining edges share no vertex with the edges of F_2, and consist only of RED edges. So we assign zeroes to all rows of B_2 corresponding to vertices in $V(G) \setminus D$.

It remains to verify Claim 18. Since $|D| \leq 4n$, every subset D' of D (with $|D'| \geq 4$) induces at most $(5/4)|D'|$ edges; since $|D| \geq 4$, we have $(5/4)|D'| \leq 2(|D'| - 1)$. Note that the number of edges in any graph with at most $1 \leq \ell \leq 4$ vertices is at most $2\ell - 2$. So we may invoke Theorem 19 below and justify the claim. ◀

▶ **Theorem 19** (Nash-Williams (see Theorem 3.5.4 in [10]).). *Let $H = (V, E)$ be an undirected graph such that for each non-empty subset $X \subseteq V$, the number of edges with both end points in X is at most $2(|X| - 1)$. Then E can be partitioned as $E = E_1 \cup E_2$ such that (V, E_1) and (V, E_2) are both forests.*

4 Lower bounds for classical schemes

In this section, we present our justification for Theorem 2.

Canonical query schemes. Consider an (m, n, s, t)-scheme. Let us use M to denote the array of s-bits into which probes are made. With each element $x \in [m]$ of the universe such a scheme associates three addresses $(a(x), b(x), c(x)) \in [s]^3$, where the first probe is made to location $a(x)$; if the bit there is 0, then the second probe is made to $b(x)$, otherwise the probe is made to $c(x)$. We will assume that that the query scheme has the following *canonical form*. On query "Is x in S?", the answer is determined as follows: if $M[a(x)] = 1$, then return $M[b(x)]$, else return $M[(c(x)]$, where 0 is treated as false/no and 1 as true/yes. We refer to such schemes as *canonical schemes*. It is easy to see that by at most doubling the memory a scheme can be made canonical.

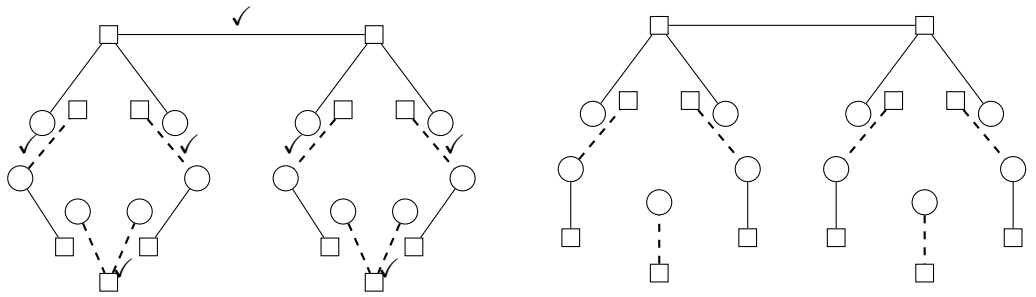
▶ **Proposition 20.** *If there is an $(m, n, s, t = 2)$-scheme, then there is an $(m, n, 2s, t = 2)$-canonical scheme.*

From now on we will assume that the scheme is canonical.

▶ **Definition 21** (The bipartite graph associated with a scheme)**.** *With the scheme σ, we associate a directed bipartite graph G_σ with vertex sets B and C, whose elements we refer to using $[s]$. For each $x \in [m]$ we add an edge $e(x) = (b(x), c(x))$ in G_σ with label $(x, a(x))$. The value $a(x)$ will be called the color of the edge; so $e(x)$ and $e(y)$ have the same color if and only if the first probes for the two queries "Is x in S?" and "Is y in S?" are made to the same location, namely $a(x) = a(y)$. We will use $\vec{e}$ to refer the oriented version of the edges of G_σ. We say that two distinct oriented edges, $\vec{e}_1$ and $\vec{e}_2$ are parallel if (i) $\vec{e}_1$ and $\vec{e}_2$ have the same color, and (ii) they are both oriented in the same direction (both from B to C or C to B).*

To store a set S, one must find an assignment to the locations where the first probes are made. This amounts to choosing an orientation for each color, and orienting the edges either from B to C or from C to B; the values in the array is then essentially forced because the protocol is assumed to be canonical. For this assignment to be valid, in the resulting directed graph, we must have the property that if $x \in S$ and $y \notin S$, then $\mathsf{head}(\vec{e}(x)) \neq \mathsf{head}(\vec{e}(y))$. We refer to such an orientation as a *safe* orientation for S.

We obtain our lower bound by establishing that if the data structure uses very small space, then there is a set S of size at most n edges, whose edges cannot be oriented safely.



(a) A forbidden configuration.

(b) The problem case.

Figure 4 $n = 7$.

Forbidden configuration. Fix n of the form $2\ell - 1$. Please refer to Figure 4a. Most edges in the figure come in pairs of solid and dotted edges, which are placed parallel to each other. For each edge, one vertex is a circle and the other is a square, to indicate that one of them comes from B and the other from C (we do not specify which is which). The elements labelling the edges are all distinct; however, we allow the edges to have the same color even if they are not explicitly depicted as being parallel. We say that such graph F is forbidden configuration of order n, if there is a subset $S \subseteq [m]$ ($|S| \leq n$) of elements appearing in the labels on the edges of F such that F is not safe for S. For example, Figure 4a is a forbidden configuration of order $n = 7$, where the set S is indicated by ✓. Our lower bound result Theorem 2 follows immediately from the following lemma.

▶ **Lemma 22.** *Fix an odd n $(3 \leq n \leq \log m)$ and an $(m, n, s, t = 2)$-scheme σ. If G_σ does not contain any forbidden configuration of order n, then $s \geq cm^{1-2/(n+3)}$, for a constant $c > 0$ independent of n.*

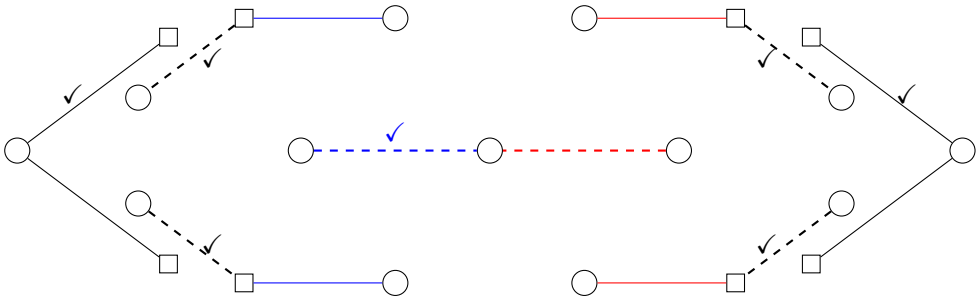


Figure 5 The second forbidden configuration ($n = 7$).

Proof Sketch. Consider $(m, n, s, t = 2)$-scheme σ. In G_σ we have m edges. Then, (on average) a vertex has degree about m/s and each colour has m/s edges. We start from an edge $e = (v, w)$, and from each of v and w, we build a tree as follows. Let us start with $v_0 = v$. We have m/s choices for an edge; for each choice of edge e of the form (v_0, v_1), we jump to an edge of the same colour (parallel to the first one), that takes us from a vertex v_1' to a vertex v_2. We continue this process, alternately expanding to a neighbor or jumping to a parallel edge for k steps in all. If we set k such that $(m/s)^k > s$, we obtain a "cycle". (Note that this is not a cycle in the usual graph-theoretic sense, because we jump from an edge to a parallel edges in alternate steps.) We delete these edges from the graph, and repeat this for the other vertex w of the starting edge e. Let us illustrate this for $n = 7$. Suppose $s \ll m^{1-2/(n+3)} = m^{4/5}$, that is, $(m/s) \gg m^{1/5}$. In particular, with $k = 4$, we have $(m/s)^k \gg m^{4/5} \gg s$, and at some point a vertex must repeat (we have only $2s$ vertices). A situation in such a case, with two cycles hanging off an edge corresponds to Figure 4a. We allow the two cycles to share vertices, but the edges involved must be distinct. Now, to see that this configuration has no safe orientation, first choose either orientation for the top edge e, say towards left. Then, the directions of all edges are forced in that cycle and we soon find edges corresponding to an element in the set and another corresponding to an element not in the set that point to the same vertex.

Unfortunately, there are other cases to consider, besides the ideal case of two cycles attached to an edge as in Figure 4a: (i) the cycles may not form right at the top, instead we might have to allow an initial path leading to the cycle; (ii) the cycle may not end with two tree edges pointing into it. Instead, it might be formed when two paths of a tree jump on to the same parallel edge. The first case presents no real difficulties; in fact, in this case the resulting configuration is not safe for even smaller sets. The second case presents genuine difficulties. For example, we might encounter a situation depicted in Figure 4b, where the three edges at the bottom are parallel. Note that all tree edges in this case can be forced away from their roots to obtain a safe orientation. The idea now is the following. If we encounter such a cycle, we put it aside and mark the middle vertex at the bottom as its terminal vertex. We have removed only a minuscule number of edges from the graph, so we can continue the exploration for a forbidden structure in the remaining graph. If we ever find a configuration corresponding to Figure 4a, we are done. Otherwise, we accumulate many edge disjoint bad cycles. Soon enough (by the time $s + 1$ bad cycles are encountered), two of them must have the same terminal vertex. We put these bad cycles together (as illustrated in Figure 5) and again obtain a forbidden configuration. The discussion above uses $k = 4$ and $n = 7$ for illustration, but the same argument applies for other k, and, in general, yields a configuration without any safe orientation for a set of size $n = 2k - 1$, whenever, $s \ll m^{1-1/(k+1)} = m^{1-2/(n+3)}$. The detailed argument will appear in the full version of the paper. ◀

5 Quantum non-adaptive schemes for $n = 2$ and $t = 2, 3$

In this section, we show that the lower bound in 1 is tight for two cases: $s_{QN}(m, n = 2, t = 2) = O(\sqrt{m})$ and $s_{QN}(m, n = 2, t = 3) = O(m^{1/3})$; the schemes we give are non-adaptive and only use the fact that the XOR of two bits can be computed using one quantum query. The proofs are algebraic.

5.1 Case $t = 2$

We identify $[m]$ with $A \times B$, where each of the sets has about $\sqrt{m}$ elements; A and B are disjoint. We will have two array indexed by A (we call them X_1 and X_2) and two arrays indexed by B (we call them Y_1 and Y_2).

Query: On receiving the element $x = (x_1, x_2) \in A \times B$, the algorithm returns

$$(X_1[x_1] + Y_1[x_2])(X_2[x_1] + Y_2[x_2]) \pmod 2,$$

which is a polynomial of degree two. Note that both $X_1[a] + Y_1[b]$ and $X_2[a] + Y_2[b]$ can be computed in parallel with one quantum query each. Thus, the scheme requires only two non-adaptive queries.

Storage: Given a pair of elements $\{\alpha_1, \alpha_2\} \subseteq [m]$, we need to show how values will be assigned to the four arrays: X_1, X_2, Y_1, Y_2. It will be easier to describe and analyse our storage algebraically. We view X_1, X_2 as functions from A to $\{0, 1\}$ and Y_1, Y_2 as functions from B to $\{0, 1\}$. For $a \in A$, let $\delta_a : A \to \{0, 1\}$ be defined by $\delta_a(z) = 1$ iff $z = a$; similarly for $b \in B$, let $\delta_b : B \to \{0, 1\}$ be defined by $\delta_b(z) = 1$ iff $z = b$. We have three cases based on the number of components $\ell \in \{0, 1, 2\}$ where α_1 and α_2 agree.

$\ell = 2$: We have only one element (a, b). We set $X_1 \equiv \delta_a$, $Y_1 = 0$, $X_2 \equiv 0$ and $Y_2 \equiv \delta_b$. The query polynomial reduces to the monomial $\delta_a(x_1)\delta_b(x_2)$, which is what we want.

$\ell = 1$: Say the set is $\{(a, b), (a', b)\}$. We set $X_1 \equiv \delta_a + \delta_{a'}$, $Y_1 \equiv 0$, $X_2 \equiv 0$ and $Y_2 \equiv \delta_b$. The query polynomial reduces to $(\delta_a(x_1) + \delta_{a'}(x_2))\delta_b(z_2) = \delta_a(z_1)\delta_b(z_2) + \delta_{a'}(z_1)\delta_b(z_2)$, which is what we want.

$\ell = 0$: Say the set is $\{(a, b), (a', b')\}$. We set $X_1 \equiv \delta_a$, $Y_1 \equiv \delta_{b'}$, $X_2 \equiv \delta_{a'}$ and $Y_2 \equiv \delta_b$. The query polynomial evaluates $(\delta_a(z_1)+\delta_{b'}(z_2))(\delta_{a'}(z_1)+\delta_b(z_2)) = \delta_a(z_1)\delta_b(z_2)+\delta_{a'}(z_1)\delta_{b'}(z_2)$ (mod 2), which is what we want.

5.2 Case $t = 3$

Let us identify $[m]$ with $A \times B \times C$, where each of the sets has roughly $m^{1/3}$ elements; we will assume that A, B and C are disjoint. We have six arrays, two indexed by A (we call them X_1 and X_2), two indexed by B (we call them Y_1 and Y_3) and two indexed by C (we call them Z_2 and Z_3); the subscripts indicate which query probes the corresponding array.

Query: On receiving the element $e = (x, y, z)$, the algorithm returns

$$(X_1[x] + Y_1[y])(X_2[x] + Z_2[z])(Y_3[y] + Z_3[z]) \pmod 2,$$

which is a polynomial of degree 3.

Storage: Given a pair of elements $\{\alpha_1, \alpha_2\}$, we need to show how values will be assigned to the six arrays. Let $\alpha = (a, b, c)$ and $\beta = (a', b', c')$. We define functions of the form $\delta_a : A \to \{0, 1\}$, $\delta_b : B \to \{0, 1\}$ and $\delta_c : C \to \{0, 1\}$ as before. Also 0 and 1 when denoting functions correspond to the all 0's and the all 1's functions. We have four cases, depending on the number of places $\ell \in \{0, 1, 2, 3\}$ where α_1 and α_2 agree.

$\ell = 3$: The set has only one element (a, b, c), say. The arrays are as follows. $X_1 \equiv \delta_a$; $Z_2 \equiv \delta_c$; $Y_3 \equiv \delta_b$, the other three arrays are 0. So, the query function becomes $(\delta_a(x) + 0)(\delta_c(z) + 0)(\delta_b(y) + 0) = \delta_a(x)\delta_b(y)\delta_c(z)$, which yields 1 iff $(x, y, z) = (a, b, c)$.

$\ell = 2$: Say $\alpha_1 = (a, b, c)$ and $\alpha_2 = (a, b, c')$. We set $X_1 \equiv \delta_a$, $Y_1 \equiv 0$, $X_2 \equiv 0$, $Z_2 \equiv \delta_c + \delta_{c'}$, $Y_3 \equiv \delta_b$ and $Z_3 \equiv 0$. Then, the query function becomes $(\delta_a(x)+0)(\delta_c(z)+\delta_{c'}(z))(\delta_b(y)+0)$, which reduces to $\delta_a(x)\delta_b(y)\delta_c(z) + \delta_a(x)\delta_b(y)\delta_{c'}(z)$, that is, the function that evaluates to 1 precisely when the input is (a, b, c) or (a, b, c'). The other cases are symmetric.

$\ell = 1$: Say $\alpha_1 = (a, b, c)$ and $\alpha_2 = (a, b', c')$. We set $X_1 \equiv 1 + \delta_a$, $Y_1 \equiv \delta_b + \delta_{b'}$, $X_2 \equiv 1 + \delta_a$, $Z_2 = \delta_c + \delta_{c'}$, $Y_3 = \delta_b$ $Z_3 = \delta_{c'}$. Our query polynomial then evaluates to

$$(1 + \delta_a + \delta_b + \delta_{b'})(1 + \delta_a + \delta_c + \delta_{c'})(\delta_b + \delta_{c'}), \tag{11}$$

where, to simplify notation, we just write δ_a instead of $\delta_a(x)$, etc. Applying the rule $gh = (g + h + 1)h$ twice, we obtain $(\delta_c + \delta_{b'})(1 + \delta_a + \delta_c + \delta_{c'})(\delta_b + \delta_{c'})$. Then, combining the first and last factors, we obtain, $(\delta_b\delta_c + \delta_{b'}\delta_{c'})(1 + \delta_a + \delta_c + \delta_{c'})$. Expanding this mod 2, we obtain $(\delta_b\delta_c + \delta_{b'}\delta_{c'})\delta_a$, which yields 1 iff $x \in \{(a, b, c), (a, b', c')\}$, as required.

$\ell = 0$ **(the two elements differ on all coordinates):** We set $X_1 \equiv \delta_a$, $X_2 \equiv \delta_{a'}$, $Y_1 \equiv \delta_{b'}$, $Y_2 \equiv \delta_b$, $Z_1 \equiv \delta_{c'}$, $Z_2 \equiv \delta_c$. The query expression evaluates to

$$(\delta_a + \delta_{b'})(\delta_{a'} + \delta_c)(\delta_b + \delta_{c'}).$$

Focus on the middle factor. If we pick $\delta_{a'}$ from that factor, then we are forced to pick $\delta_{b'}$ from the previous, which forces us to pick $\delta_{c'}$ from the last (to avoid getting 0); if we pick δ_c from the middle factor, then we are forced to pick δ_b from the last and then δ_a from the first. All other terms are 0. The final expression with two terms is $\delta_a\delta_b\delta_c + \delta_{a'}\delta_{b'}\delta_{c'}$, as required.

References

1. Noga Alon and Uriel Feige. On the power of two, three and four probes. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 346–354. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496809.
2. Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. Two new schemes in the bitprobe model. In M. Sohel Rahman, Wing-Kin Sung, and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018, Proceedings*, volume 10755 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2018. doi:10.1007/978-3-319-75172-6_7.
3. Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. Improved bounds for two query adaptive bitprobe schemes storing five elements. *CoRR*, abs/1910.03651, 2019. URL: http://arxiv.org/abs/1910.03651, arXiv:1910.03651.
4. Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. Improved bounds for two query adaptive bitprobe schemes storing five elements. *Theor. Comput. Sci.*, 838:208–230, 2020. doi:10.1016/j.tcs.2020.07.036.
5. Mirza Galib Anwarul Husain Baig and Deepanjan Kesh. Two improved schemes in the bitprobe model. *Theoretical Computer Science*, 806:543–552, 2020. doi:10.1016/j.tcs.2019.08.033.
6. Mirza Galib Anwarul Husain Baig, Deepanjan Kesh, and Chirag Sodani. An improved scheme in the two query adaptive bitprobe model. In Charles J. Colbourn, Roberto Grossi, and Nadia Pisanti, editors, *Combinatorial Algorithms - 30th International Workshop, IWOCA 2019, Pisa, Italy, July 23-25, 2019, Proceedings*, volume 11638 of *Lecture Notes in Computer Science*, pages 22–34. Springer, 2019. doi:10.1007/978-3-030-25005-8_3.

7 Mirza Galib Anwarul Husain Baig, Deepanjan Kesh, and Chirag Sodani. A two query adaptive bitprobe scheme storing five elements. In Gautam K. Das, Partha Sarathi Mandal, Krishnendu Mukhopadhyaya, and Shin-Ichi Nakano, editors, *WALCOM: Algorithms and Computation - 13th International Conference, WALCOM 2019, Guwahati, India, February 27 - March 2, 2019, Proceedings*, volume 11355 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2019. doi:10.1007/978-3-030-10564-8_25.

8 Clark T. Benson. Minimal regular graphs of girths eight and twelve. *Canadian Journal of Mathematics*, 18:1091–1094, 1966. doi:10.4153/CJM-1966-109-8.

9 H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM J. Comput.*, 31(6):1723–1744, June 2002. doi:10.1137/S0097539702405292.

10 Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

11 Mohit Garg and Jaikumar Radhakrishnan. Set membership with non-adaptive bit probes. In *Proc. 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 38:1–38:13, 2017. doi:10.4230/LIPIcs.STACS.2017.38.

12 Mohit Garg and Jaikumar Radhakrishnan. Set membership with a few bit probes. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 776–784, San Diego, CA, USA, January 4-6, 2015. doi:10.1137/1.9781611973730.53.

13 Deepanjan Kesh. Space Complexity of Two Adaptive Bitprobe Schemes Storing Three Elements. In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2018.12.

14 Felix Lazebnik, V. Ustimenko, and Andrew Woldar. A new series of dense graphs of high girth. *Bull. AMS*, 32, December 1994. doi:10.1090/S0273-0979-1995-00569-0.

15 Moshe Lewenstein, J. Ian Munro, Patrick K. Nicholson, and Venkatesh Raman. Improved explicit data structures in the bitprobe model. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 630–641. Springer, 2014. doi:10.1007/978-3-662-44777-2_52.

16 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016. URL: https://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computation-and-quantum-information-10th-anniversary-edition?format=HB.

17 Jaikumar Radhakrishnan, Venkatesh Raman, and S. Srinivasa Rao. Explicit deterministic constructions for membership in the bitprobe model. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 290–299. Springer, 2001. doi:10.1007/3-540-44676-1_24.

18 Jaikumar Radhakrishnan, Pranab Sen, and Srinivasan Venkatesh. The quantum complexity of set membership. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 554–562. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892143.

19 Jaikumar Radhakrishnan, Smit Shah, and Saswata Shannigrahi. Data structures for storing small sets in the bitprobe model. In *Proc. 18th Annual European Symposium on Algorithms Algorithms*, pages 159–170, Liverpool, UK, September 6-8, 2010. doi:10.1007/978-3-642-15781-3_14.

20 R Wenger. Extremal graphs with no c4's, c6's, or c10's. *Journal of Combinatorial Theory, Series B*, 52(1):113–116, 1991. doi:10.1016/0095-8956(91)90097-4.

A Explicit construction of graphs

We say that a graph on L vertices is explicitly, if the adjacency matrix of L can be constructed in polynomial time in L.

- We will use the construction due to Wenger [20] to exhibit explicit graphs with girths 8. Wenger constructs a graph $H_k(p)$ with $2p^k$ vertices and $2p^{k+1}$ edges, and shows that if p is prime, then $H_3(p)$ has girth at least 8. In the bipartite graph $H_k(p)$ the vertices are represented as k-tuples of numbers $\{0, \ldots, p\}$ and two vertices are connected based on a simple arithmetic check involving addition and multiplication modulo p. In our application, given a number L, we set p to be the smallest prime that is at least $L^{1/(k+1)}$. Then, $H_k(p)$ has at most $2^{k+2}L$ vertices and at least $2L^{1+1/k}$ edges. Thus, we obtain graphs with the following parameters: $g = 8$, $c(g = 8) = 2^5$, $\tau(g = 8) = 1/3$. Our application for $g = 8$ (see Figure 1) uses these parameters.
- For girth 12, we use a construction due to Benson [8]. Theorem 2 of the paper presents an algebraic construction where the graph is obtained by considering point-line incidences for points and lines of a quardic surface in the projective 6-space $P(6, q)$. The degree of each vertex of this graph is $q + 1$. On page 1093, the number of vertices in this graph is computed to be $(q + 1)(1 + q^2 + q^4)$. So, to get the graph suitable for our applications, we take q to be the smallest prime such that $(q + 1)(1 + q^2 + q^4) \geq L$ and use this construction. Then, it is easy to see that the number of vertices in this graph is $\mathcal{O}(L)$ and the number of edges is at least $L^{1+1/5}$.
- Lazebnik, Ustimenko and Woldar [14] exhibit dense graphs for various values of girth. Their Corollary 3.3 shows graphs with girth at least $2s + 2$, with $v \leq 2q^{(3s-3)/2}$ vertices if s is odd and at most $2q^{(3s-2)/2}$ vertices if s is even. The graph has $\frac{1}{2}vq$ edges. To construct the graphs for our application, fix L and let q be the smallest prime larger than $L^{2/(3s-3)}$ or $L^{2/(3s-2)}$ (depending on whether s is odd or even) and consider the graph obtained from the above construction. If the graph has fewer than L vertices, then we put together disjoint copies of it, to obtain one with number of vertices between L and $2L$. It can be verified that this graph has $O(L)$ vertices and $\Omega(L^{1+2/(3s-3)})$ or $\Omega(L^{1+2/(3s-2)})$ edges (depending on whether s is odd or even). In our application (see Figure 1), we use graphs with girth $4r$ and $4r - 2$. Setting $2s + 2 = 4r$, i.e., $s = 2r - 1$ (an odd number), we obtain a graph with $\Omega(L^{1+1/(3r-3))})$ edges; similarly setting $2s + 2 = 4r - 2$, i.e., $s = 2r - 2$ (an even number), we obtain a graph with $\Omega(L^{1+1/(3r-4)})$ edges.

B Examples that show Lemma 9 is tight

The bound shown Lemma 9 is tight in the following sense: for each positive even integer g, there exists a bipartite graph with girth g and $\lfloor 4g/3 \rfloor + 1$ GREEN edges that cannot be safely oriented. For example, the graph consisting of three edge-disjoint s-t paths, each of length $2k$, has girth $g = 4k$; but we can designate a set of $n = 3k + 1$ edges GREEN for which the graph has no safe orientation. For this graph $n = 3k + 1$ and $\lfloor 3g/4 \rfloor = 3k$. A similar example, with three edge-disjoint paths of length $2k + 1$, shows that the above lemma is tight for $g = 4k + 2$. Figure 6 shows these examples for $k = 2$.

C Proof of Lemma 17

Consider the random graph on N vertices where each edge is picked independently with probability $p = (1/50)N^{-5/6}$. The probability that G is not $(4N^{1/6}, 5/4)$-locally sparse is at most (we use the union bound over the choice of subsets of size $\ell \leq 4N^{1/6}$, and for each set over all choice of 1.24ℓ edges for simplicity we ignore floors and ceilings):

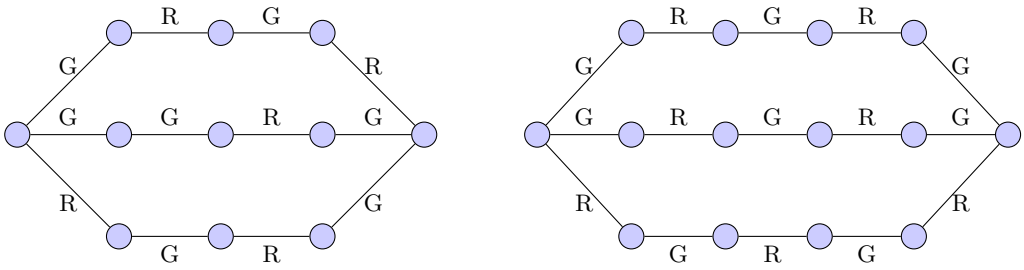


Figure 6 Examples of graphs with girth $g = 10$ and $g = 8$ that cannot be safely oriented.

$$\begin{aligned}\sum_{\ell=5}^{4N^{1/6}} \binom{N}{\ell}\binom{\ell^2}{5/4\ell} p^{(5/4)\ell} &\le \sum_{\ell=5}^{4N^{1/6}} \left(\frac{eN}{\ell}\right)^{\ell} \left(\frac{e\ell^2}{5/4\ell}\right)^{(5/4)\ell} p^{(5/4)\ell} \\ &\le \sum_{\ell=5}^{4N^{1/6}} 8^{\ell} \left(N\ell^{1/4}\right)^{\ell} p^{(5/4)\ell} \\ &\le \sum_{\ell=5}^{4N^{1/6}} \left(\frac{1}{7}\right)^{\ell} \left(N\ell^{1/4}\right)^{\ell} N^{-(5/6)(5/4)\ell}.\end{aligned}$$

By considering terms corresponding to (say) $\ell < N^{1/12}$ and $\ell \ge N^{1/12}$ separately, we see that the last sum is $o(1)$. Thus, with high probability there is no set of size up to $4N^{1/6}$ that violates the local sparsity condition. Furthermore, with high probability the number of edges in the graph is at least $pN^2/2 = \Omega(N^{7/6})$. Thus, there exists an $(4N^{1/6}, 5/4)$-sparse graph with $\Omega(N^{7/6})$ edges.

D Non-adaptive quantum bounds

We give an upper bound on $s_Q(m, n = 2k+1, t = 2)$ for the non-adaptive classical scheme, where $k \in \mathbb{N}$. The arrangement of the element and bits is similar to the classical adaptive scheme we described in Section 2. The first probe is on the corresponding edge array and the second is an equality probe on the rows corresponding to the vertices of the edge. We AND the two probes to answer membership queries. We obtain

$$s_Q(m, n = 2k+1, t = 2) = \begin{cases} \mathcal{O}(v^{1+\frac{4}{3n-9}}) & \text{if } 4|(n+1); \\ \mathcal{O}(v^{1+\frac{4}{3n-7}}) & \text{if } 4 \nmid (n+1). \end{cases} \tag{12}$$

▶ **Definition 23** (Non-adaptive Quantum (G, K)-scheme). *Let G be an un-directed graph with N vertices and M edges; let K be a positive integer. We refer to the following as a (non adaptive) quantum (G, K)-scheme. The storage consists of two bit arrays, A and B. To answer a membership query the decision tree will make the first probe to array A and the second probe to array B.*

Edge array: *An array $A : E(G) \to \{0, 1\}$, indexed by edges of G.*

Vertex array: *A two dimensional array $B : V \times [K] \to \{0, 1\}$.*

Elements: *We identify our universe of elements $[m]$ with a subset of $E(G) \times [K]$ (so we must ensure that the graph has at least m/K edges); thus, each element $x \in [m]$ will be referred to as (e, i).*

Query: *We represent an edge of G as an ordered pair of the form $e = (v_0, v_1)$. To process the query for the element $x = (e, i)$, we return the value $A[e] \cdot (B[v_0, i] \oplus B[v_1, i])$.*

Space: *We will ensure that $MK \geq m$. The space used by this scheme is then $NK + M$ bits (NK for the N vertex array and K for the edge array). By choosing the graph G and the parameter K appropriately we will show that our schemes use small space.*

As in the classical adaptive setting, an edges e is coloured GREEN if $(e, i) \in S$ for some i. Values can be assigned consistently to the arrays if there is no cycle in the graph consisting entirely of GREEN edges. This idea is formalized in the lemma below.

▶ **Lemma 24.** *Let G be a graph with N vertices M edges and girth g such that $n < g$. Then, $s_A(m, n, 2) \leq M + N\lceil m/M \rceil$.*

Hardness Results for Laplacians of Simplicial Complexes via Sparse-Linear Equation Complete Gadgets

Ming Ding
ETH Zürich, Switzerland

Rasmus Kyng
ETH Zürich, Switzerland

Maximilian Probst Gutenberg
ETH Zürich, Switzerland

Peng Zhang
Rutgers University, Piscataway, NJ, USA

Abstract

We study linear equations in combinatorial Laplacians of k-dimensional simplicial complexes (k-complexes), a natural generalization of graph Laplacians. Combinatorial Laplacians play a crucial role in homology and are a central tool in topology. Beyond this, they have various applications in data analysis and physical modeling problems. It is known that nearly-linear time solvers exist for graph Laplacians. However, nearly-linear time solvers for combinatorial Laplacians are only known for restricted classes of complexes.

This paper shows that linear equations in combinatorial Laplacians of 2-complexes are as hard to solve as general linear equations. More precisely, for any constant $c \geq 1$, if we can solve linear equations in combinatorial Laplacians of 2-complexes up to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$, then we can solve general linear equations with polynomially bounded integer coefficients and condition numbers up to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$. We prove this by a nearly-linear time reduction from general linear equations to combinatorial Laplacians of 2-complexes. Our reduction preserves the sparsity of the problem instances up to poly-logarithmic factors.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Mathematics of computing → Computations on matrices; Mathematics of computing → Algebraic topology

Keywords and phrases Simplicial Complexes, Combinatorial Laplacians, Linear Equations, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.53

Category Track A: Algorithms, Complexity and Games

Related Version *Extended Version*: `https://arxiv.org/abs/2202.05011` [14]

Funding *Rasmus Kyng*: The research leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation.
Maximilian Probst Gutenberg: The research leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 53; pp. 53:1–53:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Simplicial Complexes, Homology, and Combinatorial Laplacians

We study linear equations whose coefficient matrices are combinatorial Laplacians of k-dimensional abstract simplicial complexes (k-complexes), which generalize the well-studied graph Laplacians. An abstract simplicial complex $\mathcal{K}$ is a family of sets, known as simplices, closed under taking subsets, i.e., every subset of a set in $\mathcal{K}$ is also in $\mathcal{K}$. The dimension of $\mathcal{K}$ is the maximum dimension among all simplices in $\mathcal{K}$. A geometric notion of abstract simplicial complexes is simplicial complexes, under which a k-simplex is the convex hull of $k+1$ vertices (for example, 0,1,2-simplices are vertices, edges, and triangles, respectively). In particular, complexes in 1 dimension are graphs; combinatorial Laplacians in 1-complexes are graph Laplacians.

Nearly-linear time solvers exist for linear equations in graph Laplacians [42, 26, 27, 39, 11, 31, 24], and some natural generalizations such as connection Laplacians [29] and directed Laplacians [10, 9]. However, nearly-linear time solvers for linear equations in combinatorial Laplacians are only known for very restricted classes of 2-complexes [8, 3]. We ask whether one can extend these nearly-linear solvers to general combinatorial Laplacians.

Combinatorial Laplacians are defined via boundary operators of the chain spaces of an oriented complex. Given an oriented simplicial complex $\mathcal{K}$, a k-chain is a (signed) weighted sum of the k-simplices in $\mathcal{K}$. The boundary operator ∂_k is a linear map from the k-chain space to the $(k-1)$-chain space; in particular, it maps a k-simplex to a signed sum of its boundary $(k-1)$-simplices, where the signs are determined by the orientations of the k-simplex and its boundary $(k-1)$-simplices. For example, ∂_1 is the oriented vertex-edge incidence matrix. The combinatorial Laplacian $\mathcal{L}_k$ is defined to be $\partial_{k+1}\partial_{k+1}^\top + \partial_k^\top \partial_k$. In particular, $\mathcal{L}_0 = \partial_1 \partial_1^\top$ is the graph Laplacian.

Combinatorial Laplacians play an important role in both pure mathematics and applied areas. These matrices originate in the study of discrete Hodge decomposition [18]: The kernel of $\mathcal{L}_k$ is isomorphic to the kth homology space of $\mathcal{K}$. The properties of combinatorial Laplacians have been studied in many works [20, 15, 16, 17, 36]. A central problem in homology theory is evaluating the Betti number of the kth homology space, which equals the rank of $\mathcal{L}_k$. In the case of homology over the reals, computing the rank of $\mathcal{L}_k$ can be reduced to solving a poly-logarithmic number of linear equations in $\mathcal{L}_k$ [2]. Computation of Betti numbers over the reals is a key step in numerous problems in applied topology, computational topology, and topological data analysis [47, 21, 5, 19, 6]. In addition, combinatorial Laplacians have applications in statistical ranking [25, 45], graphics and image processing [35, 44], electromagnetism and fluid mechanics [13], data representations [7], cryo-electron microscopy [46], biology [41]. We refer to the reader to [34] for an accessible survey.

The reader may be puzzled that despite a vast literature on combinatorial Laplacians and their central role in topology, little is known about solving linear equations in these matrices except in very restricted cases [8, 3]. In this paper, we show that approximately solving linear equations in general combinatorial Laplacians is as hard as approximately solving general linear equations over the reals, which explains the lack of special-purpose solvers for this class of equations. More precisely, if one can solve linear equations in combinatorial Laplacians of general 2-complexes to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$[1] for some constant

[1] $\tilde{O}$ hides poly-logarithmic factors in following parameters of the input: ratio of maximum and minimum non-zero singular values, the maximum ratio of non-zero entries (in absolute value), and the inverse of the accuracy parameter.

$c \geq 1$, then one can solve general linear equations with polynomially bounded integer coefficients and condition numbers up to high accuracy in time $\tilde{O}((\#\text{ of nonzero coefficients})^c)$. A recent breakthrough shows that general linear equations can be solved up to high accuracy in time $\widetilde{O}((\#\text{ of nonzero coefficients})^{2.27159})$ [40, 38], which for sparse linear equations is asymptotically faster than the long-standing runtime barrier of fast matrix multiplication [43], which currently achieves a running time of $\tilde{O}(n^{2.3728596})$ [1]. Understanding the optimal value of c is a major open problem in numerical linear algebra. Our result, viewed positively, shows that one can reduce the problem of designing fast solvers for general linear equations to that for combinatorial Laplacians.

1.2 Hardness Results Based on Linear Equations

Kyng and Zhang [33] initiated the study of hardness results for solving structured linear equations. They showed that solving linear equations in a slight generalization of graph Laplacians such as 2-commodity Laplacians, 2-dimensional truss stiffness matrices, and 2-total-variation matrices is as hard as solving general linear equations.

Suppose given an invertible matrix $\boldsymbol{A}$ and a vector $\boldsymbol{b}$ over the reals, we want to approximately solve $\boldsymbol{Ax} = \boldsymbol{b}$, i.e., find $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{b}\|_2$ for some ϵ.

▶ **Definition** ((Informal) Sparse-linear-equation completeness of matrix family $\mathcal{B}$)**.** *Consider a family of matrices $\mathcal{B}$, and suppose that for any instance $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ we can produce matrix $\boldsymbol{B} \in \mathcal{B}$, vector $\boldsymbol{c}$, and accuracy parameter δ, such that if we can solve $\boldsymbol{By} = \boldsymbol{c}$ up to error δ, then we can produce $\tilde{\boldsymbol{x}}$ that solves $\boldsymbol{Ax} = \boldsymbol{b}$ to the desired accuracy.*

If, given $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$, we can compute $(\boldsymbol{B}, \boldsymbol{c}, \delta)$ in $\tilde{O}(\text{nnz}(\boldsymbol{A}))$ time with $\text{nnz}(\boldsymbol{B}) = \tilde{O}(\text{nnz}(\boldsymbol{A}))$ then we say that the class $\mathcal{B}$ is sparse-linear-equation complete*.*

In our preliminaries in Section 2, we state a formal definition of sparse-linear equation completeness that also extends to non-invertible matrices.

The reason of using the term "completeness" is that if a solver with runtime $\tilde{O}((\#\text{ of nonzero coefficients})^c)$ is known for the class $\mathcal{B}$, then a solver with runtime $\tilde{O}((\#\text{ of nonzero coefficients})^c)$ exists for general matrices. Such solvers are known for the classes of Laplacian Matrices, Directed Laplacian Matrices, Connection Laplacian Matrices, etc., all with $c = 1$. Thus, if any of these classes were sparse-linear-equation complete, we would immediately get nearly-linear time solvers for general linear equations.

In this language, Kyng and Zhang [33] showed that 2-commodity Laplacians, 2-dimensional truss stiffness matrices, and 2-total-variation matrices are all sparse-linear-equation complete. We note that [32] considered a larger family of hardness assumptions based on linear equations, which, among other things, can express weaker hardness statements based on weaker reductions.

1.3 Our Contributions

In the terminology established above, our main result can be stated very succinctly:

▶ **Theorem 1.1** (Informal First Main Theorem)**.** *Linear equations in combinatorial Laplacians of 2-complexes are sparse-linear-equation complete.*

In fact, we show this by showing an even simpler problem is sparse-linear-equation complete, namely linear equations in the boundary operator of a 2-complex.

▶ **Theorem 1.2** (Informal Second Main Theorem)**.** *Linear equations in the boundary operators ∂_2 of 2-complexes are sparse-linear-equation complete.*

This result is formally stated in Theorem 3.1. Below, in Section 1.3.1, we sketch how our first main theorem above follows from our second main theorem. We give a formal proof of this in the full version [14].

Our proof establishes the sparse-linear-equation completeness of boundary operators of a 2-complex via a two-step reduction. In our first reduction step, we show sparse-linear-equation completeness of a very simple class of linear equations which we call *difference-average equations*: these are equations where every row either restricts the difference of two variables: $\boldsymbol{x}(i) - \boldsymbol{x}(j) = b$, or it sets one variable to be the average of two others: $\boldsymbol{x}(i) + \boldsymbol{x}(j) = 2\boldsymbol{x}(k)$. This reduction was implicitly proved in [33] as an intermediate step. In this paper, we make the reduction explicit, which may be of independent interest, as this reduction class is likely to be a good starting point for many other hardness reductions. One can think of this step as analogous to showing that 3-SAT is NP-complete: It gives us a simple starting point for proving the hardness of other problems. The formal theorem statement appears in Theorem 2.9. In our second reduction step, we reduce a given difference-average equation problem to a linear equation in the boundary operator of a 2-complex.

Both the two steps preserve the number of nonzero coefficients in the linear equations up to a logarithmic factor, and only blow up the coefficients and condition numbers polynomially. The reductions are also robust to error in the sense that to solve the original problem to high accuracy, it suffices to solve the reduced problem to accuracy at most polynomially higher. Finally, we can compose the two reductions to show that solving linear equations in 2-complex boundary operators to high accuracy is as hard as solving general linear equations with polynomially bounded integer coefficients and condition numbers to high accuracy.

We give more details on both reductions below, but first we describe why solving linear equations in combinatorial Laplacians $\mathcal{L}_1$ is also sparse-linear-equation complete.

1.3.1 Hardness for Combinatorial Laplacians From Hardness for Boundary Operators

Our main technical result, Theorem 3.1, shows that the class of linear equations in the boundary operators of 2-complexes is sparse-linear-equation complete. But, as the following simple lemma shows, we can reduce the problem of solving in a boundary operator ∂_2 to solving in the corresponding combinatorial Laplacian $\mathcal{L}_1$, and hence the latter problem must be at least as hard. This then immediately implies our first main result, Theorem 1.1. The reduction is captured in the following lemma.

▶ **Lemma 1.3** (Informal reduction from boundary operators to combinatorial Laplacians in 2-complexes)**.** *Suppose we can solve linear equations in combinatorial Laplacians of 2-complexes to high accuracy in nearly-linear time. Then, we can solve linear equations in boundary operators ∂_2 of 2-complexes to high accuracy in nearly-linear time.*

The proof is by standard arguments which we sketch below. In the full version of the paper [14], we will formally state the theorem and provide a rigorous proof. Suppose we have a high-accuracy solver for combinatorial Laplacians of 2-complexes. Using this, we want to obtain a solver for linear equations in the boundary operator ∂_2. Note that when the equation $\partial_2 \boldsymbol{f} = \boldsymbol{d}$ is infeasible, we measure the solution quality by $\|\partial_2 \boldsymbol{f} - \boldsymbol{\Pi}_{\partial_2} \boldsymbol{d}\|_2$ where $\boldsymbol{\Pi}_{\partial_2}$ denotes the orthogonal projection onto the image $\mathrm{im}(\partial_2)$. The minimum over $\boldsymbol{f}$ of the quantity $\|\partial_2 \boldsymbol{f} - \boldsymbol{\Pi}_{\partial_2} \boldsymbol{d}\|_2$ is zero, which is obtained by setting $\boldsymbol{f} = \partial_2^\dagger \boldsymbol{d}$ (where $\partial_2^\dagger$ is the Moore-Penrose pseudo-inverse of ∂_2). The equation $\partial_2 \boldsymbol{f} = \boldsymbol{d}$ is feasible exactly when $\boldsymbol{\Pi}_{\partial_2} \boldsymbol{d} = \boldsymbol{d}$.

A central and basic fact in the study of simplicial homology is that $\mathrm{im}(\partial_1^\top) \cap \mathrm{im}(\partial_2) = \{\mathbf{0}\}$. This follows from $\partial_1 \partial_2 = \mathbf{0}$ (the boundary of a boundary is zero, which gives $\mathrm{im}(\partial_2) \subseteq \ker(\partial_1)$) and the general fact in linear algebra that $\ker(\boldsymbol{A})$ is the orthogonal complement of $\mathrm{im}(\boldsymbol{A}^\top)$ so

that $\ker(\partial_1)$ is orthogonal to $\mathrm{im}(\partial_1^\top)$. Intuitively, the fact that the boundary of a boundary is zero generalizes that the boundary of a disc is a circular path, and such a path has no endpoints. This implies that $\mathbf{\Pi}_{\partial_2}\partial_1^\top = \mathbf{0}$. Now, suppose that $\tilde{\boldsymbol{x}}$ approximately solves $\mathcal{L}_1\boldsymbol{x} = \boldsymbol{d}$, i.e. $\mathcal{L}_1\tilde{\boldsymbol{x}} \approx \boldsymbol{d}$. We can rewrite this as $\partial_1^\top\partial_1\tilde{\boldsymbol{x}} + \partial_2\partial_2^\top\tilde{\boldsymbol{x}} \approx \boldsymbol{d}$. Now, if we apply $\mathbf{\Pi}_{\partial_2}$ on both sides, $\mathbf{\Pi}_{\partial_2}\boldsymbol{d} \approx \mathbf{\Pi}_{\partial_2}\partial_2\partial_2^\top\tilde{\boldsymbol{x}} = \partial_2\partial_2^\top\tilde{\boldsymbol{x}}$. Thus if we set $\tilde{\boldsymbol{f}} = \partial_2^\top\tilde{\boldsymbol{x}}$, then we have $\mathbf{\Pi}_{\partial_2}\boldsymbol{d} \approx \partial_2\tilde{\boldsymbol{f}}$, which matches our notion of $\tilde{\boldsymbol{f}}$ approximately solving the (possibly infeasible) linear equation $\partial_2\boldsymbol{f} = \boldsymbol{d}$. This means that if we can approximately solve linear equations in $\mathcal{L}_1$, we can solve linear equations in ∂_2. This way we can also argue that if we can solve linear equations in $\partial_2\partial_2^\top$, then we can solve linear equations in ∂_2. Finally, one should note that $\mathrm{nnz}(\mathcal{L}_1) = O(\mathrm{nnz}(\partial_2))$ and that using our definition of condition number (see Section 2), both have polynomially related condition number[2]. This also means a high accuracy solve in one can be converted to a high accuracy solve in the other.

1.3.2 Linear Equations in $\partial_2\partial_2^\top$

In addition to the many applications discussed in Section 1.1, the problem of solving linear equations in $\partial_2\partial_2^\top$ also arises when using Interior Point Methods to solve a generalized max-flow problem in higher-dimensional simplicial complexes as defined in [36]. We sketch how this inverse problem arises when using an Interior Point Method in the full version of the paper [14]. By a similar argument as Lemma 1.3, we can show that if we can solve linear equations in $\partial_2\partial_2^\top$ to high accuracy in nearly-linear time, then we can solve linear equations in ∂_2 to high accuracy in nearly-linear time.

1.3.3 Sparse-Linear-Equation Completeness of Difference-Average Equations

Our first reduction transforms general linear equations with polynomially bounded integer entries and condition numbers into difference-average equations. We first transform a general linear equation instance to a linear equation instance such that the coefficient matrix has row sum zero and the sum of positive coefficients in each row is a power of 2, by introducing a constant number of more variables and equations. Then, we transform each single equation to a set of difference-average equations by bit-wise pairing and replacing each pair of variables with a new variable via an average equation.

1.3.4 Sparse-Linear-Equation Completeness of Boundary Operators of Simplicial Complexes

Our second reduction transforms difference-average linear equations into linear equations in the boundary operators of 2-complexes. Solving $\partial_2\boldsymbol{f} = \boldsymbol{d}$ can be interpreted as computing a flow $\boldsymbol{f}$ in the triangle space of a 2-complex subject to pre-specified edge demands $\boldsymbol{f}$.

Our reduction is inspired by a reduction in [36] that proves NP-hardness of computing maximum *integral* flows in 2-complexes via a reduction from 3-coloring problems in graphs. However, the correctness of their reduction heavily relies on that the flow values in the 2-complex are 0-1 integers, which does not apply in our setting. In addition, it is unclear how to encode linear equations as a graph coloring problem even if fractional colors are allowed.

We employ some basic building blocks used in [36] including punctured spheres and tubes. However, we need to carefully arrange and orient the triangles in the 2-complex to encode both the positive and negative coefficients in difference-average equations, and we need to express the averaging relations not covered by the previous work.

[2] This is because ∂_1 has polynomially bounded singular values.

An important aspect of our contribution is that we carefully control the number of nonzeros of the boundary operator matrix that we construct, and we bound the condition number of this matrix and how error propagates from an approximate solution to the boundary operator problem back to the original difference-average equations. In order to do so, we develop explicit triangulation algorithms that specify the precise number of triangles needed to triangulate each building block and allow a detailed error and condition number analysis.

We remark that our constructed 2-complex does not admit an embedding into a sphere in 3 dimensions. Recent work [3] has shown that simplicial complexes with a known embedding into $\mathbb{R}^3$ have non-trivial linear equation solvers, but the full extent to which embeddability can lead to better solvers remains an open question.

We analyze our construction in the Real RAM model. However, it can be transferred to the fixed point arithmetic model with $(\log N)^{O(1)}$ bits per number, where N is the size of the problem instance.

1.4 Related Works

Generalized Flows. One can generalize the notions of flows, demands vectors, circulations, and cuts to higher-dimensional simplicial complexes [17, 36]. Recall that in a graph, flows and circulations are defined on a vector space over edges, while demands and cuts are defined on a vector space over vertices. On a connected graph, a demand vector is a vector orthogonal to the all-ones vector, i.e. in the kernel of the boundary operator ∂_0. A flow that routes demand $\boldsymbol{d}$, is a vector $\boldsymbol{f}$ such that $\partial_1 \boldsymbol{f} = \boldsymbol{d}$.

More generally, on a k-complex, we say a demand vector is a vector $\boldsymbol{d}$ on $(k-1)$-simplices with $\boldsymbol{d} \in \ker(\partial_{k-1})$. We say a flow is a vector $\boldsymbol{f}$ on k-simplices, and that the flows $\boldsymbol{f}$ routes demand $\boldsymbol{d}$ if $\partial_k \boldsymbol{f} = \boldsymbol{d}$. Given a demand vector $\boldsymbol{d} \in \ker(\partial_{k-1})$ and a capacity vector $\boldsymbol{c}$ for the k-simplices, a reasonable generalization of the max-flow problem to k-complexes is to compute a flow $\boldsymbol{f}$ satisfying $\partial_k \boldsymbol{f} = \alpha \boldsymbol{d}$ and $\mathbf{0} \leq \boldsymbol{f} \leq \boldsymbol{c}$ to maximize the flow value α.

Solving Linear Equations. Linear equations are ubiquitous in computational mathematics, computer science, engineering, physics, biology, and economics. Currently, the best known algorithm for solving general dense linear equations in dimensions $n \times n$ runs in time $\tilde{O}(n^\omega)$, where $\omega < 2.3728596$ is the matrix multiplication constant [1]. For sparse linear equations with N nonzero coefficients and condition number κ, the best known approximate algorithms run in time $\widetilde{O}(\min\{N^{2.27159}, N\kappa\})$, where the first runtime is from [40, 38] and the second is by the conjugate gradient[3] [23].

In contrast to general linear equations, linear equations in graph Laplacians and its generalizations can be solved asymptotically faster, as mentioned earlier. In addition, faster solvers are also known for restricted classes of total-variation matrices [28], stiffness matrices from elliptic finite element systems [4], and 2 and 3-dimensional truss stiffness matrices [12, 30]. An interesting open question is to what extent one can generalize these faster solvers to more classes of matrices.

Reduction From Sparse Linear Equations. [32] defines a parameterized family of hypotheses for runtime of solving sparse linear equations. Under these hypotheses, they prove hardness of approximately solving packing and covering linear programs. For example, if one can solve a packing linear program up to ϵ accuracy in time $\tilde{O}(\#\text{ of nonzero coefficients} \times \epsilon^{-0.165})$, then one can solve linear equations in time asymptotically faster than $\tilde{O}(\#\text{ of nonzero coefficients} \times$ condition number of matrix), which is the runtime of conjugate gradient.

[3] If the coefficient matrix is symmetric positive semidefinite, the runtime is $\tilde{O}(N\sqrt{\kappa})$.

2 Preliminaries

2.1 Simplicial Homology

We define the basic concepts of simplicial homology. We recommend the reader the books [37] and [22] for a more complete treatment.

Simplicial Complexes. A *k-dimensional simplex* (or k-simplex) $\sigma = \text{conv}\{v_0, \ldots, v_k\}$ is the convex hull of $k+1$ affinely independent points $v_0, \ldots, v_k$. For example, 0,1,2-simplices are vertices, edges, and triangles, respectively. A *face* of σ is the convex hull of a non-empty subset of $\{v_0, v_1, \ldots, v_k\}$. An *orientation* of σ is given by an ordering Π of its vertices, written as $\sigma = [v_{\Pi(0)}, \ldots, v_{\Pi(k)}]$, such that two orderings define the same orientation if and only if they differ by an even permutation. If Π is even, then $[v_{\Pi(0)}, \ldots, v_{\Pi(k)}] = [v_0, \ldots, v_k]$; if Π is odd, then $[v_{\Pi(0)}, \ldots, v_{\Pi(k)}] = -[v_0, \ldots, v_k]$.

A *simplicial complex* $\mathcal{K}$ is a finite collection of simplices such that (1) for every $\sigma \in \mathcal{K}$ if $\tau \subset \sigma$ then $\tau \in \mathcal{K}$ and (2) for every $\sigma_1, \sigma_2 \in \mathcal{K}$, $\sigma_1 \cap \sigma_2$ is either empty or a face of both σ_1, σ_2. The *dimensions* of $\mathcal{K}$ is the maximum dimension of any simplex in $\mathcal{K}$. We refer to a simplicial complex in k dimensions as a k-complex.

Boundary Operators. A k-chain is a formal sum of the oriented k-simplices in $\mathcal{K}$ with the coefficients over $\mathbb{R}$. Let $C_k(\mathcal{K})$ denote the kth chain space. The *boundary operator* is a linear map $\partial_k : C_k(\mathcal{K}) \to C_{k-1}(\mathcal{K})$ such that for an oriented k-simplex $\sigma = [v_0, v_1, \ldots, v_k]$,

$$\partial_k(\sigma) = \sum_{i=0}^{k} (-1)^i [v_0, \ldots, \hat{v}_i, \ldots, v_k],$$

where $[v_0, \ldots, \hat{v}_i, \ldots, v_k]$ is the oriented $(k-1)$-simplex obtained by removing v_i from σ, and $(-1)^i$ is its *induced orientation*. The operator ∂_k can be written as a matrix in $n_{k-1} \times n_k$ dimensions, where n_d is the number of d-simplices in $\mathcal{K}$. The (i,j)th entry of ∂_k is ± 1 if the ith $(k-1)$-simplex is a face of the jth k-simplex where the sign is determined by the orientations, and 0 otherwise. See Figure 1 and Eq. (1) for an example.

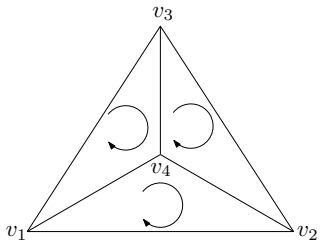


Figure 1 An example of boundary operator and oriented triangulation. We orient 2-simplices clockwise, and orient 1-simplices by the increasing order of vertex indices.

$$\partial_2 = \begin{array}{c} \\ [v_1,v_2] \\ [v_2,v_3] \\ [v_1,v_3] \\ [v_1,v_4] \\ [v_2,v_4] \\ [v_3,v_4] \end{array} \begin{array}{c} \begin{array}{ccc} [v_1,v_4,v_2] & [v_2,v_4,v_3] & [v_1,v_3,v_4] \end{array} \\ \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \end{array}. \tag{1}$$

An important property of the boundary operator is that applying the boundary operator twice results in the zero operator, i.e.,

$$\partial_{k-1}\partial_k = \mathbf{0}. \tag{2}$$

This implies $\mathrm{im}(\partial_k) \subseteq \ker(\partial_{k-1})$. Thus, we can define the quotient space $H_k = \ker(\partial_k) \setminus \mathrm{im}(\partial_{k+1})$, referred to as the kth homology space of $\mathcal{K}$. The dimension of H_k is the kth Betti number of $\mathcal{K}$, which plays an important role in understanding the homology spaces.

Hodge Theory and Combinatorial Laplacians. Combinatorial Laplacians arise from the discrete Hodge decomposition.

▶ **Theorem 2.1** (Hodge decomposition [34]). *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times p}$ be matrices satisfying $\boldsymbol{AB} = \mathbf{0}$. Then, there is an orthogonal direct sum decomposition*

$$\mathbb{R}^n = \mathrm{im}(\boldsymbol{A}^\top) \oplus \ker(\boldsymbol{A}^\top\boldsymbol{A} + \boldsymbol{B}\boldsymbol{B}^\top) \oplus \mathrm{im}(\boldsymbol{B}).$$

By Eq. (2), it is valid to set $\boldsymbol{A} = \partial_k$ and $\boldsymbol{B} = \partial_{k+1}$. The matrix we get in the middle term is the *combinatorial Laplacian*: $\mathcal{L}_k \stackrel{\text{def}}{=} \partial_k^\top\partial_k + \partial_{k+1}\partial_{k+1}^\top$. In particular, $\mathcal{L}_0 = \partial_1\partial_1^\top$ is the graph Laplacian. The kth homology space $H_k(\mathcal{K})$ is isomorphic to $\ker(\mathcal{L}_k)$, and thus the kth Betti number of $\mathcal{K}$ equals the dimension of $\ker(\mathcal{L}_k)$.

Triangulation. A *triangulation* of a topological space $\mathcal{X}$ is a simplicial complex $\mathcal{K}$ together with a homeomorphism between $\mathcal{X}$ and $\mathcal{K}$. In this paper, the only topological spaces that we compute triangulations of are 2-dimensional manifolds. A 2-dimensional manifold can be triangulated by a 2-complex, where every edge in the 2-complex is contained in exactly one triangle (boundary edge) or two triangles (interior edge). An *oriented triangulation* of a 2-dimensional manifold is a triangulation together with an orientation for each triangle such that any two neighboring triangles induce opposite signs on their shared interior edge.

Figure 1 is an example of (oriented) triangulation: the topological space is a disk; boundary edges are $[v_1, v_2], [v_2, v_3], [v_1, v_3]$; interior edges are $[v_1, v_4], [v_2, v_4], [v_3, v_4]$; the orientation of each triangle is clockwise.

2.2 Notation for Matrices and Vectors

We use parentheses to denote entries of a matrix or a vector: Let $\boldsymbol{A}(i,j)$ bet the (i,j)th entry of a matrix $\boldsymbol{A}$, and let $\boldsymbol{x}(i)$ bet the ith entry of a vector $\boldsymbol{x}$. We use $\mathbf{1}_n, \mathbf{0}_n$ to denote n-dimensional all-one vector and all-zero vector, respectively. We define $\|\boldsymbol{x}\|_{\max} = \max_{i\in[n]} |\boldsymbol{x}(i)|$, $\|\boldsymbol{x}\|_1 = \sum_{i\in[n]} |\boldsymbol{x}(i)|$. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, we use $\boldsymbol{A}(i)$ to denote the ith row of $\boldsymbol{A}$ and $\mathrm{nnz}(\boldsymbol{A})$ the number of nonzero entries of $\boldsymbol{A}$. Without loss of generality, we assume that $\mathrm{nnz}(\boldsymbol{A}) \geq \max\{d, n\}$. We let $\|\boldsymbol{A}\|_{\max} = \max_{i,j} |\boldsymbol{A}(i,j)|$. We use $\mathrm{im}(\boldsymbol{A})$ to denote the image (i.e., the column space) of $\boldsymbol{A}$ and $\mathrm{null}(\boldsymbol{A})$ the null space of $\boldsymbol{A}$. We let $\boldsymbol{\Pi}_{\boldsymbol{A}} = \boldsymbol{A}(\boldsymbol{A}\boldsymbol{A}^\top)^\dagger\boldsymbol{A}^\top$ be the orthogonal projection onto $\mathrm{im}(\boldsymbol{A})$, where $\boldsymbol{M}^\dagger$ is the pseudo-inverse of $\boldsymbol{M}$. Let $\lambda_{\max}(\boldsymbol{A})$ be the maximum eigenvalue of $\boldsymbol{A}$ and $\lambda_{\min}(\boldsymbol{A})$ the minimum *nonzero* eigenvalue of $\boldsymbol{A}$. Similarly, let $\sigma_{\max}(\boldsymbol{A})$ be the maximum eigenvalue of $\boldsymbol{A}$ and $\sigma_{\min}(\boldsymbol{A})$ the minimum *nonzero* singular value of $\boldsymbol{A}$. The condition number of $\boldsymbol{A}$, denoted by $\kappa(\boldsymbol{A})$, is the ratio of the maximum to the minimum *nonzero* singular value of $\boldsymbol{A}$.

We define a function U that takes a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{b}$ as arguments and returns the maximum of $\|\cdot\|_{\max}$ of all the arguments, that is, $U(\boldsymbol{A}, \boldsymbol{b}) = \max\{\|\boldsymbol{A}\|_{\max}, \|\boldsymbol{b}\|_{\max}\}$.

2.3 Systems of Linear Equations

We define approximately solving linear equations in a general form, following [33]. For more details, we refer the readers to Section 2.1 of [33].

▶ **Definition 2.2** (Linear Equation Problem (LE)). *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, a vector $\boldsymbol{b} \in \mathbb{R}^d$, we refer to the linear equation problem for the tuple $(\boldsymbol{A}, \boldsymbol{b})$, denoted by* LE $(\boldsymbol{A}, \boldsymbol{b})$*, as the problem of finding an $\boldsymbol{x} \in \mathbb{R}^n$ such that $\boldsymbol{x} \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$.*

▶ **Fact 2.3.** *Let $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$. Then,*

$$\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^\dagger\boldsymbol{A}^\top\boldsymbol{b} = \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b} \quad \text{and} \quad \|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\|_2^2 = \|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{A}})\boldsymbol{b}\|_2^2 .$$

By the above fact, solving LE $(\boldsymbol{A}, \boldsymbol{b})$ is equivalent to finding an $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}$. This equation is known as the normal equation, and it is always feasible. If $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$, then $\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b} = \boldsymbol{b}$.

In practice, we are more interested in *approximately* solving linear equations, since numerical errors are unavoidably in data collection and computation and approximate solvers may run faster.

▶ **Definition 2.4** (Linear Equation Approximation Problem (LEA)). *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, vectors $\boldsymbol{b} \in \mathbb{R}^d$, and an error parameter $\epsilon \in (0, 1]$, we refer to linear equation approximate problem for the tuple $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$, denoted by* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$*, as the problem of finding an $\boldsymbol{x} \in \mathbb{R}^n$ such that $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2 \le \epsilon \|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2$.*

▶ **Fact 2.5.** *Let $\boldsymbol{x}$ be a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$*. Then,*

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \le \|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\|_2^2 + \epsilon^2 \|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2^2 .$$

The definition of the approximate error in Definition 2.4 is equivalent to several error notions that are commonly used in solving linear equations. In particular,

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2 = \left\|\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top\boldsymbol{b}\right\|_{(\boldsymbol{A}^\top\boldsymbol{A})^\dagger} = \|\boldsymbol{x} - \boldsymbol{x}^*\|_{\boldsymbol{A}^\top\boldsymbol{A}} .$$

2.3.1 Matrix Classes

We are interested in linear equations whose coefficient matrices belonging to the following matrix classes.

1. $\mathcal{G}$ refers to the class of *General Matrices* that have integer entries and do not have all-0 rows and all-0 columns. We refer to linear equations whose coefficient matrix is in $\mathcal{G}$ as *general linear equations*.
2. $\mathcal{DA}$ refers to the class of *Difference-Average Matrices* whose rows fall into two categories:
   a. A *difference row* which has exactly two nonzero entries 1 and -1;
   b. An *average row* which has exactly three nonzero entries 1, 1, and -2.

   Multiplying a difference row vector to a column vector $\boldsymbol{x}$ gives $\boldsymbol{x}(i) - \boldsymbol{x}(j)$; multiplying an average row vector to $\boldsymbol{x}$ gives $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k)$. We refer to linear equations whose coefficient matrix is in $\mathcal{DA}$ as *difference-average linear equations*.
3. $\mathcal{B}_2$ refers to the class of *Boundary Operator Matrices ∂_2 in 2-complexes*. We refer to linear equations whose coefficient matrix is in $\mathcal{B}_2$ as *2-complex boundary linear equations*.

Our definition of "general matrices" specifies the matrix must have integer entries. However, when the input matrix is invertible, using a simple rounding argument, we can convert any linear equation into an linear equation with integer entries $\widetilde{O}(1)$ bits per entry. We caution the reader this relies on our definition of $\widetilde{O}(\cdot)$ as hiding polylogarithmic factors in the input condition number. In general, the condition number can be exponentially large – however, our results are mainly of interest when the condition number is quasipolynomially bounded.

2.3.2 Reduction Between Linear Equations

We will again follow the definition of efficient reductions in [33]. We say LEA over matrix class $\mathcal{M}_1$ is *nearly-linear time reducible* to LEA over matrix class $\mathcal{M}_2$, denoted by $\mathcal{M}_1 \leq_{nlt} \mathcal{M}_2$, if the following holds:

1. There is an algorithm that maps an arbitrary instance LEA $(\boldsymbol{M}_1, \boldsymbol{c}_1, \epsilon_1)$ where $\boldsymbol{M}_1 \in \mathcal{M}_1$ to an instance LEA $(\boldsymbol{M}_2, \boldsymbol{c}_2, \epsilon_2)$ where $\boldsymbol{M}_2 \in \mathcal{M}_2$ such that there is another algorithm that can map a solution to LEA $(\boldsymbol{M}_2, \boldsymbol{c}_2, \epsilon_2)$ to a solution to LEA $(\boldsymbol{M}_1, \boldsymbol{c}_1, \epsilon_1)$.
2. Both the two algorithms run in time $\widetilde{O}(\mathrm{nnz}(\boldsymbol{M}_1))$.
3. In addition, we can guarantee $\mathrm{nnz}(\boldsymbol{M}_2) = \widetilde{O}(\mathrm{nnz}(\boldsymbol{M}_1))$, and $\epsilon_2^{-1}, \kappa(\boldsymbol{M}_2), U(\boldsymbol{M}_2, \boldsymbol{b}_2) = \mathrm{poly}(\mathrm{nnz}(\boldsymbol{M}_1), \epsilon_1^{-1}, \kappa(\boldsymbol{M}_1), U(\boldsymbol{M}_1, \boldsymbol{b}_1))$.

We do not require a nearly-linear time reduction to preserve the number of variables or constraints (dimensions) of linear equations. The dimensions of the new linear equation instance that we construct can be much larger than that of the original instance. On the other hand, a reduction that *only* preserves dimensions may construct a dense linear equation instance even if the original instance is sparse. A nearly-linear time reduction that preserves *both* the number of nonzeros and dimensions would be stronger than what we achieve.

▶ **Fact 2.6.** *If $\mathcal{M}_1 \leq_{nlt} \mathcal{M}_2$ and $\mathcal{M}_2 \leq_{nlt} \mathcal{M}_3$, then $\mathcal{M}_1 \leq_{nlt} \mathcal{M}_3$.*

▶ **Definition 2.7** (Sparse linear equation complete (SLE-complete))**.** *We say LEA over a matrix class $\mathcal{M}$ is* sparse-linear-equation-complete *if $\mathcal{G} \leq_{nlt} \mathcal{M}$.*

▶ **Fact 2.8.** *Suppose LEA over $\mathcal{M}$ is SLE-complete. If one can solve all instances LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ with $\boldsymbol{A} \in \mathcal{M}$ in time $\tilde{O}(\mathrm{nnz}(\boldsymbol{A})^c)$ where $c \geq 1$, then one can solve all instances LEA $(\boldsymbol{A}', \boldsymbol{b}', \epsilon')$ with $\boldsymbol{A}' \in \mathcal{G}$ in time $\tilde{O}(\mathrm{nnz}(\boldsymbol{A}')^c)$.*

Under the above definitions, [33] implicitly shows the following results. We provide an explicit and simplified proof in the full version of the paper [14].

▶ **Theorem 2.9** (Implicitly stated in [33])**.** *LEA over $\mathcal{DA}$ is SLE-complete.*

3 Main Results

Our main result is stated in the following theorem.

▶ **Theorem 3.1.** *LEA over $\mathcal{B}_2$ is SLE-complete.*

Theorem 3.1 states that a linear equation approximation problem over a boundary operator of a 2-complex is sparse linear equation complete. See Section 2 for detailed definitions.

Although our main theorem focuses on linear equation approximate problems, we construct nearly-linear time reductions for both linear equation problem LE and its approximate counterpart LEA. We first reduce LE instances $(\boldsymbol{A}, \boldsymbol{b})$ (and LEA instances $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$) over difference-average matrices to those over 2-complex boundary operator matrices, *under the assumption* $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$ (stated in Theorem 3.2 and 3.3). In this case, the constructed 2-complexes have unit edge weights. We then provide a slightly modified nearly-linear time reduction for LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon)$ over difference-average matrices to LEA over 2-complex boundary operator matrices *without assuming* $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$ (stated in Theorem 3.4). In this case, we introduce polynomially bounded edge weights for the constructed 2-complexes.

▶ **Theorem 3.2.** *Given a linear equation instance LE $(\boldsymbol{A}, \boldsymbol{b})$ where $\boldsymbol{A} \in \mathcal{DA}$ and $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$, we can reduce it to an instance LE (∂_2, γ) where $\partial_2 \in \mathcal{B}_2$, in time $O(\mathrm{nnz}(\boldsymbol{A}))$, such that a solution to LE (∂_2, γ) can be mapped to a solution to LE $(\boldsymbol{A}, \boldsymbol{b})$ in time $O(\mathrm{nnz}(\boldsymbol{A}))$.*

▶ **Theorem 3.3.** *Given a linear equation instance* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *where* $\boldsymbol{A} \in \mathcal{DA}$ *and* $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$*, we can reduce it to an instance* LEA $(\partial_2, \gamma, \epsilon^{B_2})$ *where* $\partial_2 \in \mathcal{B}_2$ *and* $\epsilon^{B_2} \leq \frac{\epsilon^{DA}}{42\,\mathrm{nnz}(\boldsymbol{A})}$*, in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*, such that a solution to* LEA $(\partial_2, \gamma, \epsilon^{B_2})$ *can be mapped to a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*.*

▶ **Theorem 3.4.** *Given an instance* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *where* $\boldsymbol{A} \in \mathcal{DA}$*, we can reduce it to an instance* LEA $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma, \epsilon^{B_2})$ *where* $\partial_2 \in \mathcal{B}_2$ *and* $\boldsymbol{W}$ *is a diagonal matrix with nonnegative diagonals, in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*. Let* s, ϵ, K, U *denote* $\mathrm{nnz}(\boldsymbol{A}), \epsilon^{DA}, \kappa(\boldsymbol{A}), U(\boldsymbol{A}, \boldsymbol{b})$*, respectively. Then, we can guarantee that*

$$\mathrm{nnz}(\partial_2) = O(s),\ U(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma) = O\left(sU\epsilon^{-1}\right),$$
$$\epsilon^{B_2} = \Omega(\epsilon U^{-1}s^{-1}),\ \kappa(\boldsymbol{W}^{1/2}\partial_2) = O\left(s^{15/2}K^2\epsilon^{-2}\right)$$

and a solution to LEA $(\boldsymbol{W}^{1/2}\partial_2, \boldsymbol{W}^{1/2}\gamma, \epsilon^{B_2})$ *can be mapped to a solution to* LEA $(\boldsymbol{A}, \boldsymbol{b}, \epsilon^{DA})$ *in time* $O(\mathrm{nnz}(\boldsymbol{A}))$*.*

We refer the reader to the full version of the paper [14] for a formal proof of Theorem 3.2, 3.3, and 3.4.

3.1 Overview of Our Proof

Multiplying a 2-complex boundary operator $\partial_2 \in \mathbb{R}^{m\times t}$ to a vector $\boldsymbol{f} \in \mathbb{R}^t$ can be interpreted as transforming flows in the triangle space to demands in the edge space. Given $\gamma \in \mathbb{R}^m$, solving LE (∂_2, γ) can be interpreted as finding flows $\boldsymbol{f}$ in the triangle space subject to edge demands γ. We will encode difference-average linear equations as a 2-complex flow network.

Encoding Linear Operations. In difference-average linear equations, linear operations include subtraction, addition, and multiplication. We observe a simple fact: If we glue two triangles Δ_1, Δ_2 with the same orientation, then the net flow $\partial_2\boldsymbol{f}$ on the shared edge is $\boldsymbol{f}(\Delta_1) - \boldsymbol{f}(\Delta_2)$ (see Figure 2 (a)); if we glue two triangles Δ_1, Δ_2 with opposite orientations, then the net flow $\partial_2\boldsymbol{f}$ on the shared edge is $\boldsymbol{f}(\Delta_1) + \boldsymbol{f}(\Delta_2)$ (see Figure 2 (b)). Given an equation $\boldsymbol{a}^\top\boldsymbol{x} = b$ with the nonzero coefficients being ± 1, we can encode it by gluing more triangles as above and setting the demand of the shared edge to be b. To handle the coefficient -2 in an average equation, say $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k)$, we implicitly interpret it as $\boldsymbol{x}(i) + \boldsymbol{x}(j) - \boldsymbol{x}(k_1) - \boldsymbol{x}(k_2)$ together with an additional difference equation $\boldsymbol{x}(k_1) = \boldsymbol{x}(k_2)$ (see Figure 2 (c)).

Encoding a Variable. We use a sphere to encode a variable involved in many equations. We can obtain an oriented triangulation of the sphere and set all the edge demand to be 0 so that all the triangles on the sphere must have an equal flow value (see Figure 2 (d)).

Putting It All Together. We represent each variable using a triangulated sphere. To add a constraint between variables, in each corresponding sphere we add a "hole" if the variable coefficient is ± 1 and add two "holes" if the coefficient is 2. Then we attach a "tube" to each of these holes. Similarly, we can obtain an oriented triangulation of each tube and set the edge demand properly so that all the triangles on the tube must have the same flow value to the triangles on the sphere that the tube is attached to. We then connect these tubes corresponding to different variables so that they share common edges. Depending on how the tubes connect and on the net flow demand on the shared edges, we can represent different linear constraints on the variables.

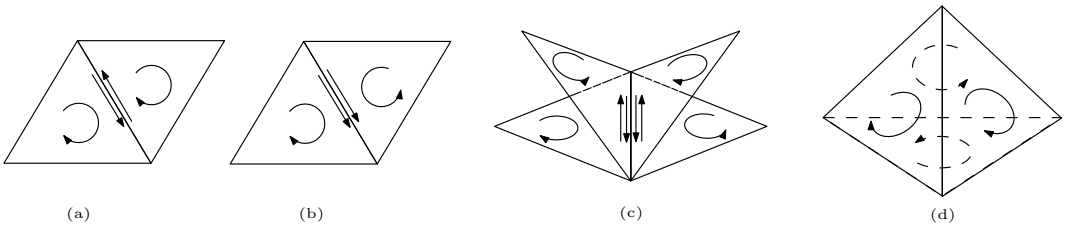


Figure 2 An illustration for encoding linear operations and variables: (a) encodes the "subtraction" with two identically oriented triangles. (b) encodes the "addition" with two oppositely oriented triangles. (c) encodes the "multiplication" by a coefficient 2 in an average equation. (d) encodes a variable with an identically oriented triangulated sphere, and this will later allow us to use several "copies" of this variable, by making holes in this sphere and attaching a tube to each such hole.

Discussion.

1. *Why encode difference/average equations rather than directly encoding general equations with integer coefficients?*
   We can generalize the above encoding method to encode a general equation $\boldsymbol{g}^\top \boldsymbol{y} = c$ with arbitrary integer coefficients into a 2-complex with roughly $\|\boldsymbol{g}\|_1$ tubes. However, the encoding size required to express a general system of linear equations $\boldsymbol{G}\boldsymbol{y} = \boldsymbol{c}$ this way can be as large as $\Omega(\mathrm{nnz}(\boldsymbol{G})\,\|\boldsymbol{G}\|_{\max})$. This dependence on $\|\boldsymbol{G}\|_{\max}$ is prohibitive, and makes for a fairly weak result.

   On the other hand, we can first reduce the general linear equations $\boldsymbol{G}\boldsymbol{y} = \boldsymbol{c}$ into difference-average linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\|\boldsymbol{A}\|_{\max} = 2$ and $\mathrm{nnz}(\boldsymbol{A}) = O\left(\mathrm{nnz}(\boldsymbol{G}) \log \|\boldsymbol{G}\|_{\max}\right)$ (by Lemma A.1 in [14]). Then we can encode $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ into a 2-complex. The encoding size required to express the the difference-average linear equations as a 2-complex is thus $O(\mathrm{nnz}(\boldsymbol{A}))$ (by Lemma 4.2). Thus, the overall encoding size required to express the original linear equation $\boldsymbol{G}\boldsymbol{y} = \boldsymbol{c}$ is now $\widetilde{O}\left(\mathrm{nnz}(\boldsymbol{G})\right)$, exponentially improving the dependence on $\|\boldsymbol{G}\|_{\max}$.

   Therefore, the two-step reduction is a nearly-linear time reduction while the one-step reduction is not.

2. *Why encode into a 2-complex rather than a 1-complex?*
   We do not expect that general linear equations with integer coefficients can be efficiently encoded using a 1-complex. This would immediately imply a nearly-linear time solver for general linear equations, as fast solvers for 1-complex operators exist (using Laplacian linear equation solvers).

4 Reducing Exact Solvers for $\mathcal{DA}$ to $\mathcal{B}_2$ Assuming the Right-Hand Side Vector in the Image of the Coefficient Matrix

In this section, we describe a nearly-linear time reduction from instances LE $(\boldsymbol{A}, \boldsymbol{b})$ over $\mathcal{DA}$ to instances LE over $\mathcal{B}_2$, under the assumption that $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$, and analyze its runtime and the size of the reduced problem. In the full version of the paper [14], we show that the same reduction with a carefully chosen error parameter reduces linear equation approximate problem LEA over $\mathcal{DA}$ to LEA over $\mathcal{B}_2$, assuming $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$. In addition, we slightly modify the reduction algorithm to drop the assumption $\boldsymbol{b} \in \mathrm{im}(\boldsymbol{A})$ for LEA.

4.1 Reduction Algorithm

Recall that an instance LE $(\boldsymbol{A}, \boldsymbol{b})$ over $\mathcal{DA}$ only consists of two types of linear equations: (1) *Difference equation*: $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$; (2) *Average equation*: $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k) = 0$. Suppose LE $(\boldsymbol{A}, \boldsymbol{b})$ has d_1 difference equations and d_2 average equations. Without loss of generality, we reorder all the equations so that the first d_1 equations are difference equations and the rest are average equations. The following algorithm ReduceDATo$\mathcal{B}_2$ constructs a 2-complex and a system of linear equations in its boundary operator.

Algorithm Reduce$\mathcal{DA}$To$\mathcal{B}_2$.
Input: an instance LE $(\boldsymbol{A}, \boldsymbol{b})$ where $\boldsymbol{A} \in \mathcal{DA}$ is a $d \times n$ matrix and $\boldsymbol{b} \in \mathbb{R}^d$.
Output: $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ where $\partial_2 \in \mathcal{B}_2$ is an $m \times t$ matrix, $\gamma \in \mathbb{R}^m$, and $\boldsymbol{\Delta}^c$ is a set of n triangles.

1. For each $i \in [n]$ and variable $\boldsymbol{x}(i)$ in LE $(\boldsymbol{A}, \boldsymbol{b})$, we construct a sphere $\mathcal{S}_i$.
2. For each $q \in [d_1]$, which corresponds to a difference equation $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$, we add a *loop*[4] α_q with a net flow demand $\boldsymbol{b}(q)$. Then,
   a. we add a *boundary component*[5] $\beta_{q,i}$ on $\mathcal{S}_i$, and a boundary component $\beta_{q,j}$ on $\mathcal{S}_j$;
   b. we construct a tube $\mathcal{T}_{q,i}$ with boundary components $\{-\beta_{q,i}, \alpha_q\}$, and a tube $\mathcal{T}_{q,j}$ with boundary components $\{-\beta_{q,j}, -\alpha_q\}$.

   See Figure 3 for an illustration[6].

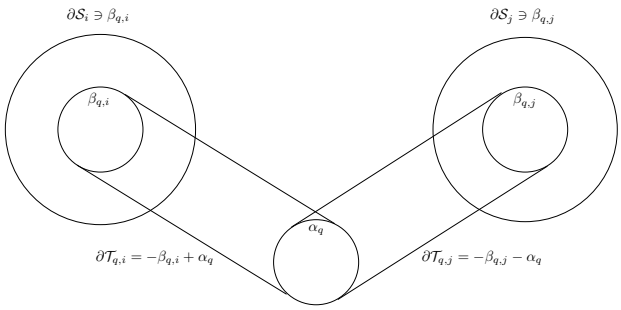


Figure 3 The construction for a difference equation $\boldsymbol{x}(i) - \boldsymbol{x}(j) = \boldsymbol{b}(q)$. For a topological space X, we use ∂X to denote its boundary.

3. For each $q \in \{d_1+1, \ldots, d\}$, which corresponds to an average equation $\boldsymbol{x}(i)+\boldsymbol{x}(j)-2\boldsymbol{x}(k) = \boldsymbol{b}(q) = 0$, we add a loop α_q with zero net flow demand. Then,
   a. we add a boundary component $\beta_{q,i}$ on $\mathcal{S}_i$, a boundary component $\beta_{q,j}$ on $\mathcal{S}_j$, and two boundary components $\beta_{q,k,1}, \beta_{q,k,2}$ on $\mathcal{S}_k$;
   b. we construct a tube $\mathcal{T}_{q,i}$ with boundary components $\{-\beta_{q,i}, \alpha_q\}$, a tube $\mathcal{T}_{q,j}$ with boundary components $\{-\beta_{q,j}, \alpha_q\}$, and two tubes $\mathcal{T}_{q,k,1}, \mathcal{T}_{q,k,2}$ with boundary components $\{-\beta_{q,k,1}, -\alpha_q\}$ and $\{-\beta_{q,k,2}, -\alpha_q\}$, respectively.

   See Figure 4 for an illustration[7].

[4] In topology, a loop in a topological space X is a path whose initial point is equal to its terminal point.
[5] In topology, a boundary of a topological space X is a set of points that can be approached from both X and the outside of X. A boundary component is a connected component of the boundary. Here, sphere $\mathcal{S}_i$ does not have a boundary, but we can hollow a "hole" by adding a boundary component.
[6] Note that since the loop α_q has demand $\boldsymbol{b}(q)$, our construction is different from identifying the boundary component α_q of $\mathcal{T}_{q,i}$ and the boundary component $-\alpha_q$ of $\mathcal{T}_{q,j}$.
[7] As four tubes are connected to a single loop, to avoid the intersection of tubes before attaching the loop, a higher-dimensional space is required.

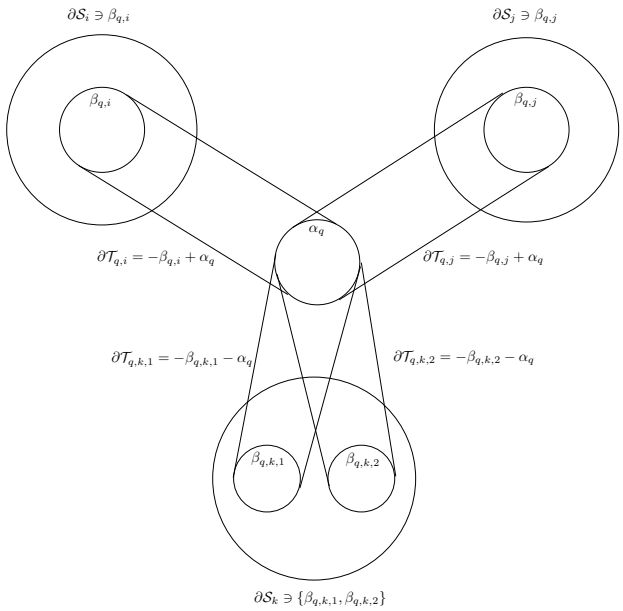


Figure 4 The construction for an average equation $\boldsymbol{x}(i) + \boldsymbol{x}(j) - 2\boldsymbol{x}(k) = 0$.

4. For each $i \in [n]$, the punctured sphere $\mathcal{S}_i$ and the tubes connected to $\mathcal{S}_i$ form a continuous topological space. We construct an oriented triangulation for this space such that the induced orientation of each edge on a loop α_q is consistent with the orientation of α_q. We will describe this oriented triangulation subroutine in Section 4.1.1. Let $\mathcal{K}$ be the oriented 2-complex. Let ∂_2 be the boundary operator of $\mathcal{K}$.
5. Each edge on a loop α_q has net demand $\boldsymbol{b}(q)$; each other edge has net demand 0. Let γ be the vector of the net flow demands.
6. On each triangulated sphere $\mathcal{S}_i$, we choose an arbitrary triangle $\Delta_i \in \mathcal{S}_i$ as the *central triangle*. Let $\boldsymbol{\Delta}^c$ be the set of all the central triangles.
7. We return $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$.

The following algorithm MapSoln$\mathcal{B}_2$to$\mathcal{DA}$ maps a solution $\boldsymbol{f}$ to LE (∂_2, γ) to a solution $\boldsymbol{x}$ to LE $(\boldsymbol{A}, \boldsymbol{b})$.

Algorithm MapSoln$\mathcal{B}_2$to$\mathcal{DA}$.
Input: a tuple $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$, where $\boldsymbol{A} \in \mathcal{DA}$ is a $d \times n$ matrix, $\boldsymbol{b} \in \mathbb{R}^d$, $\boldsymbol{f} \in \mathbb{R}^t$, and $\boldsymbol{\Delta}^c$ is the set of n central triangles.
Output: a vector $\boldsymbol{x} \in \mathbb{R}^n$.
1. If $\boldsymbol{A}^\top \boldsymbol{b} = 0$, we return $\boldsymbol{x} = \boldsymbol{0}$.
2. Otherwise, we set $\boldsymbol{x}(i) = \boldsymbol{f}(\Delta_i)$, where $\Delta_i \in \boldsymbol{\Delta}^c$ is the central triangle on sphere $\mathcal{S}_i$.

4.1.1 Oriented Triangulation for Punctured Spheres and Tubes

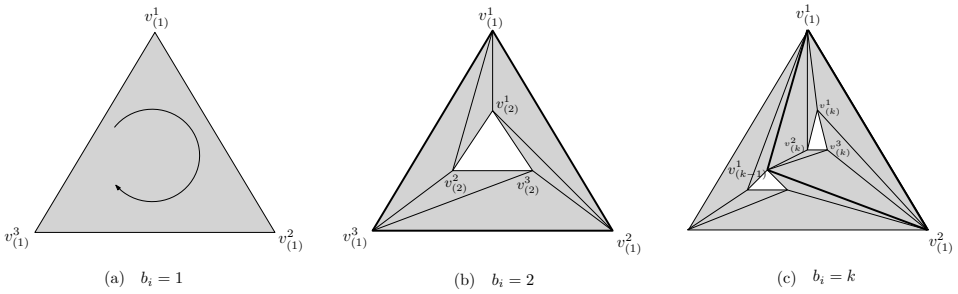
We provide a concrete triangulation subroutine for the benefit of algorithm analysis.

Oriented Triangulation for Punctured Spheres

By our construction, each sphere $\mathcal{S}_i$ has $b_i = \sum_{q=1}^{d} |\boldsymbol{A}(q, i)|$ boundary components. We will create $\tilde{t}_i$ triangles and $\tilde{m}_i$ edges on $\mathcal{S}_i$, based on b_i.

1. If $b_i = 1$ (see Figure 5 (a)), the punctured sphere is topologically equivalent to a disk. In this case, $\mathcal{S}_i$ can be triangulated using a single triangle $[v^1_{(1)}, v^2_{(1)}, v^3_{(1)}]$, thus $\tilde{t}_i = 1, \tilde{m}_i = 3$.
2. If $b_i = 2$ (see Figure 5 (b)), the punctured sphere is topologically equivalent to an annulus. We subdivide the triangle $[v^1_{(1)}, v^2_{(1)}, v^3_{(1)}]$ obtained in the previous case by adding 6 interior edges between vertices of the inner and the outer boundaries: $[v^1_{(1)}, v^1_{(2)}], [v^1_{(1)}, v^2_{(2)}], [v^2_{(1)}, v^1_{(2)}], [v^2_{(1)}, v^3_{(2)}], [v^3_{(1)}, v^2_{(2)}], [v^3_{(1)}, v^3_{(2)}]$, thus $\tilde{t}_i = 6, \tilde{m}_i = 12$.
3. Generally, if $b_i = k$ (see Figure 5 (c)), we subdivide the rightmost triangle $[v^1_{(1)}, v^2_{(1)}, v^1_{(k-1)}]$ obtained in the case of $b_i = k - 1$ with the same method. By induction, we have

$$\tilde{t}_i = 5b_i - 4, \qquad \tilde{m}_i = 9b_i - 6, \qquad \text{for} \quad b_i \geq 1. \tag{3}$$



(a) $b_i = 1$ (b) $b_i = 2$ (c) $b_i = k$

Figure 5 Oriented triangulation of punctured spheres. The light area represents the "holes" defined by boundary components.

The orientation for triangles on the same sphere should be identical. Without loss of generality, we orient all triangles clockwise. Note that with this triangulation method, all boundary components are composed of 3 edges.

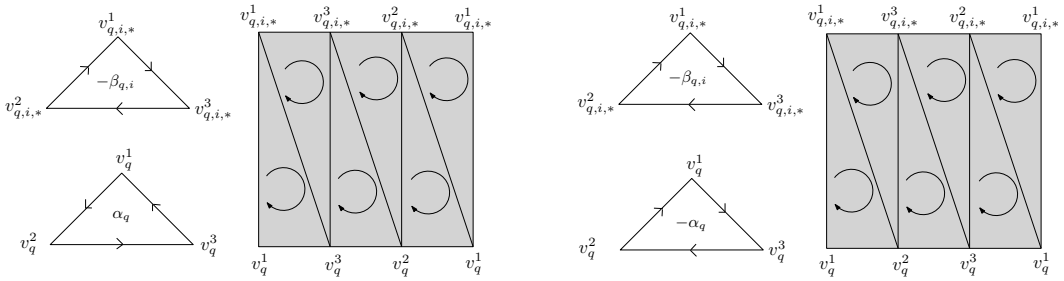
Oriented Triangulation for Tubes

A tube is defined by two boundary components. By our construction, for every tube connected to $\mathcal{S}_i$, one of the two boundary components is always $-\beta_{q,i,*}$[8], and the other one is $\pm\alpha_q$, whose orientation depends on the sign of the entry $\boldsymbol{A}(q,i)$. Without loss of generality, we orient anti-clockwise for all α_q, thus clockwise for all $-\alpha_q$. Therefore, there are two possibilities of boundary component combinations.

1. If $\boldsymbol{A}(q,i) > 0$ (see Figure 6 (a)), then the two boundary components have opposite orientations: $-\beta_{q,i,*} = [v^1_{q,i,*}, v^3_{q,i,*}, v^2_{q,i,*}]$ and $\alpha_q = [v^1_q, v^2_q, v^3_q]$. We triangulate by matching $v^1_{q,i,*}$ to v^1_q, $v^2_{q,i,*}$ to v^2_q, and $v^3_{q,i,*}$ to v^3_q.
2. If $\boldsymbol{A}(q,i) < 0$ (see Figure 6 (b)), then the two boundary components have identical orientations: $-\beta_{q,i,*} = [v^1_{q,i,*}, v^3_{q,i,*}, v^2_{q,i,*}]$ and $-\alpha_q = [v^1_q, v^3_q, v^2_q]$. We triangulate by matching $v^1_{q,i,*}$ to v^1_q, $v^3_{q,i,*}$ to v^2_q, and $v^2_{q,i,*}$ to v^3_q.

In either case, only 6 triangles and 12 edges are required for an oriented triangulation of any tube $\mathcal{T}_{q,i,*}$. Again, we orient all triangles clockwise.

[8] We introduce a third element $* \in \{1, 2\}$ in the subscript of $\beta_{q,k,*}$, which is activated only when $\boldsymbol{A}(q,k) = -2$.



(a) $\mathbf{A}(q,i) > 0$ (b) $\mathbf{A}(q,i) < 0$

Figure 6 Oriented triangulation of tubes with opposite or identical boundary orientations.

4.2 Algorithm Runtime and Problem Size

In this section, we show that the reduction algorithm ReduceDAToB2 and the solution mapping algorithm MapSolnB2toDA both run in linear time, and ReduceDAToB2 constructs a 2-complex whose size is linear in the number of nonzeros in the input linear equations.

▶ **Lemma 4.1** (Runtime). *Given a difference-average instance* LE $(\boldsymbol{A}, \boldsymbol{b})$ *where* $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, *Algorithm* ReduceDAToB2$(\boldsymbol{A}, \boldsymbol{b})$ *returns* $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ *in time* $O(\mathrm{nnz}(\boldsymbol{A}))$. *Given a solution* $\boldsymbol{f}$ *to* LE (∂_2, γ), *Algorithm* MapSolnB2toDA$(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{\Delta}^c)$ *returns* $\boldsymbol{x}$ *in time* $O(n)$.

Proof. For reduction, ReduceDAToB2$(\boldsymbol{A}, \boldsymbol{b})$ calls the tube triangulation subroutine for $\|\boldsymbol{A}\|_1$ times, and the punctured sphere triangulation subroutine for n times. The tube triangulation subroutine runs in time $O(1)$ since the there are a constant number of triangles in a tube; and the punctured sphere triangulation subroutine runs in time $O(\|\boldsymbol{A}(:,j)\|_1)$ for the jth call, $j \in [n]$. Putting all together, the total runtime of ReduceDAToB2$(\boldsymbol{A}, \boldsymbol{b})$ is $O\left(\|\boldsymbol{A}\|_1 + \sum_{j\in[n]} \|\boldsymbol{A}(:,j)\|_1\right) \le O(\mathrm{nnz}(\boldsymbol{A}))$, where we use the fact $\|\boldsymbol{A}\|_{\max} = 2$.

For solution mapping, the runtime of the algorithm MapSolnB2toDA is obvious. ◀

▶ **Lemma 4.2** (Size of ∂_2). *Given a difference-average instance* LE $(\boldsymbol{A}, \boldsymbol{b})$, *let* $(\partial_2, \gamma, \boldsymbol{\Delta}^c)$ *be returned by* ReduceDAToB2$(\boldsymbol{A}, \boldsymbol{b})$. *Suppose* $\partial_2 \in \mathbb{R}^{m\times t}$. *Then,*

1. $t \le 22\,\mathrm{nnz}(\boldsymbol{A})$;
2. $m \le 33\,\mathrm{nnz}(\boldsymbol{A})$;
3. $\mathrm{nnz}(\partial_2) \le 66\,\mathrm{nnz}(\boldsymbol{A})$.

Proof. We first compute the total number of triangles in the constructed 2-complex $\mathcal{K}$. For sphere $\mathcal{S}_j$, we have $\tilde{t}_j = 5b_j - 4$ triangles by (3), where $b_j = \sum_{i\in[d]} |\boldsymbol{A}(i,j)|$. Therefore, the number of triangles of all spheres is

$$\sum_{j=1}^{n} \tilde{t}_j = \sum_{j=1}^{n}(5\sum_{i\in[d]} |\boldsymbol{A}(i,j)| - 4) = 5\,\|\boldsymbol{A}\|_1 - 4n.$$

Moreover, each boundary component on spheres corresponds to a tube, and each tube has 6 triangles. Hence, the number of triangles of all tubes is $6\,\|\boldsymbol{A}\|_1$. Putting spheres and tubes together, we get

$$t = 11\,\|\boldsymbol{A}\|_1 - 4n \le 22\,\mathrm{nnz}(\boldsymbol{A}),$$

where the last inequality is because entries of $\boldsymbol{A}$ are bounded by 2.

Next, we compute the total number of edges in $\mathcal{K}$. By construction, each triangle has 3 incident edges and each edge is shared by a constant number of triangles (2 for interior edges, and 4 for boundary edges). Thus, we have

$$m \leq 1.5t \leq 33\,\mathrm{nnz}(\boldsymbol{A}).$$

Since each column of ∂_2 has exactly 3 nonzero entries, we have

$$\mathrm{nnz}(\partial_2) = 3t \leq 66\,\mathrm{nnz}(\boldsymbol{A}).$$ ◀

References

1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

2 Mitali Bafna and Nikhil Vyas. Optimal fine-grained hardness of approximation of linear equations. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

3 Mitchell Black, William Maxwell, Amir Nayyeri, and Eli Winkel. Computational topology in a collapsing universe: Laplacians, homology, cohomology. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.

4 Erik G Boman, Bruce Hendrickson, and Stephen Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM Journal on Numerical Analysis*, 46(6):3264–3284, 2008.

5 Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

6 Frédéric Chazal, Vin de Silva, Marc Glisse, and Steve Oudot. *The Structure and Stability of Persistence Modules*. Springer, October 2016.

7 Charles K Chui, HN Mhaskar, and Xiaosheng Zhuang. Representation of functions on big data associated with directed graphs. *Applied and Computational Harmonic Analysis*, 44(1):165–188, 2018.

8 Michael B Cohen, Brittany Terese Fasy, Gary L Miller, Amir Nayyeri, Richard Peng, and Noel Walkington. Solving 1-laplacians in nearly linear time: Collapsing and expanding a topological ball. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 204–216. SIAM, 2014.

9 Michael B Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 898–909. IEEE, 2018.

10 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 410–419, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055463.

11 Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log 1/2 n time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 343–352. ACM, 2014.

12 Samuel I Daitch and Daniel A Spielman. Support-graph preconditioners for 2-dimensional trusses. *arXiv preprint cs/0703119*, 2007. arXiv:cs/0703119.

13 Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *Discrete differential geometry*, pages 287–324. Springer, 2008.

14 Ming Ding, Rasmus Kyng, Maximilian Probst Gutenberg, and Peng Zhang. Hardness results for laplacians of simplicial complexes via sparse-linear equation complete gadgets. *arXiv preprint*, 2022. arXiv:2202.05011.

15 Xun Dong and Michelle L Wachs. Combinatorial laplacian of the matching complex. *the electronic journal of combinatorics*, pages R17–R17, 2002.

16 Art Duval, Caroline Klivans, and Jeremy Martin. Simplicial matrix-tree theorems. *Transactions of the American Mathematical Society*, 361(11):6073–6114, 2009.

17 Art M Duval, Caroline J Klivans, and Jeremy L Martin. Cuts and flows of cell complexes. *Journal of Algebraic Combinatorics*, 41(4):969–999, 2015.

18 Beno Eckmann. Harmonische funktionen und randwertaufgaben in einem komplex. *Commentarii Mathematici Helvetici*, 17(1):240–255, 1944.

19 Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Soc., 2010.

20 Joel Friedman. Computing betti numbers via combinatorial laplacians. *Algorithmica*, 21(4):331–346, 1998.

21 Robert Ghrist. Barcodes: The persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

22 Allen Hatcher. *Algebraic topology*. Cambridge University Press, 2000.

23 M R Hestenes and E Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 1952.

24 Arun Jambulapati and Aaron Sidford. Ultrasparse Ultrasparsifiers and Faster Laplacian System Solvers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 540–559. SIAM, 2021.

25 Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial hodge theory. *Mathematical Programming*, 127(1):203–244, 2011.

26 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching Optimality for Solving SDD Linear Systems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 235–244, USA, October 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.29.

27 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE, 2011.

28 Ioannis Koutis, Gary L Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011.

29 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 842–850, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897640.

30 Rasmus Kyng, Richard Peng, Robert Schwieterman, and Peng Zhang. Incomplete nested dissection. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 404–417, 2018.

31 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016.

32 Rasmus Kyng, Di Wang, and Peng Zhang. Packing LPs are hard to solve accurately, assuming linear equations are hard. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 279–296. SIAM, 2020.

33 Rasmus Kyng and Peng Zhang. Hardness results for structured linear systems. *SIAM Journal on Computing*, 49(4):FOCS17–280, 2020.

34 Lek-Heng Lim. Hodge laplacians on graphs. *Siam Review*, 62(3):685–715, 2020.

35 Wenye Ma, Jean-Michel Morel, Stanley Osher, and Aichi Chien. An l 1-based variational model for retinex theory and its application to medical images. In *CVPR 2011*, pages 153–160. IEEE, 2011.

36 William Maxwell and Amir Nayyeri. Generalized max-flows and min-cuts in simplicial complexes. *arXiv preprint*, 2021. `arXiv:2106.14116`.

37 James R Munkres. *Elements of algebraic topology*. CRC press, 2018.

38 Zipei Nie. Matrix anti-concentration inequalities with applications. *arXiv preprint*, 2021. `arXiv:2111.05553`.

39 Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, pages 333–342, 2014.

40 Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 504–521. SIAM, 2021.

41 Michael T Schaub, Austin R Benson, Paul Horn, Gabor Lippner, and Ali Jadbabaie. Random walks on simplicial complexes and the normalized hodge 1-laplacian. *SIAM Review*, 62(2):353–391, 2020.

42 Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.

43 Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

44 Yiying Tong, Santiago Lombeyda, Anil N Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM transactions on graphics (TOG)*, 22(3):445–452, 2003.

45 Qianqian Xu, Qingming Huang, Tingting Jiang, Bowei Yan, Weisi Lin, and Yuan Yao. Hodgerank on random graphs for subjective video quality assessment. *IEEE Transactions on Multimedia*, 14(3):844–857, 2012.

46 Ke Ye and Lek-Heng Lim. Cohomology of cryo-electron microscopy. *SIAM Journal on Applied Algebra and Geometry*, 1(1):507–535, 2017.

47 Afra J. Zomorodian. *Topology for Computing*, volume 16. Cambridge university press, 2005.

Two-Commodity Flow Is Equivalent to Linear Programming Under Nearly-Linear Time Reductions

Ming Ding
ETH Zürich, Switzerland

Rasmus Kyng
ETH Zürich, Switzerland

Peng Zhang
Rutgers University, Piscataway, NJ, USA

Abstract

We give a nearly-linear time reduction that encodes any linear program as a 2-commodity flow problem with only a small blow-up in size. Under mild assumptions similar to those employed by modern fast solvers for linear programs, our reduction causes only a polylogarithmic multiplicative increase in the size of the program, and runs in nearly-linear time. Our reduction applies to high-accuracy approximation algorithms and exact algorithms. Given an approximate solution to the 2-commodity flow problem, we can extract a solution to the linear program in linear time with only a polynomial factor increase in the error. This implies that any algorithm that solves the 2-commodity flow problem can solve linear programs in essentially the same time. Given a directed graph with edge capacities and two source-sink pairs, the goal of the 2-commodity flow problem is to maximize the sum of the flows routed between the two source-sink pairs subject to edge capacities and flow conservation. A 2-commodity flow problem can be formulated as a linear program, which can be solved to high accuracy in almost the current matrix multiplication time (Cohen-Lee-Song JACM'21). Our reduction shows that linear programs can be approximately solved, to high accuracy, using 2-commodity flow as well.

Our proof follows the outline of Itai's polynomial-time reduction of a linear program to a 2-commodity flow problem (JACM'78). Itai's reduction shows that exactly solving 2-commodity flow and exactly solving linear programming are polynomial-time equivalent. We improve Itai's reduction to nearly preserve the problem representation size in each step. In addition, we establish an error bound for approximately solving each intermediate problem in the reduction, and show that the accumulated error is polynomially bounded. We remark that our reduction does not run in strongly polynomial time and that it is open whether 2-commodity flow and linear programming are equivalent in strongly polynomial time.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Linear programming

Keywords and phrases Two-Commodity Flow Problems, Linear Programming, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.54

Category Track A: Algorithms, Complexity and Games

Related Version *Extended Version*: https://arxiv.org/abs/2201.11587 [6]

Funding *Rasmus Kyng*: The research leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 54; pp. 54:1–54:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Multi-commodity maximum flow is a very well-studied problem, which can be formulated as a linear program. In this paper, we show that general linear programs can be very efficiently encoded as a multi-commodity maximum flow programs. Many variants of multi-commodity flow problems exist. We consider one of the simplest directed variants, 2-commodity maximum through-put flow. Given a directed graph with edge capacities and two source-sink pairs, this problem requires us to maximize the sum of the flows routed between the two source-sink pairs, while satisfying capacity constraints and flow conservation at the remaining nodes. In the rest of the paper, we will simply refer to this as *the 2-commodity flow problem*. We abbreviate this problem as 2CF. Our goal is to relate the hardness of solving 2CF to that of solving linear programs (LPs). 2-commodity flow is easily expressed as a linear program, so it is clearly no harder than solving LPs. We show that the 2-commodity flow problem can encode a linear program with only a polylogarithmic blow-up in size, when the program has polynomially bounded integer entries and polynomially bounded solution norm. Our reduction runs in nearly-linear time. Given an approximate solution to the 2-commodity flow problem, we can recover, in linear time, an approximate solution to the linear program with only a polynomial factor increase in the error. Our reduction also shows that an exact solution to the flow problem yields an exact solution to the linear program.

Multi-commodity flow problems are extremely well-studied and have been the subject of numerous surveys [23, 38, 40, 1, 50], in part because a large number of problems can be expressed as variants of multi-commodity flow. Our result shows a very strong form of this observation: In fact, general linear programs can be expressed as 2-commodity flow problems with essentially the same size. Early in the study of these problems, before a polynomial-time algorithm for linear programming was known, it was shown that the *undirected* 2-commodity flow problem can be solved in polynomial time [16]. In fact, it can be reduced to two undirected single commodity maximum flow problems. In contrast, directed 2-commodity flow problems were seemingly harder, despite the discovery of non-trivial algorithms for some special cases [8, 9].

Searching for Multi-Commodity Flow Solvers. Alon Itai [17] proved a polynomial-time reduction from linear programming to 2-commodity flow, before a polynomial-time algorithm for linear programming was known. For decades, the only major progress on solving multi-commodity flow came from improvements to general linear program solvers [24, 19, 43, 48]. Leighton et al. [33] showed that undirected capacitated k-commodity flow in a graph with m edges and n vertices can be approximately solved in $\widetilde{O}(kmn)$ time, completely routing all demands with $1+\epsilon$ times the optimal congestion, albeit with a poor dependence on the error parameter ϵ. This beats solve-times for linear programming in sparse graphs for small k, even with today's LP solvers that run in current matrix multiplication time, albeit with much worse error. This result spurred a number of follow-up works with improvements for low-accuracy algorithms [14, 11, 35]. Later, breakthroughs in achieving almost- and nearly-linear time algorithms for undirected single-commodity maximum flow also lead to faster algorithms for undirected k-commodity flow [21, 45, 41], culminating in Sherman's introduction of area-convexity to build a $\widetilde{O}(mk\epsilon^{-1})$ time algorithm for approximate undirected k-commodity flow [46].

Solving Single-Commodity Flow Problems. Single commodity flow problems have been an area of tremendous success for the development of graph algorithms, starting with an era of algorithms influenced by early results on maximum flow and minimum cut [12] and later

the development of powerful combinatorial algorithms for maximum flow with polynomially bounded edge capacities [7, 10, 15]. Later, a breakthrough nearly-linear time algorithm for electrical flows by Spielman and Teng [47] lead to the *Laplacian paradigm*. A long line of work explored direct improvements and simplifications of this result [25, 26, 22, 42, 28, 18]. This also motivated a new line of research on undirected maximum flow [3, 31, 21, 45], which in turn lead to faster algorithms for directed maximum flow and minimum cost flow [36, 37, 34, 20, 49, 13] building on powerful tools using mixed-ℓ_2, ℓ_p-norm minimizing flows [27] and inverse-maintenance ideas [2]. Certain developments are particularly relevant to our result: For a graph $G = (V, E)$ these works established high-accuracy algorithms with $\widetilde{O}(|E|)$ running time for computing electrical flow [47] and $O(|E|^{4/3})$ running time for unit capacity directed maximum flow [36, 20], and $\widetilde{O}(\min(|E|^{1.497}, |E| + |V|^{1.5}))$ running time for directed maximum flow with general capacities [13, 49].

Solving General Linear Programs. As described in the previous paragraphs, there has been tremendous success in developing fast algorithms for single-commodity flow problems and undirected multi-commodity flow problems, albeit in the latter case only in the low-accuracy regime (as the algorithm running times depend polynomially on the error parameter). In contrast, the best known algorithms for directed multi-commodity flow simply treat the problem as a general linear program, and use a solver for general linear programs.

The fastest known solvers for general linear programs are based on interior point methods [19], and in particular central path methods [43]. Recently, there has been significant progress on solvers for general linear programs, but the running time required to solve a linear program with roughly n variables and $\widetilde{O}(n)$ constraints (assuming polynomially bounded entries and polytope radius) is stuck at the $\widetilde{O}(n^{2.372\cdots})$, the running time provided by LP solvers that run in current matrix multiplication time [4]. Note that this running time is in the RealRAM model, and this algorithm cannot be translated to fixed point arithmetic with polylogarithmic bit complexity per number without additional assumptions on the input, as we describe the paragraphs on numerical stability below. To compare these running times with those for single-commodity maximum flow algorithms on a graph with $|V|$ vertices and $|E|$ edges, observe that in a sparse graph with $|E| = \widetilde{O}(|V|)$, by writing the maximum flow problem as a linear program, we can solve it using general linear program solvers and obtain a running time of $\widetilde{O}(|V|^{2.372\cdots})$, while the state-of-the art maximum flow solver obtains a running time of $\widetilde{O}(|V|^{1.497})$ on such a sparse graph. On dense graphs with $|E| = \Theta(|V|^2)$, the gap is smaller but still substantial: The running time is $\widetilde{O}(|V|^2)$ using maximum flow algorithms vs. $\widetilde{O}(|V|^{2.372\cdots})$ using general LP algorithms.

How Hard Is It to Solve Multi-Commodity Flow? The many successes in developing high-accuracy algorithms for single-commodity flow problems highlight an important open question: Can multi-commodity flow be solved to high accuracy faster than general linear programs? We rule out this possibility, by proving that any linear program (assuming it is polynomially bounded and has integer entries) can be encoded as a multi-commodity flow problem in nearly-linear time. This implies that any improvement in the running time of (high-accuracy) algorithms for sparse multi-commodity flow problems would directly translate to a faster algorithm for solving sparse linear programs to high accuracy, with only a polylogarithmic increase in running time.

Previous work by Kyng and Zhang [30] had shown that fast algorithms for multi-commodity flow were unlikely to arise from combining interior point methods with special-purpose linear equation solvers. Concretely, they showed that the linear equations that arise

in interior point methods for multi-commodity flow are as hard to solve as arbitrary linear equations. This ruled out algorithms following the pattern of the known fast algorithms for high-accuracy single-commodity flow problems. However, it left open the broader question if some other family of algorithms could succeed. We now show that, in general, a separation between multi-commodity flow and linear programming is not possible.

1.1 Background: Numerical Stability of Linear Program Solvers and Reductions

Current research on fast algorithms for solving linear programs generally relies on assuming bounds on (1) the size of the program entries and (2) the norm of all feasible solutions. Generally, algorithm running time depends logarithmically on these quantities, and hence to make these factors negligible, entry size and feasible solution norms are assumed to be polynomially bounded, for example in [4]. We will refer to a linear program satisfying these assumptions as *polynomially bounded*. More precisely, we say a linear program with N non-zero coefficients is *polynomially bounded* if it has coefficients in the range $[-X, X]$ and $\|\boldsymbol{x}\|_1 \leq R$ for all feasible $\boldsymbol{x}$ (i.e. the polytope of feasible solutions has radius of R in ℓ_1 norm), and $X, R \leq O(N^c)$ for some constant c. In fact, if there exists a feasible solution $\boldsymbol{x}$ satisfying $\|\boldsymbol{x}\|_1 \leq R$, then we can add a constraint $\|\boldsymbol{x}\|_1 \leq R$ to the LP (which can be rewritten as linear inequality constraints) so that in the new LP, all feasible solutions have ℓ_1 norm at most R. This only increases the number of nonzeros in the LP by at most a constant factor.

Interior Point Methods and Reductions With Fixed Point Arithmetic. Modern fast interior point methods for linear programming, such as [4], are analyzed in the RealRAM model. In order to implement these algorithms using *fixed point arithmetic* with polylogarithmic bit complexity per number, instead of RealRAM, additional assumptions are required. For example, this class of algorithms relies on computing matrix inverses, and these must be approximately representable using polylogarithmic bit complexity per entry. This is not possible, if the inverses have exponentially large entries, which may occur even in polynomially bounded linear programs. For example, consider a linear program feasibility problem $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$, with constraint matrix $\boldsymbol{A} \in \mathbb{R}^{2n \times 2n}$ given by

$$\boldsymbol{A}(i,j) = \begin{cases} 1 & \text{if } i < n \text{ and } i = j \\ -2 & \text{if } i < n \text{ and } i+1 = j \\ 2 & \text{if } i \geq n \text{ and } i = j \\ -1 & \text{if } i \geq n \text{ and } i+1 = j \\ 0 & \text{o.w.} \end{cases}$$

Such a linear program is polynomially bounded for many choices of $\boldsymbol{b}$, e.g. $\boldsymbol{b} = \boldsymbol{e}_{2n}$. Unfortunately, for the vector $\boldsymbol{x} \in \mathbb{R}^{2n}$ given by

$$\boldsymbol{x}(i) = \begin{cases} 2^{-i-1} & \text{if } i \leq n \\ 0 & \text{o.w.} \end{cases}$$

we have $\boldsymbol{A}\boldsymbol{x} = 2^{-n}\boldsymbol{e}_n$, and from this one can see that $\boldsymbol{A}^{-1}$ must have entries of size at least $\Omega(2^n/n)$. This will cause algorithms such as [4] to perform intermediate calculations with n bits per number, increasing the running time by a factor roughly n.

Modern interior point methods can be translated to fixed precision arithmetic with various different assumptions leading to different per entry bit complexity (see [4] for a discussion of one standard sufficient condition). Furthermore, if the problem has polynomially bounded

condition number (when appropriately defined), then we expect that polylogarithmic bit complexity per entry should suffice, at least for highly accurate approximate solutions, by relying on fast stable numerical linear algebra [5], although we are not aware of a complete analysis of this translation.

If a linear program with integer entries is solved to sufficiently small additive error, the approximate solution can be converted into an exact solution, e.g. see [43, 32, 4] for a discussion of the necessary precision and for a further discussion of numerical stability properties of interior point methods. Polynomially bounded linear programs with integer coefficients may still require exponentially small additive error for this rounding to succeed.

We give a reduction from general linear programming to 2-commodity flow, and like [4], we assume the program is polynomially bounded to carry out the reduction. We also use an additional assumption, namely that the linear program is written using integral entries[1]. We do not make additional assumptions about polynomially bounded condition number of problem. This means we can apply our reduction to programs such as the one above, despite [4] not obtaining a reasonable running time on such programs using fixed point arithmetic.

Our analysis of our reduction uses the RealRAM model like [4] and other modern interior point method analysis, however, it should be straightforward to translate our reduction and error analysis to fixed point arithmetic with polylogarithmically many bits, because all our mappings are simple linear transformations, and we never need to compute or apply a matrix inverse.

Rounding Linear Programs to Have Integer Entries. It is possible to give some fairly general and natural sufficient conditions for when a polynomially bounded linear program can be rounded to have integral entries, one example of this is having a polynomially bounded *Renegar's condition number*. Renegar introduced this condition number for linear programs in [44]. For a given linear program, suppose that perturbing the entries of the program by at most δ each does not change the feasibility of the the linear program, and let δ^* be the largest such δ. Let U denote the maximum absolute value of entries in the linear program. Then $\kappa = U/\delta^*$ is Renegar's condition number for the linear program.

Suppose we are given a polynomially bounded linear program $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ (also referred to as $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$), with polytope radius at most R, and Renegar's condition number κ also bounded by a polynomial. We wish to compute a vector $\boldsymbol{x} \geq \boldsymbol{0}$ with an ϵ additive error on each constraint and in the optimal value. We can reduce this problem for instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ to a polynomially bounded linear program instance with integral input numbers. Specifically, we round the entries of $\boldsymbol{A}$ down to $\tilde{\boldsymbol{A}}$ and those of $\boldsymbol{b}, \boldsymbol{c}$ up to $\tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$ all by at most $\min\{\frac{\epsilon}{3R}, \frac{U}{\kappa R}\}$ such that each entry of $\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$ only needs a logarithmic number of bits. Suppose $\tilde{\boldsymbol{x}}$ is a solution to $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}})$ with $\frac{\epsilon}{3}$ additive error on each constraint and in the optimal value. Then,

$$\boldsymbol{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} + (\boldsymbol{A} - \tilde{\boldsymbol{A}})\tilde{\boldsymbol{x}} \leq \boldsymbol{b} + \epsilon\boldsymbol{1}, \qquad \boldsymbol{c}^\top \tilde{\boldsymbol{x}} \geq \tilde{\boldsymbol{c}}^\top \tilde{\boldsymbol{x}}^* - \frac{2}{3}\epsilon$$

where $\boldsymbol{1}$ is the all-one vector, $\tilde{\boldsymbol{x}}^*$ is an optimal solution to $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}})$. In addition, the optimal value of $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}})$ is greater than or equal to that of $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$. So, $\tilde{\boldsymbol{x}}$ is a solution to $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ with ϵ additive error as desired. Since each entry of $\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$ has a logarithmic number of bits,

[1] W.l.o.g., by scaling, this is the same as assuming the program is written with polynomially bounded fixed precision numbers of the form k/D where k is an integer, and D is an integral denominator shared across all entries, and both k and D are polynomially bounded.

we can scale all of them to polynomially bounded integers without changing the feasible set and the optimal solutions. Thus we see that if we restrict ourselves to polynomially linear programs with polynomially bounded Renegar's condition number, and we wish to solve the program with small additive error, we can assume without loss of generality that the program has integer coefficients.

1.2 Previous Work

Our paper follows the proof by Itai [17] that linear programming is polynomial-time reducible to 2-commodity flow. However, it is also inspired by recent works on hardness for structured linear equations [30] and packing/covering LPs [29], which focused on obtaining nearly-linear time reductions in somewhat related settings. These works in turn were motivated by the last decade's substantial progress on fine-grained complexity for a range of polynomial time solvable problems, e.g. see [51]. Also notable is the result by Musco et al. [39] on hardness for matrix spectrum approximation.

1.3 Our Contributions

In this paper, we explore the hardness of 2-commodity maximum throughput flow, which for brevity we refer to as the 2-commodity flow problem or 2CF. We relate the difficulty of 2CF to that of linear programming (LP) by developing an extremely efficient reduction from the former to the latter. The main properties of our reduction are described by the informal theorem statement below. We give a formal statement of Theorem 1.1 as Theorem 3.1 in Section 3.

▶ **Theorem 1.1** (Main Theorem (Informal)). *Consider any polynomially bounded linear program* $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ *with integer coefficients and N non-zero entries. In nearly-linear time, we can convert this linear program to a 2-commodity flow problem which is feasible if and only if the original program is. The 2-commodity flow problem has $\widetilde{O}(N)$ edges and has polynomially bounded integral edge capacities. Furthermore, any solution to the 2-commodity flow instance with at most ϵ additive error on each constraint and value at most ϵ from the optimum can be converted to a solution to the original linear program with additive error $\widetilde{O}(\text{poly}(N)\epsilon)$ on each constraint and similarly value within $\widetilde{O}(\text{poly}(N)\epsilon)$ of the optimum.*

This implies that, for any constant $a > 1$, if any 2-commodity flow instance with polynomially bounded integer capacities can be solved with ϵ additive error in time $\widetilde{O}\left(|E|^a \cdot \text{poly}\log(1/\epsilon)\right)$, then any polynomially bounded linear program can be solved to ϵ additive error in time $\widetilde{O}\left(N^a \cdot \text{poly}\log(1/\epsilon)\right)$.

Note that in our definition any 2CF problem is already an LP, and so no reduction in the other direction is necessary. Our notion of approximate solutions to LPs and 2CF problems is also such that treating a 2CF problem as an LP and solving it approximately ensures that the 2CF is approximately solved w.r.t. to the our approximate solution definition for 2CF.

We obtain Theorem 1.1, our main result, by making several improvements to Itai's reduction from LP to 2CF. Recall that a linear program with N non-zero coefficients is polynomially bounded if it has coefficients in the range $[-X, X]$ and $\|\boldsymbol{x}\|_1 \leq R$ for all feasible $\boldsymbol{x}$, where $X, R \leq O(N^c)$ for some constant c. Firstly, while Itai produced a 2CF with the number of edges on the order of $\Theta\left(N^2 \log^2 X\right)$, we show that an improved gadget can reduce this to $O\left(N \log X\right)$. Thus, in the case of polynomially bounded linear programs, where $\log X = O(\log N)$, we get an only polylogarithmic multiplicative increase in the number of non-zero entries from N to $\widetilde{O}(N)$, whereas Itai had an increase in the number of nonzeros by a factor $\widetilde{O}(N)$, from N to $\widetilde{O}(N^2)$.

Secondly, Itai used very large graph edge capacities that require $O\left((N \log X)^{1.01}\right)$ many bits *per edge*, letting the capacities grow exponentially given an LP with polynomially bounded entries. We show that when the feasible polytope radius R is bounded, we can ensure capacities remain a polynomial function of the initial parameters N, R, and X. In the important case of polynomially bounded linear programs, this means the capacities stay polynomially bounded.

Thirdly, while Itai only analyzed the chain of reductions under the case with exact solutions, we generalize the analysis to the case with approximate solutions by establishing an error analysis along the chain of reductions. We show that the error only grows polynomially during the reduction. Moreover, to simplify our error analysis, we observe that additional structures can be established in many of Itai's reductions. For instance, we propose the notion of a *fixed flow network*, which consists of a subset of edges with equal lower and upper bound of capacity. It is a simplification of Itai's (l, u) network with general capacity (both lower and upper bounds on the amount of flow).

Open Problems. Our reductions do not suffice to prove that a strongly polynomial time algorithm for 2-commodity flow would imply a strongly polynomial time algorithm for linear programming. In a similar vein, it is unclear if a more efficient reduction could exist for the case of linear programs that are not polynomially bounded. We leave these as very interesting open problems.

Finally, our reductions do not preserve the "shape" of the linear program, in particular, a dense linear program may be reduced to a sparse 2-commodity flow problem with a similar number of edges as there are non-zero entries in the original program. It would be interesting to convert a dense linear program into a dense 2-commodity flow problem, e.g. to convert a linear program with m constraints and n variables (say, $m \leq n$) into a 2-commodity flow problem with $\widetilde{O}(n)$ edges and $\widetilde{O}(m)$ vertices.

2 Preliminaries

2.1 Notation

Matrices and Vectors. We use parentheses to denote entries of a matrix or a vector: Let $\boldsymbol{A}(i, j)$ denote the (i, j)th entry of a matrix $\boldsymbol{A}$, and let $\boldsymbol{x}(i)$ denote the ith entry of a vector $\boldsymbol{x}$. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, we use $\boldsymbol{a}_i^\top$ to denote the ith row of a matrix $\boldsymbol{A}$ and $\mathrm{nnz}(\boldsymbol{A})$ to denote the number of nonzero entries of $\boldsymbol{A}$. Without loss of generality, we assume that $\mathrm{nnz}(\boldsymbol{A}) \geq \max\{m, n\}$. For any vector $\boldsymbol{x} \in \mathbb{R}^n$, we define $\|\boldsymbol{x}\|_{\max} = \max_{i \in [n]} |\boldsymbol{x}(i)|$, $\|\boldsymbol{x}\|_1 = \sum_{i \in [n]} |\boldsymbol{x}(i)|$. For any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, we define $\|\boldsymbol{A}\|_{\max} = \max_{i,j} |\boldsymbol{A}(i, j)|$.

We define a function X that takes an arbitrary number of matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{k_1}$, vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k_2}$, and scalars $K_1, \ldots, K_{k_3}$ as arguments, and returns the maximum of $\|\cdot\|_{\max}$ of all the arguments, i.e.,

$$X(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{k_1}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k_2}, K_1, \ldots, K_{k_3}) = \max\left\{\|\boldsymbol{A}_1\|_{\max}, \ldots, \|\boldsymbol{A}_{k_1}\|_{\max}, \|\boldsymbol{b}_1\|_{\max}, \ldots, \|\boldsymbol{b}_{k_2}\|_{\max}, |K_1|, \ldots, |K_{k_3}|\right\}.$$

2.2 Problem Definitions

In this section, we formally define the problems that we use in the reduction. These problems fall into two categories: one is related to linear programming and linear equations, and the other is related to flow problems in graphs. In addition, we define the errors for approximately solving these problems.

2.2.1 Linear Programming and Linear Equations With Positive Variables

For the convenience of our reduction, we define linear programming as a "decision" problem. We can solve the optimization problem $\max\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ by binary searching its optimal value via the decision problem.

▶ **Definition 2.1** (Linear Programming (LP)). *Given a matrix $\boldsymbol{A} \in \mathbb{Z}^{m\times n}$, vectors $\boldsymbol{b} \in \mathbb{Z}^m$ and $\boldsymbol{c} \in \mathbb{Z}^n$, an integer K, and $R \geq \max\{1, \max\{\|\boldsymbol{x}\|_1 : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}\}$, we refer to the LP problem for $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ as the problem of finding a vector $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$ satisfying*

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \text{ and } \boldsymbol{c}^\top \boldsymbol{x} \geq K$$

if such an $\boldsymbol{x}$ exists and returning "infeasible" otherwise.

We will reduce linear programming to linear equations with nonnegative variables (LEN), and then to linear equations with nonnegative variables and small integer coefficients (k-LEN).

▶ **Definition 2.2** (Linear Equations with Nonnegative Variables (LEN)). *Given $\boldsymbol{A} \in \mathbb{Z}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Z}^m$, and $R \geq \max\{1, \max\{\|\boldsymbol{x}\|_1 : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}\}$, we refer to the LEN problem for $(\boldsymbol{A}, \boldsymbol{b}, R)$ as the problem of finding a vector $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$ satisfying $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ if such an $\boldsymbol{x}$ exists and returning "infeasible" otherwise.*

▶ **Definition 2.3** (k-LEN (k-LEN)). *The k-LEN problem is an LEN problem $(\boldsymbol{A}, \boldsymbol{b}, R)$ where the entries of $\boldsymbol{A}$ are integers in $[-k, k]$ for some given $k \in \mathbb{Z}_+$.*

We employ the following additive error notion. We append a letter "A" to each problem name to denote its approximation version, e.g., LP Approximate Problem is abbreviated to LPA.

▶ **Definition 2.4** (Approximation Errors). *We always require $\boldsymbol{x} \geq \boldsymbol{0}$. In addition,*

1. *Error in objective: $\boldsymbol{c}^\top \boldsymbol{x} \geq K$ is relaxed to $\boldsymbol{c}^\top \boldsymbol{x} \geq K - \epsilon$;*
2. *Error in constraint:*
   a. *The inequality constraint $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ is relaxed to $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \leq \epsilon\boldsymbol{1}$, where $\boldsymbol{1}$ is the all-1 vector;*
   b. *The equality constraint $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is relaxed to $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty \leq \epsilon$.*

Based on Definition 2.4, we can define the approximate version of LP. The approximate version of LEN and k-LEN can be found in the full version of the paper [6].

▶ **Definition 2.5** (LP Approximate Problem (LPA)). *An LPA instance is given by an LP instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ and an error parameter $\epsilon \in [0, 1]$, which we collect in a tuple $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon)$. We say an algorithm solves the LPA problem, if, given any LPA instance, it returns a vector $\boldsymbol{x} \geq \boldsymbol{0}$ such that*

$$\boldsymbol{c}^\top \boldsymbol{x} \geq K - \epsilon$$
$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + \epsilon\boldsymbol{1}$$

where $\boldsymbol{1}$ is the all-1 vector, or it correctly declares that the associated LP instance is infeasible.

2.2.2 Flow Problems

A *flow network* is a directed graph $G = (V, E)$, where V is the set of vertices and $E \subset V \times V$ is the set of edges, together with a vector of edge capacities $\boldsymbol{u} \in \mathbb{Z}^{|E|}_{>0}$ that upper bound the amount of flow passing each edge. A *2-commodity flow network* is a flow network together with two source-sink pairs $s_i, t_i \in V$ for each commodity $i \in \{1, 2\}$.

Given a 2-commodity flow network $(G = (V, E), \boldsymbol{u}, s_1, t_1, s_2, t_2)$, a *feasible 2-commodity flow* is a pair of flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \in \mathbb{R}^{|E|}_{\geq 0}$ that satisfies

1. capacity constraint: $\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e),\ \forall e \in E$, and
2. conservation of flows: $\sum_{u:(u,v)\in E} \boldsymbol{f}_i(u,v) = \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v,w),\ \forall i \in \{1,2\}, v \in V \setminus \{s_i, t_i\}$[2].

Similar to the definition of LP, we define 2-commodity flow problem as a decision problem. We can solve a decision problem by solving the corresponding optimization problem.

▶ **Definition 2.6** (2-Commodity Flow Problem (2CF)). *Given a 2-commodity flow network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with $R \geq 0$, we refer to the 2CF problem for $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ satisfying*

$$F_1 + F_2 \geq R$$

if such flows exist and returning "infeasible" otherwise.

To reduce LP to 2CF, we need a sequence of variants of flow problems.

▶ **Definition 2.7** (2-Commodity Flow with Required Flow Amount (2CFR)). *Given a 2-commodity flow network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with $R_1, R_2 \geq 0$, we refer to the 2CFR for $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R_1, R_2)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ satisfying*

$$F_1 \geq R_1, \quad F_2 \geq R_2$$

if such flows exist and returning "infeasible" otherwise.

▶ **Definition 2.8** (Fixed Flow Constraints). *Given a set $F \subseteq E$ in a 2-commodity flow network, we say the flows $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ satisfy* fixed flow constraints on F *if*

$$\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) = \boldsymbol{u}(e),\ \forall e \in F.$$

Similarly, given a set $F \subseteq E$ in a 1-commodity flow network, we say the flow $\boldsymbol{f} \geq \boldsymbol{0}$ satisfies fixed flow constraints on F *if*

$$\boldsymbol{f}(e) = \boldsymbol{u}(e),\ \forall e \in F.$$

▶ **Definition 2.9** (2-Commodity Fixed Flow Problem (2CFF)). *Given a 2-commodity flow network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with a subset of edges $F \subseteq E$, we refer to the 2CFF problem for the tuple $(G, F, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ which also satisfies the fixed flow constraints on F if such flows exist and returning "infeasible" otherwise.*

▶ **Definition 2.10** (Selective Fixed Flow Problem (SFF)). *Given a 2-commodity network $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ together with three edge sets $F, S_1, S_2 \subseteq E$, we refer to the SFF problem for $(G, F, S_1, S_2, \boldsymbol{u}, s_1, t_1, s_2, t_2)$ as the problem of finding a feasible 2-commodity flow $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq \boldsymbol{0}$ such that for each $i \in \{1,2\}$, flow $\boldsymbol{f}_i(e) > 0$ only if $e \in S_i$, and $\boldsymbol{f}_1, \boldsymbol{f}_2$ satisfy the fixed flow constraints on F, if such flows exist, and returning "infeasible" otherwise.*

▶ **Definition 2.11** (Fixed Homologous Flow Problem (FHF)). *Given a flow network with a single source-sink pair $(G, \boldsymbol{u}, s, t)$ together with a collection of disjoint subsets of edges $\mathcal{H} = \{H_1, \ldots, H_h\}$ and a subset of edges $F \subseteq E$ such that F is disjoint from all the sets in $\mathcal{H}$, we refer to the FHF problem for $(G, F, \mathcal{H}, \boldsymbol{u}, s, t)$ as the problem of finding a feasible flow $\boldsymbol{f} \geq \boldsymbol{0}$ such that*

[2] Note that for commodity i, this constraint includes the case of $v \in \{s_{\bar{i}}, t_{\bar{i}}\}, \bar{i} = \{1,2\} \setminus i$.

$$\boldsymbol{f}(e_1) = \boldsymbol{f}(e_2), \quad \forall e_1, e_2 \in H_k, 1 \le k \le h,$$

and $\boldsymbol{f}$ satisfies the fixed flow constraints on F, if such flows exist, and returning "infeasible" otherwise.

▶ **Definition 2.12** (Fixed Pair Homologous Flow Problem (FPHF)). *An FPHF is an FHF problem $(G, F, \mathcal{H}, \boldsymbol{u}, s, t)$ where every set in $\mathcal{H}$ has size 2.*

Now, we define errors for the above flow problems.

▶ **Definition 2.13** (Approximation Errors). *We always require flows $\boldsymbol{f} \ge \boldsymbol{0}$ and $\boldsymbol{f}_1, \boldsymbol{f}_2 \ge \boldsymbol{0}$. In addition,*

1. *Error in congestion: the capacity constraints are relaxed to:*

$$\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \le \boldsymbol{u}(e) + \epsilon, \quad \forall e \in E.$$

There are several variants corresponding to different flow problems.

   a. *If $e \in F$ is a fixed-flow edge, the fixed-flow constraints are relaxed to*

$$\boldsymbol{u}(e) - \epsilon \le \boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \le \boldsymbol{u}(e) + \epsilon, \quad \forall e \in F$$

   b. *If G is a 1-commodity flow network, we replace $\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e)$ by $\boldsymbol{f}(e)$.*

2. *Error in demand: the conservation of flows is relaxed to*

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}_i(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v,w) \right| \le \epsilon, \ \forall v \in V \backslash \{s_i, t_i\}, i \in \{1,2\} \tag{1}$$

There are several variants of this constraint corresponding to different flow problems.

   a. *If the problem is with flow requirement F_i, then besides Eq. (1), we add demand constraints for s_i and t_i with respect to commodity i:*

$$\left| \sum_{w:(s_i,w)\in E} \boldsymbol{f}_i(s_i,w) - F_i \right| \le \epsilon, \qquad \left| \sum_{u:(u,t_i)\in E} \boldsymbol{f}_i(u,t_i) - F_i \right| \le \epsilon, \ i \in \{1,2\} \tag{2}$$

   b. *If G is a 1-commodity flow network, Eq. (1) can be simplified as*

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}(v,w) \right| \le \epsilon, \ \forall v \in V \backslash \{s,t\}$$

3. *Error in type: the selective constraints are relaxed to*

$$\boldsymbol{f}_{\bar{i}}(e) \le \epsilon, \quad \forall e \in S_i, \ \bar{i} = \{1,2\} \backslash i.$$

4. *Error in (pair) homology: the (pair) homologous constraints are relaxed to*

$$|\boldsymbol{f}(e_1) - \boldsymbol{f}(e_2)| \le \epsilon, \ \forall e_1, e_2 \in H_k, H_k \in \mathcal{H}.$$

Based on Definition 2.13, we define the approximate version of 2CF. Again, the approximate version of the rest flow problems can be found in the full version of the paper [6].

▶ **Definition 2.14** (2CF Approximate Problem (2CFA)). *A 2CFA instance is given by a 2CF instance* $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R)$ *and an error parameter* $\epsilon \in [0, 1]$*, which we collect in a tuple* $(G, \boldsymbol{u}, s_1, t_1, s_2, t_2, R, \epsilon)$*. We say an algorithm solves the 2CFA problem, if, given any 2CFA instance, it returns a pair of flows* $\boldsymbol{f}_1, \boldsymbol{f}_2 \geq 0$ *that satisfies*

$$\boldsymbol{f}_1(e) + \boldsymbol{f}_2(e) \leq \boldsymbol{u}(e) + \epsilon, \ \forall e \in E \tag{3}$$

$$\left| \sum_{u:(u,v)\in E} \boldsymbol{f}_i(u,v) - \sum_{w:(v,w)\in E} \boldsymbol{f}_i(v,w) \right| \leq \epsilon, \ \forall v \in V \backslash \{s_i, t_i\}, i \in \{1,2\} \tag{4}$$

$$\left| \sum_{w:(s_i,w)\in E} \boldsymbol{f}_i(s_i,w) - F_i \right| \leq \epsilon, \qquad \left| \sum_{u:(u,t_i)\in E} \boldsymbol{f}_i(u,t_i) - F_i \right| \leq \epsilon, \ i \in \{1,2\} \tag{5}$$

where $F_1 + F_2 = R$ [3]*; or it correctly declares that the associated 2CF instance is infeasible. We refer to the error in* (3) *as error in congestion, error in* (4) *and* (5) *as error in demand.*

3 Main Results

▶ **Theorem 3.1.** *Given an LPA instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon^{lp})$ *where* $\boldsymbol{A} \in \mathbb{Z}^{m\times n}, \boldsymbol{b} \in Z^m, \boldsymbol{c} \in \mathbb{Z}^n, K \in \mathbb{Z}, \epsilon^{lp} \geq 0$ *and* $\boldsymbol{A}$ *has* $\operatorname{nnz}(\boldsymbol{A})$ *nonzero entries, we can reduce it to a 2CFA instance* $(G = (V, E), \boldsymbol{u}, s_1, t_1, s_2, t_2, R^{2cf}, \epsilon^{2cf})$ *in time* $O(\operatorname{nnz}(\boldsymbol{A}) \log X)$ *where* $X = X(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K)$*, such that*

$$|V|, |E| = O(\operatorname{nnz}(\boldsymbol{A}) \log X),$$
$$\|\boldsymbol{u}\|_{\max}, R^{2cf} = O(\operatorname{nnz}^3(\boldsymbol{A}) R X^2 \log^2 X),$$
$$\epsilon^{2cf} = \Omega\left(\frac{1}{\operatorname{nnz}^7(\boldsymbol{A}) R X^3 \log^6 X}\right) \epsilon^{lp}.$$

If the LP instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ *has a solution, then the 2CF instance* $(G^{2cf}, \boldsymbol{u}^{2cf}, s_1, t_1, s_2, t_2, R^{2cf})$ *has a solution. Furthermore, if* $\boldsymbol{f}^{2cf}$ *is a solution to the 2CFA (2CF) instance, then in time* $O(\operatorname{nnz}(\boldsymbol{A}) \log X)$*, we can compute a solution* $\boldsymbol{x}$ *to the LPA (LP, respectively) instance, where the exact case holds when* $\epsilon^{2cf} = \epsilon^{lp} = 0$*.*

Our main theorem immediately implies the following corollary.

▶ **Corollary 3.2.** *If we can solve any 2CFA instance* $(G = (V, E), \boldsymbol{u}, s_1, t_1, s_2, t_2, R^{2cf}, \epsilon)$ *in time* $O\left(|E|^c \operatorname{poly}\log\left(\frac{\|\boldsymbol{u}\|_1}{\epsilon}\right)\right)$ *for some small constant* $c \geq 1$*, then we can solve any LPA instance* $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R, \epsilon)$ *in time* $O\left(\operatorname{nnz}^c(\boldsymbol{A}) \operatorname{poly}\log\left(\frac{\operatorname{nnz}(\boldsymbol{A}) R X(\boldsymbol{A},\boldsymbol{b},\boldsymbol{c},K)}{\epsilon}\right)\right)$.

3.1 Overview of Our Proof

In this section, we will explain how to reduce an LP instance to a 2-commodity flow (2CF) instance by a chain of efficient reductions. In each step, we reduce a decision problem P to a decision problem Q. We guarantee that (1) the reduction runs in nearly-linear time[4], (2) the

[3] If we encode 2CF as an LP instance, and approximately solve the LP with at most ϵ additive error. Then, the approximate solution also agrees with the error notions of 2CF, except that we get $F_1 + F_2 \geq R - \epsilon$ instead of $F_1 + F_2 \geq R$. This inconsistency can be eliminated by setting $\epsilon' = 2\epsilon$, and slightly adjusting F_1, F_2 to F_1', F_2' such that $F_1' + F_2' \geq R$. This way, we obtain an approximate solution to 2CF with at most ϵ' additive error.

[4] Linear in the size of problem P, poly-logarithmic in the maximum magnitude of all the numbers that describe P, the feasible set radius, and the inverse of the error parameter if an approximate solution is allowed.

Table 1 A summary of notation used in the reduction from LP to 2CF.

Exact problems	Input	Output
LP (Def. 2.1)	$\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R$	$\boldsymbol{x}$
LEN (Def. 2.2)	$\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R}$	$\tilde{\boldsymbol{x}}$
2-LEN (Def. 2.3)	$\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R}$	$\bar{\boldsymbol{x}}$
1-LEN (Def. 2.3)	$\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R}$	$\hat{\boldsymbol{x}}$
FHF (Def. 2.11)	$G^h, F^h, \mathcal{H}^h = \{H_1, \cdots, H_h\}, \boldsymbol{u}^h, s, t$	$\boldsymbol{f}^h$
FPHF (Def. 2.12)	$G^p, F^p, \mathcal{H}^p = \{H_1, \cdots, H_p\}, \boldsymbol{u}^p, s, t$	$\boldsymbol{f}^p$
SFF (Def. 2.10)	$G^s, F^s, S_1, S_2, \boldsymbol{u}^s, s_1, t_1, s_2, t_2$	$\boldsymbol{f}^s$
2CFF (Def. 2.9)	$G^f, F^f, \boldsymbol{u}^f, s_1, t_1, s_2, t_2$	$\boldsymbol{f}^f$
2CFR (Def. 2.7)	$G^r, \boldsymbol{u}^r, \bar{s}_1, \bar{t}_1, \bar{s}_2, \bar{t}_2, R_1, R_2$	$\boldsymbol{f}^r$
2CF (Def. 2.6)	$G^{2cf}, \boldsymbol{u}^{2cf}, \bar{\bar{s}}_1, \bar{t}_1, \bar{\bar{s}}_2, \bar{t}_2, R^{2cf}$	$\boldsymbol{f}^{2cf}$

size of Q is nearly-linear in the size of P, and (3) that P is feasible implies that Q is feasible, and an approximate solution to Q can be turned to an approximate solution to P with only a polynomial blow-up in error parameters, in linear time.

We follow the outline of Itai's reduction [17]. A summary of the problem notation used in the reduction from LP to 2CF is given in Table 1. Itai first reduced an LP instance to a 1-LEN instance (linear equations with nonnegative variables and ± 1 coefficients). A 1-LEN instance can be cast as a single-commodity flow problem subject to additional homologous constraints and fixed flow constraints (i.e., FHF). Then, Itai dropped these additional constraints step by step, via introducing a second commodity of flow and imposing lower bound requirements on the total amount of flows routed between the source-sink pairs. However, in the worst case, Itai's reduction from 1-LEN to FHF enlarges the problem size quadratically and is thus inefficient. One of our main contributions is to improve this step so that the problem representation size is preserved along the reduction chain.

Our second main contribution is an upper bound on the errors accumulated during the process of mapping an approximate solution to the 2CF instance to an approximate solution to the LP instance. We show that the error only grows by polynomial factors. Itai only considered exact solutions between these two instances.

We will explain the reductions based on the exact versions of the problems. At the end of this section, we will discuss some intuitions of our error analysis.

3.1.1 Reducing Linear Programming to Linear Equations With Nonnegative Variables and ± 1 Coefficients

Given an LP instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, K, R)$ where $R \geq \max\{1, \max\{\|x\|_1 : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}\}$, we want to compute a vector $\boldsymbol{x} \geq \boldsymbol{0}$ satisfying

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{c}^\top \boldsymbol{x} \geq K$$

or to correctly declare infeasible. We introduce slack variables $\boldsymbol{s}, \alpha \geq \boldsymbol{0}$ and turn the above inequalities to equalities:

$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}, \boldsymbol{c}^\top \boldsymbol{x} - \alpha = K$$

which is an LEN instance $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{b}}, \tilde{R})$. Comparing to Itai's proof, we need to track two additional parameters: the polytope radius R and the maximum magnitude of the input entries X.

We then reduce the LEN instance to linear equations with ± 2 coefficients (2-LEN) by bitwise decomposition. For each bit, we introduce a carry term. Different from Itai's reduction, we impose an upper bound for each carry variable. We show that this upper bound does not change problem feasibility and it guarantees that the polytope radius only increases polynomially. The following example demonstrates this process.

$$5x_1 + 3x_2 - 7x_3 = -1 \quad \Rightarrow \quad (x_1 + x_2 - x_3)2^0 + (x_2 - x_3)2^1 + (x_1 - x_3)2^2 = -1 \cdot 2^0$$

It can be decomposed to 3 linear equations, together with carry terms $(c_i - d_i)$, where $c_i, d_i \geq 0$:

$$\begin{aligned} x_1 + x_2 - x_3 - 2(c_0 - d_0) &= -1 \\ x_2 - x_3 + (c_0 - d_0) - 2(c_1 - d_1) &= 0 \\ x_1 - x_3 + (c_1 - d_1) &= 0 \end{aligned}$$

Next, we reduce the 2-LEN instance $(\bar{\boldsymbol{A}}, \bar{\boldsymbol{b}}, \bar{R})$ to a 1-LEN instance $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{b}}, \hat{R})$ by replacing each ± 2 coefficient variable with two new equal-valued variables.

All the above three reduction steps run in nearly-linear time, and the problem sizes increase nearly-linearly.

3.1.2 Reducing Linear Equations With Nonnegative Variables and ± 1 Coefficients to Fixed Homologous Flow Problem

One of our main contributions is a linear-time reduction from 1-LEN to FHF (single-commodity fixed homologous flow problem). Our reduction is similar to Itai's reduction, but more efficient.

Itai observed that a single linear equation $\hat{\boldsymbol{a}}^\top \hat{\boldsymbol{x}} = \hat{b}$ with ± 1 coefficients can be represented as a fixed homologous flow network. We improve his construction by creating a *sparser* flow network in which the number of edges is proportional to the number of nonzero coefficients in the linear equations. Figure 1 depicts our gadget. Our gadget has a source vertex s, a sink vertex t, and two additional vertices J^+ and J^-. Each variable $\hat{\boldsymbol{x}}(i)$ with coefficient $\hat{\boldsymbol{a}}(i) \neq 0$ corresponds to an edge: There is an edge from s to J^+ if $\hat{\boldsymbol{a}}(i) = 1$, and there is an edge from s to J^- if $\hat{\boldsymbol{a}}(i) = -1$. The amount of flow passing this edge encodes the value of $\hat{\boldsymbol{x}}(i)$. Thus, the difference between the total amount of flow entering J^+ and that entering J^- equals to $\hat{\boldsymbol{a}}^\top \hat{\boldsymbol{x}}$. To force $\hat{\boldsymbol{a}}^\top \hat{\boldsymbol{x}} = \hat{b}$, we add two edges e_1, e_2 from J^+ to t and one edge e_3 from J^- to t; we require e_1 and e_3 to be homologous and require e_2 to be a fixed flow with value $\hat{b}$.

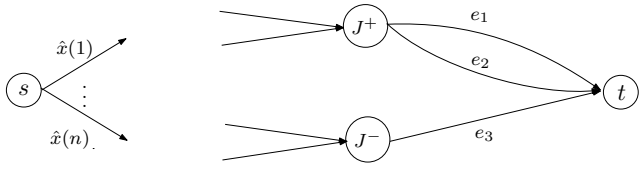


Figure 1 The gadget of reducing a single linear equation $\hat{\boldsymbol{a}}^\top \hat{\boldsymbol{x}} = \hat{b}$ with ± 1 coefficients to a fixed homologous flow network (FHF).

We can generalize this construction to encode a system of linear equations $\hat{\boldsymbol{A}}\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$, where $\hat{\boldsymbol{A}} \in \boldsymbol{Z}^{\hat{m} \times \hat{n}}$, $\hat{\boldsymbol{b}} \in \boldsymbol{Z}^{\hat{m}}$. Specifically, we create a gadget as above for each individual equation, and then glue all the source (sink) vertices for all the equations together as the source (sink, respectively) of the graph (see Figure 2). In addition to requiring e_i^+, e_i^- to be homologous for each single linear equation, to guarantee the variable values to be consistent in these

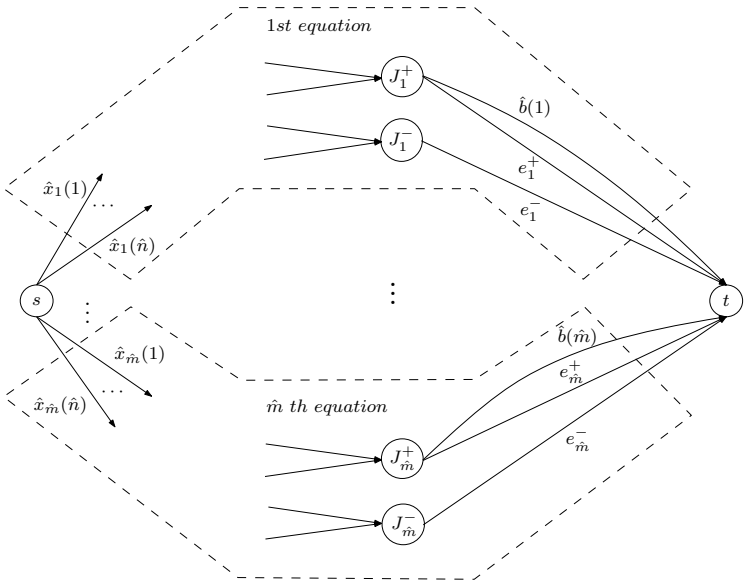


Figure 2 The reduction from 1-LEN to FHF.

equations, we also require the edges corresponding to the same variable $\hat{\boldsymbol{x}}(i)$ in different equations to be homologous, i.e., $\{\hat{x}_1(i), \ldots, \hat{x}_{\hat{m}}(i)\}$. We can check that the number of the vertices is linear in the number of equations; the number of the edges and the total size of the homologous sets are both linear in the number of nonzero coefficients of the linear equation system.

3.1.3 Dropping the Homologous and Fixed Flow Constraints

To reduce FHF to 2CF (2-commodity flow problem), we need to drop the homologous and fixed flow constraints. The reduction has three main steps.

Reducing FHF to SFF. Given an FHF instance, we can reduce it to a fixed homologous flow instance in which each homologous edge set has size 2 (FPHF). To drop the homologous requirement in FPHF, we introduce a second commodity of flow with source-sink pair (s_2, t_2), and for each edge, we carefully select the type(s) of flow that can pass through this edge. Specifically, given two homologous edges (v, w) and (y, z), we construct a constant-sized gadget (see Figure 3): We introduce new vertices vw, vw', yz, yz', construct a directed path

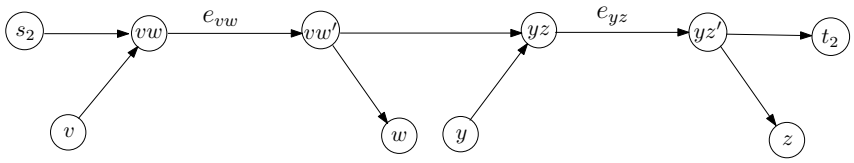


Figure 3 The gadget of reducing a pair of homologous edges $(v, w), (y, z)$ to a selective fixed flow network (SFF).

$P : s_2 \to vw \to vw' \to yz \to yz' \to t_2$, and add edges $(v, vw), (vw', w)$ and $(y, yz), (yz', z)$. Now, there is a directed path $P_{vw} : v \to vw \to vw' \to w$ and a directed path $P_{yz} : y \to yz \to yz' \to z$. Paths P and P_{vw} (P_{yz}) share an edge $e_{vw} = (vw, vw')$ ($e_{yz} = (yz, yz')$, respectively). We select e_{vw} and e_{yz} for both flow $\boldsymbol{f}_1^s$ and $\boldsymbol{f}_2^s$, select the rest of the edges along P for only $\boldsymbol{f}_2^s$, and select the rest of the edges along P_{vw}, P_{yz} for only $\boldsymbol{f}_1^s$. By this

construction, in this gadget, we have $\boldsymbol{f}_2^s(e_{vw}) = \boldsymbol{f}_2^s(e_{yz})$ being the amount of flow routed in P, $\boldsymbol{f}_1^s(e_{vw})$ and $\boldsymbol{f}_1^s(e_{yz})$ being the amount of flow routed in P_{vw} and P_{yz}, respectively. Next, we choose e_{vw} and e_{yz} to be fixed flow edges with equal capacity; this guarantees the same amount of $\boldsymbol{f}_1^s$ is routed through P_{vw} and P_{yz}. The new graph is an SFF instance.

Reducing SFF to 2CFF. Next, we will drop the selective requirement of the SFF instance. For each edge (x, y) selected for flow i, we construct a constant-sized gadget (see Figure 4): We introduce two vertices xy, xy', construct a direct path $s_i \to xy' \to xy \to t_i$, and add

Figure 4 The gadget of reducing a selective edge (x, y) for commodity i to a 2-commodity fixed flow network (2CFF).

edge (x, xy) and (xy', y). This gadget simulates a directed path from x to y for flow $\boldsymbol{f}_i^f$, and guarantees no directed path from x to y for flow $\boldsymbol{f}_{\bar{i}}^f$ so that $\boldsymbol{f}_{\bar{i}}^f$ cannot be routed from x to y. We get a 2CFF instance.

Reducing 2CFF to 2CF. It remains to drop the fixed flow constraints. The gadget we will use is similar to that used in the last step. We first introduce new sources $\bar{s}_1, \bar{s}_2$ and sinks $\bar{t}_1, \bar{t}_2$. Then, for each edge (x, y) with capacity u, we construct a constant-sized gadget (see Figure 5).

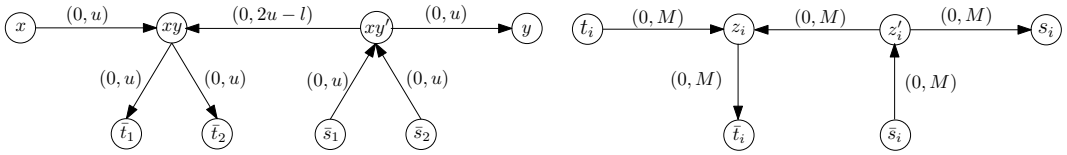


Figure 5 The gadget of reducing 2CFF to a 2-commodity flow with required flow amount (2CFR). l, u are lower and upper edge capacity of (x, y), respectively: $l = u$ if (x, y) is a fixed flow edge; $l = 0$ if (x, y) is a non-fixed flow edge.

We introduce two vertices xy, xy', add edges $(\bar{s}_1, xy'), (\bar{s}_2, xy'), (xy, \bar{t}_1), (xy, \bar{t}_2), (xy', xy)$, and $(x, xy), (xy', y)$. This simulates a directed path from x to y that both flow $\boldsymbol{f}_1^r$ and $\boldsymbol{f}_2^r$ can pass through. We let (xy', xy) have capacity u if (x, y) is a fixed flow edge and $2u$ otherwise; we let all the other edges have capacity u. Assume all the edges incident to the sources and the sinks are saturated, then the total amount of flows routed from x to y in this gadget must be u if (x, y) is a fixed flow edge and no larger than u otherwise. Moreover, since the original sources and sinks are no longer sources and sinks now, we have to satisfy the conservation of flows at these vertices. For each $i \in \{1, 2\}$, we create a similar gadget involving $\bar{s}_i, \bar{t}_i$ to simulate a directed path from t_i to s_i (the original sink and source), and let the edges incident to $\bar{s}_i, \bar{t}_i$ have capacity M, the sum of all the edge capacities in the 2CFF instance. This gadget guarantees that assuming the edges incident to $\bar{s}_i$ and $\bar{t}_i$ are saturated, the amount of flow routed from t_i to s_i through this gadget can be any number at most M. To force the above edge-saturation assumptions to hold, we require the amount of flow $\boldsymbol{f}_i^r$ routed from $\bar{s}_i$ to $\bar{t}_i$ to be no less than $2M$ for each $i \in \{1, 2\}$.

Now, this instance is close to a 2CF instance except that we require a lower bound for each flow value instead of a lower bound for the sum of two flow values. To handle this, we introduce new sources $\bar{\bar{s}}_1, \bar{\bar{s}}_2$ and for each $i \in \{1, 2\}$, we add an edge $(\bar{\bar{s}}_i, \bar{s}_i)$ with capacity $2M$, the lower bound required for the value of $\boldsymbol{f}_i^r$.

One can check that in each reduction step, the reduction time is nearly linear and the problem size increases nearly linearly. In addition, given a solution to the 2CF instance, one can construct a solution to the LP instance in nearly linear time.

3.1.4 Computing an Approximate Linear Program Solution From an Approximate 2-Commodity Flow Solution

We establish an error bound for mapping an approximate solution to 2CFA to an approximate solution to LPA. Below we outline the intuition behind our error analysis for flow problems. We will keep track of multiple types of error (e.g., error in congestion, demand, selective types, and homology depending on the problem settings). Now suppose we reduce problem P to problem Q using a certain gadget, and then we map a solution to Q back to a solution to P. We observe that each error notion of P is an additive accumulation of the multiple error notions of Q. This is because we have to map the flows of Q passing through a gadget including multiple edges back to a flow of P passing through a single edge. Each time we remove an edge, various errors related to this edge and incident vertices get transferred to its neighbors. Thus, the total error accumulation by the solution mapping can be polynomially bounded by the number of edges. So, the final error only increases polynomially.

References

1 Cynthia Barnhart, Niranjan Krishnan, and Pamela H. Vance. Multicommodity flow problems. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 2354–2362. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-74759-0_407.

2 L. Chen, G. Goranci, M. Henzinger, R. Peng, and T. Saranurak. Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1135–1146, November 2020. doi:10.1109/FOCS46700.2020.00109.

3 Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 273–282, 2011.

4 Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

5 James Demmel, Ioana Dumitriu, and Olga Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, 2007.

6 Ming Ding, Rasmus Kyng, and Peng Zhang. Two-commodity flow is equivalent to linear programming under nearly-linear time reductions. *arXiv preprint*, 2022. arXiv:2201.11587.

7 Efim A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

8 James R. Evans. A combinatorial equivalence between A class of multicommodity flow problems and the capacitated transportation problem. *Mathematical Programming*, 10(1):401–404, December 1976. doi:10.1007/BF01580684.

9 James R. Evans. The simplex method for integral multicommodity networks. *Naval Research Logistics Quarterly*, 25(1):31–37, March 1978. doi:10.1002/nav.3800250104.

10 Shimon Even and R. Endre Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975.

11 Lisa K. Fleischer. Approximating Fractional Multicommodity Flow Independent of the Number of Commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, January 2000. doi:10.1137/S0895480199355754.

12 Lester Randolph Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

13 Yu Gao, Yang P. Liu, and Richard Peng. Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao. *arXiv:2101.07233 [cs]*, January 2021. arXiv:2101.07233.

14 Naveen Garg and Jochen Könemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. *SIAM Journal on Computing*, 37(2):630–652, January 2007. doi:10.1137/S0097539704446232.

15 Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, September 1998. doi:10.1145/290179.290181.

16 T. C. Hu. Multi-Commodity Network Flows. *Operations Research*, 11(3):344–360, June 1963. doi:10.1287/opre.11.3.344.

17 Alon Itai. Two-commodity flow. *Journal of the ACM (JACM)*, 25(4):596–611, 1978.

18 Arun Jambulapati and Aaron Sidford. Ultrasparse Ultrasparsifiers and Faster Laplacian System Solvers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 540–559. SIAM, 2021.

19 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311, 1984.

20 Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit Capacity Maxflow in Almost $ O (m$\hat{}${$4/3$}$) $ Time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130. IEEE, 2020.

21 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 217–226. SIAM, 2014.

22 Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, pages 911–920, 2013.

23 Jeff L. Kennington. A Survey of Linear Cost Multicommodity Network Flows. *Operations Research*, 26(2):209–236, 1978.

24 Leonid G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

25 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching Optimality for Solving SDD Linear Systems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 235–244, USA, October 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.29.

26 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE, 2011.

27 Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 902–913, 2019.

28 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016.

29 Rasmus Kyng, Di Wang, and Peng Zhang. Packing LPs are hard to solve accurately, assuming linear equations are hard. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 279–296. SIAM, 2020.

30 Rasmus Kyng and Peng Zhang. Hardness results for structured linear systems. *SIAM Journal on Computing*, 49(4):FOCS17–280, 2020.

31 Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, pages 755–764, 2013.

32 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

33 T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast Approximation Algorithms for Multicommodity Flow Problems. *Journal of Computer and System Sciences*, 50(2):228–243, April 1995. doi:10.1006/jcss.1995.1020.

34 Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 803–814, 2020.

35 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, pages 121–130, New York, NY, USA, June 2010. Association for Computing Machinery. doi:10.1145/1806689.1806708.

36 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

37 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

38 Thomas Magnanti, R Ahuja, and J Orlin. Network flows: theory, algorithms, and applications. *PrenticeHall, Upper Saddle River, NJ*, 1993.

39 Cameron Musco, Praneeth Netrapalli, Aaron Sidford, Shashanka Ubaru, and David P. Woodruff. Spectrum Approximation Beyond Fast Matrix Multiplication: Algorithms and Hardness. *arXiv:1704.04163 [cs, math]*, January 2019. arXiv:1704.04163.

40 A. Ouorou, P. Mahey, and J.-Ph. Vial. A Survey of Algorithms for Convex Multicommodity Flow Problems. *Management Science*, 46(1):126–147, 2000.

41 Richard Peng. Approximate undirected maximum flows in o (m polylog (n)) time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1862–1867. SIAM, 2016.

42 Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, pages 333–342, 2014.

43 James Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988.

44 James Renegar. Incorporating Condition Measures into the Complexity Theory of Linear Programming. *SIAM Journal on Optimization*, 5(3):506–524, August 1995. doi:10.1137/0805026.

45 Jonah Sherman. Nearly Maximum Flows in Nearly Linear Time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 263–269, October 2013. doi:10.1109/FOCS.2013.36.

46 Jonah Sherman. Area-convexity, l∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017.

47 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 81–90, New York, NY, USA, June 2004. Association for Computing Machinery. doi:10.1145/1007352.1007372.

48 Pravin M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337. IEEE Computer Society, 1989.

49 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, MDPs, and L1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 859–869, New York, NY, USA, June 2021. Association for Computing Machinery. doi:10.1145/3406325.3451108.

50 I.-Lin Wang. Multicommodity network flows: A survey, Part I: Applications and Formulations. *International Journal of Operations Research*, 15(4):145–153, 2018.

51 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *Journal of the ACM*, 65(5):27:1–27:38, August 2018. doi:10.1145/3186893.

High-Probability List-Recovery, and Applications to Heavy Hitters

Dean Doron
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Mary Wootters
Departments of Computer Science and Electrical Engineering, Stanford University, CA, USA

Abstract

An error correcting code $\mathcal{C}\colon \Sigma^k \to \Sigma^n$ is efficiently list-recoverable from input list size ℓ if for any sets $\mathcal{L}_1, \ldots, \mathcal{L}_n \subseteq \Sigma$ of size at most ℓ, one can efficiently recover the list $\mathcal{L} = \{x \in \Sigma^k : \forall j \in [n], \mathcal{C}(x)_j \in \mathcal{L}_j\}$. While list-recovery has been well-studied in error correcting codes, all known constructions with "efficient" algorithms are not efficient in the parameter ℓ. In this work, motivated by applications in algorithm design and pseudorandomness, we study list-recovery with the goal of obtaining a good dependence on ℓ. We make a step towards this goal by obtaining it in the weaker case where we allow a *randomized* encoding map and a small failure probability, and where the input lists are derived from unions of codewords. As an application of our construction, we give a data structure for the heavy hitters problem in the strict turnstile model that, for some parameter regimes, obtains stronger guarantees than known constructions.

2012 ACM Subject Classification Theory of computation → Error-correcting codes; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases List recoverable codes, Heavy Hitters, high-dimensional expanders

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.55

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2020/162/` [11]

Funding *Dean Doron*: The work was done at Stanford, supported by NSF award CCF-1763311.
Mary Wootters: Supported by NSF award CCF-1844628 and NSF-BSF award CCF-1814629.

Acknowledgements We would like to thank Jelani Nelson and Amnon Ta-Shma for helpful conversations. We thank Mahdi Cheraghchi, Venkat Guruswami, and Badih Ghazi for pointing out relevant related work. We also thank anonymous reviewers for helpful comments and for pointing out related works.

1 Introduction

Let $\mathcal{C}\colon \Sigma^k \to \Sigma^n$ be an *error correcting code*. We say that $\mathcal{C}$ is (efficiently) *list-recoverable*[1] from list-size ℓ with output list-size L if, for any lists $\mathcal{L}_1, \ldots, \mathcal{L}_n \subseteq \Sigma$ with $|\mathcal{L}_i| \leq \ell$ for all i, there is an (efficient) algorithm to recover the list

$$\mathcal{L} = \{x \in \Sigma^k : \forall i \in [n], \mathcal{C}(x)_i \in \mathcal{L}_i\},$$

and $|\mathcal{L}| \leq L$. List recovery has historically been studied in the context of *list-decodable* codes, where it has been used as a tool to obtain efficient list-decoding algorithms (see, e.g., [17, 16, 19, 27, 22]). However, even though efficient list-recovery algorithms have been

[1] In this paper we focus on *zero-error* list-recovery, which is the definition given here. Other works focus on the more general problem of list-recovery from errors, in which $\mathcal{C}(x)_i$ needs to be in $\mathcal{L}_i$ only for some fraction of the i-s.

developed, all of them have a poor dependence on the parameter ℓ. For example, Hemenway, Ron-Zewi, and Wootters [22] presents near-linear-time (in n) list-recovery algorithms, but the output list $\mathcal{L}$ has size doubly-exponential in ℓ.

In this work, we are motivated by the following goal (which we do not fully achieve):

▶ **Goal 1.** *For $\ell \geq 2$, design a family of codes $\mathcal{C} \colon \Sigma^k \to \Sigma^n$ so that:*

1. *$\mathcal{C}$ can be encoded in time $O(n)$;*
2. *The* rate *k/n of the code is a constant (independent of n **and** ℓ);*
3. *The* alphabet size *$|\Sigma|$ is polynomial in ℓ (and independent of n);*
4. *The code $\mathcal{C}$ can be list-recovered in time $O(n \cdot \ell)$ (linear in both n **and** ℓ), with output list size $|\mathcal{L}| = O(\ell)$.*

To the best of our knowledge, this goal is open even if we allow the output list size $|\mathcal{L}|$ and the running time to depend polynomially on ℓ, rather than linearly.

Goal 1 is desirable for several reasons. First, it represents a bottleneck in our understanding of algorithmic coding theory, and it seems likely that achieving it would involve developing new techniques that would be useful elsewhere. Second, list-recovery with reasonable dependence on ℓ is related to questions in pseudorandomness, where the the parameter ℓ is often very large (see our discussion in Section 1.2). Third, as we explore in this paper, obtaining Goal 1 has applications in algorithm design, in particular to algorithms for heavy hitters.

Probabilistic list-recovery with good dependence on ℓ

In this work, we make progress on Goal 1 by achieving a relaxed version where the encoding map $\mathcal{C} \colon \Sigma^k \to \Sigma^n$ is allowed to be *randomized*, and where the input lists are generated from unions of codewords; we must succeed with high probability over the randomness in $\mathcal{C}$. In particular, our main result implies the following theorem.

▶ **Theorem 2** (informal; weaker than main result)**.** *For all $\ell > 0$, there is a randomized encoding map $\mathcal{C} \colon \Sigma^k \to \Sigma^n$ so that*

1. *$\mathcal{C}$ can be encoded in time $O(n)$;*
2. *The rate of $\mathcal{C}$, k/n, is a constant independent of ℓ and n;*
3. *The alphabet size $|\Sigma|$ is polynomial in ℓ (and independent of n);*
4. *For any list $x^{(1)}, x^{(2)}, \ldots, x^{(\ell)} \in \Sigma^k$, there is an algorithm that runs in time $O(n\ell \operatorname{polylog} \ell)$ that has the following guarantee. With probability at least $1 - o(1)$ over the randomness of $\mathcal{C}$, given the lists $\mathcal{L}_i = \{\mathcal{C}(x^{(j)})_i : j \in [\ell]\}$, the algorithm returns a list $\mathcal{L}$ so that $x^{(i)} \in \mathcal{L}$ for all i, and so that $|\mathcal{L}| = O(\ell)$.*

This statement is weaker than our main result because in fact our result still holds even if a random subset of the lists $\mathcal{L}_i$ in Item 4 are erased, and moreover the result still holds when some of the lists $\mathcal{L}_i$ in Item 4 contain some extra "distractor" symbols that occur according to any sufficiently "nice" distribution. We defer the formal statement of our list-recovery guarantee to Section 2.

Our code is essentially an *expander code* with *aggregated symbols*. That is, we begin with an expander code $\mathcal{C}_0 \colon \Sigma_0^k \to \Sigma_0^n$, as in [39], and we aggregate together the symbols as in [1]. (We discuss this construction in more detail below.) Our recovery algorithm uses ideas from previous algorithms, propagating information around the underlying expander graph given some advice. What makes our work different are the facts that (a) we leverage the randomness of $\mathcal{C}$ and a small failure probability, and (b) our underlying expander graph comes

from a *high-dimensional expander*.[2] In particular, using the randomness in $\mathcal{C}$ we are able to obtain an algorithm with running time nearly-linear in ℓ, and using a high-dimensional expander we are able to boost our success probability to a level appropriate for an application to heavy hitters, which we discuss next.

Motivation from Heavy Hitters

One of the reasons we are interested in Goal 1 is because of the potential algorithmic applications of such a code. To illustrate this potential, we work out an application of our construction to the *heavy hitters* problem. **We emphasize that our focus is on the parameter regime where N is very large, specifically $\log N \gg \mathbf{poly}(1/\varepsilon)$.** In particular, we are interested in optimizing the dependence on N, rather than on ε.

The set-up is as follows. We are given a stream of updates $(x^{(i)}, \Delta^{(i)})$, for $x^{(i)}$ in some universe $\mathcal{U}$ of size N, and $\Delta^{(i)} \in \mathbb{R}$. For all m, x, we assume that $f(x) \triangleq \sum_{j\in[m]} \Delta^{(j)} \cdot \mathbf{1}_{x^{(j)}=x} > 0$. we think of $f(x)$ as the "frequency" of item x. The Δ-s are updates: we may add or remove some quantity of each item x, provided that $f(x)$ never becomes negative. This is called the *strict turnstile model.* The goal is to maintain a small data structure (a "sketch") so that, after m (efficient) updates $(x^{(i)}, \Delta^{(i)})$, we can (efficiently) query the data structure to return a list of *ε-heavy hitters.* That is, we would like to recover a list $\mathcal{L}$ of size at most $O(1/\varepsilon)$ that contains all $x \in \mathcal{U}$ so that $f(x) \geq \varepsilon \cdot \|f\|_1 \triangleq \sum_{x\in\mathcal{U}} f(x)$.

The beautiful *Count-Min Sketch* (CMS) data structure of Cormode and Muthukrishnan [7] gives a solution to this problem. It uses optimal space $O(\varepsilon^{-1} \log N)$ and has update time $O(\log N)$. However, the query time to return all $O(1/\varepsilon)$ heavy hitters is large, $O(N \log N)$ (essentially, one performs a point query for each $x \in \mathcal{U}$ to see if it is a heavy hitter). The work [7] showed how to alleviate this with a so-called "dyadic trick," bringing the query time to $O(\log^2 N)$ at the cost of an extra $\log N$ factor in both the space and update time.[3] (See Table 1 for a summary of the parameters in these and other works).

The starting point for our work is the work of Larsen, Nelson, Nguyễn and Thorup [29]. That work studied a much more general problem – heavy hitters for all ℓ_p norms in the general turnstile model – but for the special case of the ℓ_1 norm and the strict turnstile model, they were able to get a nearly optimal algorithm, with the same space and update time complexity as the original CMS, but with query time $O(\varepsilon^{-1} \log^{1+\gamma} N)$ for any constant $\gamma > 0$. That work highlighted a connection to list-recovery (see [29, Section C]; a similar connection is also present in earlier works on group testing and compressed sensing, for example [24, 35, 36, 13, 12]), which is one of our motivations to study Goal 1.

The approach of [29] was the following (we have modified the description to be more explicitly coding-theoretic). To perform an update on an item $x \in \mathcal{U}$, encode it as $\mathcal{C}(x) \in \Sigma^n$ with our (randomized) encoding function. Then insert each symbol $\mathcal{C}(x)_j$ into n different ε-heavy hitters data structures that work on universe Σ (this could be a small CMS sketch, or something else). To query all of the heavy hitters, we first query each smaller data structure to find a list $\mathcal{L}_j$. Notice that since $|\Sigma| \ll |\mathcal{U}|$, it does not matter that the query algorithm for the small data structures is slow. Now, we do list-recovery on the lists $\mathcal{L}_j$ to recover a list $\mathcal{L}$ that contains all of the heavy hitters.[4]

[2] We note that the construction of Dinur et al. [9] is similar to ours, also using an ABNNR-style [1] symbol aggregation with a high-dimensional expander. However, in that work they have a more ambitious goal – list-decoding with no randomness in the encoder – but in return the parameters are not close to those in Goal 1.

[3] See also the work by Cormode and Hadjieleftheriou [6] who consider a generalization of the dyadic trick that trades off between the query time and the overhead in update time and space.

[4] Provided that the output $\mathcal{L}$ of the list-recovery algorithm is not too large, we can use an additional large CMS data structure to efficiently do point queries on each item $x \in \mathcal{L}$, pruning it down to $O(1/\varepsilon)$.

Table 1 Some relevant results on ε-heavy hitters in the strict turnstile model where the universe has size N, for $\log N \gg \mathrm{poly}(1/\varepsilon)$. We consider schemes with failure probability $\delta \geq 1/\mathrm{poly}(N)$; see the discussion in Section 1.3 for smaller failure probability where the works marked with $\star$ shine. The $\tilde{O}$ notation hides $\log\log(N)$ factors and $\log(1/\varepsilon)$ factors. Above, c is a constant independent of N and ε, and γ is any constant larger than 0. Unfortunately, the failure probability for our algorithm is only $N^{-\mathrm{poly}(\varepsilon)}$, rather that N^{-c} for some constant c. By repeating our algorithm $\mathrm{poly}(1/\varepsilon)$ times we can boost the success probability to N^{-c}. We note that each of Space, Update, Query time for [7] (with the dyadic trick) and [31] can be multiplied by ε^c if one replaces the failure probability with $N^{-\varepsilon^c}$ and the results from [29, Theorem 9] remain the same for that larger failure probability.

Reference	Space	Update	Query	Failure probability
[7]	$O\left(\frac{\log N}{\varepsilon}\right)$	$O(\log N)$	$O(N \log N)$	N^{-c}
[7] ("dyadic trick")	$O\left(\frac{\log^2 N}{\varepsilon}\right)$	$O(\log^2 N)$	$O\left(\frac{\log^2 N}{\varepsilon}\right)$	N^{-c}
[29]	$O\left(\frac{\log N}{\varepsilon}\right)$	$O(\log N)$	$O\left(\frac{\log^{1+\gamma} N}{\varepsilon}\right)$	N^{-c}
[31]*	$\widetilde{O}\left(\log^2 N\right)$	$\widetilde{O}\left(\varepsilon \log^2 N\right)$	$\frac{1}{\varepsilon}\,\mathrm{poly}(\log N)$	N^{-c}
[5]*	$O\left(\frac{\log N}{\varepsilon}\right)$	$\widetilde{O}(\log N)$	$\frac{1}{\varepsilon}\,\mathrm{poly}(\log N)$	0
This work	$O\left(\frac{\log N}{\varepsilon}\right)$	$O(\log N)$	$O\left(\frac{\log N}{\varepsilon}\right)$	$N^{-\mathrm{poly}(\varepsilon)}$
This work	$O\left(\frac{\log N}{\varepsilon^c}\right)$	$O\left(\frac{\log N}{\varepsilon^c}\right)$	$O\left(\frac{\log N}{\varepsilon^c}\right)$	N^{-c}

However, as Goal 1 remains open, [29] did not use a list-recoverable code to obtain their results. Instead, they (like us) took advantage of the fact that the lists $\mathcal{L}_j$ can be viewed as random variables over the randomness in the encoding map $\mathcal{C}$, and then use a construction based on "cluster-preserving clustering" to solve the problem. While in some sense this construction must be a list-recoverable code for randomized input lists, it is not clear (to us) how to extract a natural code out of it: the work [29] took the perspective of graph clustering, rather than coding theory. In contrast, our code is very natural in the context of coding theory, as it is simply an expander code with aggregated symbols (albeit using a high-dimensional expander for the underlying graph).

As an example of the utility of our construction, we plug our randomized list-recoverable code (as in Theorem 2) into the framework of [29]. This gives us an algorithm for heavy hitters that, in some parameter regimes, even slightly outperforms that of [29]. When ε is constant and N is growing, we are able to improve the query time from $O(\log^{1+\gamma} N)$ to $O(\log N)$. In particular, we prove the following theorem. (See Table 1 for a comparison to other work when $\log N \gg \mathrm{poly}(1/\varepsilon)$).

▶ **Theorem 3** (informal; see Theorem 5.11 in the full version)**.** *There is a data structure that solves the heavy hitters problem in the strict turnstile model, that uses space $O(\varepsilon^{-1}\log N)$, update time $O(\log N)$, and query time $O(\varepsilon^{-1}\log N\,\mathrm{polylog}(1/\varepsilon))$, with failure probability $\delta = N^{-\Theta(\varepsilon^3)}$, as long as $\varepsilon \geq (\log N)^{-\Omega(1)}$.*

By repeating this data structure $O(\varepsilon^{-3})$ times, we obtain a data structure that takes space $O(\varepsilon^{-4}\log N)$, update time $O(\varepsilon^{-3}\log N)$ and query time $O(\varepsilon^{-4}\log N\,\mathrm{polylog}(1/\varepsilon))$, with failure probability $\delta = N^{-c}$.

Our algorithm has the added property that a successful $\mathcal{L}$ of size $O(1/\varepsilon)$ not only contains all the true heavy hitters, but also does not contain "false-positives", in the sense that each $x \in \mathcal{L}$ satisfies, say, $f(x) \geq \frac{\varepsilon}{4}\|f\|_1$. This property also applies to most previous heavy hitters algorithms.

Contributions

To summarize, our main contributions are the following.

1. **A code with probabilistic list-recovery.** We give a natural code construction that achieves a probabilistic version of Goal 1, as per Theorem 2. Our code construction leverages recent progress in high-dimensional expanders in order to succeed with high probability. We hope that our construction and techniques may be used in the the future to make further progress on Goal 1.
2. **Proof of concept: application to heavy hitters.** As an illustration of the utility of our construction – and as an proof-of-concept meant to encourage study of Goal 1 – we obtain a new data structure for ε-heavy hitters in the strict turnstile model. Our data structure has slightly stronger guarantees than existing constructions for failure probability $1/\operatorname{poly}(N)$ when ε is constant and the universe size N is growing (although it is outperformed by previous work when ε is small compared to $1/\log(N)$).

1.1 Construction Overview

In this section, we give a brief overview of our probabilistically list-recoverable code. We use this code to solve the ε-heavy-hitters problem following the paradigm described above, by using small heavy-hitters sketches for each symbol of the (randomized) encoding $\mathcal{C}(x)$ of $x \in \mathcal{U}$.

At a high level, we construct our code $\mathcal{C}\colon \Sigma_0^k \to \Sigma^{n'}$ as follows. We start with some base code $\mathcal{C}_0\colon \Sigma_0^k \to \Sigma_0^n$, as well as a bipartite expander graph $G = (R, L, E)$, where $L = [n]$ and $R = [n']$, for some $n' = O(n)$.[5] We will need $\mathcal{C}_0$ and G to have specific properties, which we will come to below. For $x \in \Sigma_0^k$, we generate the encoding $\mathcal{C}(x)$ as follows. For $j \in [n']$, the encoded symbol $\mathcal{C}(x)_j$ will be gotten as the concatenation of the symbols $\mathcal{C}_0(x)_i$ for $i \in \Gamma_G(j)$, where $\Gamma_G(j)$ denotes the neighbors of j in the graph G. This sort of "aggregation along an expander" technique, introduced in [1], has become a standard distance amplification technique in error correcting codes. Because of the concatenation, our final alphabet Σ will be $\Sigma = \Sigma_0^{m_2}$.

To perform list recovery, we will start with a small piece of "advice," and then recover the (hopefully unique) message x consistent with that advice. We will generate our final list $\mathcal{L}$ by iterating over all possible values of the advice. Towards this end, we will choose some coordinate $j \in [n']$ for which $\mathcal{L}_j$ is not erased, and some $\sigma^\star \in \mathcal{L}_j$ as our guess for $\mathcal{C}_0(x)|_{\Gamma_G(j)}$ to act as our advice. Given this advice $\sigma^\star$, we wish to keep propagating information until we obtain enough coordinates of $\mathcal{C}_0$ that would allow us to uniquely determine x; this amounts to decoding the code $\mathcal{C}_0$ from erasures.

In the exposition below, we start with a naive attempt to do this propagation, and build up the properties that we will need $\mathcal{C}_0$ and G to satisfy as we refine it. Our construction is depicted in Figure 1.

A naive attempt

Our first attempt (which will not work) is the following. Let $j \in [n']$ be as above, so we assume that we are given as advice the m_2 symbols $\mathcal{C}_0(x)|_{\Gamma_G(j)}$; our goal is to recover (a hopefully unique) x given this advice and given the input lists $\mathcal{L}_{j'}$ for $j' \in [n']$. Choose some coordinate $j' \in [n']$ such that $\Gamma_G(j) \cap \Gamma_G(j') \neq \emptyset$. As we already know the symbols in the

[5] Using the notation of the full version, $\Sigma_0 = \mathbb{F}_q$, $\Sigma = \mathbb{F}_q^{m_2}$ for some constant m_2, and $n' = |V_2| = O(n)$.

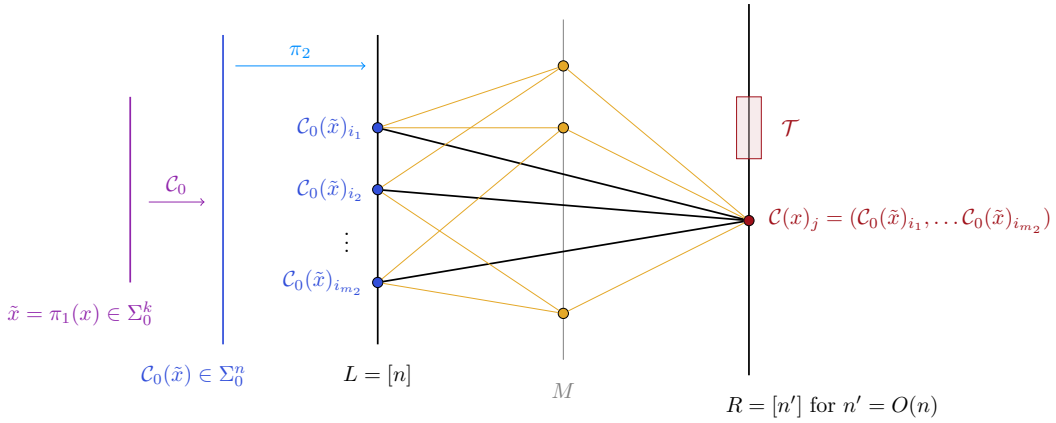


Figure 1 Illustration of our construction. The coordinates of the inner code $\mathcal{C}_0$ live on the vertices of L. The final code $\mathcal{C}$ consists of symbols aggregated by vertices in R. The randomness in the encoding comes from the permuations π_1 and π_2, which scramble the messages in Σ^k and the coordinates in $[n]$, respectively. We use the vertices in $\mathcal{T} \subseteq R$ to define parity checks that partially define the code $\mathcal{C}_0$. The "middle layer" M is not used in the definition of the code, but is a necessary auxiliary structure for our recovery algorithm.

coordinates indexed by $\Gamma_G(j)$, this gives us partial information about $\mathcal{C}(x)_{j'}$ in the form of $|\Gamma_G(j) \cap \Gamma_G(j')|$ elements of Σ_0 in known locations. One can hope that this information would be enough to pinpoint a specific entry in the list $\mathcal{L}_{j'}$, allowing us to recover all symbols of $\mathcal{C}_0(x)$ in the coordinates indexed by $\Gamma_G(j')$, and keep going in the same manner until enough information is propagated.

Clearly, when we have no guarantee on the input lists $\mathcal{L}_i$, this approach fails miserably, as it may be the case that $\mathcal{L}_{j'}$ contains numerous elements in $\Sigma_0^{m_2}$ that agree in some of the m_2 locations, and the information coming from our advice for j will not uniquely pin down an element of $\mathcal{L}_{j'}$. However, note that for a *completely random* input list $\mathcal{L}_{j'}$, such an attempt would be successful with probability at least $1 - |\mathcal{L}_{j'}| / |\Sigma_0|$, and we could set the parameters in such a way that $|\mathcal{L}_{j'}| \ll |\Sigma_0|$. That is, in this case it would become reasonably likely that the choice of $\sigma^\star \in \mathcal{L}_j$ would uniquely pin down an element $\sigma \in \mathcal{L}_{j'}$, allowing us to propagate information to another vertex in the graph. The hope is that we could propagate this information throughout the graph, using the fact that G is an expander to guarantee that most vertices will be determined. Of course, the problem with this is that we do not want to assume that the input lists are completely random, but this leads us to our next attempt, where we inject randomness into the encoding procedure.

Injecting randomness

While we won't get completely random lists $\mathcal{L}_j$ as we might have wanted for the naive attempt, we can make the input lists randomized via a randomized encoding. More specifically, our base code $\mathcal{C}_0$ will be deterministic, and to apply $\mathcal{C}$ we will make use of two permutations: a permutation π_1 acting on the universe $\mathcal{U}$ and a permutation π_2 acting on $[n]$. More formally, given $x \in \Sigma_0^k$, we first apply $\pi_1(x)$ and apply the encoding $\mathcal{C}_0$ to $\pi_1(x)$. Next, we permute the coordinates of the outcome according to π_2. Finally, we aggregate symbols according to G, yielding $\mathcal{C}(x) \in \Sigma^{n'}$. Roughly speaking, the first permutation – which will be pairwise independent – will make $\mathcal{C}_0(x)$ uniformly distributed over the code's image, even conditioned the value of $\mathcal{C}_0(x')$ for some $x' \neq x$. The second permutation will make sure that querying

any particular symbol $\mathcal{C}_0(x)_j$ symbol will behave like sampling a uniformly random symbol in $\mathcal{C}_0(x)$, and even more strongly, combined with π_1 it will behave like a random sampling from a nearly uniform distribution over Σ_0.

Analyzing the permutation-aided construction carefully, we are able to show that indeed, with probability roughly $1-\eta$ for $\eta \approx |\mathcal{L}_{j'}|/\sqrt{|\Sigma_0|}$, we can pinpoint a single list element of $\mathcal{L}_{j'}$. One conceptual observation that will help us establish that result is the fact that the distribution of symbols in most codewords of a high-rate code is close to uniform, and indeed we will need the rate of $\mathcal{C}_0$ to be very high (see Section 3.3 of the full version). We leave the more technical details to Section 5 of the full version.

Although promising, this approach is still problematic. We start with $m_2 = O(1)$ symbols that we know, and at each iteration the set of revealed coordinates grows by a small constant factor, using the expansion properties of G. As initially our sets are of constant size, we cannot hope for success probability much greater than $1-\eta$ for the initial propagation steps. A failure probability of η, even if we disregard the need for a problematic union bound over all propagation steps, is far too large for us, and in particular for our application to heavy hitters. The problem described here is common to various expander-based techniques, and in this work we resolve it by choosing G to be a special expander graph that comes from a high-dimensional expander, and by choosing $\mathcal{C}_0$ to be a suitable *Tanner code*. We discuss these modifications next.

Using high-dimensional expanders to get a good head start

We resolve the issue described above – that we cannot possibly get a good failure probability if we start with only a few known symbols – by using techniques from high-dimensional expanders. Suppose that, starting with only the advice $\sigma^\star$ for m_2 symbols of $\mathcal{C}_0(x)$, we could deterministically identify a large subset $\mathcal{T} \subseteq [n']$ for which we know all symbols of $\mathcal{C}_0(x)$ indexed by $\Gamma_G(\mathcal{T})$. This way, concentration bounds can kick in, and hopefully each propagation step would be successful with probability roughly $\eta^{|\mathcal{T}|}$, provided we can get a enough independence between query attempts at the same propagation step. We defer the independence issue to the full version (this ends up following from the amount of independence we have in our permutations π_1 and π_2), and concentrate on obtaining such a $\mathcal{T}$.

Recall that we work over the bipartite expander graph $G = (R = [n'], L = [n], E)$. We will construct G', a tripartite extension of G, with an added middle layer M, $|M| = O(n)$, having the following property. Identify each vertex j of R with a subset $\Gamma_G(j) \subset [n]$ of cardinally m_2 in the natural way. Each vertex in M is identified with a subset $S \subset [n]$ of cardinality m_1, for $1 < m_1 < m_2$, such that S is connected to all its m_1 elements on the left, and to all its supersets on the right. More specifically, each vertex j in R will be connected to all $\binom{m_2}{m_1}$ subsets of $\Gamma_G(j)$ in M. (See Figure 1 for an illustration.)

We will choose the code $\mathcal{C}_0$ to be a *Tanner code* with respect to the structure of the graph G. That is, as before, we associate the n symbols of a codeword $\mathcal{C}_0(x)$ with the left hand vertices L of G, and we define $\mathcal{C}_0$ so that a codeword $\mathcal{C}_0(x)$ is a labeling of L so that to following property holds: For every j in an appropriate subset $\mathcal{T} \subset R$, the labels on the vertices of $\Gamma_G(j)$ form a codeword in some error correcting code $\mathcal{C}_{00}$ of length m_2 with good distance; in particular, given any m_1 symbols of $\mathcal{C}_{00}(x')$ for some x', we can recover all of $\mathcal{C}_{00}(x')$.

The reason to choose $\mathcal{C}_0$ like this is the following. Say we know that j and j' are in the set $\mathcal{T}$, and that they have a common neighbor in M. This implies that $|\Gamma_G(j) \cap \Gamma_G(j')| \geq m_1$, since there is some set of size m_1 that both of those sets contain. In particular, by our choice of $\mathcal{C}_{00}$, once we know the symbols of $\Gamma_G(j)$, we can *deterministically* reveal all symbols of

$\Gamma_G(j')$ by decoding $\mathcal{C}_{00}$. Then we can continue this process until we recover the symbols in $\Gamma_G(j)$ for all $j \in \mathcal{T}$. By counting constraints, it turns out that we can choose $\mathcal{T}$ to be large and still have a high-rate code $\mathcal{C}_0$. This gives us our set $\mathcal{T}$ so that we can deterministically fill in the symbols of $\Gamma_G(\mathcal{T})$ to use as a head start and increase our success probability.

How do we construct such a tripartite graph, that on the one hand has not too many vertices in R and M (i.e., $R = O(n)$ and $M = O(n)$), but on the other hand has favorable intersection and expansion properties? This is where *high-dimensional expanders* enter the picture, and indeed the tripartite graph comes from an $(m_2 - 1)$-dimensional simplicial complex (see the full version for the formal definitions). A similar object was used by Dinur et al. [9] as a *double sampler*, and in Dikstein et al. [8] as a *multilayer agreement sampler*. We note that the construction of [9] is quite similar to ours, as they also use the symbol-aggregation technique of Alon et al. [1]; the main difference in the construction is that we use a very specific inner code $\mathcal{C}_0$ that uses the structure of G as part of its parity checks, while the work of [9] chooses $\mathcal{C}_0$ to be an arbitrary code with good distance.

In our actual construction, the code $\mathcal{C}_0$ is a bit more involved, and its constraints arise both from the special subset $\mathcal{T}$ of R and from an *additional* bipartite expander. Each of the two types of constraints is helpful for a different aspect of our algorithm. Roughly speaking, the constraints that come from $\mathcal{T} \subseteq R$ help us as described above (filling in the set $\Gamma_G(\mathcal{T})$ to get a head start). The other constraints are there to ensure that the final code $\mathcal{C}_0$ has good enough distance to allow for the final unique decoding. All in all, we are able to achieve a set $\mathcal{T}$ that has size about $|\mathcal{T}| \approx \text{poly}(\varepsilon) \cdot n$. We remark that this is the point where we don't quite get the failure probability that we want, resulting in a sub-optimal dependence on ε for our application to heavy hitters: we want failure probability $\exp(-n)$ (we will choose n logarithmic in N, so this would be $\text{poly}(1/N)$), and we end up with failure probability $\exp(-|\mathcal{T}|) = \exp(-\text{poly}(\varepsilon)n)$.

There are plenty of details that are swept under the rug in the description above, including implementation details needed to keep the recovery algorithm linear-time. We give the recovery algorithm in detail in the full version of the paper. We present our list-recovery algorithm in the context of a query algorithm for heavy hitters, since for our analysis we want to focus on the distribution of input lists that arises from the heavy hitters example, and it is easiest to present everything together. In particular, the input lists do not arise simply from the union of ℓ codewords $\mathcal{C}(x)$, but (a) may be erased if the corresponding small data structure failed, and (b) may contain extraneous symbols that arise from items $x^{(i)}$ that appear in the stream that are not heavy hitters.

1.2 Motivating Goal 1 from Pseudorandomness

In this section, we briefly explain why Goal 1 – and in particular, getting a good dependence on the parameter ℓ – is of interest in pseudorandomness. There is a tight connection between error correcting codes and fundamental constructions in pseudorandomness, notably the equivalence between (strong) seeded extractors and list-decodable codes [41, 40]. It turns out that list recovery can also play a prominent role in the study of related objects from extractor theory. In *seeded condensers*, first studied in [38], the goal is to "improve" the quality of a random source $\boldsymbol{X}$ using few additional random bits. A bit more formally, given a random variable $\boldsymbol{X} \sim \{0,1\}^n$ with min-entropy k, a condenser $\mathsf{Cond}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is such that $\mathsf{Cond}(\boldsymbol{X}, U_d)$ has min-entropy k', where we want the entropy rate to improve, namely, $\frac{k'}{m} \gg \frac{k}{n}$, and to maintain a small entropy gap $m - k'$. (For the formal definition,

see, e.g., [18].) List recoverable codes in the *errors* model[6] give seeded condensers, and vice versa. More specifically, the input and output entropies k and k' are almost in one-to-one correspondence with the (logarithm of the) output and input list sizes, $\log|\mathcal{L}|$ and $\log \ell$ (for the precise statement, see [10]). Thus, to get meaningful condensers from list-recoverable codes, the dependence between L and ℓ needs to be good, in all regime of parameters, and in particular handle ℓ that grows arbitrarily with the message length. In fact, the best list-recoverable code in this regime is the (folded) Parvaresh-Vardy code [18], giving $|\mathcal{L}| \approx \ell$.[7] The connection between condensers and list-recoverable codes was recently utilized in the *computational* setting to construct nearly-optimal pseudorandom generators for polynomial-sized circuits [10].

The model of *zero-error list recovery*, described in Goal 1 (when $|\mathcal{L}|$ depends nicely on ℓ and ℓ can be arbitrary), has applications to pseudorandomness too. A (strong) *disperser* is a function $\mathsf{Disp}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that for any random variable $\boldsymbol{X} \sim \{0,1\}^n$ with sufficient min-entropy, the support of $\mathsf{Disp}(\boldsymbol{X}, U_d)$ is large. Such dispersers have found several applications, and are tightly connected to open problems in expander graphs. It is not hard to show, and we do so in Appendix B of the full version, that dispersers, in some parameter regime, are equivalent to zero-error list-recoverable codes. We are not aware of this equivalence being stated elsewhere. For completeness, we note that dispersers in another parameter regime give rise to erasure list-decodable codes [3].

Finally, observe that in order to get good pseudorandomness primitives from list-recoverable codes, efficient recovery is not an issue, and all that is needed is an efficient *encoding*.

Even though a probabilistic guarantee as in Theorem 2 does not immediately yield improved pseudorandom objects, it is our hope that our progress on Goal 1 is a first step towards achieving that goal, which would imply improved dispersers.

1.3 Related Work

Algorithmic List-Recovery

List-recovery was originally introduced as an avenue towards *list-decoding*, where the goal is, given a vector $z \in \Sigma^n$, to recover the list $\mathcal{L}$ of all messages $x \in \Sigma^k$ so that $\mathcal{C}(x)$ is sufficiently close to z in Hamming distance. For example, the celebrated list-decoding algorithm of Guruswami and Sudan for Reed-Solomon codes [17] is in fact a list-recovery algorithm. However, the Guruswami-Sudan algorithm stops working at the so-called *Johnson bound*, which in the context of list-recovery means that the rate k/n of the code can be at most $1/\ell$. Since the Guruswami-Sudan algorithm, there has been a great deal of work, mostly based on algebraic constructions, aimed at surpassing the Johnson bound for list-decoding and list-recovery. In particular, the works [16, 19, 27, 28] show variations of Reed-Solomon codes, like folded RS codes and multiplicity codes, can be efficiently list-decoded and list-recovered beyond the Johnson bound. For list-recovery, these constructions are able to obtain rate $k/n = \Omega(1)$, but unfortunately the size of the lists $\mathcal{L}$ returned (and in particular the running time of the algorithm that returns that list) is at least quasipolynomial in ℓ [16, 28], and sometimes exponential in ℓ. Moreover, those constructions naturally have large alphabet sizes, polynomial in n. In order to reduce the alphabet size, constructions using algebraic

[6] In the errors model, we are given $\mathcal{L}_1, \ldots, \mathcal{L}_n \subseteq \Sigma$ with $|\mathcal{L}_i| \leq \ell$ for all i, and we require the list $\mathcal{L} = \left\{x \in \Sigma^k : \Pr_{i\in[n]}[\mathcal{C}_i(x) \in \mathcal{L}_i] \geq 1-\gamma\right\}$ to be small, for some error parameter γ.

[7] Note, however, that the rate of the code in [18] is only $k^{-\Omega(1)}$ for k being the message length.

geometry codes have been used (e.g. [20, 21, 15]), although these works still have parameters with an exponential dependence on ℓ. Moreover, all of the works mentioned above have polynomial – and not linear – time recovery algorithms. Using expander-based techniques (e.g. that of Alon et al. [2]), these algorithms can be improved to near-linear time in n (e.g., as in [22]), but at the cost of increasing the dependence on ℓ to doubly-exponential.

In addition to algebraic constructions, there have also been a few constructions of purely graph-based codes, which are more similar to our constructions. The work of Guruswami and Indyk [14] gives a linear-time algorithm for list-recovery of graph-based codes, which does even better in the setting of *mixture-recovery* (similar to the setting that we study here) where the input lists are generated from unions of codewords. That work achieves output list size $|\mathcal{L}|$ exactly equal to ℓ, but has rate $O(1/\ell)$ and the alphabet size is exponential in ℓ. The work of Hemenway and Wootters [23] gives an $O(n)$-time algorithm for list-recovering graph-based codes (the expander codes of [39, 42], with an appropriate inner code); these can have high rate (close to 1), but unfortunately the dependence on ℓ in other parameters is *quadruply*-exponential.

The work of Dinur, Harsha, Kaufman, Livni-Navon and Ta-Shma [9], which directly inspired our work, used *double-samplers* derived from high-dimensional expanders, combined with an expander-based symbol aggregation technique of ABNNR that we also use [1]. The goal of that work was to give an efficient list-decoding algorithm for *any* code that follows the ABNNR construction. This is much more general that what we are aiming to do (since we get to carefully design our code before applying the ABNNR construction), and also the goal is different (list-decoding in the worst case, rather than randomized list-recovery). That work is able to get efficient (polynomial-time) algorithms, but when one tries to turn their algorithm into a list-recovery algorithm in the most direct way, the parameters are not close to those in Goal 1; in particular, the algorithm is only $\text{poly}(n)$-time, and the dependence on ℓ is again exponential. It is not clear (to us) how to use the approach of [9] to achieve Goal 1.

We also mention a recent work of Dikstein, Dinur, and Harsha [8] that suggests an approach for constructing locally testable codes. In particular, as in our construction they also use the underlying graph (an *agreement expander* coming from a high-dimensional expander) both for symbol manipulation and for defining the parity checks. However, their goal is quite different than ours: they obtain locally testable codes via lifting a set of "smaller" locally testable codes, extending the natural Tanner tests.

Heavy Hitters

The first work with provable guarantees for the heavy hitters problem was by Misra and Gries [32], which applied to the *cash register* model where each of the updates $\Delta^{(i)}$ are equal to 1. We work in the more general strict turnstile model described above. For the strict turnstile model, the Count-Min Sketch data structure of [7] above already gets good results, and the best current results for the parameter regime we are motivated by (in particular, with failure probability $1/\text{poly}(N)$, and where $\log N \gg \text{poly}(1/\varepsilon)$) are those of [29] described above. It is known [25] that $\Omega(\varepsilon^{-1} \log N)$ words of memory are required for this setting, and thus the space used by these works are optimal.

We next mention three works that study heavy hitters when the failure probability is extremely small (or zero) [31, 5, 33]. Relative to our work, these works achieve – as with [29] – a better dependence on ε but worse dependence on N; however, these works can additionally get away with extremely small or even zero failure probability. In [31], Li et al. modify the Count-Min Sketch by looking at different hash functions, and they present a data

structure with failure probability δ with space $\widetilde{O}\left(\log(\varepsilon N)\left(\varepsilon^{-1}+\log(1/\delta)\right)\right)$, update time $\widetilde{O}(\log^2(1/\varepsilon)\log(\varepsilon N)\left(1+\varepsilon\log(1/\delta)\right)$, and query time $\widetilde{O}(\varepsilon^{-1}\log^2(1/\varepsilon)\log(\varepsilon N)\log(1/\delta))$. For $\delta = N^{-c}$ and $\log N \gg \text{poly}(1/\varepsilon)$, this gives the parameters stated in Table 1. However, when δ is much smaller – for example, $\delta = N^{-\Omega(1/\varepsilon)}$ – this gives better results than the works previously discussed, and in particular implies a result that is *uniform* over all sets of heavy hitters by union bounding over the $N^{O(1/\varepsilon)}$ choices for such sets. In [5], Cheraghchi and Nakos give a randomized construction of a data structure that also solves the heavy hitters problem uniformly over all streams $x^{(1)}, x^{(2)}, \ldots$ (that is, with error probability *zero* assuming that the data structure was constructed correctly). This scheme uses space $O(\varepsilon^{-1}\log(N\varepsilon))$, has update time $\tilde{O}(\log^2(1/\varepsilon)\log(\varepsilon N))$, and query time $\varepsilon^{-1}\,\text{polylog}(N)$.[8] That work actually provides solutions to several problems, not just heavy hitters, via a construction of *list-disjunct matrices*. Finally, we mention the work of Nelson et al. [33], which gives a fully deterministic construction of a data structure for heavy hitters (and more generally for ℓ_∞/ℓ_1 sparse recovery) with zero error probability; the space and query time is $O(\varepsilon^{-2}\,\text{poly}\log(N))$, and the update time is $O(\varepsilon^{-1}\,\text{poly}\log(n))$.[9]

We note that there are algorithms that achieve $O(\log N)$ update and query time for constant ε, but with only a constant failure probability. For example, such an algorithm is given in the full version of [29] (see [30, Theorem 10]).

One can generalize to the *general turnstile* model, where there is no guarantee that $f(x)$ is positive at each point in the stream, and one can generalize to ℓ_p-heavy hitters, where the goal is to return all x so that $|f(x)| \geq \varepsilon\|f\|_p$. There has been a great deal of work along both of these lines; see [29] and the references therein. In particular, for ℓ_p heavy hitters in the general turnstile model, the work [29] gives a data structure with space $O(\varepsilon^{-p}\log N)$, update time $O(\log N)$, and query complexity $\varepsilon^{-p}\,\text{polylog}(n)$.

We briefly discuss the approach of [29], in order to illustrate the differences between their approach and ours. While that work inspired the list-recovery approach we take, and they also use error correcting codes and expander graphs, the construction itself is quite different. That work takes the perspective of *graph clustering*. In more detail, their sketch can output a graph in which each heavy hitter is represented by a well-connected cluster in the graph. They then develop a clustering algorithm that can recover the clusters, and hence the heavy hitters. In order to make the connection to graph clustering, they first encode x with an error correcting code $\mathcal{C}_0$ as we do; but they only need this code to have good distance, as they do not go down the list-recovery route. Then they break $\mathcal{C}_0(x)$ up into n' chunks. Before putting the j-th of these chunks into the j-th smaller data structure, they append it with tags $h_j(x)$ and $\{h_{\Gamma(j)_i}(x)\}$, where the h_j are hash functions and Γ is the adjacency function for an expander graph G. Thus, the j-th chunk of $\mathcal{C}_0(x)$ is essentially connected by edges in G to the other chunks of $\mathcal{C}_0(x)$, and in particular the chunks of $\mathcal{C}_0(x)$ form a cluster that can be recovered by a clustering algorithm.

We note that [4] was also inspired by [29], and builds on their approach to develop differentially private heavy-hitters algorithms. In fact, that work even casts the scheme of [29] as a list-recovery scheme, in a relaxed definition of list-recovery that is different

[8] We note that here the guarantee is to return a list $\mathcal{L}$ of size $O(1/\varepsilon)$ containing all the true heavy hitters, although in both [31] and [5], the list is allowed to contain elements with frequencies $f(x) \ll \varepsilon\|f\|_1$, while most of the heavy-hitters work surveyed above, including ours, does not have such false positives.

[9] We note that in [33], the query time to recover the list of the heavy hitters is $\Omega(N)$ and the space involves a single factor of $\log(N)$, but the "dyadic trick" can be used to obtain the bounds mentioned above.

from our relaxed version in Theorem 2. In particular, their notion of list-recovery will not handle input lists $\mathcal{L}_1, \ldots, \mathcal{L}_n$ that are generated by any ℓ distinct messages, as we handle in Theorem 2.[10]

Algorithmic applications of list-recovery

Our work is inspired by the use of list-recovery in [29], but there is a rich history of using list-recoverable codes in similar algorithmic applications. One example is *group testing*, where the goal is to identify d "positive" items out of a universe of size N, given tests of the form $\bigvee_{i \in I} \mathbf{1}[i \text{ is positive}]$ for subsets $I \subset [N]$. A classic construction of Kautz and Singleton [26] reduces this question to the question of list-recovery. This connection, and elaborations on it, has been exploited in several works, which aim to both minimize the number of tests and to develop sublinear-time algorithms to recover the set of positive items [24, 35].

A second example, even closer to our work, is in *compressed sensing*. In compressed sensing, the goal is to approximately recover an approximately sparse vector $v \in \mathbb{R}^N$ given linear measurements Av for some $A \in \mathbb{R}^{t \times N}$. The heavy-hitters problem is closely related, as a (linear) solution to the heavy hitters problem can approximately recover the support of v. List-recoverable codes have been used in the context of compressed sensing in a similar way as it was used in [29]: associate each $i \in [N]$ with a message, and encode it with a list-recoverable code to get a codeword $\mathcal{C}(i) = (c_1, \ldots, c_n) \in \Sigma^n$. Then reduce the compressed sensing problem to n smaller instances of the same problem for vectors of length $|\Sigma|$: for each $j \in [n]$, we have a vector $w^{(j)}$ indexed by Σ so that the entry $w^{(j)}_\sigma$ is obtained by aggregating all of the coordinates v_i of v so that $\mathcal{C}(i)_j = \sigma$. Now we can either recurse or solve these smaller problems in another way. Previous works [36, 13, 12] have observed that a good list-recoverable code (e.g., satisfying Goal 1) would solve this problem. However, they ran into the same issue that we did, namely that we do not know of any such codes. Instead, they either used sub-optimal codes or developed work-arounds, as we describe below.

The work of Ngo et al. [36] was, to the best of our knowledge, the first to apply list-recovery in compressed sensing. We mention two results in that work that use a framework quite similar to that of [29] (and thus to ours), making explicit use of (sub-optimal) list-recoverable codes. The first result is based on the list-recoverability of Reed-Solomon codes. As RS codes do not achieve Goal 1, this results in a sub-optimal number of measurements, but is nice and simple. The second is based on Parvaresh-Vardy (PV) codes. PV codes have good rate and output list-size, but unfortunately the alphabet size is very large. To get around this, [36] (inspired by the work [35] on group testing mentioned above) considered a code constructed by repeatedly concatenating PV codes with themselves. This does not lead to a code that achieves Goal 1 – the rate depends on ℓ, and either the alphabet size or the rate must depend on n – but they are able to make these dependencies not too bad. This leads to schemes with near-optimal number of measurements t, although the schemes only work for non-negative signals. Further, since PV codes do not have near-linear-time algorithms, the recovery algorithm runs in time $\text{poly}(t)$ rather than near-linear in t.

[10] In a bit more detail, the notion of list-recovery in [4] allows for $\mathcal{L}_1, \ldots, \mathcal{L}_n$ to be generated by ℓ messages $x^{(1)}, \ldots, x^{(\ell)}$, *provided* that the messages lie in distinct "buckets," according to any fixed bucketing of the message space. The choice of the code may depend on the bucketing. In this language, the result of [29] (see [4, Theorem 3.6]) says that it is possible to obtain a code with constant rate, output list size $L = O(\ell)$, and alphabet size that is polynomial in the number of buckets, and a polynomial-time decoding algorithm. If the number of buckets is $\text{poly}(\ell)$, the alphabet size is also $\text{poly}(\ell)$, as we would hope, but as the number of buckets grows (to approach the general case with $|\Sigma|^k$ buckets of size one, where there is no "bucketing" restriction) the alphabet size grows accordingly.

The work of Gilbert et al. [13] follows a similar outline, using the Loomis-Whitney-based codes of [34]. For $d > 0$ some integer parameter, these are codes $\mathcal{C}\colon [N] \to [N^{1/d}]^{d-1}$ are $(\ell, \ell^{d/(d-1)})$-list-recoverable in time $O(\ell^{d/(d-1)} \log N)$. In terms of the desiderata of Goal 1, this does give near-linear-time recovery with good dependence on ℓ; however the alphabet size is huge, growing exponentially in the message length. In [13], they deal with this by applying the scheme mentioned above recursively until the alphabet size becomes manageable. As a result, they are able to get a nearly optimal number of measurements, with a recovery time that depends polynomially (but not linearly) on $\log N$, and with an extremely small error probability, smaller than $1/\operatorname{poly}(N)$.

We also mention [12], which uses list-recoverable codes (PV codes) in a more complicated way to achieve a near-optimal compressed sensing algorithm in the uniform ("forall") model. They also treat the indices i as messages and encode them with a list-recoverable code, but they develop more machinery – using an expander to add linking information between the symbols for example – in order to reduce to the list-recovery problem.

1.4 Open Questions and Future Work

In this work we have made progress towards Goal 1 by constructing a *randomized* code that supports, with high probability, linear-time list-recovery from certain lists. This was enough for our application to heavy hitters, but many open questions still remain.

1. The most obvious open question is to fully attain Goal 1. In addition to furthering our knowledge in algorithmic coding theory, it seems likely that attaining Goal 1 (or the techniques used to do it), would have other applications in algorithm design, as well as in pseudorandomness (as per Section 1.2).
2. While we are able to use techniques from high-dimensional expansion to obtain a failure probability of $N^{-\Omega(\varepsilon^3)}$ (in the setting of ε-heavy hitters), we would like a failure probability of $N^{-\Omega(1)}$.
3. In this paper we studied only *zero-error* list-recovery (or, more accurately, list-recovery from a small fraction of erasures). While this question is interesting and challenging on its own, one can ask about extending our results to list-recovery from errors. In particular, this might lead to improved heavy-hitters schemes in the general turnstile model.
4. We motivated our "probabilistic list-recovery" model by an application to heavy hitters. However, we hope that there are many other algorithmic applications for such a model and for our construction. Indeed, there are several algorithmic applications of list-recovery mentioned in Section 1.3 (e.g., [24, 35, 36, 13]) that explicitly use list-recoverable codes and would be improved by codes that achieve Goal 1. It is our hope that some of these applications could also be improved by better constructions of the probabilistically list-recoverable codes that we study here. As one example, if one could obtain Theorem 2 with $|\Sigma| = \widetilde{O}(\ell)$ (rather than polynomial in ℓ), then by the construction of Kautz and Singleton mentioned above [26] this would yield optimal constructions of probabilistic group testing matrices with sublinear-time decoding, matching a recent result of [37] in a black-box way.

2 Randomized List Recovery

Inspecting our main theorem's proof (in particular, Section 5 in the full version), we can extract a list recovery result for our (randomized) code $\mathcal{C}$ that tolerates a small fraction of erasures. Our randomized encoding can handle input lists that come from a union of

codewords $\{\boldsymbol{\mathcal{C}}(x) : x \in \mathcal{L}_0\}$ for some $\mathcal{L}_0 \subseteq \mathbb{F}_q^k$; this is what we stated in Theorem 2. Moreover, our algorithm can also handle some extra "distractor symbols," provided that those symbols are randomized and unlikely to collide with the symbols that come from $\mathcal{L}_0$. In order to state this formally, we first give a definition that captures the sort of input lists that our algorithm can handle.

▶ **Definition 4.** *Let $\boldsymbol{\mathcal{C}} \colon \mathbb{F}_q^k \to \Sigma^{n'}$ be a randomized encoding, for $\Sigma = \mathbb{F}_q^b$. Consider a randomized function of $\mathcal{C} \sim \boldsymbol{\mathcal{C}}$ and a set of messages $\mathcal{L}_0 \subseteq \mathbb{F}_q^k$, that outputs lists $\mathcal{L}_1, \ldots, \mathcal{L}_{n'} \subseteq \Sigma$. We say that such a function is (t, η)-*nice *w.r.t. $\boldsymbol{\mathcal{C}}$ if the following holds for all $\mathcal{L}_0 \subseteq \mathbb{F}_q^k$ (note that the lists $\mathcal{L}_i$ can depend on $\mathcal{L}_0$):*

1. *For any $i \in [n']$,*
   - *with probability at least $1 - \eta$, $|\mathcal{L}_i| = O(|\mathcal{L}_0|)$, and,*
   - *with probability 1, $\boldsymbol{\mathcal{C}}(x)_i \in \mathcal{L}_i$ for all $x \in \mathcal{L}_0$.*
2. *For any $x \in \mathcal{L}_0$ and $i \in [n']$, with probability at least $1 - \eta$ it holds that $(\boldsymbol{\mathcal{C}}(x)_i)_j \neq \sigma_j$ for every $j \in [b]$ and $\sigma \in \mathcal{L}_i \setminus \{\boldsymbol{\mathcal{C}}(y)_i : y \in \mathcal{L}_0\}$.*

Furthermore, we require that the above properties should hold t-wise independently across the lists. Namely, for any $x \in \mathcal{L}_0$, whether (1) and (2) hold for some $i \in [n']$ is independent of whether it holds for any $t - 1$ other values of $i' \in [n']$.

To illustrate this definition, we give a few examples.

▶ **Example 5** (lists from a union of codewords)**.** The simplest example of a nice distribution is the function that gives

$$\mathcal{L}_i = \{\boldsymbol{\mathcal{C}}(x)_i : x \in \mathcal{L}_0\}.$$

That is, the lists $\mathcal{L}_i$ are just given by the union of the codewords in $\mathcal{L}_0$. To see that this is $(\eta = 0, t = n')$-nice, observe that both (1) and (2) hold deterministically, with probability 1. Indeed, (1) holds by construction, and (2) holds because there are no $\sigma \in \mathcal{L}_i \setminus \{\boldsymbol{\mathcal{C}}(y)_i : y \in \mathcal{L}_0\}$, so the condition is trivial.

▶ **Example 6** (lists with random distractor symbols)**.** Another natural example of a nice distribution is the example above, with some uniformly random extra "distractor" symbols. That is,

$$\mathcal{L}_i = \{\boldsymbol{\mathcal{C}}(x)_i : x \in \mathcal{L}_0\} \cup \{\sigma_{i,h} : h \in [r]\}$$

where $r > 0$ is some parameter and where $\sigma_{i,h}$ are i.i.d. and uniform in Σ. Again, this satisfies item (1) deterministically, provided that $r = O(|\mathcal{L}_0|)$. For (2), we can compute the probability of a collision between the distractor symbols $\{\sigma_{i,j} : j \in [r]\}$ and a given codeword $\boldsymbol{\mathcal{C}}(x)$ for $x \in \mathcal{L}_0$:

$$\begin{aligned}\Pr\left[(\boldsymbol{\mathcal{C}}(x)_i)_j \neq \sigma_j \ \forall j \in [b], \sigma \in \{\sigma_{i,h} : h \in [r]\}\right] &= \left(1 - \frac{1}{q}\right)^{br} \\ &\leq \exp(-br/q).\end{aligned}$$

In particular, when $q \gg br$, this is $1 - O\left(\frac{br}{q}\right)$. Thus, this distribution is (η, t)-nice where $\eta = O(br/q)$ and $t = n$.

Finally, we note that the distribution of distractor symbols that arises in our heavy hitters application is also nice for the code $\mathcal{C}$ that we use. The first point of Item (1) holds because the lists $\mathcal{L}_i$ can only become too large if the inner InnerHH fails and includes items that are not $\varepsilon/4$-heavy hitters. The second point of Item (1) holds because our instantiation of InnerHH has only one-sided error. Item (2) holds even for any $\sigma \in \mathcal{L}_i \setminus \{\mathcal{C}(x)_i\}$, which follows from Lemma 5.4 in the full version.

With this definition in place, we can now state our main theorem for list-recovery. Theorem 7 generalizes Theorem 2, because it allows for input lists with some extra "distractor" symbols, as per Definition 4.

▶ **Theorem 7.** *There exist constants $c > 1$ and $\gamma \in (0, 1)$ such that the following holds for any positive integers k and $\ell \leq k^\gamma$. There exists a randomized encoding $\mathcal{C} \colon \mathbb{F}_q^k \to \Sigma^{n'}$, for $q = \mathrm{poly}(\ell)$, $\Sigma = \mathbb{F}_q^{O(1)}$ and $n' = \Theta(k)$, and a randomized list recovery algorithm $\mathcal{A}$ running in time $\ell^c \cdot k$, with the following guarantee.*

For some constant $\eta < 1$, and an integer $t = \frac{k}{\mathrm{poly}(\ell)}$, for any list of messages $\mathcal{L}_0 \subseteq \mathbb{F}_q^k$ of size ℓ, and any distribution over input lists $\mathcal{L}_1, \ldots, \mathcal{L}_{n'}$ to $\mathcal{A}$ which are randomized functions of $\mathcal{C}$ and $\mathcal{L}_0$ and are (t, η)-nice w.r.t. $\mathcal{C}$, the list recovery algorithm $\mathcal{A}$, with probability $1 - \ell^{-\Omega(k)}$ (over the randomness of the encoding and the lists), outputs $\mathcal{L} \subseteq \mathbb{F}_q^k$ of size $O(\ell)$ such that $\mathcal{L}_0 \subseteq \mathcal{L}$. Furthermore, the encoding time of $\mathcal{C}$ is $O(k \log \ell)$, with a preprocessing step which takes $\mathrm{poly}(k)$ time.

We stress that unlike in standard state-of-the-art efficient list recovery algorithms, here we have a good dependence on ℓ, namely $q = \mathrm{poly}(\ell)$ and $|\mathcal{L}| = O(\ell)$.

We hope that Theorem 7 will find more applications. As discussed in Section 1.3, there are many algorithmic applications of list-recovery in the literature, and several previous applications of list-recovery have ended up with sub-optimal parameters due to the unavailability of codes that achieve Goal 1. It seems possible that Theorem 7 (or a further improvement on our techniques) could lead to improved results in (non-uniform or "for-each") group testing or compressed sensing.

References

1. Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on information theory*, 38(2):509–516, 1992.
2. Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *36th Annual Symposium on Foundations of Computer Science (FOCS 1995)*, pages 512–519. IEEE, 1995.
3. Avraham Ben-Aroya, Dean Doron, and Amnon Ta-Shma. Near-optimal erasure list-decodable codes. In *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
4. Mark Bun, Jelani Nelson, and Uri Stemmer. Heavy hitters and the structure of local privacy. *ACM Transactions on Algorithms (TALG)*, 15(4):1–40, 2019.
5. Mahdi Cheraghchi and Vasileios Nakos. Combinatorial group testing and sparse recovery schemes with near-optimal decoding time. In *61st Annual Symposium on Foundations of Computer Science (FOCS 2020)*. IEEE, 2020. To appear.
6. Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, 2010.
7. Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

8 Yotam Dikstein, Irit Dinur, Prahladh Harsha, and Noga Ron-Zewi. Locally testable codes via high-dimensional expanders. *arXiv preprint*, 2020. `arXiv:2005.01045`.

9 Irit Dinur, Prahladh Harsha, Tali Kaufman, Inbal Livni Navon, and Amnon Ta-Shma. List decoding with double samplers. In *ACM-SIAM 38th Annual Symposium on Discrete Algorithms (SODA 2019)*, pages 2134–2153. SIAM, 2019.

10 Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. In *61st Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1057–1068. IEEE, 2020.

11 Dean Doron and Mary Wootters. High-probability list-recovery, and applications to heavy hitters. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2021. Manuscript.

12 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. For-all sparse recovery in near-optimal time. *ACM Transactions on Algorithms (TALG)*, 13(3):1–26, 2017.

13 Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. ℓ_2/ℓ_2-foreach sparse recovery with low risk. In *International Colloquium on Automata, Languages, and Programming (ICALP 2013)*, pages 461–472. Springer, 2013.

14 Venkatesan Guruswami and Piotr Indyk. Linear-time list decoding in error-free settings. In *International Colloquium on Automata, Languages, and Programming (ICALP 2004)*, pages 695–707. Springer, 2004.

15 Venkatesan Guruswami and Swastik Kopparty. Explicit subspace designs. *Combinatorica*, 36(2):161–185, 2016.

16 Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

17 Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. In *39th Annual Symposium on Foundations of Computer Science (FOCS 1998)*, pages 28–37. IEEE, 1998.

18 Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. *Journal of the ACM (JACM)*, 56(4):1–34, 2009.

19 Venkatesan Guruswami and Carol Wang. Linear-algebraic list decoding for variants of reed–solomon codes. *IEEE Transactions on Information Theory*, 59(6):3257–3268, 2013.

20 Venkatesan Guruswami and Chaoping Xing. Folded codes from function field towers and improved optimal rate list decoding. In *44th Annual Symposium on Theory of Computing (STOC 2012)*, pages 339–350. ACM, 2012.

21 Venkatesan Guruswami and Chaoping Xing. List decoding Reed-Solomon, algebraic-geometric, and Gabidulin subcodes up to the Singleton bound. In *45th Annual Symposium on Theory of Computing (STOC 2012)*, pages 843–852. ACM, 2013.

22 Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes and applications. *SIAM Journal on Computing*, pages FOCS17–157, 2019.

23 Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. *Information and Computation*, 261:202–218, 2018.

24 Piotr Indyk, Hung Q. Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *ACM-SIAM 21st Annual Symposium on Discrete Algorithms (SODA 2010)*, pages 1126–1142. SIAM, 2010.

25 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for ℓ_p samplers, finding duplicates in streams, and related problems. In *ACM SIGMOD-SIGACT-SIGART 30th Annual Symposium on Principles of Database Systems*, pages 49–58, 2011.

26 William Kautz and Roy Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10(4):363–377, 1964.

27 Swastik Kopparty. List-decoding multiplicity codes. *Theory of Computing*, 11(1):149–182, 2015.

28 Swastik Kopparty, Noga Ron-Zewi, Shubhangi Saraf, and Mary Wootters. Improved decoding of folded Reed-Solomon and multiplicity codes. In *59th Annual Symposium on Foundations of Computer Science (FOCS 2018)*, pages 212–223. IEEE, 2018.

29 Kasper Green Larsen, Jelani Nelson, Huy L. Nguyễn, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In *57th Annual Symposium on Foundations of Computer Science (FOCS 2016)*, pages 61–70. IEEE, 2016.

30 Kasper Green Larsen, Jelani Nelson, Huy L. Nguyễn, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. arxiv:1604.01357 [cs.DS], 2016.

31 Yi Li, Vasileios Nakos, and David P. Woodruff. On low-risk heavy hitters and sparse recovery schemes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116, pages 19:1–19:13. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2018.

32 Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

33 Jelani Nelson, Huy L. Nguyễn, and David P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. *Linear Algebra and its Applications*, 441:152–167, 2014.

34 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):1–40, 2018.

35 Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable error-correcting list disjunct matrices and applications. In *International Colloquium on Automata, Languages, and Programming (ICALP 2011)*, pages 557–568. Springer, 2011.

36 Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable compressed sensing by list-recoverable codes and recursion. In *29th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14, pages 230–241. LIPIcs, 2012.

37 Eric Price and Jonathan Scarlett. A fast binary splitting approach to non-adaptive group testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

38 Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In *31st Annual Symposium on Theory of Computing (STOC 1999)*, pages 159–168, 1999.

39 Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

40 Amnon Ta-Shma and David Zuckerman. Extractor codes. *IEEE Transactions on Information Theory*, 50(12):3015–3025, 2004.

41 Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM (JACM)*, 48(4):860–879, 2001.

42 Gillés Zémor. On expander codes. *IEEE Transactions on Information Theory*, 47(2):835–837, 2001.

Almost Optimal Bounds for Sublinear-Time Sampling of k-Cliques in Bounded Arboricity Graphs

Talya Eden
Boston University, MA, USA
MIT, Cambridge, MA, USA

Dana Ron
Tel Aviv University, Israel

Will Rosenbaum
Amherst College, MA, USA

Abstract

Counting and sampling small subgraphs are fundamental algorithmic tasks. Motivated by the need to handle massive datasets efficiently, recent theoretical work has examined the problems in the sublinear time regime. In this work, we consider the problem of sampling a k-clique in a graph from an almost uniform distribution. Specifically the algorithm should output each k-clique with probability $(1 \pm \epsilon)/n_k$, where n_k denotes the number of k-cliques in the graph and ϵ is a given approximation parameter. To this end, the algorithm may perform degree, neighbor, and pair queries. We focus on the class of graphs with arboricity at most α, and prove that the query complexity of the problem is

$$\Theta^*\left(\min\left\{n\alpha, \max\left\{\left(\frac{(n\alpha)^{k/2}}{n_k}\right)^{\frac{1}{k-1}}, \frac{n\alpha^{k-1}}{n_k}\right\}\right\}\right),$$

where n is the number of vertices in the graph, and $\Theta^*(\cdot)$ suppresses dependencies on $(\log n/\epsilon)^{O(k)}$.

Our upper bound is based on defining a special auxiliary graph H_k, such that sampling edges almost uniformly in H_k translates to sampling k-cliques almost uniformly in the original graph G. We then build on a known edge-sampling algorithm (Eden, Ron and Rosenbaum, ICALP19) to sample edges in H_k. The challenge is simulating queries to H_k while being given query access only to G. Our lower bound follows from a construction of a family of graphs with arboricity α such that in each graph there are n_k k-cliques, where one of these cliques is "hidden" and hence hard to sample.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases sublinear time algorithms, graph algorithms, cliques, arboricity, uniform sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.56

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2012.04090 [24]

Funding *Talya Eden*: This research was partially supported by Ben Gurion Postdoctoral Scholarship.
Dana Ron: This research was partially supported by the Israel Science Foundation (grant No. 1041/18).

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 56; pp. 56:1–56:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We consider the problem of sampling k-cliques in sublinear-time. Sampling subgraphs is a fundamental computational task in randomized algorithms, statistics, data science, and many other disciplines. Sampling k-cliques, and triangles in particular, has numerous applications across various fields, see, e.g. [38, 7, 19, 47, 16] and references therein. The best exact combinatorial algorithm for this task is an $O((n/\log n)^k)$-time algorithm by Vassilevska [46], and Chen et al. [14] proved that, under the exponential time hypothesis, there is no $n^{o(k)}$-time algorithm.

Motivated by the need to handle massive datasets efficiently, we consider algorithms that are given query access to the graph, in the form of degree, neighbor and pair queries.[1] We refer to this model as the *general query model*. Our goal is to design an algorithm that samples k-cliques while performing as few queries as possible.

Fichtenberger, Gao, and Peng [32] recently studied the problem of sampling *arbitrary* subgraphs. They assumed access to the above queries, as well as access to uniform edge samples. Thus, they considered a strictly stronger model. Specifically, for sampling k-cliques uniformly at random, their algorithm has expected complexity[2] $\tilde{O}(m^{k/2}/n_k)$, where m and n_k denote the number of edges and k-cliques in the graph, respectively. Their result is known to be essentially tight, due to a lower bound by Eden and Rosenbaum [28].

However, the lower bound of [28] only holds when considering the worst-case over all possible inputs. In this work we ask whether the lower bound can be circumvented when considering graphs with bounded arboricity. The arboricity of a graph G, denoted $\alpha(G)$, is the minimal number of forests required to cover its edge set. Up to a factor of 2, it is equivalent to the average degree of the densest subgraph in G. Hence, arboricity is a natural and useful measure of density "everywhere". Graphs with bounded arboricity constitute an important and rich family of graphs, including planar graphs, minor-closed graphs, graphs with bounded treewidth, and preferential attachment graphs. On the applied side, in most real-world graphs the arboricity is at most an order of magnitude larger than the average degree, while the maximum degree could be up to three or four orders of magnitude larger [34, 30, 43]. Many applied algorithms exploit the property of bounded arboricity in order to design faster algorithms for clique and dense subgraph counting and listing [30, 33, 41, 37, 17, 9]. Furthermore, in a recent work, Eden, Mossel and Ron presented an algorithm for approximating the arboricity in sublinear time [21], whose output can be used as input to our algorithm (as an upper bound on the arboricity of the graph).

We seek algorithms that, given a parameter α and query access to a graph whose arboricity is upper bounded by α, "beat" the aforementioned lower bound when α is sufficiently bounded away from $\sqrt{m}$ (recall that the arboricity of a graph is always at most $\sqrt{m}$).

Such an algorithm was recently designed for the special case of sampling edges (2-cliques) almost uniformly. Here and elsewhere, when we say "almost uniformly" we mean in the strong sense of *pointwise-close* to uniform. Namely, where *each* element is returned with almost equal probability. We further discuss the benefits of this notion as compared to the strictly weaker notion of proximity with respect to the total variation distance in Section 1.1.2.

---

[1] Degree queries return the degree, $d(v)$, of a given vertex v; neighbor queries return the i^{th} neighbor of v for any given vertex v and $1 \leq i \leq d(v)$; and pair queries return whether there is an edge between a given pair of vertices.

[2] Throughout the introduction, when we discuss the "complexity" of previous results, we mean the running time of the algorithm. The query complexity is always bounded by the minimum between the running time and $n + m \leq n\alpha$.

Specifically, it is shown in [23] that the complexity of the almost-uniform edge sampling problem is $\Theta^*(n\alpha/m)$.[3] Comparing this to the $\Theta^*(n/\sqrt{m})$ complexity of the problem in general graphs [29, 45], exhibits an improvement by (roughly) a factor of $\sqrt{m}/\alpha$. In particular, for graphs with constant arboricity, this implies an exponential improvement, from $O^*(\sqrt{n})$ to $O^*(1)$. Similar improvements were obtained for the related question of approximately counting the number of k-cliques in the graph, where Eden, Ron and Seshadhri [26] obtained significant improvements for the class of bounded arboricity graphs, compared to the (essentially optimal) result for the general case [27].

In this work we show that, indeed, the complexity of the task of sampling k-cliques for *any* constant $k \geq 3$, is significantly better for the class of graphs of bounded arboricity, as compared to general graphs.

▶ **Theorem 1.1.** *Let $\varepsilon \in (0,1)$ be a constant. There exists an almost uniform sampling algorithm for k-cliques in graphs with arboricity at most α, that returns each k-clique in the graph with probability $\frac{(1\pm\varepsilon)}{n_k}$. Given a constant factor estimate of n_k,[4] the query complexity of the algorithm is*

$$O^*\left(\min\left\{n\alpha, \max\left\{\left(\frac{(n\alpha)^{k/2}}{n_k}\right)^{\frac{1}{k-1}}, \frac{n\alpha^{k-1}}{n_k}\right\}\right\}\right).$$

The running time is the same as the second term of the minimum.

While our upper bound on the complexity might seem unnatural at first glance, we also prove an almost-matching lower bound, thus resolving the complexity of the problem up to $(\log n/\varepsilon)^{O(k)}$ factors.

▶ **Theorem 1.2.** *Let $\mathcal{A}$ be an algorithm that given query access to a graph with arboricity at most α, returns each k-clique with probability $\Theta\left(\frac{1}{n_k}\right)$. Then the query complexity of $\mathcal{A}$ is*

$$\Omega^*\left(\min\left\{n\alpha, \max\left\{\left(\frac{(n\alpha)^{k/2}}{n_k}\right)^{\frac{1}{k-1}}, \frac{n\alpha^{k-1}}{n_k}\right\}\right\}\right).$$

We note that we chose not to parameterize our bounds in terms of m, but rather, only in terms of n, α, k and n_k. Hence, both the upper bound and the lower bound are stated for worst case m, which is $n\alpha$. We note that it is possible to obtain a finer expression that does depend on m for both bounds. However, for the sake of exposition, we chose not to include an additional parameter.

1.1 Discussion of the results

1.1.1 Comparison to previous results

For simplicity, assume that ε and k are constants, and ignore lower order terms. To compare the complexity of our algorithm to the upper bound of [32], consider, for example, the family of graphs G with n vertices, $m = n^{3/2}$ edges and arboricity $\alpha = n^{1/2}$, and assume that

[3] We use O^*, Ω^* and Θ^* to suppress a dependence on functions $g(\log n, k, 1/\varepsilon)$, which are at most $(\log n/\varepsilon)^{O(k)}$, where ε is the given approximation parameter.

[4] If the algorithm is not provided with an estimate of n_k, then an estimate of n_k can be obtained by applying the algorithm of [26] whose expected query complexity is dominated by the runtime of Theorem 1.1.

$k = 3$ and $n_3 = n^2$. Then the complexity of the problem in the general case grows like[5] $\tilde{\Theta}(m^{3/2}/n_3) = \tilde{\Theta}(n^{1/4})$, while we get $\Theta^*(n^{1/8})$. That is, we obtain a quadratic improvement, despite the fact that we work in a strictly weaker query model.

If we also allow access to uniform edge queries, then our algorithm can be adapted to run in time $O^*\left(\max\left\{\left(\frac{m^{k/2}}{n_k}\right)^{\frac{1}{k-1}}, \frac{m\alpha^{k-2}}{n_k}\right\}\right)$. To compare this to the $O(m^{k/2}/n_k)$ upper bound of [32] (for k-cliques), observe that if the first term in our bound is the dominant one, then we get the bound of [32] taken to the power of $1/(k-1)$. If the second term is the dominant one, then we improve on the bound of [32] by a factor of $(\sqrt{m}/\alpha)^{k-2}$ (recall that for every graph, $\alpha \leq \sqrt{m}$).

1.1.2 The importance of point-wise uniform sampling

In our results we measure "almost uniformity" with respect to pointwise distance between distributions. This notion of approximately uniform is a very strong one, as it requires every element to be returned with almost equal probability. In contrast, one could also consider the strictly weaker requirement that the distribution is close to uniform with respect to total variation distance (TVD). Here, it might be the case that the distributions assigns zero probability to an ε-fraction of the domain elements.

Sampling almost uniformly with respect to TVD may be sufficient in some contexts. However, there are scenarios in which the stronger notion of pointwise almost uniform sampling is crucial. Consider a domain in which each element has some significance score attributed to it, and assume that a small fraction of the domain elements have non-zero score and the others have score zero. If we have access to a distribution whose TVD distance to uniform is larger than the fraction of elements with positive score, then it is useless if we want to get any information regarding the (non-zero) significance scores of the elements. This is in contrast to having access to a distribution that is point-wise close to uniform, even for constant point-wise distance, where each element is returned with probability $\Theta(1/N)$, where N is the size of the domain. In the latter case, the probability of hitting non-zero score elements is slightly reduced as compared to uniform sampling, but does not fall to zero as in the former case.

For a more concrete example, consider protein networks, where cliques correspond to folding sites [18, 13, 1]. One can use point-wise sampling in order to get access to the folding sites of the protein at question, and then continue to further study their surrounding neighborhood. In TVD sampling however, it might be the case that exactly the folding sites of interest are the ones "missed" by the TVD sampler. Furthermore, as biological networks tend to huge and sparse [3, 39], allowing for improved results in bounded arboricity graphs is of major interest.

1.1.3 Approximate counting vs. point-wise sampling

We first observe that the complexities of approximately counting edges and point-wise almost uniformly sampling edges (in both bounded arboricity and general graphs) are of the same order. In contrast, for $k \geq 3$, the two complexities might differ significantly in bounded arboricity graphs. For example, consider the case of triangles ($k = 3$), $\alpha = O(1)$, and

[5] We note that the $\Omega(m^{k/2}/n_k)$ lower bound of [28] also holds for the task of sampling k-cliques almost uniformly.

$n_3 = \Theta(n)$. The complexity of sampling triangles almost uniformly is $\Theta^*(n^{1/4})$, while the complexity of approximately counting problem, is $O^*(1)$. This implies an exponential gap between the complexities of counting and sampling for certain ranges of parameters.[6] [7]

1.2 The high level ideas behind the clique-sampling algorithm

We start by briefly describing the ideas behind the algorithm of [23] for sampling edges almost uniformly, which we employ both as a subroutine and as a starting point of our algorithm for sampling k-cliques. We then turn to describe our algorithm. Throughout, we assume that an upper bound α on the arboricity of the input graph is known.

1.2.1 The edge sampling algorithm

Let L_0 be the set of all vertices in the graph with degree at most (roughly) α (so that almost all the vertices in the graph belong to L_0). The edge sampling algorithm samples a vertex v_0 uniformly at random, and if v_0 is in L_0, it performs a short random walk $v_0, v_1, \ldots, v_j$ of length j, for an index j chosen uniformly in $[\log n]$. If at any point the walk returns to L_0, then the algorithm aborts, and otherwise, it returns the last edge traversed.

The analysis of the algorithm relies on a layered decomposition of the graph vertices. The vertices in L_0 comprise the first layer. Subsequent layers are defined inductively: a vertex v is in L_j if (1) it is not in any of the layers L_i for $i < j$, and (2) most of its neighbors are in layers $L_0, L_2, \ldots, L_{j-1}$. While the algorithm is completely oblivious to the levels of the encountered vertices v_i for $i > 0$, using the aforementioned layering, it can be shown that each edge is sampled with almost equal probability $\approx \frac{1}{n\alpha}$.

1.2.2 The auxiliary graph H_k and the clique-sampling algorithm

In order to sample k-cliques in G, we first define an auxiliary graph H_k, whose edges correspond to k-cliques of G. Specifically, for each $(k-1)$-clique Q in G, there is a node v_Q in H_k, and for each k-clique C in G, there is a single edge in H_k between a pair of nodes $v_Q, v_{Q'}$ corresponding to two of its $(k-1)$-cliques, Q and Q'. Specifically, the first two $(k-1)$-cliques according to an ordering on all $(k-1)$-cliques, which will be defined later on. We say that C is *assigned* to Q and Q'. When $k = 2$, this assignment is uniquely determined (since every 2-clique (edge) contains exactly two 1-cliques (vertices)), and we have $H_2 = G$. For larger values of k, the assignment rule is such that Q and Q' both contain the vertex in C that has minimum degree in G. In general, since there is a one-to-one correspondence between the edges in H_k and the k-cliques of G, sampling an almost uniform edge in H_k is equivalent to sampling an almost uniform k-clique in G. An important observation is that if G has arboricity at most α, then so does H_k.

Given the aforementioned relation between k-cliques in G and edges in H_k, the basic underlying idea of our algorithm is emulating the edge sampling algorithm of [23] on the graph H_k, *while only having query access to the graph* G. Indeed this approach is natural (having defined H_k). However, emulating the edge sampling algorithm by performing random walks on H_k requires us to overcome several challenges:

[6] We note that the separation between approximately counting triangles and sampling triangles almost uniformly was already mentioned in [23] in passing as a preliminary result. However, [23] did not include any proof details nor a full characterization of the complexity of the sampling problem (for any $k \geq 3$).

[7] We believe that the algorithm of [26] can be adapted to sample a k-clique almost uniformly with respect to TVD with essentially the same complexity. However, this is not immediate.

1. We do not have query access to uniformly random nodes of H_k;
2. Determining whether a node in H_k is in layer L_0 cannot be performed by a single degree query (as was the case in [23]);
3. In order to sample a random neighbor of a node v_Q in H_k, we must sample a k-clique in G that is assigned to Q. (In [23] this could be implemented by a single neighbor query.) The emulation is "noisy" in the sense that it obtains only approximate answers to queries on H_k. In particular, it only estimates the degrees of nodes in H_k and selects nodes according to a distribution that is close to uniform. This is in contrast to the [23] algorithm, which gets precise answers to its queries. Hence, we must prove that the new algorithm still returns an edge (in H_k) that is close to uniform.
4. Emulating each query on H_k is implemented by performing multiple queries on G. Hence, one of the main challenges of this work is in bounding the query complexity of the clique-sampling algorithm.

We now outline how we address these challenges.

Addressing the challenges

Challenge 1: Emulating uniform node queries. The algorithm of [23] starts by sampling vertices uniformly at random in G. As stated previously, we do not have direct access to uniform node samples in H_k. Instead, in order to sample nodes in H_k, we recursively invoke our algorithm for sampling $(k-1)$-cliques in G almost uniformly. This results in a distribution that is only close to uniform, but we prove that this is sufficient for our needs.

Challenge 2: Determining whether a node belongs to $\mathbf{L_0(H_k)}$. Recall that in the edge sampling algorithm of [23], L_0 is the set of all vertices with degree roughly α. Therefore, in that algorithm, checking if a vertex belongs to L_0 requires a single degree query. In our setting, the degree of a node V_Q in H_k is equivalent to the number of k-cliques that are assigned to Q in G. Hence,

given a sampled node v_Q in H_k, we implement a procedure to check whether $v_Q \in L_0 = L_0(H_k)$, by trying to approximate the number of k-cliques that are assigned to Q in G. To do so efficiently, we replace the threshold α used to define L_0 in [23], by a value $\tau \geq \alpha$, where we will explain how τ is chosen later in the presentation.

Challenge 3: Emulating a random neighbor query. We next explain how we emulate a random neighbor query for a node v_Q in H_k (so as to emulate a random walk on H_k). Let $\mathcal{A}(Q)$ denote the set of k-cliques assigned to Q. By the definition of H_k, sampling an edge incident to v_Q translates to sampling a k-clique C in $\mathcal{A}(Q)$. Let u be the minimum degree vertex in Q, and define $d(Q) = d(u)$, where $d(u)$ is u's degree (in G). As explained above, for $k > 2$, the assignment rule is such that if a k-clique C is assigned to Q, then u is also the minimum degree vertex in C. Hence, in order to select a random neighbor of v_Q in H_k, we need only consider k-cliques C obtained from Q by adding a vertex with degree at least $d(u) = d(Q)$ (that neighbors all vertices in Q). By dealing separately with the case that $d(Q) \leq \sqrt{n\alpha}$ and the case that $d(Q) > \sqrt{n\alpha}$, we can design a procedure that for every $(k-1)$-clique Q samples each k-clique in $\mathcal{A}(Q)$ with probability (roughly) $\frac{1}{\min\{d(Q),\sqrt{n\alpha}\}}$ (and may fail to output any k-clique).

Given the above, to emulate a random neighbor query from a node v_Q in H_k such that $v_Q \notin L_0$ (so that $|\mathcal{A}(Q)| \geq \tau$), we repeat the above sampling attempts $O^*\left(\left\lceil \frac{\min\{d(Q),\sqrt{n\alpha}\}}{\tau} \right\rceil\right)$ times. This process succeeds in obtaining a uniformly distributed k-clique in $\mathcal{A}(Q)$ with high probability. For a node v_Q in L_0 (where we don't have a lower bound on $|\mathcal{A}(Q)|$), performing this number of attempts implies that each k-clique in $\mathcal{A}(Q)$ is obtained with probability $1/\tau$.

An inductive analysis shows that a single invocation of the above emulation of the random walk on H_k returns each k-clique in G with probability roughly $\frac{1}{n\alpha \cdot \tau^{k-2}}$. The $(n\alpha)$ term in the denominator comes from the base of the induction, i.e., sampling a uniform 2-clique (edge) in G, and the term τ^{k-2} stems from the $k-2$ recursive calls, where in each level of recursion, we "lose" a factor of $1/\tau$. Therefore, the overall success probability of a single attempt to sample an edge in H_k is roughly $\frac{n_k}{n\alpha \cdot \tau^{k-2}}$. Hence, $O^*(\frac{n\alpha \cdot \tau^{k-2}}{n_k})$ repetitions are sufficient so that, with high probability, an almost uniformly distributed k-clique in G is returned.

Challenge 4: Proving correctness. Given the above approach, we are able to emulate the basic algorithm of [23] on the auxiliary graph H_k. Hence, to prove correctness we follow the ideas of [23]. However, since the emulation on H_k results in "noisy" answers to node, degree and neighbor queries, so that we cannot immediately rely on the correctness of the algorithm of [23]. Hence, we must (re-)prove that the emulation algorithm induces a distribution on the edges (of H_k) that is close to uniform. This is done by carefully keeping track of the divergence from uniformity that is caused due to the noisy answers to queries throughout the execution of the algorithm. We note that the main challenge lies not in the proof of correctness, but rather in bounding the complexity of the clique-sampling algorithm, as discussed next.

Challenge 5: Bounding the query complexity. As discussed above, to sample a k-clique in G with high probability, we perform $t = O^*\left(\frac{n\alpha \cdot \tau^{k-2}}{n_k}\right)$ repetitions of the random walk emulation on H_k. In each such emulation, there is a sequence of $k-1$ recursive calls to sample i-cliques for $i \in [2, \ldots, k]$ by performing a random walk on the graph H_i. Whenever a random neighbor query is emulated on a node in H_i for $i > 2$, $r = O^*\left(\frac{\min\{d(T), \sqrt{n\alpha}\}}{\tau}\right)$ queries are performed in G. Conditioned on τ being sufficiently larger than α, we get that the expected number of queries in each such emulation is just $O^*(1)$ (while the maximum is $O^*\left(\frac{\sqrt{n\alpha}}{\tau}\right)$). This implies that the expected total query complexity is $O^*\left(\frac{n\alpha \cdot \tau^{k-2}}{n_k}\right)$. As for the maximum running time, we can get an upper bound of $O^*\left(\frac{n\alpha \cdot \tau^{k-2}}{n_k} + \frac{\sqrt{n\alpha}}{\tau}\right)$ by aborting the algorithm if it performs a larger number of queries, while still obtaining an output distribution as desired.

Hence, we get a certain tradeoff between the expected query complexity and the maximum one (for "hard to sample" cliques). In particular, if we set $\tau = \Theta^*(\alpha)$, we get that the expected query complexity is $O^*\left(\frac{n\alpha^{k-1}}{n_k}\right)$, as in the case of counting, while the maximum query complexity is $O^*\left(\frac{n\alpha^{k-1}}{n_k} + \sqrt{n/\alpha}\right)$. The upper bound in Theorem 1.1 is derived by setting τ so that the two summands in the expression $O^*\left(\frac{n\alpha \cdot \tau^{k-2}}{n_k} + \frac{\sqrt{n\alpha}}{\tau}\right)$ are equal.

1.3 A discussion of related random-walk based sampling algorithms

The idea of sampling k-subgraphs (i.e., subgraphs of size k) from a graph G using random walks on an auxiliary graph (in which nodes and edges correspond to subgraphs), is not new, see, e.g., [7, 47, 42, 12, 11]. However, our approach, and in particular our definition of the auxiliary graph H_k, differs from previous ones in several ways, which are crucial for sampling k-cliques according to a distribution that is ε-pointwise close to uniform, with sublinear query complexity. Below we discuss the main aspects of difference between our approach and the aforementioned previous works.

1. The task. We start by noting that in the aforementioned random-walk based works, the focus was on sampling from a distribution whose support is *all* connected k-subgraphs, while we focus on sampling from a distribution whose support is the set of k-cliques. An algorithm for sampling k-subgraphs can be directly adapted to output only k-cliques using rejection sampling, however this could significantly increase the complexity.
2. The distance measure. In most previous works, the considered distance measure is the total variation distance, while our result considers the strictly stronger pointwise distance measure.[8]
3. The definition of the auxiliary graph. The auxiliary graph considered in the aforementioned works, which we denote here by H'_k, is defined as follows. There is a node in H'_k for each subgraph of size $k-1$ in G, and there is an edge between two nodes in H'_k if the two corresponding $(k-1)$-subgraphs differ by a single vertex. Hence, similarly to our auxiliary graph, H_k, edges in the auxiliary graph correspond to the objects that we would like to sample (k-subgraphs in previous works, and k-cliques in ours). However, the way we define the edge-set of H_k is pivotal to the analysis of our algorithm. In particular, we put an edge between two nodes in H_k not only if the union between the corresponding $(k-1)$ cliques is a k-clique, but also if this k-clique is assigned to the two $(k-1)$-cliques. Our assignment rule is tailored to bound the query complexity of the algorithm (based on the degrees of vertices in the cliques). Also note that if the original graph G is connected, then so is H'_k, while H_k is typically not connected.
4. Performing random walks on the auxiliary graph. Recall that in our context, where we are given only query access to G and would like to minimize the number of queries, we have to overcome several challenges in the emulation of random walks on H_k. These do not arise in previous works: The random walk starts from an arbitrary node in H'_k (an arbitrary $(k-1)$-subgraph), and each step in the walk is simply implemented by selecting a random neighbor of one of the vertices in the current $(k-1)$-subgraph.
5. The complexity of the sampling algorithm. As noted above, our focus is on bounding the query complexity of the algorithm, and indeed we get an almost tight bound based on our definition of H_k and the details of the emulation of random walks on H_k given query access to G. In the aforementioned works, the complexity of the algorithms was shown to depend on the mixing time of H'_k, and one of the main challenges of these works is in analyzing it. Indeed, in [11] it is proved that even if the mixing time of G is relatively small, the mixing time of H'_k may be a factor of $\rho(G)^{k-2}$ larger, where $\rho(G)$ is the ratio between the maximum and minimum degree in G (and hence may be large (e.g., $\Omega(n)$) even in bounded-arboricity graphs).

1.4 Overview of the lower bound

The first and last terms in the lower bound of Theorem 1.2 follow directly from a lower bound of $\Omega^*\left(\min\left\{n\alpha, \frac{n\alpha^{k-1}}{n_k}\right\}\right)$ by [26] for the task of approximately counting the number of k-cliques. They prove that any algorithm that performs fewer queries than the above expression cannot distinguish with high probability between two families of graphs, one with n_k k-cliques, and one with no k-cliques. It follows that any uniform sampling algorithm cannot perform fewer queries. Therefore, our main focus is on proving the term $\Omega^*\left(\left(\frac{(n\alpha)^{k/2}}{n_k}\right)^{\frac{1}{k-1}}\right)$ (which, as noted previously, may be much larger than the $\Omega^*\left(\frac{n\alpha^{k-1}}{n_k}\right)$ term).

[8] In [11], Bressan combined a random-walk based approach with a (linear in m) preprocessing of the graph, in order to obtain an exact uniform sampler.

To this end, we construct a family of graphs (with arboricity at most α), such that in each graph, among the n_k k-cliques that it contains, there is one "hidden" k-clique. This clique is hidden in the sense that any algorithm that (always) performs less than $\left(\frac{(n\alpha)^{k/2}}{n_k}\right)^{\frac{1}{k-1}}/c$ queries (for a sufficiently large constant c) cannot sample this clique with probability $\Omega(1/n_k)$.[9]

The above idea is formalized by defining a process that answers the queries of a sampling algorithm "on the fly" while constructing a random graph in the family. All graphs in the family have the same underlying structure, and they differ in the choice of clique vertices and in the labeling of (part of) the edges. Here we give the high-level idea of the underlying structure, and the intuition for the lower-bound expression.

In each graph in the family, the hidden clique is over a subset S of k vertices that all have (high) degree $\Theta(\ell)$ where $\ell = \sqrt{n\alpha}$. The total number of high-degree vertices is $\Theta(\ell)$ as well (so that all graphs graphs in the family have $\Theta(\ell^2) = \Theta(n\alpha)$ edges[10]). Other than the clique edges, there are no other edges between the high-degree vertices. See Figure 1 for an illustration. Intuitively, in order to reveal the hidden clique, the algorithm must first reveal one edge (u, u') in the clique and then reveal $k-2$ additional edges between u and the other edges in the clique.[11] We prove that in each query, the probability of revealing the first edge of the clique is $O(k^2/\ell^2)$, and the probability of revealing any consecutive edge is $O(k/\ell)$.[12]

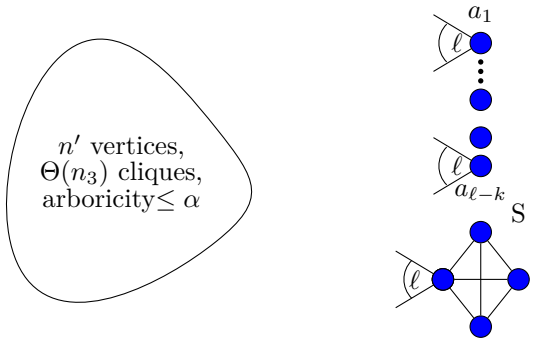


Figure 1 The underlying structure of the graphs in the lower bound construction for $k = 4$. For a more detailed description and figure, see the full version of this paper [24].

The intuition for the upper bound $O(k^2/\ell^2)$ on the probability of revealing the first edge is that the number of clique edges is $\binom{k}{2}$, while the total number of edges is $\Theta(\ell^2)$. Similarly, the rough intuition for the upper bound of $O(k/\ell)$ on revealing each additional edge in the clique is that each clique vertex has $k-1$ neighbors in the clique and a total of $\Theta(\ell)$ neighbors. In order to provide a formal argument, we define an auxiliary bipartite graph

[9] We note that this does not preclude the possibility that the expected complexity of the algorithm is smaller (as discusses in Subsection 1.2.2).

[10] As stated in the introduction, we note that since all of the graphs have $n\alpha$ edges, our lower bound does not exclude the possibility of a refined upper bound that also depends on the number of edges m. (And indeed, it is possible to replace some of the $n\alpha$ terms in the upper bound with m terms. However, we chose not to further complicate the exposition of the algorithm and therefore we also present the lower bound in terms of worst-case m.)

[11] The algorithm may alternatively try to reveal $k/2$ edges in the clique that do not have common endpoints (or some other combination of edges that together reveals all clique vertices), but this is not advantageous for the algorithm.

[12] We note that whenever the term $\frac{(n\alpha)^{k/(2(k-1))}}{kn_k^{1/(k-1)}}$ dominates the last term in the lower bound of Theorem 1.2, it is smaller than $\sqrt{n/\alpha}$.

whose nodes correspond to graphs that are consistent with all previous queries (and answers) and either contain a "witness" clique edge that corresponds to the query of the algorithm (one side of the graph), or do not (the other side). The edges of the bipartite graph are defined by certain transformations from witness graphs to non-witness graphs. By analyzing the degrees of nodes on both sides of this auxiliary graph, we obtain the aforementioned bounds on the probability of revealing edges in the hidden clique.

Given these probability upper bounds, if an algorithm performs T queries, then the probability that it reveals the hidden clique is upper bounded by $T \cdot \frac{k^2}{\ell^2} \cdot \left(T \cdot \frac{k}{\ell}\right)^{k-2}$. If we want this expression to be $\Omega(1/n_k)$, the number of queries T must be $\Omega^*\left(\left(\frac{(n\alpha)^{k/2}}{n_k}\right)^{1/(k-1)}\right)$.

1.5 Related Work

The works most related to ours were mentioned in earlier subsections of the introduction. In Appendix A, we give a broader view of recent advances on sublinear-time approximate counting and uniform sampling algorithms.

1.6 Organization

We start with some preliminaries in Section 2. Due to the page limitation, in this extended abstract we only describe the algorithm and a sketch of the analysis for the case of $k = 3$ (triangles) – see Section 3.

The full algorithm and analysis for the general case, as well as the lower bound, can be found in the full version of this paper [24].

2 Preliminaries

Let $G = (V, E)$ be a graph over n vertices and arboricity at most α. Each vertex $v \in V$ has a unique id in $[n]$, denoted $id(v)$. Let $\mathcal{C}_k$ denote the set of k-cliques of G, and let $n_k = |\mathcal{C}_k|$. For a vertex v, let $\Gamma(v) = \Gamma_G(v)$ denote its set of neighbors and let $d(v) = d_G(v) = |\Gamma(v)|$. We sometimes refer to edges as oriented, meaning that we consider each edge from both its endpoints.

Access to G is given via the following types of queries: (1) A degree query, $\mathsf{deg}(v)$, returns the degree $d(v)$ of the vertex v; (2) A neighbor query, $\mathsf{nbr}(v, i)$ for $i \in [d(v)]$, returns the i^{th} neighbor of v; (3) A pair query, $\mathsf{pair}(v, v')$, returns whether $(v, v') \in E$.

▶ **Definition 2.1** (Ordering of the vertices.)**.** *We define an ordering on the graph's vertices, where $u \prec v$ if $d(u) < d(v)$ or if $d(u) = d(v)$ and $id(u) < id(v)$.*

▶ **Definition 2.2** (Cliques' degree and neighbors)**.** *For a k-clique C, let v be the minimal vertex in v_C according to $\prec$. We define the degree of C in G to be $d(C) = d(v)$. We define the neighbor-set of C, denoted $\Gamma(C) = \Gamma(v)$, to be the set of v's neighbors in G.*

▶ **Definition 2.3** (Cliques id and an ordering of cliques)**.** *For a k-clique C, let its id, $id(C)$ be a concatenation of its vertices ordered by $\prec$. We extend the order $\prec$ to cliques, so that for two k cliques C, C', $C \prec C'$ if $d(C) < d(C')$ or if $d(C) = d(C')$ and $id(C) < id(C')$.*

▶ **Definition 2.4** (Assignment of k-cliques to $(k-1)$-cliques)**.** *We* assign *each k-clique C to its two first $(k-1)$-cliques according to $\prec$. For every $(k-1)$ clique Q, we denote its set of assigned k-cliques by $\mathcal{A}(Q)$, and let $a(Q) = |\mathcal{A}(Q)|$. We refer to $a(Q)$ as the assigned cliques degree of Q.*

Note that for every $k \geq 3$, $a(Q) \leq d(Q)$.

▶ **Observation 2.5.** *By Definition 2.4, for $k \geq 3$, if Q and Q' are assigned a k-clique C, then $d(Q) = d(Q') = d(C)$. Hence, if a k-clique C is assigned to a $(k-1)$-cliques Q such that $C = Q \cup \{w\}$, then $d(Q) = d(C) \leq d(w)$.*

We shall sometimes abuse notation and let $\{Q, u\}$ denote $Q \cup \{w\}$.

We are now ready to define the auxiliary graph H_k, which is central to our algorithm.

▶ **Definition 2.6** (The graph H_k)**.** *Given a graph G, we define the graph $H_k(G) = H_k = (V_{H_k}, E_{H_k})$ as follows. For every $(k-1)$-clique Q in G there is a node v_Q in V_{H_k}. For every k-clique C in G, there is an edge in H_k between the two $(k-1)$-cliques that C is assigned to, as defined in Definition 2.4.*

For the sake of clarity, throughout the paper, we refer to the vertices of H_k as nodes. Note that for the special case of $k = 2$, we have that $H_2(G) = G$, and each edge (2-clique) in G, is assigned to both its endpoints. More generally, Definition 2.6 implies a one-to-one correspondence between the set of edges incident to a node v_Q in $H_k(G)$ and the set $\mathcal{A}(Q)$ of k-cliques assigned to Q in G. This in turn implies that the degree of a node v_Q in $H_k(G)$ equals the assigned cliques degree of Q, $a(Q)$. By the comment following Definition 2.4, the degree of v_Q in $H_k(G)$ is upper bounded by the degree of Q in G.

The last claim in this section concerns the arboricity of $H_k(G)$.

▷ Claim 2.7. Let G be a graph of arboricity at most α. Then $H_k(G)$ has arboricity at most α.

3 The case of $k = 3$: sampling triangles

As a warmup, in this section we describe our algorithm for the case of sampling triangles and provide the structure of its analysis. To ease readability, some of the claims we present are loosely stated (the precise and general claims appear (and are proved) in the next section). Since the graph G is fixed throughout the presentation, we use the shorthand H_3 for $H_3(G)$.

In addition to receiving as input n, α, and ϵ, as well as being given query access to G, the sampling algorithm, Sample-Triangle, receives the parameter $\overline{n}_3$, which is assumed to be a constant factor estimate of the number of triangles, n_3. Such an estimate can be obtained by running the algorithm of [26], without asymptotically increasing the expected complexity of our algorithm.

To sample a triangle in G, the algorithm Sample-Triangle repeatedly invokes the procedure Sample-Edge-Auxiliary-Tri on the graph H_3, while ensuring that the number of queries does not exceed a certain threshold. For the sake of conciseness, from this point on we view the parameters that Sample-Triangle receives, as global variables for all other procedures.

Sample-Triangle$(n, \alpha, \varepsilon, \overline{n}_3)$

1. Set $\tau = \max\left\{\frac{\sqrt{\overline{n}_3}}{(n\alpha)^{1/4}}, \alpha\right\}$.
2. While the number of queries does not exceed $r = c \cdot \min\left\{n\alpha, \max\left\{\frac{\sqrt{n\alpha}}{\tau}, \frac{n\alpha\tau}{\overline{n}_3}\right\}\right\}$ for a sufficiently large constant c:
   a. Invoke Sample-Edge-Auxiliary-Tri(τ), and if an edge in H_3 is returned, then **return** the corresponding 3-clique (triangle) in G.

▶ **Theorem 3.1** (loosely stated)**.** *The algorithm* Sample-Triangle *is a pointwise ε-close to uniform sampling algorithm for triangles (3-cliques) in graphs with arboricity at most α. The query complexity of the algorithm is $O^*\left(\min\left\{n\alpha, \max\left\{\left(\frac{(n\alpha)^{3/4}}{\sqrt{n_3}}\right), \frac{n\alpha^2}{n_3}\right\}\right\}\right)$.*

We defer the proof of the theorem to the end of this section, and continue to describe the procedure Sample-Edge-Auxiliary-Tri. This procedures (tries to) return an (almost) uniformly distributed edge in the auxiliary graph H_3 (corresponding to a triangle in G), so that each edge is returned with probability (roughly) $\frac{1\pm\Theta(\varepsilon)}{n\alpha\tau}$, and it is the main procedure used in order to prove Theorem 3.1.

As will be explained in more detail momentarily, the setting of τ in Sample-Triangle (together with the random coins of the algorithm) determines a set $L_0(H_3)$ of nodes in H_3. Recall that the degree of a node v_Q in H_3 (where Q is a 2-clique, i.e., an edge in G) is the number of triangles (3-cliques) that are assigned to Q according to the assignment rule of Definition 2.4 (denoted $a(Q)$). With high probability over the randomness of the algorithm, all nodes $v_Q \in H_3$ whose degree (in H_3) at most τ belong to $L_0(H_3)$, and all nodes $v_Q \in H_3$ with degree greater than 2τ do not belong to $L_0(H_3)$ (the rest of the nodes can belong to either set). We use $E(L_0(H_3))$ to denote the edges in H_3 that are incident to nodes in $L_0(H_3)$.

Sample-Edge-Auxiliary-Tri first invokes the subroutine Sample-Edge-L_0-Auxiliary-Tri that either returns a (close to) uniform edge (v_0, v_1) among the edges of $E(L_0(H_3))$ or returns FAIL. The procedure then chooses an index j in $[0, \ldots, \log(n\alpha)]$ uniformly at random, and performs a random walk of length j on H_3, by traversing at each step to a uniformly selected neighbor of the last visited node. This is done by invoking the procedure Sample-Neighbor-Auxiliary-Tri. If at any point the last visited node belongs to $L_0(H_3)$ (which is verified by the procedure Define-L_0-Auxiliary-Tri), then the procedure fails. Otherwise, the last traversed edge in the walk is returned.

Sample-Edge-Auxiliary-Tri(τ).

1. Set $s = \log(n\alpha)$ and set $\beta = \varepsilon/(2s+2)$.
2. Invoke Sample-Edge-L_0-Auxiliary-Tri(β, τ), and let $e_0 = (v_{Q_0}, v_{Q_1})$ be the returned edge if one was returned. Otherwise, **return** FAIL.
3. Choose $j \in [0, \ldots, s]$ uniformly at random.
4. For $i = 1$ to j do:
   a. If Define-L_0-Auxiliary-Tri($v_{Q_i}, \delta = \beta/\overline{n}_3, \beta, \tau$)=YES, then **return** FAIL.
   b. Invoke Sample-Neighbor-Auxiliary-Tri(v_{Q_i}, β, τ) to sample an edge $(v_{Q_i}, v_{Q_{i+1}})$ in H_3.
5. **Return** $(v_{Q_j}, v_{Q_{j+1}})$.

▶ **Lemma 3.2** (loosely stated). *The procedure* Sample-Edge-Auxiliary-Tri *returns an edge in H_3 so that each edge is returned with probability (roughly) $\frac{(1\pm\varepsilon)}{n\alpha\tau}$. Furthermore, the expected running time of a single invocation of the procedure is $O^*(1)$, and the maximum running time is $O^*(\sqrt{n\alpha}/\tau)$.*

Before presenting the subroutines Sample-Edge-L_0-Auxiliary-Tri, Sample-Neighbor-Auxiliary-Tri and Define-L_0-Auxiliary-Tri invoked in Sample-Edge-Auxiliary-Tri, we describe a simple procedure, Samp-High-Deg-Nbr, used by these subroutines. The procedure gets a 2-clique (edge) Q as input, and tries to sample a higher degree neighbor of Q in G, so that each such neighbor is returned with probability $1/\min\{d(Q), \sqrt{n\alpha}\}$. As mentioned in the introduction, we shall make use of the procedure Sample-Basic-Edge by [29], that returns every edge in G with probability (roughly) $(1 \pm \beta)/(n\alpha)$, given an approximation parameter β.

Samp-High-Deg-Nbr(Q, β).

1. Let u be the min degree vertex of Q, and query $d(u)(= d(Q))$.
2. If $d(Q) \leq \sqrt{n\alpha}$, then query the i^{th} neighbor of u in G, for a uniformly selected index $i \in d(u)$. If $d(w) \geq d(Q)$, then return w.
3. If $d(Q) > \sqrt{n\alpha}$, then:
   a. Invoke Sample-Basic-Edge(β) and if an edge is returned then denote it (w, x). Otherwise, return FAIL.
   b. Query $d(w)$ and if $d(w) > d(Q)$, then return the endpoint w with probability $d(w)/\sqrt{n\alpha}$. Otherwise, return FAIL.

▷ Claim 3.3 (loosely stated). Let Q be a 2-clique (edge) in G. The procedure Samp-High-Deg-Nbr either returns a neighbor of Q in G (that is, a neighbor in G of the min-degree vertex of Q), or fails. Each $w \in \Gamma(Q)$ such that $d(w) \geq d(Q)$ is returned with (roughly) equal probability $(1 \pm \beta)/\min\{d(Q), \sqrt{n\alpha}\}$. The query and time complexity of the procedure are $O^*(\log n)$.

We turn to present the subroutines used by Sample-Edge-Auxiliary-Tri, starting with the subroutine Define-L_0-Auxiliary-Tri, that defines the aforementioned set $L_0(H_3) \subseteq V(H_3)$. Namely, $L_0(H_3)$ is determined according to the output of the subroutine, so that

$$L_0(H_3) = \{v_Q \in V_{H_3} \ : \ \text{Define-}L_0\text{-Auxiliary-Tri}(v_Q, \delta, \beta, \tau) = \ \text{YES} \ \}$$

(where we assume that the randomness of the subroutine is uniquely determined for each v_Q). Hence Define-L_0-Auxiliary-Tri can be thought of as an *oracle* that returns whether a given v_Q belongs to $L_0(H_3)$ or not.

Define-L_0-Auxiliary-Tri$(v_Q, \delta, \beta, \tau)$.

1. Let Q be the 2-clique (edge) in G corresponding to v_Q.
2. For $i = 1$ to $r = \frac{\min\{d(Q), \sqrt{n\alpha}\}}{\tau} \cdot \ln(1/\delta)$ do:
   a. Invoke Samp-High-Deg-Nbr$(Q, \beta/10)$ to (try and) sample a neighbor w_i of Q.
   b. If Q and w_i form a triangle C, and C is assigned to Q, then let $\chi_{w_i} = 1$.
3. Let $\tilde{a} = \frac{1}{r}\sum_{i=1}^{r} \chi_{w_i}$.
4. If $\tilde{a} < 1.5\tau/d(Q)$ then return YES. Otherwise, return NO.

Recall that $\mathcal{A}(Q)$ is the set of cliques assigned to Q, and $a(Q) = |\mathcal{A}(Q)|$.

▷ Claim 3.4 (loosely stated). With high probability Define-L_0-Auxiliary-Tri determines a set $L_0(H_3)$ so that the following holds for every $v_Q \in H_3$.

- If $a(Q) \leq \tau$, then $v_Q \in L_0(H_3)$ and the subroutine returns YES.
- If $a(Q) > 2\tau$, then $v_Q \notin L_0(H_3)$ and the subroutine returns NO.

Furthermore, the query and time complexities of the subroutine are $O^*\left(\frac{\min\{d(Q), \sqrt{n\alpha}\}}{\tau}\right)$.

Given Claim 3.4, from here on we assume that for every $Q \in L_0(H_3)$, $dt(Q) \leq 2\tau$, and for every $Q \notin L_0(H_3), a(Q) > \tau$.

We now present the subroutine Sample-Edge-L_0-Auxiliary-Tri that is used for sampling edges in H_3 that are incident to nodes in $L_0(H_3)$. In order to sample a uniform edge in $E(L_0(H_3))$, we first need to sample a node in $L_0(H_3)$. Recall that the nodes of H_3 correspond to edges in G. Hence, to sample a node in $L_0(H_3)$, the procedure first invokes Sample-Basic-Edge to sample an edge in G (almost) uniformly at random, and then checks if the corresponding node is in $L_0(H_3)$ (by invoking Define-L_0-Auxiliary-Tri).

Sample-Edge-L_0-Auxiliary-Tri(β, τ).

1. Invoke Sample-Basic-Edge$(\beta/4)$. Let Q be the returned 2-clique if one was returned. Otherwise FAIL.
2. If Define-L_0-Auxiliary-Tri$(v_Q, \delta, \beta/4, \tau)$=NO then FAIL.
3. Repeat at most $r = O^*\left(\frac{\min\{d(Q), \sqrt{n\alpha}\}}{2\tau}\right)$ times or until a neighbor is sampled:
   a. Invoke Samp-High-Deg-Nbr$(Q, \beta/10)$ to (try and) sample a neighbor w of Q.
4. If no neighbor is sampled, then return FAIL.
5. Check if $\{Q, w\}$ is a triangle assigned to Q. If so, return $C = \{Q, w\}$.

▷ Claim 3.5. Consider an invocation of Sample-Edge-L_0-Auxiliary-Tri with parameters β and τ. The subroutine Sample-Edge-L_0-Auxiliary-Tri returns every edge in $E(L_0(H_3))$ with probability (roughly) $(1 \pm \beta)/(2n\alpha\tau)$. The expected query and time complexity of the subroutine are $O^*(1)$ and the maximum query and time complexity are $O^*\left(\frac{\sqrt{n\alpha}}{\tau}\right)$.

Finally, the subroutine Sample-Neighbor-Auxiliary-Tri (tries to) sample a neighbor of a node v_Q in the auxiliary graph H_3. That is, it tries to sample a triangle $C \in \mathcal{A}(Q)$.

Sample-Neighbor-Auxiliary-Tri(v_Q, β).

1. Let Q be the 2-clique (edge) in G corresponding to v_Q.
2. Repeat at most $r = \frac{\min\{d(Q), \sqrt{n\alpha}\}}{\tau} \cdot \ln(1/\beta)$ times:
   a. Invoke Samp-High-Deg-Nbr$(Q, \beta/10)$ to (try and) sample a neighbor w of Q.
   b. If Q and w form a triangle C, and C is assigned to Q, then **return** the edge $(v_Q, v_{Q'})$ in H_3 that corresponds to C.

▷ Claim 3.6 (loosely stated). For a given node $v_Q \in V(H_3)$ such that $Q \notin L_0(H_3)$, with probability at least $1 - \beta$, the subroutine Sample-Neighbor-Auxiliary-Tri returns a neighbor of v_Q in H_3, so that each neighbor of v_Q is returned with probability (roughly) $(1 \pm \beta)/a(Q)$. The query and time complexity of the subroutine are $O^*\left(\frac{\min\{d(Q), \sqrt{n\alpha}\}}{\tau}\right)$.

Finally, we sketch the proof of (the loosely stated) Theorem 3.1.

Proof sketch of Theorem 3.1. By Lemma 3.2, Sample-Edge-Auxiliary-Tri returns every specific edge in H_3 with probability (roughly) $(1 \pm \varepsilon)/(n\alpha\tau)$. Since there is a one-to-one correspondence between edges in H_3 to triangles in G, this implies that each triangle is returned with probability (roughly) $(1 \pm \varepsilon)/(n\alpha\tau)$, and that with probability (roughly) $n_3/(n\alpha\tau)$, some triangle is returned. Hence, the expected number of invocations of the loop is $O(n\alpha\tau/n_3)$. Furthermore, by Claim 3.2, the expected complexity of each invocation is $O^*(1)$. It can be proven using standard concentration bounds, that with high probability, a triangle will be returned before the number of allowed queries r is exceeded. Therefore, with high probability, a triangle is returned, and since all triangle are almost equally likely to be the returned one, it holds that each triangle is returned with probability (roughly) $1/n_3$.

Furthermore, since the expected complexity of each invocation of Sample-Edge-Auxiliary is $O^*(1)$ and the maximum is $O^*(\sqrt{n\alpha}/\tau)$, it follows that the query and time complexity of Sample-Triangle is $O^*(r + \sqrt{n\alpha}/\tau) = O^*\left(\min\left\{n\alpha, \max\left\{\frac{\sqrt{n\alpha}}{\tau}, \frac{n\alpha\tau}{n_3}\right\}\right\}\right)$. Plugging $\tau = \max\left\{\frac{\sqrt{n_3}}{(n\alpha)^{1/4}}, \alpha\right\}$, we get that the query complexity of Sample-Triangle is bounded by

$$O^*\left(\min\left\{n\alpha, \max\left\{\frac{(n\alpha)^{3/4}}{\sqrt{n_3}}, \frac{n\alpha^2}{n_3}\right\}\right\}\right),$$

as claimed. ◀

References

1 Balázs Adamcsek, Gergely Palla, Illés J Farkas, Imre Derényi, and Tamás Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006.

2 Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica*, 80(2):668–697, 2018.

3 Uri Alon. *An introduction to systems biology: design principles of biological circuits*. Chapman and Hall/CRC, 2006.

4 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:20, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2019.6.

5 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Transactions on Algorithms (TALG)*, 16(4):1–27, 2020.

6 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Triangle estimation using tripartite independent set queries. In *30th International Symposium on Algorithms and Computation (ISAAC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

7 Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th International Conference on Data Mining*, pages 91–100. IEEE, 2012.

8 Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Bipartite independent set oracles and beyond: Can it even count triangles in polylogarithmic queries? *arXiv preprint*, 2021. arXiv:2110.03836.

9 Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Slobodan Mitrovic, and Ronitt Rubinfeld. Parallel algorithms for small subgraph counting. *CoRR*, abs/2002.08299, 2020. arXiv:2002.08299.

10 Amartya Shankha Biswas, Talya Eden, and Ronitt Rubinfeld. Towards a decomposition-optimal algorithm for counting and sampling arbitrary motifs in sublinear time. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 55:1–55:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.55.

11 Marco Bressan. Efficient and near-optimal algorithms for sampling connected subgraphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1132–1143, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451042.

12 Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Motif counting beyond five nodes. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(4):1–25, 2018.

13 Broto Chakrabarty and Nita Parekh. Naps: network analysis of protein structures. *Nucleic acids research*, 44(W1):W375–W382, 2016.

14 Jianer Chen, Xiuzhen Huang, Iyad A Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006.

15 Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2916–2935. SIAM, 2020.

16 Xiaowei Chen, Yongkun Li, Pinghui Wang, and John CS Lui. A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment*, 10(3), 2016.

17 Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference*, pages 589–598, 2018.

18 Dhruba Deb, Saraswathi Vishveshwara, and Smitha Vishveshwara. Understanding protein structure from a percolation perspective. *Biophysical journal*, 97(6):1787–1794, 2009.

19 Nurcan Durak, Ali Pinar, Tamara G. Kolda, and C. Seshadhri. Degree relations of triangles in real-world networks and graph models. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 1712–1716, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2396761.2398503.

20 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. In *Foundations of Computer Science , 2015 IEEE 56th Annual Symposium on*, pages 614–633. IEEE, 2015.

21 Talya Eden, Saleet Mossel, and Dana Ron. Approximating the arboricity in sublinear time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2404–2425. SIAM, 2022. doi:10.1137/1.9781611977073.96.

22 Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. Sampling multiple edges efficiently. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 51:1–51:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.51.

23 Talya Eden, Dana Ron, and Will Rosenbaum. The arboricity captures the complexity of sampling edges. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 52:1–52:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.52.

24 Talya Eden, Dana Ron, and Will Rosenbaum. Almost optimal bounds for sublinear-time sampling of k-cliques: Sampling cliques is harder than counting. *CoRR*, abs/2012.04090, 2020. arXiv:2012.04090.

25 Talya Eden, Dana Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The arboricity connection. *SIAM J. Discrete Math.*, 33(4):2267–2285, 2019. doi:10.1137/17M1159014.

26 Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1467–1478, 2020. doi:10.1137/1.9781611975994.89.

27 Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. *SIAM Journal on Computing*, 49(4):747–771, 2020. doi:10.1137/18M1176701.

28 Talya Eden and Will Rosenbaum. Lower bounds for approximating graph parameters via communication complexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques 2018*, pages 11:1–11:18, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.11.

29 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 7:1–7:9, 2018. doi:10.4230/OASIcs.SOSA.2018.7.

30 David Eppstein and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. In *International Symposium on Experimental Algorithms*, pages 364–375. Springer, 2011.

31 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.

32 Hendrik Fichtenberger, Mingze Gao, and Pan Peng. Sampling arbitrary subgraphs exactly uniformly in sublinear time. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 45:1–45:13, 2020. doi:10.4230/LIPIcs.ICALP.2020.45.

33 Irene Finocchi, Marco Finocchi, and Emanuele G Fusco. Clique counting in mapreduce: Algorithms and experiments. *Journal of Experimental Algorithmics (JEA)*, 20:1–20, 2015.

34 Gaurav Goel and Jens Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 159–167. Springer, 2006.

35 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008. doi:10.1002/rsa.20203.

36 Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM Journal on Discrete Mathematics*, 25(3):1365–1411, 2011.

37 Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th international conference on world wide web*, pages 441–449, 2017.

38 Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

39 Idit Kosti, Predrag Radivojac, and Yael Mandel-Gutfreund. An integrated regulatory network reveals pervasive cross-regulation among transcription and splicing factors. *PLoS computational biology*, 8(7):e1002603, 2012.

40 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 472–482. Springer, 2015. doi:10.1007/978-3-662-48054-0_39.

41 Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 815–824, 2015.

42 Tanay Kumar Saha and Mohammad Al Hasan. Finding network motifs using mcmc sampling. In *Complex Networks VI*, pages 13–24. Springer, 2015.

43 Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems*, 54(3):677–710, 2018.

44 Jakub Tetek. Approximate triangle counting via sampling and fast matrix multiplication. *CoRR*, abs/2104.08501, 2021. To appear in ICALP 2022. arXiv:2104.08501.

45 Jakub Tětek and Mikkel Thorup. Edge sampling and graph parameter estimation via vertex neighborhood accesses. *CoRR*, 2021. To appear in STOC 2022. arXiv:2107.03821.

46 Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254–257, 2009.

47 Pinghui Wang, John CS Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(2):1–27, 2014.

A Related Work

We note that some of the works were mentioned before, but we repeat them here for the sake of completeness. In recent years, there has been an increasing interest in the questions of subgraph approximate counting and uniform sampling in sublinear-time. These works differ by the query model, graph class of G and the subgraph H at question.

The general graph query model. The first works on estimating the number of subgraph counts were by Feige [31] and Goldreich and Ron [35], who presented algorithms for approximately counting the number of k-cliques in a graph for $k = 2$ (edges).[13] Later, Gonen, Ron and Shavitt [36] gave essentially optimal bounds for the problem of approximately counting the number of stars in a graph. In [20, 27], Eden, Levi, Ron and Seshadhri and Eden, Ron and Seshadhri presented essentially optimal query complexity bounds for the problems of approximately counting triangles and k-cliques. In [44], Tětek gave improved running time bounds for the regime where the query complexity is linear in previous works.

In [28], Eden and Rosenbaum presented a framework for proving subgraph counting lower bounds using reduction from communication complexity, which allowed them to reprove the lower bounds for all of the variants listed above.

Augmented model. In [2], Aliakbarpour, Biswas, Gouleakis, Peebles, and Rubinfeld and Yodpinyanee suggested a model that also allows for uniform edge samples. In that model they presented improved bounds for the approximate star counting problem. In that model, Assadi, Kapralov and Khanna [4] considered the problem of approximate counting of arbitrary subgraphs H. The expected query complexity of their algorithm is $\tilde{O}\left(\min\left\{m, \frac{m^{\rho(H)}}{n_H}\right\}\right)$, where $\rho(H)$ is the fractional edge cover of H,[14] and n_H is the number of copies of H in G. In particular, for the case of k-clique (and odd-cycle counting) the complexity of their algorithm is $O(m^{k/2})$, and this is optimal. In [10], Biswas, Eden and Rubinfeld have refined the complexity of approximating and uniformly sampling arbitrary motifs to $O^*\left(\min\left\{m, \mathsf{decomp\text{-}cost}(G, H, D^*(H))\right\}\right)$, where $D^*(H)$ is an optimal decomposition of H, and $\mathsf{decomp\text{-}cost}$ is the decomposition cost of H in G.

Set query model. In [5], Beame, Har-Peled, Ramamoorthy and Sinha suggested two new models that allow what they refer to as *independent set* (IS) and *bipartite independent set* (BIS) queries. They considered the problem of estimating the number of edges and gave $O^*(n^{2/3})$ and $O^*(1)$ algorithms for this problem using IS and BIS queries, respectively. The first result was later improved by Chen, Levi and Waingarten [15] who settled the complexity of the problem to $\Theta^*(n/\sqrt{m})$. In [6], Bhattacharya, Bishnu, Ghosh, and Mishra later have generalized the BIS model to tripartite set queries, where they considered the problem of triangle counting, and in [8], Bishnu, Ghosh, and Mishra settled the complexity of approximately counting triangles using BIS queries to $\Theta^*\left(\min\left\{\frac{m}{\sqrt{T}}, \frac{m^{3/2}}{T}\right\}\right)$, where T denotes the number of triangles.

[13] Feige considered a model that only allows for degree queries, and presented a factor 2 approximation algorithm, and also proved that with no additional queries this approximation factor cannot be improved in sublinear time. Goldreich and Ron then considered this question allowing also for neighbor queries. In this model the proved an $(1 \pm \varepsilon)$-factor approximation algorithm with the same complexity as the previous one (as well as a matching lower bound).

[14] The fractional edge cover of a graph $H = (V_H, E_H)$ is a mapping $\psi : E_H \to [0, 1]$ such that for each vertex $a \in V_H$, $\sum_{e \in E_H, a \in e} \psi(e) \geq 1$. The fractional edge-cover number $\rho(H)$ of H is the minimum value of $\sum_{e \in E_H} \psi(e)$ among all fractional edge covers ψ.

Uniform sampling. In [29], Eden and Rosenbaum initiated the study of sampling subgraphs (almost) uniformly at random. They considered the general graph query model, and presented upper and matching lower bounds for the problem of sampling edges almost uniformly. Their algorithm matches the complexity of the counting variant of the problem. Their algorithm's dependency on ε was later improved by Tětek and Thorup [45] to $\log(1/\varepsilon)$, so that for all practical purposes the new algorithm allows to sample from essentially the uniform distribution. They also present an algorithm that works in a stronger model that allows for hash-based neighbor queries, and outputs an edge from *exactly* the uniform distribution.

In [32], Fichtenberger, Gao and Peng proved that in the augmented edge model, *exact* uniform sampling of arbitrary subgraphs can be performed in $O\left(\frac{m^{\rho(H)}}{n_H}\right)$ time. This matches the upper bound of [4] for the counting variant. The aforementioned bound of [10], also holds for this setting, refining over the complexity of [32].

In [22], Eden, Mossel and Rubinfeld presented an algorithm for sampling multiple edges efficiently. Their algorithm was later shown to be optimal by Tětek and Thorup [45].

Graphs G with bounded arboricity. In [25, 26], Eden, Ron and Seshadhri first studied the problem of sublinear approximate counting in bounded arboricity graphs. They presented improved algorithm for edges, star and k-clique counting in the general graph model, parameterized by the arboricity. In [23], Eden Ron and Rosenbaum presented an improved algorithm for almost uniform sampling of edges in bounded arboricity graphs, in the general graph query model.

Approximating the arboricity in sublinear time. In [21], Eden, Mossel and Ron presented an algorithm for approximating the arboricity in $\tilde{O}(n/\alpha)$ time. Their algorithm returns a value $\hat{\alpha}$ such that, with high probability, $\hat{\alpha} \in [\alpha, \alpha \cdot c\log^2 n]$, for some constant c. It can also be shown that the algorithm of McGregor, Tench, Vorotnikova and Vu [40] can be adapted to the adjacency list query model,[15] resulting in a $(1 \pm \varepsilon)$-multiplicative approximation in $\tilde{O}(m/\alpha)$ complexity. The output of these algorithms can be used as input to our algorithm (as well as all aforementioned sublinear-time algorithms that rely on getting an upper bound on the arboricity as input).

[15] This requires some care, as their algorithm relies on sampling edges uniformly at random, which is not immediately implementable in the adjacency list model. However, it can be shown that only edges with high degree endpoints (roughly ones with degree above α) are of interest, and these can be sampled uniformly with an additive overhead of $\tilde{O}(m/\alpha)$.

On Sampling Symmetric Gibbs Distributions on Sparse Random Graphs and Hypergraphs

Charilaos Efthymiou ✉
Computer Science, University of Warwick, Coventry, UK

Abstract

In this paper, we present a novel, polynomial time, algorithm for approximate sampling from symmetric Gibbs distributions on the sparse random graph and hypergraph. The examples of symmetric distributions we consider here include some important distributions on spin-systems and spin-glasses. These are: the q-state antiferromagnetic Potts model for $q \geq 2$, including the (hyper)graph Ising model and random colourings. The uniform distribution over the Not-All-Equal solutions of a random k-SAT formula. Finally, we consider sampling from the *spin-glass* distribution called the k-spin model, i.e., this is the "diluted" version of the well-known Sherrington-Kirkpatrick model. Spin-glasses give rise to very intricate distributions which are also studied in mathematics, in neural computation, computational biology and many other areas. To our knowledge, this is the first rigorously analysed efficient sampling algorithm for spin-glasses which operates in a non trivial range of the parameters of the distribution.

We present, what we believe to be, an elegant sampling algorithm. Our algorithm is unique in its approach and does not belong to any of the well-known families of sampling algorithms. We derive it by investigating the power and the limits of the approach that was introduced in [Efthymiou: SODA 2012] and combine it, in a novel way, with powerful notions from the Cavity method.

Specifically, for a symmetric Gibbs distribution μ on the random (hyper)graph whose parameters are within an appropriate range, our sampling algorithm has the following properties: with probability $1-o(1)$ over the instances of the input (hyper)graph, it generates a configuration which is distributed within total variation distance $n^{-\Omega(1)}$ from μ. The time complexity is $O((n \log n)^2)$, where n is the size of the input (hyper)graph.

We make a notable progress regarding impressive predictions of physicists relating phase-transitions of Gibbs distributions with the efficiency of the corresponding sampling algorithms. For most (if not all) the cases we consider here, our algorithm outperforms by far any other sampling algorithms in terms of the permitted range of the parameters of the Gibbs distributions.

The use of notions and ideas from the Cavity method provides a new insight to the sampling problem. Our results imply that there is a lot of potential for further exploiting the Cavity method for algorithmic design.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Randomness, geometry and discrete structures; Mathematics of computing → Discrete mathematics

Keywords and phrases spin-system, spin-glass, sparse random (hyper)graph, approximate sampling, efficient algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.57

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2007.07145

Funding *Charilaos Efthymiou*: EPSRC New Investigator Award (grant no. EP/V050842/1) and Centre of Discrete Mathematics and Applications (DIMAP), The University of Warwick.

Acknowledgements The author of this work would like to thank Amin Coja-Oghlan for the fruitful discussions.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 57; pp. 57:1–57:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Random constraint satisfaction problems (r-CSPs) have been the subject of intense study in combinatorics, computer science and statistical physics. In computer science the study of random CSPs is motivated by a wealth of applications, e.g., they are used as algorithmic benchmarks for hard problems such as the graph colouring, or the k-SAT, they are studied as models for statistical inference, they are also used as gadgets for cryptographic constructions, or reductions in complexity theory [23, 20, 19, 26].

Physicists, independently, have been studying random CSPs as models of *disordered systems* using the so-called *Cavity Method* (e.g. see [32, 29]). The Cavity method originates from the groundbreaking ideas in physics which got Giorgio Parisi the 2021 Nobel Prize in Physics. With its very impressive predictions and its deep intuition, the Cavity Method attracted the interest of both computer scientists and mathematicians, despite its lack of mathematical rigour. In the last two-decades, or so, ideas from the Cavity method have blended the study of random CSPs in computer science and have yield some beautiful results and breakthroughs in the area e.g., [11, 1, 34, 12, 24].

A fundamental notion in physicists' predictions is that of the Gibbs distribution. Using the Cavity method, physicists make predictions relating phase-transitions of Gibbs distributions with the efficiency of the sampling algorithms. Establishing rigorously these connections is a very challenging task and, despite any recent advances, most of the central questions still remain open. In this work, we introduce a novel approach to the sampling problem that exploits *intuition* from the Cavity method, as well as *mathematical tools* and ideas that were developed for the study of random CSPs in conjunction with the Cavity method. Our approach yields efficient algorithms with notable performance with respect to the allowed region of the parameters of the problem.

We remark that this is not the first attempt to employ ideas from the Cavity method for algorithmic design. The celebrated heuristics *Belief Propagation* and *Survey Propagation* [5, 33] are prominent examples of physicists' attempt to turn the Cavity method into an algorithm. Despite their impressive empirical performance, we lack a rigorous mathematical analysis for these heuristics with respect to generating samples from Gibbs distributions.

Our algorithm here is for sampling from what we call *symmetric Gibbs distributions*. This includes important examples such as the (hyper)graph q-colourings and its generalisation the q-state Potts model for $q \geq 2$, the symmetric variants of k-SAT such as the not-all-equal k-SAT (k-NAE-SAT). A notable case is the *spin-glass* k-spin model, i.e., the same spin-glasses that Parisi studied in the 80's. Spin-glasses give rise to very intricate distributions which are also studied in mathematics, in neural computation, computational biology and many other areas [37]. For us, the underlying geometry is an instance of the random graph, or hypergraph of constant expected degree $d > 0$.

For most of the above distributions it is extremely challenging to sample from. This is not only because the underlying geometry is random. Each of the aforementioned distributions exhibits special features that make the analysis of known sampling techniques intractable. E.g., in the interesting region of parameters for k-NAE SAT, or hypergraph colourings, we have untypical configurations with *non-local* freezing of the variables, the spin-glasses exhibit local, randomly induced, *frustration* phenomena, etc.

An additional motivation for this work comes from our desire to investigate the power and the limits of the (well-known) sampling method that is introduced in [15]. The method in [15], (or any of its subsequent works) does not exploit the Cavity method. On a high level, the approach summarises as follows: having the graph G at the input, the algorithm initially

removes all the edges and generates a configuration for the empty graph. Then, iteratively, the algorithm puts the edges back one by one. If G_i is the subgraph we have at iteration i, our objective is to have a configuration σ_i which is distributed very close to the Gibbs distribution on G_i. The idea is to generate σ_i by *updating* appropriately the configuration of G_{i-1}, i.e., update σ_{i-1} to generate efficiently σ_i. Once all the edges are put back, the algorithm outputs the configuration of G.

The algorithm in [15], relies heavily on properties that are special to graph colourings, for this reason it is restricted to this distribution. The same holds for similar approaches in the area, i.e., the algorithm is specific to the distribution it is sampling from. This is a drawback because every time we consider a new distribution we have to design a new algorithm from scratch. With our approach here, we avoid this. We introduce a sampling algorithm that the Gibbs distribution we are sampling from is a *parameter*, e.g., similarly to Glauber dynamics.

Previous approaches in the area rely on the *correlation decay* condition called *tree-uniquess* to establish the accuracy of the algorithm. For our purposes, requiring such a condition can be too restrictive. On one hand, for many of the distributions we consider here, we are far from establishing their tree uniqueness region. Actually, for many of them it is notoriously difficult to establish it even for a very limited rage of their parameters. On the other hand, it seems that Gibbs uniqueness is too restrictive a condition for distributions on the hypergraph. With our approach here we give a new insight to the problem by showing that we can exploit notions about the Gibbs distributions that we typically encounter in the study of the Cavity method and random CSPs. For example, we use notions like the *broadcasting probabilities* encountered in the study of the extremality of Gibbs distributions for random CSPs [7, 29, 16], or the *contiguity* between the Gibbs distribution and its corresponding *teacher-student model* for the study of the so-called free energy and its fluctuations [1, 11, 30, 8].

To appreciate our results, we remark that for most of the distributions we consider here, in order to prove that, e.g., the MCMC sampler works anywhere near the region of the parameters that our algorithm allows, would require major breakthroughs in the area of Markov chains, with repercussions that go beyond the problems we consider in this work.

The reader should not confuse the bounds we get here with those for "worst-case" instances. For worst case instances, usually, the parametrisation is w.r.t. the *maximum degree* of the underlying (hyper)graph, whereas for the random (hyper)graph, the natural parametrisation is w.r.t. the *expected degree*. Typically for the random (hyper)graphs here the maximum degree is *unbounded*, i.e., $\Theta(\log n/\log\log n)$, while the expected degree d is a *fixed* number.

It is important to mention that having an algorithm which allows a lot of freedom for the parameters comes at a certain cost. The approximation guarantees of our algorithm are not equally strong to, e,g., those of the MCMC approach. That is, even though the state of the art of MCMC can be far more restrictive in the parameters it allows, it provides stronger approximation guarantees. Roughly speaking, our results is summarised as follows: for a symmetric Gibbs distribution μ on the random (hyper)graph which satisfies our set of conditions, we have an approximation sampling algorithm such that with probability $1-o(1)$ over the instances of the input (hyper)graph, it generates a configuration which is distributed within total variation distance $n^{-\Omega(1)}$ from μ. The time complexity is $O((n\log n)^2)$.

Concluding, the idea of "iteratively adding edges and updating" turns out to be quite powerful, particularly when we combine it with notions and ideas from the Cavity method. It allows us to sample from distributions that, prior to this work, was impossible to sample. Our approach leads to, what we believe to be, an elegant sampling algorithm which deviates from [15] and follow up approaches not only on the phenomena of the Gibbs distributions that it utilises, but also on its basic description. Our work shows how powerful the notions

from the Cavity method can be, i.e., even in the context of sampling algorithms. We believe that there is a lot of potential towards the direction of using ideas from the Cavity method for the sampling problem in order to get even stronger algorithms.

Results for specific distributions appear in Section 1.2. Particularly, we show results for the anti-ferro q-state Potts model on graphs and hypergraphs, for any $q \geq 2$. This includes the "zero-temperature" case (hyper)graph colourings, as well as the anti-ferro Ising model. Also, we show results for the k-NAE-SAT, for $k \geq 2$, and the *spin-glass* called the k-spin model. This is the diluted version of the celebrated and extensively studied *Sherrington-Kirkpatrick* mean field model [31, 38]. Spin-glasses are studied in mathematics, in neural computation, computational biology and many other areas [37]. To our knowledge, this is the first rigorous analysis of efficient algorithm for spin-glasses. Our algorithm is by no means restricted to the cases presented here, i.e., it applies to *any* symmetric Gibbs distribution. We choose to present the specific ones because there is a common analysis framework that we can use.

1.1 General Overview

Let the fixed k-uniform hypergraph $H_k = (V, E)$. Clearly, the graph case corresponds to having $k = 2$. A Gibbs distribution on H_k is specified by the set of spins $\mathcal{A}$ and the weight functions $(\psi_e)_{e\in E}$ such that $\psi_e : \mathcal{A}^k \to \mathbb{R}_{\geq 0}$. The Gibbs distribution $\mu = \mu_H$ is on the set of configurations $\mathcal{A}^V$, i.e., the assignments of spins to the vertices of H_k, such that each $\sigma \in \mathcal{A}^V$ gets probability measure

$$\mu(\sigma) \propto \prod_{e\in E} \psi_e(\sigma(x_{e,1}), \sigma(x_{e,2}), \ldots, \sigma(x_{e,k})),$$

where $x_{e,i}$ is the i-th vertex in the hyperedge e. We assume a predefined order for the vertices in each hyperedge. The symbol $\propto$ stands for "proportional to".

In many situations, we allow ψ_e to vary with e. For example, in k-NAE-SAT, or the k-spin model each ψ_e is chosen independently, according to a predefined distribution. For this early exposition of the results the reader may very well assume that all ψ_e's are the same and fixed.

Roughly speaking, μ is symmetric, if for any $\sigma, \tau \in \mathcal{A}^V$ such that σ can be obtained from τ by repermuting the spin classes, we have that $\mu(\sigma) = \mu(\tau)$. For example, suppose that $\mathcal{A} = \{\pm 1\}$. If μ is symmetric, then we have $\mu(\sigma) = \mu(\tau)$ for any two $\sigma, \tau \in \mathcal{A}^V$, such that $\sigma(x) = -\tau(x)$ for all $x \in V$.

The underlying (hyper)graph structure is random. We let $\boldsymbol{H} = \boldsymbol{H}(n, m, k)$ be the random k-uniform hypergraph on n vertices and m hyperedges. For the graph case, i.e., $k = 2$, we write $\boldsymbol{G}(n, m)$. The expected degree is denoted by d. We take d to be a constant, i.e., $m = \Theta(n)$. Our results hold for any $d > 0$, i.e., we *do not* require that "d is sufficiently large".

Consider a typical instance of $\boldsymbol{H}$, of expected degree d, and $\mu = \mu_{\boldsymbol{H}}$ a symmetric Gibbs distribution on $\boldsymbol{H}$. Suppose that we want to sample from μ. In what follows, we describe the key features of the algorithm we propose for sampling from μ.

The main problem we need to deal with is the update-rule, i.e., how to design a method that gives us the configuration σ_i from σ_{i-1}, while, at the same time, it is generic enough to apply to all symmetric Gibbs distributions. We introduce an approach that relies on an abstract process that is called *broadcasting*. This is a natural process, that can be used to generate samples of symmetric Gibbs distributions on trees. The update starts by changing the configuration of the vertices that are disconnected and become connected when we add the (hyper) edge in the iteration i. W.l.o.g. assume that we change the assignment of just one of these vertices. Starting from this disagreeing vertex, we employ a process that is a

reminiscent of a *coupling* between two broadcasting processes. It is possible that in this process the initial disagreements propagates to some of its neighbours and, in turn, the disagreement of the neighbours propagates to neighbours further away and so on. The update stops when no more disagreements are generated.

It is crucial for the accuracy of the algorithm that the disagreements do not propagate too much further from the initial one. We can bound the rate that the disagreements spread during the update in terms of certain quantities related to the Gibbs distribution called *broadcasting probabilities*. Particularly, we have two desiderata: (a) the broadcasting probabilities are sufficiently close with each other (b) we need to show that the configuration of the vertices that the update rule encounters, somehow, looks like the result of a broadcasting process.

For the first desideratum we just need to tune appropriately the parameters of the Gibbs distribution. As far as the second one is concerned, even though we expect it to be true in our setting, it is very difficult to establish it rigorously. To this end, we employ the so-called planting-trick. This is a technique that allows us to circumvent the problem of accessing properties of the Gibbs distribution by using a very accurate approximation of it, which is called the *Teacher-Student* model. Working with the later distribution is technically much easier and it allows to get our second desideratum.

Let us be more specific. In order to have an accurate algorithm we introduce a set of technical conditions for the Gibbs distribution μ which we call `SET`. For describing `SET`, we need to visit few basic concepts.

Using the weight functions $(\psi_e)_{e\in E}$ we considered before, we introduce the following distributions: for each $e \in E$, let $\mathfrak{m}_e$ be the distribution on $\mathcal{A}^e$, i.e., configurations on the vertices in e, such that

$$\mathfrak{m}_e(\sigma) \propto \psi_e(\sigma(x_{e,1}), \sigma(x_{e,2}), \ldots, \sigma(x_{e,k})) \qquad \forall \sigma \in \mathcal{A}^e. \tag{1}$$

From Cavity's perspective, the distributions $\{\mathfrak{m}_e\}_{e\in E}$ can be viewed as *fixed-points* of the so-called *BP equations*. The distributions $\mathfrak{m}_e$ are natural objects in our setting. Particularly we focus on the so-called *broadcasting probability* $\mathfrak{m}_e^i$, which is the distribution $\mathfrak{m}_e$ conditional on the configuration at $x_{e,1}$ being $i \in \mathcal{A}$.

Furthermore, we need to introduce the notion of *total variation distance* between distributions. Particularly, for any two distributions $\hat{\nu}$ and ν on $\mathcal{A}^V$ we let

$$||\hat{\nu} - \nu||_{tv} = (1/2) \sum_{\sigma\in\mathcal{A}^V} |\hat{\nu}(\sigma) - \nu(\sigma)|.$$

Also, we let $||\hat{\nu} - \nu||_\Lambda$ be the total variation distance of the marginals of $\hat{\nu}$ and ν on the vertex set $\Lambda \subseteq V$.

`SET` consists of two conditions which we call **B.1** and **B.2**. The condition **B.1** requires that for any two $\mathfrak{m}_e^i$, $\mathfrak{m}_e^j$, i.e, any two broadcasting probabilities of μ, their total variation distance is not too large. We say that **B.1** is satisfied with slack $\delta > 0$ if we have that

$$\max_{i,j\in\mathcal{A}} ||\mathfrak{m}_e^i - \mathfrak{m}_e^j||_\Lambda \leq \frac{1-\delta}{d(k-1)}, \qquad \text{where } \Lambda = \{x_{e,2}, x_{e,3}, \ldots, x_{e,k}\}.$$

Recall that k is the size of the hyper-edge in $\boldsymbol{H}$, while d is the expected degree. The above implies that for any two broadcasting, their total variation distance should be smaller than $(1-\delta)$ over the expected number of neighbours of a vertex in $\boldsymbol{H}$.

The condition **B.2** requires mutual *contiguity* between the Gibbs distribution μ and the so-called teacher-student model. We generate the pair $(\boldsymbol{H}^*, \boldsymbol{\sigma}^*)$ according to the teacher-student model by working as follows: choose $\boldsymbol{\sigma}^*$ randomly from $\mathcal{A}^V$. Given $\boldsymbol{\sigma}^*$, generate the

weighted random hypergraph $\boldsymbol{H}^*$ on n vertices and m edges, where the weight depends on $\boldsymbol{\sigma}^*$ and μ. Contiguity implies that the *typical properties* of the pair $(\boldsymbol{H}^*, \boldsymbol{\sigma}^*)$ are the same as those of the pair $(\boldsymbol{H}, \boldsymbol{\sigma})$, where $\boldsymbol{H} = \boldsymbol{H}(n, m, k)$ and $\boldsymbol{\sigma}$ is distributed as in μ. More formally, contiguity implies that for any sequence of events $(\mathcal{S}_n)_n$ we have that

$$\Pr[(\boldsymbol{H}, \boldsymbol{\sigma}) \in \mathcal{S}_n] = o(1) \qquad \text{iff} \qquad \Pr[(\boldsymbol{H}^*, \boldsymbol{\sigma}^*) \in \mathcal{S}_n] = o(1).$$

For the planting-trick we combine **B.1** with **B.2**. Roughly speaking, we use **B.1** to prove that the update rule is local when it is applied to a configuration that is from the teacher-student model. Then, **B.2**, i.e., contiguity, implies that the same is true for our original configuration σ_{i-1}. The reader can find further details and discussions about this in Section 2.

The region of the parameters that our algorithm is accurate are specified by SET. Employing technical arguments from [11, 8, 10], we show that, for symmetric distributions, the condition **B.1** is stronger than **B.2**. In that respect, the specifications of SET reflect exactly the restrictions that **B.1** imposes to the Gibbs distribution.

The general result we prove in this work is stated in the following two theorems. Also, see Section 1.2 for applications of these theorems on specific distributions.

▶ **Theorem 1.** *For $\delta \in (0, 1]$, for integer $k \geq 2$, for any $d \geq 1/(k-1)$ and integer $m = dn/k$ the following is true for our algorithm: Consider the random k-uniform hypergraph $\boldsymbol{H} = \boldsymbol{H}(n, m, k)$. Let $\mu = \mu_{\boldsymbol{H}}$ be a symmetric Gibbs distribution on $\boldsymbol{H}$ which satisfies* SET *with slack δ.*

With probability $1 - o(1)$, over the input instances $\boldsymbol{H}$ and weight functions on the edges of $\boldsymbol{H}$, our algorithm generates a configuration whose distribution $\bar{\mu}$ is such that

$$||\bar{\mu} - \mu||_{tv} \leq n^{-\frac{\delta}{55 \log(dk)}}.$$

As mentioned above, the theorem does not require d to be a "sufficiently large constant". We chose $d \geq 1/(k-1)$, because otherwise the underlying graph structure is very simple and the problem is trivial.

Let us remark that, even though the output error for the algorithm is polynomial small, we didn't optimise the constants at the exponent of the polynomial.

▶ **Theorem 2.** *For $k \geq 2$ and $d \geq 1/(k-1)$ and integer $m = dn/k$, consider the random k-uniform hypergraph $\boldsymbol{H} = \boldsymbol{H}(n, m, k)$. The time complexity of our algorithm on input $\boldsymbol{H}$ is $O((n \log n)^2)$.*

SET Versus Gibbs Uniqueness

As noted earlier, for almost all the cases of distributions we consider here, our algorithm outperforms by far the corresponding MCMC. However, it is natural to further characterise the region of parameters that our algorithm allows, not in terms of the performance of other algorithms, but in terms of *spatial mixing* conditions on the the k-uniform random hyper-tree where each non-leaf vertex has $(k-1) \times$ Poisson(d) children and $k \geq 2$. If not anything else, this could give further insight on the approach.

Establishing the so-called tree-uniqueness for many of the Gibbs distributions here turns out to be a notoriously difficult problem even for the regular tree, not to mention the *random* tree, or the *random* hyper-tree. Deriving such results is of independent interest and goes far beyond the scope of this work. The lack of rigorous result for uniqueness allows only for a discussion on the basis of conjectures coming (mainly) from physics.

For the graph case, it is natural to compare SET with the Gibbs tree-uniqueness condition on the Galton-Watson tree with offspring distribution the Poisson(d). At least for the antiferromagnetic Ising and Potts model and the 2-NAE-SAT model, the requirement of our algorithm coincides with the conjectured tree uniqueness region of the the random tree. We are not aware of any conjectures about the tree uniqueness of the spin-glass 2-spin model.

For the hyper-graph case, things seem to be more interesting. The author of this work is not aware of any physics' conjectures about the Gibbs uniqueness on the random hyper-tree. However, we believe that is interesting to include in our discussion the following (somehow easy to make) observation: For the sake of our discussion, rather than random hyper-tree consider a regular one, e.g. consider k-uniform hyper-tree where each non-leaf vertex has $\Delta(k-1)$ children, for integers $\Delta, k > 0$. Consider also a symmetric Gibbs distribution with hard constraints on this hyper-tree, e.g. the q-colouring model. Provided that Δ, k are relatively large, say $\Delta, k \geq 15$, the condition SET does not preclude configurations at the vertices at level ℓ of the tree that specify uniquely the colouring at the root, for howsoever large $\ell > 0$ we choose [1]. Of course, such colourings are *extremely untypical* with respect to the Gibbs distribution on the hyper-tree. A very similar phenomenon can be observed at the random hyper-tree, too. This aforementioned phenomenon leads us to *conjecture* for the hypergraph cases, that our algorithm allows to sample beyond the corresponding hyper-tree uniqueness region for *all* the Gibbs distributions we consider here.

The above should not be a surprise to the reader. It is well-known that for hard-constraint distributions on the hyper-graph (not necessarily random), the worst case configurations can be very problematic for the analysis of MCMC sampler, i.e., even under very mild conditions, e.g. see [22]. Our algorithm does not suffer from such problems, because it deals with typical configurations which are much nicer.

Related work

The idea of "iteratively adding edges and updating" for sampling was first introduced in [15] for sampling colourings of random graphs. The techniques and tools we introduce here for the sampling problem, rely on results developed in the study Cavity method and random CSP, e.g. see [1, 8, 10, 11].

There are two other works which follow the approach of "adding edges and updating" and use the same correlation decay approach to [15]. One is [17], an improvement of [15], which is about colourings of the random graph of sufficiently large expected degree d. The other one is [4] for the Potts model on the related random Δ-regular graph, for large Δ. From the second paper, it is conceivable that we can get an efficient algorithm only for the ferromagnetic Potts model on the random graph, provided that the expected degree d is large. Apart from colourings and ferro-Potts on the graph, we cannot rely on any of these two approaches for our endeavours. Both of them rely on the special properties of the distribution they are sampling from, thus they don't allow for other distributions. Furthermore, their tree-uniqueness requirement essentially restrict their use to considering only graphs, rather than hypergraphs. Our work here improves on both results in [17, 4] as it allows for any expected degree $d > 0$, i.e., rather than sufficiently large d.

[1] E.g., in this setting SET allows $q \approx ((k-1)\Delta)^{1/(k-1)}$, while the number of children of a non-leaf vertex is $(k-1)\Delta \gg q$.

There are other approaches to sampling from Gibbs distributions (not-necessarily on random graphs), which is different than the one we consider here. Notably, the most popular ones rely on the Markov Chain Monte Carlo Method (MCMC) [28, 21]. The literature of MCMC algorithms is vast and includes some beautiful results, just to mention a few [3, 39, 27, 25, 35, 14, 18, 6, 13].

Our results about the colourings are related to the work in [18] for MCMC sampling. Some result from [18] can be extended naturally to the Potts model, too. In that respect our approach outperforms, by far, [18] it terms of the range of the allowed parameters of the Gibbs distributions. However, we note that the MCMC algorithm achieves better approximation guarantees in the (more restricted) regions it operates.

1.2 Applications

1.2.1 The antiferromagnetic Ising Model

The Ising model on the k-uniform hypergraph $H_k = (V, E)$ is a distribution on the set of configurations $\{\pm 1\}^V$ such that each $\sigma \in \{\pm 1\}^V$ is assigned probability measure

$$\mu(\sigma) \propto \exp\left(\beta \cdot \sum_{e\in E} \prod_{x,y\in e} \mathbf{1}\{\sigma(x) = \sigma(y)\} + h \cdot \sum_{x\in V} \sigma(x)\right),$$

where $\beta \in \mathbb{R}$ is the *inverse temperature* and h is the *external field*. It is straightforward that the Ising model is symmetric only when $h = 0$. We assume $\beta < 0$, which corresponds to the *antiferromagnetic* Ising model.

For $\Delta, k > 0$ such that $\Delta > \frac{2^{k-1}-1}{k-1}$ we let the function

$$\beta_{\text{Ising}}(\Delta, k) = \log\left(\frac{\Delta(k-1)+1-2^{k-1}}{\Delta(k-1)+1}\right).$$

The uniqueness region of the antiferromagnetic Ising model on the Δ-ary tree, for $\Delta \geq 1$, is well-known. Particularly, it corresponds to temperatures β such that either $\Delta > 2$ and $\beta_{\text{Ising}}(\Delta, 2) < \beta < 0$, or $\Delta = 1$ and finite $\beta < 0$, i.e., $\beta \neq -\infty$.

▶ **Theorem 3.** *For integer $k \geq 2$ fr any $d \geq 1/(k-1)$ such that either*
1. *$d > (2^{k-1}-1)/(k-1)$ and $\beta_{\text{Ising}}(d, k) < \beta < 0$, or*
2. *$d \leq (2^{k-1}-1)/(k-1)$ and finite $\beta < 0$,*

the following is true: For the random k-uniform hypergraph $\boldsymbol{H} = \boldsymbol{H}(n, m, k)$, where $m = dn/k$, let $\mu = \mu_{\boldsymbol{H}}$ be the antiferromagnetic Ising model on $\boldsymbol{H}$, with inverse temperature β and external field $h = 0$.

There exists $c_0 > 0$ which depends only on the choice of k, d, β, such that with probability $1 - o(1)$ over the input instances $\boldsymbol{H}$, our algorithm generates a configuration with distribution $\bar{\mu}$ such that

$$||\bar{\mu} - \mu||_{tv} \leq n^{-\frac{c_0}{55\log(dk)}}.$$

The time complexity of the algorithm is $O((n \log n)^2)$ with probability 1.

▶ Remark 4. For the graph cases, i.e., $k = 2$, physics' conjecture is that we get the Gibbs uniqueness region for the random trees with expected offspring d, by somehow "pretending" that we are dealing with a d-ary tree (regardless of d being an integer, or not). From the above theorem, it is evident that the region that our algorithm operates, corresponds exactly to the region that is implied by the physics' non rigorous consideration. Someone could observe the same for the related Potts model in the following section.

1.2.2 The antiferromagnetic Potts Model and the Colouring Model

The Potts model on the k uniform hypergraph $H_k = (V, E)$ is a generalisation of the Ising model in the sense that it allows for q spins where $q \geq 2$. Particularly, each $\sigma \in [q]^V$, where $[q] = \{1, 2, \ldots, q\}$, is assigned probability measure

$$\mu(\sigma) \propto \exp\left(\beta \cdot \sum_{e \in E} \prod_{x,y \in e} \mathbf{1}\{\sigma(x) = \sigma(y)\}\right).$$

where $\beta \in \mathbb{R}$ is the *inverse temperature*. The graph case, i.e., $k = 2$, follows immediately from the above.

There is a version of the Potts model with external field, similarly to the Ising model. We do not consider the case with field because it gives rise to a non symmetric distribution. The antiferromagnetic Potts model we focus here corresponds to having $\beta < 0$.

A very interesting case of the Potts model is the *colouring model*. This is the uniform distribution over the *proper* q-colourings of the underlying (hyper)graph H_k, i.e., we do not allow configurations with monochromatic edges. The colouring model corresponds to the Potts model with $\beta = -\infty$.

For $\Delta, k > 0$ such that $\Delta > \frac{q^{k-1}-1}{k-1}$, we let the function

$$\beta_{\text{Potts}}(\Delta, q, k) = \log\left(\frac{\Delta(k-1)+1-q^{k-1}}{\Delta(k-1)+1}\right).$$

▶ **Theorem 5.** *For integer $k \geq 2$, for any $d \geq 1/(k-1)$, integers $q > 2$ and $m = dn/k$ the following is true: Assume that β, q satisfy one of the following cases:*

1. *$(q^{k-1} - 1)/(k-1) < d$ and $\beta_{\text{Potts}}(d, q, k) < \beta < 0$,*
2. *$(q^{k-1} - 1)/(k-1) > d$ and $\beta < 0$, including $\beta = -\infty$.*

Consider the random k-uniform hypergraph $\boldsymbol{H} = \boldsymbol{H}(n, m, k)$. Let $\mu = \mu_{\boldsymbol{G}}$ be the q-state antiferromagnetic Potts model on $\boldsymbol{H}$ with inverse temperature β. There exists $c_0 > 0$, which depends only on our choices of k, d, β such that with probability $1 - o(1)$ over the input instances $\boldsymbol{H}$, our algorithm generates a configuration whose distribution $\bar{\mu}$ is such that

$$||\bar{\mu} - \mu||_{tv} \leq n^{-\frac{c_0}{55\log(dk)}}.$$

The time complexity of the algorithm is $O((n \log n)^2)$ with probability 1.

Physics' (folklore) conjecture for the uniqueness of antiferromagnetic Potts model on the Galton Watson tree with offspring distribution Poisson(d) coincides with the region specified in the theorem above, for $k = 2$.

1.2.3 The k-NAE-SAT Model

For integer $k \geq 2$, let $\mathbf{F}_k(n, m)$ be a random propositional formula over the Boolean variables $x_1, \ldots, x_n$. Particularly, $\mathbf{F}_k(n, m)$ is obtained by inserting m independent random clauses of length k such that no variable appears twice in the same clause. Here, we consider formulas with $m = dn/k$ clauses for a fixed number d, i.e., on average every variable occurs in d clauses.

We focus on the "Not-All-Equal" satisfying assignments of $\mathbf{F}_k(n, m)$. A Boolean assignment σ of $x_1, \ldots, x_n$ is NAE-satisfying for $\mathbf{F}_k(n, m)$ if under both σ and its binary inverse $\bar{\sigma}$ all m clauses evaluate to "true". The random k-NAE-SAT problem is one of the standard examples of random CSPs and has received a great deal of attention. In particular, in an influential paper Achlioptas and Moore [2] pioneered the use of the second moment method for estimating the partition functions of random CSPs with the example of random k-NAE-SAT. Our focus is on sampling from the *uniform distribution* over the NAE satisyfing assignments of $\mathbf{F}_k(n, m)$. We have the following result.

▶ **Theorem 6.** *For $\delta \in (0,1]$, for $k \geq 2$, for any $1/(k-1) \leq d < (1-\delta)\frac{2^{k-1}-1}{k-1}$ and for integer $m = dn/k$, the following is true for our algorithm: Consider $\mathbf{F}_k(n,m)$ and let μ be the uniform distribution over the NAE satisfying assignments of $\mathbf{F}_k(n,m)$. With probability $1-o(1)$ over the input instances $\mathbf{F}_k(n,m)$, our algorithm generates a configuration whose distribution $\bar{\mu}$ is such that*

$$||\bar{\mu} - \mu||_{tv} \leq n^{-\frac{\delta}{55\log(dk)}}.$$

The time complexity of the algorithm is $O((n\log n)^2)$ with probability 1.

As a point of reference for the performance of our *sampling* algorithm, note that it works in a region of parameters which is very close to those of *search* algorithms for the problem, e.g. see the analysis for the renown *walk-sat* algorithm in [9].

1.2.4 The *k*-spin model

For integer $k \geq 2$, consider the k-uniform hypergraph $H_k = (V,E)$. Additionally, let $\mathbf{J} = (\mathbf{J}_e)_{e\in E}$ be a family of independent, standard Gaussians (expectation zero and variance one). The *k-spin model* on H_k at inverse temperature $\beta > 0$ is the distribution that assigns each configuration $\sigma \in \{\pm 1\}^V$ the probability measure

$$\mu(\sigma) \propto \prod_{\alpha\in E} \exp\left(\beta \mathbf{J}_e \prod_{y\in\alpha}\sigma(y)\right). \tag{2}$$

It is elementary to verify that the k-spin model is symmetric when $k \geq 2$ is an even integer. Here we consider the above distribution when the underlying (hyper)graph is an instance of $\boldsymbol{H} = \boldsymbol{H}(n,m,k)$ of expected degree d, i.e., $m = dn/k$.

The k-spin model is the diluted version of the well-known Sherrington-Kirkpatrick model which has been subject of intense study both in mathematics and physics [38, 31]. Generally, spin-glasses give rise to very intricate distributions and they are also studied in neural networks, computational biology and other areas of computer science [37]. As mentioned before, this is the first efficient algorithm for sampling from spin-glasses in a non-trivial region of their parameters. For what follows we consider the function

$$F_k(x) = \frac{|e^x - e^{-x}|}{(2^{k-1}-1)e^{-x} + e^x}. \tag{3}$$

▶ **Theorem 7.** *For $\delta \in (0,1]$, for even integer $k \geq 2$, for any $d \geq 1/(k-1)$ and for any $\beta \geq 0$ such that*

$$\mathbb{E}[F_k(\beta \mathbf{J}_0)] \leq \tfrac{1-\delta}{d(k-1)},$$

where the expectation is w.r.t. the standard Gaussian random variable $\mathbf{J}_0$, the following is true for our algorithm: Consider $\boldsymbol{H} = \boldsymbol{H}(n,m,k)$, where $m = dn/k$, and let μ be the k-spin model on $\boldsymbol{H}$ at inverse temperature β. With probability $1-o(1)$ over the input instances $\boldsymbol{H}$ and the weight functions on the edges of $\boldsymbol{H}$, our algorithm generates a configuration whose distribution $\bar{\mu}$ is such that

$$||\bar{\mu} - \mu||_{tv} \leq n^{-\frac{\delta}{55\log(dk)}}.$$

The time complexity of the algorithm is $O((n\log n)^2)$ with probability 1.

Notation

Let the hypegraph $H_k = (V, E)$ and the Gibbs distribution μ on the set of configurations $\mathcal{A}^V$. For a configuration σ, we let $\sigma(\Lambda)$ denote the configuration that σ specifies on the set of vertices Λ. We let μ_Λ denote the marginal of μ at the set Λ. For a configuration $\sigma \in \mathcal{A}^V$ we let $\mu(\cdot \mid \Lambda, \sigma)$, denote the distribution μ conditional on the configuration at Λ being $\sigma(\Lambda)$. Also, we interpret the conditional marginal $\mu_\Lambda(\cdot \mid \Lambda', \sigma)$, for $\Lambda' \subseteq V$, in the natural way.

2 Algorithmic Approach – High Level Description

To facilitate the exposition of the algorithm assume in this section that we are dealing with a graph, rather than hypergraph, while we assume that this graph is fixed.

First, we recall the algorithm: on input G, the algorithm initially removes all the edges and generates a configuration for the empty graph. Then, iteratively, we put the edges back one by one. If G_i is the subgraph we have at iteration i, our aim is to have a configuration σ_i which is distributed very close to the Gibbs distribution on G_i, for every i. We generate σ_i by updating appropriately σ_{i-1}, the configuration of G_{i-1}. Once all edges are put back, the algorithm outputs the configuration of G. One of the main challenge is to specify the *update rule*, i.e., how to generate σ_i from σ_{i-1}.

We describe the proposed rule by considering the following, simpler, problem. Consider two *high-girth*, fixed, graphs $G = (V, E)$ and $G' = (V, E')$. Assume that G and G' differ on a single edge, i.e. compared to G, the graph G' has the extra edge $e = \{u, w\}$. Let μ and μ' be the Gibbs distributions of G and G', respectively. We want to use the update rule to generate efficiently $\boldsymbol{\tau}$ a sample from μ', while we are given $\boldsymbol{\sigma}$, a sample from μ.

To facilitate our exposition, assume that we already know $\boldsymbol{\tau}(u)$ and $\boldsymbol{\tau}(w)$, and they are such that $\boldsymbol{\tau}(u) = \boldsymbol{\sigma}(u)$ and $\boldsymbol{\tau}(w) \neq \boldsymbol{\sigma}(w)$. In what follows, we focus on specifying $\boldsymbol{\tau}$ for the rest of the vertices in V.

The plan is to visit each vertex z *iteratively* and specify $\boldsymbol{\tau}(z)$. At each iteration t, we only know the configuration of $\boldsymbol{\tau}$ for the vertices in the set of vertices we have already visited, we call it $\mathcal{N}_t$. Initially we have that $\mathcal{N}_0 = \{w, u\}$. Also, let $\mathcal{D} = \{\boldsymbol{\tau}(w), \boldsymbol{\sigma}(w)\}$, i.e., $\mathcal{D}$ is the set of the spins of the initial disagreement. At iteration t we pick a vertex z which is outside $\mathcal{N}_t$ but has a neighbour $x \in \mathcal{N}_t$ which is disagreeing, i.e., $\boldsymbol{\tau}(x) \neq \boldsymbol{\sigma}(x)$. For the moment, assume that such vertex exists.

If $\boldsymbol{\sigma}(z) \notin \mathcal{D}$, then we just set $\boldsymbol{\tau}(z) = \boldsymbol{\sigma}(z)$. On the other hand, if $\boldsymbol{\sigma}(z) \in \mathcal{D}$, then we work as follows: there is a probability p_z, that depends on the configuration of $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ at $\mathcal{N}_t$, and we set

$$\boldsymbol{\tau}(z) = \begin{cases} \mathcal{D} \setminus \{\boldsymbol{\sigma}(z)\} & \text{with prob. } p_z \\ \boldsymbol{\sigma}(z) & \text{with prob. } 1 - p_z. \end{cases}$$

The first line implies that $\boldsymbol{\tau}(z)$ gets the opposite spin of $\boldsymbol{\sigma}(z)$. E.g., if $\mathcal{D} = \{\texttt{red}, \texttt{blue}\}$ and $\boldsymbol{\sigma}(z) = \texttt{red}$, then $\boldsymbol{\tau}(z) = \texttt{blue}$. Once $\boldsymbol{\tau}(z)$ is decided, set $\mathcal{N}_{t+1} = \mathcal{N}_t \cup \{z\}$ and continue with the next iteration.

It could be that in iteration t, there is no vertex z outside $\mathcal{N}_t$ which has a disagreeing neighbour inside $\mathcal{N}_t$. If this is the case, then for every z for which we have not specified $\boldsymbol{\tau}(z)$, we set $\boldsymbol{\tau}(z) = \boldsymbol{\sigma}(z)$. Once we have specified the assignment $\boldsymbol{\tau}$ for every vertex z in the graph, the update rule terminates.

The probability p_z is determined in terms of a *maximal coupling* between the marginals of μ' and μ at z, conditional on $\boldsymbol{\tau}(\mathcal{N}_t)$ and $\boldsymbol{\sigma}(\mathcal{N}_t)$. We denote these marginals as $\mu'_z(\cdot \mid \mathcal{N}_t, \boldsymbol{\tau})$ and $\mu_z(\cdot \mid \mathcal{N}_t, \boldsymbol{\sigma})$, respectively. We have

$$p_z = \max\left\{0, 1 - \frac{\mu'_z(\boldsymbol{\sigma}(z) \mid \mathcal{N}_t, \boldsymbol{\tau})}{\mu_z(\boldsymbol{\sigma}(z) \mid \mathcal{N}_t, \boldsymbol{\sigma})}\right\}.$$

One can show that the above generates a *perfect* sample from the distribution μ'. There is an issue with this approach, though. It is not clear how we can compute the probabilities p_z, efficiently. Computing p_z relies on estimating conditional marginals of Gibbs distributions. In our setting, we don't known how to estimate these marginals efficiently. To this end, we use *different* probabilities. That is, we follow the previous steps and when at the iteration t we examine a vertex z for which $\boldsymbol{\sigma}(z) \in \mathcal{D}$, we set $\boldsymbol{\tau}(z)$ such that

$$\boldsymbol{\tau}(z) = \begin{cases} \mathcal{D} \setminus \{\boldsymbol{\sigma}(z)\} & \text{with prob. } q_z \\ \boldsymbol{\sigma}(z) & \text{with prob. } 1 - q_z, \end{cases} \tag{4}$$

i.e., instead of p_z we use q_z. Recall that we choose z because it has a disagreeing neighbour $x \in \mathcal{N}_t$. Each q_z can be expressed in terms of the simpler distribution $\mathfrak{m}_\alpha$, where α is the edge between z and x. We have

$$q_z = \max\left\{0,\ 1 - \frac{\mathfrak{m}_{\alpha,z}(\boldsymbol{\sigma}(z) \mid x, \boldsymbol{\tau})}{\mathfrak{m}_{\alpha,z}(\boldsymbol{\sigma}(z) \mid x, \boldsymbol{\sigma})}\right\}. \tag{5}$$

Recall from our notation that $\mathfrak{m}_{\alpha,z}(\cdot \mid x, \boldsymbol{\tau})$ is the marginal of $\mathfrak{m}_\alpha$ on z, conditional on x being set $\boldsymbol{\tau}(x)$. Also, note from (1) that the distribution $\mathfrak{m}_\alpha$ is very simple and can be computed very fast.

A natural question at this point is what motivates the use q_z in the place of p_z. We observe that if our graphs G and G' were trees, then we would have that $q_z = p_z$. That is, for trees our update rule generates perfect samples from μ'. In some sense, our approach amounts to approximating the probabilities p_z, which are difficult to compute, with those of the tree, which we can compute very fast. In light of our assumption that our graphs G and G' are of high-girth, i.e., locally tree-like, this approximation seems quite natural.

Under certain conditions, our approach yields very good approximations of μ'. The update rule is accurate in settings where, typically, the set of vertices that change assignment does not grow "too large". To be more specific, let $\mathcal{Q}$ be the set of vertices that change configuration during the update, i.e., their configuration under $\boldsymbol{\tau}$ is different than that under $\boldsymbol{\sigma}$. Somehow, our update rule runs into trouble when $\mathcal{Q}$ induces a subgraph which contains one of the long cycles of G, or $\mathcal{Q}$ reaches u. In this case we consider that the algorithm *fails*. That is, our update rule outputs either a configuration $\boldsymbol{\tau} \in \mathcal{A}^V$, or a fail status. We establish a connection between the accuracy of the update and its failure probability, particularly we show that the smaller the failure probability the more accurate the algorithm is.

2.1 Accuracy and failure probabilities

We relate the approximation error of the update rule with failure probabilities by exploiting an interesting property of the update rule which is a reminiscent of the *detailed balance equation* from the theory of reversible Markov chains [36]. In what follows, first we describe the "detailed balance property" of the update and then we show how we use it to study the accuracy.

In the same setting as before, assume that $\boldsymbol{\sigma}(\{u, w\}) = \sigma$ and $\boldsymbol{\tau}(\{u, w\}) = \tau$, for fixed $\sigma, \tau \in \mathcal{A}^{\{u,w\}}$. The update rule can be viewed as a stochastic process that takes a configuration that agrees with σ at $\{u, w\}$ and generates either a new configuration which

agrees with τ at $\{u, w\}$, or fails. There is a natural way of defining the *reverse update* which works towards the opposite direction, i.e., takes a configuration which agrees with τ and either generates a configuration that agrees with σ at $\{u, w\}$, or fails.

For any two configuration $\kappa, \eta \in \mathcal{A}^V$, we let $\mathrm{P}_{\sigma,\tau}(\kappa, \eta)$ be the probability that on input κ the update generates η. Similarly we can define $\mathrm{P}_{\tau,\sigma}(\eta, \kappa)$ for the reverse update. The detailed balance equation relates these two probabilities, i.e., it specifies that

$$\mu(\kappa)\mathrm{P}_{\sigma,\tau}(\kappa, \eta) = \mu(\eta)\mathrm{P}_{\tau,\sigma}(\eta, \kappa).$$

Note that, in the equation above, the Gibbs distributions μ are unconditional.

We proceed by demonstrating how we use the detailed balance to get the update error. Consider the same setting as in the previous paragraphs. Let $\bar{\mu}$ be the distribution of the output of the update when the input is distributed as in $\mu(\cdot \mid \{u, w\}, \sigma)$.

We need to focus on the failure probability of both the update and the reverse update. Let $\mathtt{F}(\kappa)$ be the failure probability of the update rule on input (fixed) κ. In that respect, the failure probability is equal to $\mathbb{E}[\mathtt{F}(\boldsymbol{\kappa})]$, where the expectation is w.r.t. to $\boldsymbol{\kappa}$ which is distributed as in $\mu(\cdot \mid \{w, u\}, \sigma)$. Similarly, let $\mathtt{R}(\eta)$ be the failure probability for the reverse update on input (fixed) η. The failure probability of the reverse update is $\mathbb{E}[\mathtt{R}(\boldsymbol{\eta})]$, where $\boldsymbol{\eta}$ is distributed as in $\mu(\cdot \mid \{w, u\}, \tau)$.

Using the detail balance and an asymptotic independence result between the configuration of w and u under μ, we get the following: For any $\eta \in \mathcal{A}^V$ we have that

$$\begin{aligned}\bar{\mu}(\eta) &= \sum_{\kappa \in \mathcal{A}^V} \mu(\kappa \mid \{w, v\}, \sigma)\mathrm{P}_{\sigma,\tau}(\kappa, \eta) \approx \sum_{\kappa \in \mathcal{A}^V} \mu(\eta \mid \{w, v\}, \tau)\mathrm{P}_{\tau,\sigma}(\eta, \kappa) \\ &= \mu(\eta \mid \{w, v\}, \tau)(1 - \mathtt{R}(\eta)). \end{aligned} \tag{6}$$

The first equation is just the definition of $\bar{\mu}(\eta)$. The detailed balance with the asymptotic independence are used for the derivation with "$\approx$". The last equation follows from the observation that summing $\mathrm{P}_{\tau,\sigma}(\eta, \kappa)$ over $\kappa \in \mathcal{A}^V$ is equal to the probability that the reverse update does not fail, when the input is η.

Another observation that we use is the following one: if the update has a positive failure probability, then $\sum_\eta \bar{\mu}(\eta) < 1$. This holds because $\bar{\mu}$ gives positive measure to the failure status of the update. That is, we have that $\mathbb{E}[\mathtt{F}(\boldsymbol{\kappa})] + \sum_\eta \bar{\mu}(\eta) = 1$. Combining this equality and (6), standard derivations imply

$$||\bar{\mu} - \mu(\cdot \mid \{w, u\}, \tau)||_{tv} \approx (1/2)(\mathbb{E}[\mathtt{R}(\boldsymbol{\eta})] + \mathbb{E}[\mathtt{F}(\boldsymbol{\kappa})]).$$

Essentially the update error is equal to the average of the failure probabilities of the update and its reverse.

2.2 The failure probability on the random graph

Here we highlight the intuition behind our claim that if `SET` holds, then the failure probability is very small.

Consider a setting which is a bit simpler than that we had before. Let μ be a symmetric Gibbs distribution on $\boldsymbol{G} = \boldsymbol{G}(n, m)$. Consider the spins $c, c' \in \mathcal{A}$ such that $c \neq c'$. Let $\boldsymbol{\sigma}$ be distributed as in μ conditional on $\boldsymbol{\sigma}(u) = c'$, for some $u \in V$. We use the update process to generate a configuration $\boldsymbol{\tau}$ which is (approximately) distributed as in μ conditional on that $\boldsymbol{\tau}(u) = c$. Our focus is on the probability of failure for the update. Particularly, we argue that if `SET` holds, then the size of the set of disagreeing vertices in the update process, i.e, the vertices v such that $\boldsymbol{\sigma}(v) \neq \boldsymbol{\tau}(v)$, grows *subcritically* at each iteration.

Recall that, in the update, the disagreements start from vertex u and iteratively propagate over the graph. Assume that not too many vertices have been visited in the process. In iteration t, the process chooses the vertex z which is adjacent to the disagreeing vertex x, i.e., we already know that $\boldsymbol{\tau}(x) \neq \boldsymbol{\sigma}(x)$. The probability of disagreement propagating to z can be estimated by just using (4) and (5). The idea is to combine (4) and (5) with the randomness of $\boldsymbol{\sigma}$ and show that the probability of disagreement at z is $< 1/d$, which would imply the subcriticality for the update process.

Exploiting the randomness of $\boldsymbol{\sigma}$ means that at iteration t of the process we only have exposed the configuration of $\boldsymbol{\sigma}$ for the vertices which the process has already visited. If the process hasn't revealed the configuration of $\boldsymbol{\sigma}$ for too many vertices, then the marginal of the configuration at z should be close to $\mathfrak{m}_e(\cdot \mid x, \boldsymbol{\sigma})$, where $e = \{x, z\}$. This would imply that the disagreement probability at z is at most

$$\max_{c,c' \in \mathcal{A}} ||\mathfrak{m}_e(\cdot \mid x, c) - \mathfrak{m}_e(\cdot \mid x, c')||_z.$$

The above observation is quite interesting because the quantity above, i.e., disagreement probability, is upper bounded by using **B.1** in SET. Particularly, **B.1** implies that the above total variation distance is $\leq (1-\delta)/d$. Thus, if the above intuition is correct, then we can get the subcritical growth by exploiting the condition **B.1**. Unfortunately, with our assumptions about μ, it too difficult to argue that the marginal probability at z is very close to $\mathfrak{m}_e(\cdot \mid x, \boldsymbol{\sigma})$ in our process.

To this end, we employ the teacher-student model. We consider the pair $(\boldsymbol{G}^*, \boldsymbol{\sigma}^*)$ from the teacher-student model. We study the propagation of disagreements for the update process on the pair $(\boldsymbol{G}^*, \boldsymbol{\sigma}^*)$. There, it is simpler to argue that the distribution of z is very close to $\mathfrak{m}_e(\cdot \mid x, \boldsymbol{\sigma})$. The condition **B.1** still applies here and it implies that the growth of disagreements in $\boldsymbol{G}^*$ is subcritical. In turn, this implies that the failure probability for the specific update is very small. Subsequently, we employ contiguity, i.e., **B.2**, to argue that if the probability of failure for the case of $(\boldsymbol{G}^*, \boldsymbol{\sigma}^*)$ is very small, then the probability of failure for $(\boldsymbol{G}, \boldsymbol{\sigma})$ cannot be much larger.

References

1. Dimitris Achlioptas and Amin Coja-Oghlan. Algorithmic barriers from phase transitions. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 793–802. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.11.
2. Dimitris Achlioptas and Cristopher Moore. Random k-sat: Two moments suffice to cross a sharp threshold. *SIAM J. Comput.*, 36(3):740–762, 2006. doi:10.1137/S0097539703434231.
3. Nima Anari, Kuikui Liu, and Shayan Oveis Gharan. Spectral independence in high-dimensional expanders and applications to the hardcore model. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1319–1330. IEEE, 2020. doi:10.1109/FOCS46700.2020.00125.
4. Antonio Blanca, Andreas Galanis, Leslie Ann Goldberg, Daniel Stefankovic, Eric Vigoda, and Kuan Yang. Sampling in uniqueness from the potts and random-cluster models on random regular graphs. *SIAM J. Discret. Math.*, 34(1):742–793, 2020. doi:10.1137/18M1219722.
5. Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Struct. Algorithms*, 27(2):201–226, 2005. doi:10.1002/rsa.20057.
6. Sitan Chen, Michelle Delcourt, Ankur Moitra, Guillem Perarnau, and Luke Postle. Improved bounds for randomly sampling colorings via linear programming. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2216–2234. SIAM, 2019. doi:10.1137/1.9781611975482.134.

7 Amin Coja-Oghlan, Charilaos Efthymiou, and Nor Jaafari. Local convergence of random graph colorings. *Combinatorica*, 38(2):341–380, 2018. doi:10.1007/s00493-016-3394-x.

8 Amin Coja-Oghlan, Charilaos Efthymiou, Nor Jaafari, Mihyun Kang, and Tobias Kapetanopoulos. Charting the replica symmetric phase. *Communications in Mathematical Physics*, 359(2):603–698, 2018. doi:10.1007/s00220-018-3096-x.

9 Amin Coja-Oghlan and Alan M. Frieze. Analyzing walksat on random formulas. *SIAM J. Comput.*, 43(4):1456–1485, 2014. doi:10.1137/12090191X.

10 Amin Coja-Oghlan, Tobias Kapetanopoulos, and Noëla Müller. The replica symmetric phase of random constraint satisfaction problems. *Comb. Probab. Comput.*, 29(3):346–422, 2020. doi:10.1017/S0963548319000440.

11 Amin Coja-Oghlan, Florent Krzakala, Will Perkins, and Lenka Zdeborová. Information-theoretic thresholds from the cavity method. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 146–157. ACM, 2017. doi:10.1145/3055399.3055420.

12 Jian Ding, Allan Sly, and Nike Sun. Proof of the satisfiability conjecture for large k. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 59–68. ACM, 2015. doi:10.1145/2746539.2746619.

13 Martin E. Dyer and Alan M. Frieze. Randomly coloring graphs with lower bounds on girth and maximum degree. *Random Struct. Algorithms*, 23(2):167–179, 2003. doi:10.1002/rsa.10087.

14 Martin E. Dyer, Alan M. Frieze, Thomas P. Hayes, and Eric Vigoda. Randomly coloring constant degree graphs. *Random Struct. Algorithms*, 43(2):181–200, 2013. doi:10.1002/rsa.20451.

15 Charilaos Efthymiou. A simple algorithm for random colouring $G(n, d/n)$ using $(2 + \epsilon)d$ colours. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 272–280. SIAM, 2012. doi:10.1137/1.9781611973099.25.

16 Charilaos Efthymiou. Reconstruction/non-reconstruction thresholds for colourings of general galton-watson trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 756–774. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.756.

17 Charilaos Efthymiou. A simple algorithm for sampling colorings of $G(n, d/n)$ up to the gibbs uniqueness threshold. *SIAM J. Comput.*, 45(6):2087–2116, 2016. doi:10.1137/140977643.

18 Charilaos Efthymiou, Thomas P. Hayes, Daniel Stefankovic, and Eric Vigoda. Sampling random colorings of sparse random graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1759–1771. SIAM, 2018. doi:10.1137/1.9781611975031.115.

19 Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 534–543. ACM, 2002. doi:10.1145/509907.509985.

20 Vitaly Feldman, Will Perkins, and Santosh S. Vempala. On the complexity of random satisfiability problems with planted solutions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 77–86. ACM, 2015. doi:10.1145/2746539.2746577.

21 Alan Frieze and Eric Vigoda. A survey on the use of markov chains to randomly sample colorings. In *Combinatorics, Complexity, and Chance*, pages 53–71. Oxford University Press, January 2007. doi:10.1093/acprof:oso/9780198571278.003.0004.

22 Alan M. Frieze and Michael Anastos. Randomly coloring simple hypergraphs with fewer colors. *Inf. Process. Lett.*, 126:39–42, 2017. doi:10.1016/j.ipl.2017.06.005.

23 Andreas Galanis, Daniel Stefankovic, and Eric Vigoda. Inapproximability for antiferromagnetic spin systems in the tree nonuniqueness region. *J. ACM*, 62(6):50:1–50:60, 2015. doi:10.1145/2785964.

24 Andrei Giurgiu, Nicolas Macris, and Rüdiger Urbanke. Spatial coupling as a proof technique and three applications. *IEEE Transactions on Information Theory*, 62(10):5281–5295, 2016. doi:10.1109/TIT.2016.2539144.

25 Leslie Ann Goldberg, Russell A. Martin, and Mike Paterson. Strong spatial mixing with fewer colors for lattice graphs. *SIAM J. Comput.*, 35(2):486–517, 2005. doi:10.1137/S0097539704445470.

26 Oded Goldreich. *Candidate One-Way Functions Based on Expander Graphs*, pages 76–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-22670-0_10.

27 Thomas P. Hayes, Juan Carlos Vera, and Eric Vigoda. Randomly coloring planar graphs with fewer colors than the maximum degree. *Random Struct. Algorithms*, 47(4):731–759, 2015. doi:10.1002/rsa.20560.

28 Mark Jerrum. *Counting, Sampling and Integrating: Algorithm and Complexity*. Lectures in Mathematics ETH. Birkhäuser Basel, 1st edition, 2003.

29 Florent Krzakala, Andrea Montanari, Federico Ricci-Tersenghi, Guilhem Semerjian, and Lenka Zdeborová. Gibbs states and the set of solutions of random constraint satisfaction problems. *Proceedings of the National Academy of Sciences*, 104(25):10318–10323, 2007. doi:10.1073/pnas.0703685104.

30 Florent Krzakala and Lenka Zdeborová. Hiding quiet solutions in random constraint satisfaction problems. *Phys. Rev. Lett.*, 102:238701, June 2009. doi:10.1103/PhysRevLett.102.238701.

31 M Mezard, G Parisi, and M Virasoro. *Spin Glass Theory and Beyond*. WORLD SCIENTIFIC, 1986. doi:10.1142/0271.

32 M. Mézard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297(5582):812–815, 2002. doi:10.1126/science.1073287.

33 Marc Mézard and Andrea Montanari. *Information, Physics, and Computation*. Oxford Graduate Texts. Oxford University Press, Oxford, 2009.

34 Michael Molloy. The freezing threshold for k-colourings of a random graph. *J. ACM*, 65(2):7:1–7:62, 2018. doi:10.1145/3034781.

35 Elchanan Mossel and Allan Sly. Exact thresholds for Ising–Gibbs samplers on general graphs. *The Annals of Probability*, 41(1):294–328, 2013. doi:10.1214/11-AOP737.

36 Sheldon M Ross. *Stochastic Processes*. John Wiley & Sons, Nashville, TN, 2 edition, January 1995.

37 Daniel L Stein and Charles M Newman. *Spin Glasses and Complexity*. Princeton University Press, January 2013.

38 Michel Talagrand. The Parisi formula. *Ann. Math. (2)*, 163(1):221–263, 2006. doi:10.4007/annals.2006.163.221.

39 Eric Vigoda. Improved bounds for sampling colorings. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 51–59. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814577.

Testability and Local Certification of Monotone Properties in Minor-Closed Classes

Louis Esperet
Univ. Grenoble Alpes, CNRS, Laboratoire G-SCOP, Grenoble, France

Sergey Norin
Department of Mathematics and Statistics, McGill University, Montreal, Canada

Abstract

The main problem in the area of graph property testing is to understand which graph properties are *testable*, which means that with constantly many queries to any input graph G, a tester can decide with good probability whether G satisfies the property, or is far from satisfying the property. Testable properties are well understood in the dense model and in the bounded degree model, but little is known in sparse graph classes when graphs are allowed to have unbounded degree. This is the setting of the *sparse model*.

We prove that for any proper minor-closed class $\mathcal{G}$, any monotone property (i.e., any property that is closed under taking subgraphs) is testable for graphs from $\mathcal{G}$ in the sparse model. This extends a result of Czumaj and Sohler (FOCS'19), who proved it for monotone properties with finitely many forbidden subgraphs. Our result implies for instance that for any integers k and t, k-colorability of K_t-minor free graphs is testable in the sparse model.

Elek recently proved that monotone properties of bounded degree graphs from minor-closed classes that are closed under disjoint union can be verified by an approximate proof labeling scheme in constant time. We show again that the assumption of bounded degree can be omitted in his result.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Property testing, sparse model, local certification, minor-closed classes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.58

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2202.00543`

Funding *Louis Esperet*: supported by the French ANR Projects GATO (ANR-16-CE40-0009-01), GrR (ANR-18-CE40-0032), TWIN-WIDTH (ANR-21-CE48-0014-01), and by LabEx PERSYVAL-lab (ANR-11-LABX-0025).
Sergey Norin: supported by an NSERC Discovery grant.

Acknowledgements This work was initiated during the Graph Theory workshop in Oberwolfach, Germany, in January 2022. The authors would like to thank the organizers and participants for all the discussions and nice atmosphere (and in particular Gwenaël Joret, Chun-Hung Liu, and Ken-ichi Kawarabayashi for the discussions related to the topic of this paper). The authors would also like to thank Gábor Elek for his remarks on an earlier version of this manuscript, and his suggestion to replace minor-closed classes by classes of bounded asymptotic dimension in Theorem 4.

1 Introduction

1.1 Property testing

We say that a graph G is *ε-far from some property $\mathcal{P}$* if one needs to modify at least $\varepsilon|E(G)|$ of its adjacencies (replacing edges by non-edges and vice-versa) in order to obtain a graph satisfying $\mathcal{P}$. A property is *testable* if for any graph G, a tester can decide with good probability whether G satisfies $\mathcal{P}$ or is ε-far from $\mathcal{P}$, by only making a constant number of

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 58; pp. 58:1–58:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

queries to a given representation of G (i.e., the number of queries depends only on ε and $\mathcal{P}$, but is independent of the input graph G). The tester has *one-sided error* if it always gives the correct answer when $G \in \mathcal{P}$, and *two-sided error* otherwise.

In the *dense graph model* [28], there is a good understanding of which properties are testable with two-sided error [3] and one-sided error [5, 4]. In *the bounded degree model* [29], a sequence of papers [8, 15, 32] culminated in a proof that every property is testable with two-sided error within any *hyperfinite* graph family (this includes for instance any proper minor-closed class) [37]. The bounded degree assumption is crucial for obtaining this result and it has since then been an important open problem to obtain testability results in the weaker *sparse model*, which does not assume that the maximum degree is bounded [14, 16]. In this more general model there are two types of queries: given a vertex v, we can query the degree $d(v)$ of v in G; we can also query the i-th neighbor of v, for $1 \leq i \leq d(v)$ (all these queries are assumed to take constant time). In this model, much less is known: it was proved that bipartiteness is testable within any minor-closed class in [14], while already in the bounded degree model many simple properties are not testable in general graph classes [29], so the restriction to a sparse structured class such as a proper minor-closed class is very natural in this context. The interested reader is referred to the book of Goldreich [27] for more results and references on property testing, and especially Chapter 10 in the book, which focuses on the general graph model.

Instead of working in the sparse model as defined above, it will be enough to restrict ourselves to a single type of query: given a vertex v, we query a random neighbor of v, uniformly among the neighbors of v. Following [16], we say we make queries to the *random neighbor oracle*. Note that this type of queries can clearly be implemented is the sparse model, so this is a restriction of the model (see [16] for a comparison between these two models, and a third one were we are allowed to query a constant number of *distinct* random neighbors of a given vertex). The following was recently proved by Czumaj and Sohler [16].

▶ **Theorem 1** ([16]). *For every proper minor-closed class $\mathcal{G}$, and any finite family $\mathcal{H}$, the property of being $\mathcal{H}$-free for graphs from $\mathcal{G}$ is testable with one-sided error in the sparse model, where only queries to the random neighbor oracle are allowed.*

Here we say that a graph is *H-free* if it does not contain H as a subgraph, and *$\mathcal{H}$-free* if it is H-free for every $H \in \mathcal{H}$. Our main result is an extension of Theorem 1 to any *monotone* property, that is any property closed under taking subgraphs.

▶ **Theorem 2.** *For every proper minor-closed class $\mathcal{G}$, and any monotone property $\mathcal{P}$, the property of satisfying $\mathcal{P}$ for graphs from $\mathcal{G}$ is testable with one-sided error in the sparse model, where only queries to the random neighbor oracle are allowed.*

Note that for any monotone property $\mathcal{P}$ there is a (possibly infinite) family of graphs $\mathcal{H}$ such that $\mathcal{P}$ is precisely the property of being $\mathcal{H}$-free. This family $\mathcal{H}$ can be simply defined as the class of all the graphs that do not satisfy $\mathcal{P}$, or as the class of all the graphs that do not satisfy $\mathcal{P}$ and are minimal with this property (with respect to the subgraph relation). It follows that Theorem 2 is the natural generalization of Theorem 1, where we remove the assumption that $\mathcal{H}$ is finite. This can be seen as an analogue of the situation in the dense graph model: it was first proved that the property of being H-free (or $\mathcal{H}$-free for finite $\mathcal{H}$) was testable in this model [2], and then only much later was this extended to all monotone classes by Alon and Shapira [5]. Note that many natural monotone properties, such as being planar or k-colorable for some $k \geq 2$, do not have a finite set of minimal forbidden subgraphs. So there is a fundamental gap between being H-free and being $\mathcal{H}$-free for infinite $\mathcal{H}$.

1.2 Local certification

We now describe our second main result, which is obtained by extending the methods used in the proof of Theorem 2. We start by introducing the setting of this result: The problem of *local certification*.

In this part, all graphs are assumed to be connected. The vertices of any n-vertex graph G are assumed to be assigned distinct (but otherwise arbitrary) identifiers $(\mathrm{id}(v))_{v\in V(G)}$ from $\{1,\dots,\mathrm{poly}(n)\}$. In the remainder of this section, all graphs are implicitly labelled by these distinct identifiers (for instance, whenever we talk about a subgraph H of a graph G, we implicitly refer to the corresponding labelled subgraph of G). We follow the terminology introduced by Göös and Suomela [30].

Proofs

A *proof* for a graph G is a function $P : V(G) \to \{0,1\}^*$ (G is considered as a labelled graph, so the proof P is allowed to depend on the identifiers of the vertices of G). The binary words $P(v)$ are called *certificates*. The *size* of P is the maximum size of a certificate $P(v)$, for $v \in V(G)$.

Local verifiers

A *verifier* $\mathcal{A}$ is a function that takes a graph G, a proof P for G, and a vertex $v \in V(G)$ in input, and outputs an element of $\{0,1\}$. We say that v *accepts* the instance if $\mathcal{A}(G,P,v) = 1$ and that v rejects the instance if $\mathcal{A}(G,P,v) = 0$.

Consider an integer $r \geq 0$, a graph G, a proof P for G, and a vertex $v \in V(G)$. Let $B_r(v)$ denote the set of vertices at distance at most r from v in G. We denote by $G[v,r]$ the subgraph of G induced by $B_r(v)$, and similarly we denote by $P[v,r]$ the restriction of P to $B_r(v)$.

A verifier $\mathcal{A}$ is *local* if there is a constant $r \geq 0$, such that for any $v \in G$, $\mathcal{A}(G,P,v) = \mathcal{A}(G[v,r],P[v,r],v)$. In other words, the output of v only depends on the ball of radius r centered in v, for any vertex v of G. The constant r is called the *local horizon* of the verifier.

Proof labelling schemes

For an integer $r \geq 0$, an *r-round proof labelling scheme* for a graph class $\mathcal{G}$ is a prover-verifier pair $(\mathcal{P},\mathcal{A})$, with the following properties.

r-round: $\mathcal{A}$ is a local verifier with local horizon at most r.

Completeness: If $G \in \mathcal{G}$, then $P = \mathcal{P}(G)$ is a proof for G such that for any vertex $v \in V(G)$, $\mathcal{A}(G,P,v) = 1$.

Soundness: If $G \notin \mathcal{G}$, then for every proof P' for G, there exists a vertex $v \in V(G)$ such that $\mathcal{A}(G,P',v) = 0$.

In other words, upon looking at its ball of radius r (labelled by the identifiers and certificates), the local verifier of each vertex of a graph $G \in \mathcal{G}$ accepts the instance, while if $G \notin \mathcal{G}$, for every possible choice of certificates, the local verifier of at least one vertex rejects the instance.

The *complexity* of the labelling scheme is the maximum size of a proof $P = \mathcal{P}(G)$ for an n-vertex graph $G \in \mathcal{F}$. If we say that the complexity is $O(f(n))$, for some function f, the $O(\cdot)$ notation refers to $n \to \infty$. See [23, 30] for more details on proof labelling schemes and local certification in general.

It was proved in [25] that planar graphs have a 1-round proof labelling scheme of complexity $O(\log n)$, and that this complexity is optimal. The authors of [25] asked whether this can be extended to any proper minor-closed class. This was indeed extended in [24] to graphs embeddable in a fixed surface (see also [22] for a short proof), to graphs avoiding some small minors in [10], and more generally to any minor-closed class of bounded tree-width in [26] (in the last result, the complexity is $O(\log^2 n)$ instead of $O(\log n)$ in the other results mentioned here).

For $\varepsilon > 0$, define an *r-round ε-approximate proof labelling scheme* for some class $\mathcal{G}$ exactly as in the definition of r-round proof labelling scheme above, except that in the soundness part, the condition "If $G \notin \mathcal{G}$" is replaced by "If G is ε-far from $\mathcal{G}$" [13]. A graph class $\mathcal{G}$ is *summable* if for any $G_1, G_2 \in \mathcal{G}$, the disjoint union of G_1 and G_2 is also in $\mathcal{G}$. Elek recently proved the following result [21].

▶ **Theorem 3** ([21]). *For any $\varepsilon > 0$ and integer $D \geq 0$, and any monotone summable property $\mathcal{P}$ of a proper minor-closed class $\mathcal{G}$, there are constants $r \geq 0$ and $K \geq 0$ such that the class of graphs from $\mathcal{P}$ with maximum degree at most D has an r-round ε-approximate proof labelling scheme of complexity at most K.*

A natural problem is whether the bounded degree assumption in Elek's result can be omitted (Elek's proof crucially relies on this assumption). We prove that the bounded degree assumption can indeed be omitted.

▶ **Theorem 4.** *For any $\varepsilon > 0$ and any monotone summable property $\mathcal{P}$ of a proper minor-closed class $\mathcal{G}$, there are constants $r \geq 0$ and $K \geq 0$ such that $\mathcal{P}$ has an r-round ε-approximate proof labelling scheme of complexity at most K.*

We indeed prove a far-reaching generalization of this result (whose statement was suggested by Elek to the authors), concerning graph classes with bounded asymptotic dimension.

Asymptotic dimension

Given a graph G and an integer $r \geq 1$, we denote by G^r the graph obtained from G by adding edges between any pair of vertices at distance at most r in G. The *weak diameter* of a set S of vertices of G is the maximum distance in G between two vertices of S.

For an integer $d \geq 0$, a class of graphs $\mathcal{G}$ has *asymptotic dimension* at most d if there is a function $D : \mathbb{N} \to \mathbb{N}$ such that for any integer $r \geq 1$, any graph $G \in \mathcal{G}$ has a $(d+1)$-coloring of its vertex set such that any monochromatic[1] component of G^r has weak diameter at most $D(r)$ in G.

This notion was introduced by Gromov [31] in the more general context of metric spaces. In the specific case of graphs, it was proved that classes of bounded tree-width have asymptotic dimension at most 1, and proper minor-closed classes have asymptotic dimension at most 2 [9]. It was also proved that d-dimensional grids and families of graphs defined by the intersection of certain objects (such as unit balls) in $\mathbb{R}^d$ have asymptotic dimension d [9]. On the other hand, it is known that any class of bounded degree expanders has infinite asymptotic dimension (see [33]).

We will prove the following generalization of Theorem 4.

[1] A *monochromatic component* in a colored graph G is a connected component of a subgraph of G induced by one of the color classes.

▶ **Theorem 5.** *For any $\varepsilon > 0$ and any monotone summable property $\mathcal{P}$ of a class $\mathcal{G}$ of bounded asymptotic dimension, there are constants $r \geq 0$ and $K \geq 0$ such that $\mathcal{P}$ has an r-round ε-approximate proof labelling scheme of complexity at most K.*

Note that a monotone property $\mathcal{P}$ is summable if and only if all minimal forbidden subgraphs for $\mathcal{P}$ are connected. This includes for instance minor-closed classes whose minimal forbidden minors are connected, such as planar graphs, K_t-minor free graphs for any $t \geq 2$, graphs of bounded tree-width, graphs of bounded tree-depth, and graphs of bounded Colin de Verdière parameter.

Natural examples of non summable properties include toroidal graphs (or more generally graphs embeddable on any fixed surface other than the sphere). For monotone properties that are not necessarily summable, we prove the following.

▶ **Theorem 6.** *For any $\varepsilon > 0$ and any monotone property $\mathcal{P}$ of a proper minor-closed class $\mathcal{G}$, $\mathcal{P}$ has a 1-round ε-approximate proof labelling scheme of complexity $O(\log n)$.*

While the complexity of the scheme guaranteed by Theorem 6 is not constant as in Elek's result [21] and Theorem 4, we do not require any bounded degree assumption (as in Theorem 4), and a local horizon of 1 is sufficient. More importantly, the fact that $\mathcal{P}$ is not necessarily summable requires a completely different set of techniques, much closer from the tools used to prove Theorem 2. Theorem 6 can be thought of as an approximate answer to the question of [25] on the local certification of minor-closed classes.

Organization of the paper

We start with some preliminary results in Section 2. In Section 3, we prove the main technical contribution of this paper, a result showing that if a graph from some minor-closed class is far from a monotone property $\mathcal{P}$, then it contains linearly many edge-disjoint subgraphs of bounded size that are not in $\mathcal{P}$. In Section 4 we deduce Theorem 2 from this result and Theorem 1. Theorems 4 and 6 are proved in Section 5. We conclude in Section 6 with some remarks.

2 Preliminaries

Minor-closed classes

We denote the number of vertices of a graph G by $\mathsf{v}(G)$, and its number of edges by $\mathsf{e}(G)$. A class of graphs $\mathcal{G}$ is *minor-closed* if any minor of a graph from $\mathcal{G}$ is also in $\mathcal{G}$. A class is *proper* if it does not contain all graphs. The following was proved by Mader [35].

▶ **Theorem 7** ([35])**.** *For any proper minor-closed class $\mathcal{G}$, there is a constant C such that for any graph $G \in \mathcal{G}$, $\mathsf{e}(G) \leq C\,\mathsf{v}(G)$.*

Tree-depth

Given a rooted tree T, the *closure* of T is the graph obtained from T by adding edges between each vertex and its ancestors in the tree. The *height* of a rooted tree is the maximum number of vertices on a root-to-leaf path in the tree. The *tree-depth* of a connected graph G is the maximum height of a rooted tree T such that G is a subgraph of the closure of T, and the tree-depth of a graph G, denoted by $\mathrm{td}(G)$, is the maximum tree-depth of its connected components (equivalently, it is equal to the maximum height of a rooted *forest* F such that G is a subgraph of the closure of F).

The following was implicitly proved by Dvořák and Sereni [20] (in the proof below the actual definition of tree-width is not needed, so we omit it).

▶ **Theorem 8** ([20]). *For every proper minor-closed class $\mathcal{G}$ and every $\delta > 0$ there exists $d = d_8(\mathcal{G}, \delta) \in \mathbb{N}$ and $s = s_8(\mathcal{G}, \delta) \in \mathbb{N}$ satisfying the following. For every $G \in \mathcal{G}$ there exist $X_1, X_2, \ldots, X_s \subseteq V(G)$ such that*

- *for any $1 \leq i \leq s$, $\mathrm{td}(G[X_i]) \leq d$, and*
- *every $v \in V(G)$ belongs to at least $(1-\delta)s$ of the sets X_i.*

Proof. Let $t = \lceil \frac{2}{\delta} \rceil$. It was proved in [17] that there is a constant $k = k(t, \mathcal{G})$ such that any graph $G \in \mathcal{G}$ has a partition of its vertex set into t classes $Y_1, \ldots, Y_t$, such that the union of any $t-1$ classes Y_i induces a graph of tree-width at most k. In particular, if we define $Z_i := V(G) \setminus Y_i$ for any $1 \leq i \leq t$, then each graph $G[Z_i]$ has tree-width at most k and each vertex v lies in $t-1 = (1-\frac{1}{t})t$ sets Z_i. Dvořák and Sereni [20, Theorem 31] proved[2] that for every integer k and real $\delta > 0$, there are integers $r = r(k, \delta)$ and $d = d(k, \delta)$ such that for any graph H of tree-width at most k, H has a cover of its vertex set by r sets $X_1, \ldots, X_r$, such that each $H[X_i]$ has tree-depth at most d and each vertex lies in at least $(1-\frac{\delta}{2})r$ sets X_i. Applying this result to $H = G[Z_i]$ for any $1 \leq i \leq t$, we obtain rt sets $X'_1, \ldots, X'_{rt}$ of vertices of G, such that the subgraph $G[X'_i]$ induced by each of them has tree-depth at most d and each vertex of G lies in at least $(1-\frac{\delta}{2})r \cdot (1-\frac{1}{t})t \geq (1-\delta)rt$ sets X'_i. Thus d and $s = rt$ satisfy the conditions of the theorem. ◀

We deduce the following useful result.

▶ **Corollary 9.** *For every proper minor-closed class $\mathcal{G}$ and every $\varepsilon > 0$ there exists $d = d_9(\mathcal{G}, \varepsilon) \in \mathbb{N}$ satisfying the following. For every $G \in \mathcal{G}$ there exist $F \subseteq E(G)$ such that $|F| \leq \varepsilon\, \mathsf{e}(G)$ and $\mathrm{td}(G \setminus F) \leq d$.*

Proof. Let $\delta = \frac{\varepsilon}{2}$. We show that $d = d_8(\mathcal{G}, \delta)$ satisfies the corollary. Indeed, for $G \in \mathcal{G}$ let $X_1, X_2, \ldots, X_s \subseteq V(G)$ be as in Theorem 8. Let $F_i = E(G) \setminus E(G[X_i])$ for $i \in [s]$, then $\mathrm{td}(G \setminus F_i) \leq d$. Moreover, every edge belongs to at most $2\delta s$ sets F_i, so

$$\frac{1}{s} \sum_{i=1}^{s} |F_i| \leq \frac{1}{s} \cdot 2\delta s \cdot \mathsf{e}(G) = \varepsilon\, \mathsf{e}(G).$$

Thus, by averaging, $|F_i| \leq \varepsilon\, \mathsf{e}(G)$ for some i, and $F = F_i$ satisfies the corollary. ◀

Note that the conclusion of Corollary 9 can be shown to hold in greater generality than in the context of minor-closed classes. For instance, any class in which all graphs can be made of bounded *tree-width* by removing an arbitrarily small fraction of edges also have this property (see [20]). This includes all graphs of bounded *layered tree-width* (see [18, 39]). Typical non minor-closed examples of such classes are families of graphs that can be embedded on a fixed surface, with a bounded number of crossings per edge [19]. However, since the proof of Theorem 1 itself strongly relies on edge-contractions (and thus on the graph class $\mathcal{G}$ being minor-closed), Theorem 2 does not seem to be easily extendable beyond minor-closed classes.

[2] The property that s is bounded independently of G does not appear explicitly in the statement of their theorem, but readily follows from their proof. This will only be needed in Section 5.

3 Bounded size obstructions

3.1 General properties

A graph property $\mathcal{P}$ is a graph class that is closed under isomorphism. It will be convenient to write that $G \in \mathcal{P}$ instead of "G satisfies $\mathcal{P}$" in the remainder of the paper. A graph H is *minimally not in* $\mathcal{P}$ if $H \notin \mathcal{P}$ and any proper subgraph of H is in $\mathcal{P}$.

We will use the following result of Nešetřil and Ossona de Mendez (Lemma 6.13 in [36]) [3].

▶ **Lemma 10** ([36]). *For every integer $d \geq 1$ and every property $\mathcal{P}$, there exists $N = N_{10}(d, \mathcal{P})$ such that if H is minimally not in $\mathcal{P}$ and $\mathrm{td}(H) \leq d$ then $\mathsf{v}(H) \leq N$.*

3.2 Colorability

The conclusion of Lemma 10 is quite strong but it does not give explicit bounds on $N_{10}(d, \mathcal{P})$. For completeness, we give such an explicit bound when $\mathcal{P}$ is the property of being k-colorable. The specific question of whether 3-colorability of planar graphs was testable in the sparse model was raised by Christian Sohler at the Workshop on Local Algorithms (WOLA) in 2021. A positive answer to this question directly follows from Theorem 2, but the lemma below allows us to give an explicit bound on the query complexity of testing k-colorability in minor-closed classes (see Section 6).

Given a graph H and two vertex subsets $A, B \subseteq V(H)$, we say that (A, B) is a *proper separation* of H if $A \cup B = V(H)$, $A \setminus B$ and $B \setminus A$ are both non-empty, and there are no edges between $A \setminus B$ and $B \setminus A$ in H. We say that a graph H is *split* if there exists a proper separation (A, B) of H and an isomorphism $\phi : A \to B$ between $H[A]$ and $H[B]$ such that $\phi(v) = v$ for every $v \in A \cap B$ (see Figure 1 for an example). Equivalently, a split graph can be obtained by taking two copies of some smaller graph and, for a proper subset of vertices, identifying the two copies of the vertex subset with each other.

A connected graph H is *unsplit* if it is not split. Note that minimally non-k-colorable graphs are unsplit.

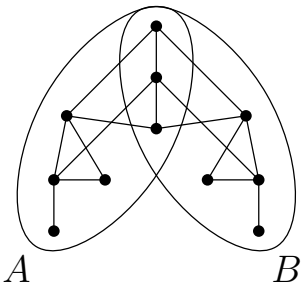


Figure 1 A split graph H and the corresponding proper separation (A, B) of H. The isomorphism $\phi : A \to B$ is the reflection symmetry with respect to the vertical axis.

The *tower function* is defined as $\mathsf{twr}(0) = 0$ and $\mathsf{twr}(i+1) = 2^{\mathsf{twr}(i)}$ for any integer $i \geq 0$.

▶ **Lemma 11.** *For every integer $d \geq 1$, there exists $N = N_{11}(d) = \mathsf{twr}(O(d))$ such that if H is an unsplit graph with $\mathrm{td}(H) \leq d$ then $\mathsf{v}(H) \leq N$.*

[3] The version we use here only needs Q to be a singleton in the statement of Lemma 6.13 in [36].

Proof. Define $k_d := 1$ and for any $0 \leq i \leq d-1$, let $k_i := (2^{\binom{k_{i+1}}{2}+d\cdot k_{i+1}})k_{i+1}+1$.

Choose a rooted tree T of height at most $d+1$ rooted at some vertex r, such that H is a subgraph of the closure of T. For each $v \in V(H)$, let T_v be the subtree of T rooted at v (consisting of v and all its descendants), and let H_v be the subgraph of H induced by $V(T_v)$. Define the *level* $\ell(v) := \mathrm{dist}_T(r,v) \in [0,d]$.

We prove by induction on $d-i$ that $\mathsf{v}(H_v) \leq k_i$ for every v with $\ell(v) = i$. The base case $i = d$ trivially holds, as $k_d = 1$.

For the induction step, let $u_1, \ldots, u_m$ be all the children of a vertex v with $\ell(v) = i$, and let A_v be the set consisting of v and its ancestors. Then $|A_v| = i+1$. For each $j \in [m]$, let $H_j^+ = H[A_v \cup V(H_{u_j})]$. Note that there are at most $2^{\binom{k_{i+1}}{2}+d\cdot k_{i+1}}$ distinct (labelled) graphs G on at most $|A_v| + k_{i+1} \leq d + k_{i+1}$ vertices such that the subgraph of G induced by the first $|A_v|$ vertices is isomorphic to $H[A_v]$. As $\mathsf{v}(H_{u_j}) \leq k_{i+1}$, if $m > 2^{\binom{k_{i+1}}{2}+d\cdot k_{i+1}}$ there exist $j \neq j'$ and an isomorphism $\phi : V(H_j^+) \to V(H_{j'}^+)$ such that $\phi(w) = w$ for every $w \in A_v$. Such an isomorphism would imply that H is split, and so $m \leq 2^{\binom{k_{i+1}}{2}+d\cdot k_{i+1}}$. It follows that $\mathsf{v}(H_v) \leq mk_{i+1}+1 \leq (2^{\binom{k_{i+1}}{2}+d\cdot k_{i+1}})k_{i+1}+1 = k_i$, as desired.

By taking $N := k_0$ we obtain that $\mathsf{v}(H) \leq N$. It can be checked from the definition of $(k_i)_{0\leq i\leq d}$ that $N = k_0$ is at most a tower function of $O(d)$. ◀

3.3 A linear Erdős-Posá property

We use Corollary 9 and Lemma 10 to deduce the following result, which is the main technical contribution of this paper.

▶ **Theorem 12.** *For every proper minor-closed class $\mathcal{G}$, every $\varepsilon > 0$, and every property $\mathcal{P}$, there exists $\delta > 0$ and an integer N such that for every $G \in \mathcal{G}$ either*

- *there exists $F \subseteq E(G)$ with $|F| \leq \varepsilon\,\mathsf{e}(G)$ such that $G \setminus F$ is in $\mathcal{P}$, or*
- *there exist edge-disjoint subgraphs $G_1, \ldots, G_m$ of G that are not in $\mathcal{P}$, such that $m \geq \delta\,\mathsf{e}(G)$ and for every $1 \leq i \leq m$, $\mathsf{v}(G_i) \leq N$.*

Proof. By Theorem 7, there exists C such that $\mathsf{e}(G) \leq C\mathsf{v}(G)$ for every $G \in \mathcal{G}$. Let $d := d_9(\mathcal{G}, \varepsilon/2)$ and let $N := N_{10}(d, \mathcal{P})$. We show that $\delta := \frac{\varepsilon}{2NC}$ satisfies the theorem.

Let $G_1, \ldots, G_m$ be a maximal collection of edge-disjoint subgraphs of G that are not in $\mathcal{P}$, and such that $\mathsf{v}(G_i) \leq N$. If $m \geq \delta\mathsf{e}(G)$ the theorem holds, so we assume that $m < \delta\mathsf{e}(G)$.

Let $F' = \bigcup_{i=1}^m E(G_i)$. Then

$$|F'| \leq C\sum_{i=1}^{m} \mathsf{v}(G_i) \leq CmN < cN\delta\mathsf{e}(G) \leq \frac{\varepsilon}{2}\,\mathsf{e}(G).$$

Let $G' = G \setminus F'$. By the choice of d, it follows from Corollary 9 that there exists $F'' \subseteq E(G')$ such that $|F''| \leq \frac{\varepsilon}{2}\,\mathsf{e}(G)$ and $\mathrm{td}(G' \setminus F'') \leq d$.

Let $G'' = G' \setminus F''$. Suppose first that G'' is not in $\mathcal{P}$, and let H be a minimal subgraph of G'' that is not in $\mathcal{P}$. As H is minimally not in $\mathcal{P}$, it follows from Lemma 10 that $\mathsf{v}(H) \leq N$. Thus adding H to the collection $G_1, \ldots, G_m$ contradicts its maximality.

It follows that G'' is in $\mathcal{P}$, but $G'' = G \setminus F$, where $F = F' \cup F''$ and $|F| \leq \varepsilon\,\mathsf{e}(G)$, and so the theorem holds. ◀

4 Property testing in the sparse model

The model

As alluded to in the introduction, we work in the sparse model, only using queries to the random neighbor oracle. That is, given an input graph G, the tester only does a constant number of queries to the input, all of the following type: given a vertex v, return a random neighbor of v (uniformly at random among all the neighbors of v in G). The vertex v itself can be taken to be a random vertex of G, but does not need to. The computation of a random vertex of G and a random neighbor of a given vertex of G are assumed to take constant time in this model.

We are now ready to prove Theorem 2.

Proof of Theorem 2. Let $\mathcal{G}$ be a proper-minor class and $\mathcal{P}$ be a monotone property. Let ε be given. Let $\delta > 0$ and N be obtained by applying Theorem 12 to $\mathcal{G}$, $\mathcal{P}$ and ε, and let $\mathcal{H}$ be the (finite) set of all graphs of at most N vertices that are not in $\mathcal{P}$. We now run the tester of Theorem 1 for testing whether a graph $G \in \mathcal{G}$ is $\mathcal{H}$-free or δ-far from being $\mathcal{H}$-free.

Assume first that $G \in \mathcal{P}$. If G contains a graph $H \in \mathcal{H}$ as a subgraph, then since $\mathcal{P}$ is monotone, we have $H \in \mathcal{P}$, which is a contradiction. Hence, G is $\mathcal{H}$-free, and it follows that the one-sided tester of Theorem 1 accepts G with probability 1. Assume now that G is ε-far from $\mathcal{P}$. By Theorem 12, there exist at least $\delta \mathsf{e}(G)$ edge-disjoint subgraphs of G that are all in $\mathcal{H}$, and thus one needs to remove at least one edge in each of these $\delta \mathsf{e}(G)$ edge-disjoint subgraphs to obtain an $\mathcal{H}$-free graph. As $\mathcal{P}$ is monotone, G is δ-far from being $\mathcal{H}$-free, and it follows that the tester of Theorem 1 rejects G with probability at least $\frac{2}{3}$, as desired. This concludes the proof of Theorem 2. ◀

5 Local Certification

We recall that in this part, all graphs are assumed to be connected.

5.1 Summable properties

Before we prove Theorem 5, we will need the following consequence of a result of Brodskiy, Dydak, Levin and Mitra [12] (obtained by taking $r = 2$ in Theorem 2.4 in their paper). This can be seen as an analogue of Theorem 8 where tree-depth is replaced by the weaker notion of weak diameter, while proper minor-closed classes are replaced by the more general classes of bounded asymptotic dimension.

▶ **Theorem 13** ([12]). *Let $\mathcal{G}$ be a class of graphs of bounded asymptotic dimension and let $\delta > 0$ be a real number. Then there exist two constants $D = D_{13}(\mathcal{G}, \delta) \in \mathbb{N}$ and $s = s_{13}(\mathcal{G}, \delta) \in \mathbb{N}$ satisfying the following. For every $G \in \mathcal{G}$ there exist $X_1, X_2, \ldots, X_s \subseteq V(G)$ such that*

- *for any $1 \leq i \leq s$, each connected component of $G[X_i]$ has weak diameter at most D in G, and*
- *every $v \in V(G)$ belongs to at least $(1-\delta)s$ of the sets X_i.*

It can be noted that if a subset S of vertices of a graph G is such that $G[S]$ has bounded tree-depth, then $G[S]$ has bounded diameter [36], and thus S has bounded weak diameter in G. It follows that in the special case of proper minor-closed classes, Theorem 8 implies Theorem 13 in a strong form.

Proof of Theorem 5. Fix any real number $\varepsilon > 0$ and an integer $d \geq 0$. Let $\mathcal{G}$ be a class of bounded asymptotic dimension, and let $\mathcal{P}$ be a monotone summable property of $\mathcal{G}$. Let $\delta = \frac{\varepsilon}{2}$. By Theorem 13, there exist two constants $D = D_{13}(\mathcal{G}, \delta) \in \mathbb{N}$ and $s = s_{13}(\mathcal{G}, \delta) \in \mathbb{N}$ satisfying the following. For every $G \in \mathcal{G}$ there exist $X_1, X_2, \ldots, X_s \subseteq V(G)$ such that

- for any $1 \leq i \leq s$, each component of $G[X_i]$ has weak diameter at most D in G, and
- every $v \in V(G)$ belongs to at least $(1-\delta)s$ of the sets X_i.

For any $v \in V(G)$, we define the proof $P(v)$ as (a binary representation of) the set of indices $I(v) \subseteq \{1, 2, \ldots, s\}$ such that $v \in X_i$. This proof has constant size (depending only of $\mathcal{P}$ and ε).

For every vertex v, the local verifier $\mathcal{A}(G, P, v)$ first checks that $I(v)$ contains at least $(1-\delta)s$ integers from $\{1, 2, \ldots, s\}$. If this is not the case, then v rejects the instance. In the remainder, we call a *monochromatic component of color i* a maximal connected subset of vertices v of G such that $i \in I(v)$. We omit the color if it is irrelevant in the discussion. Note that each vertex v belongs to $|I(v)|$ monochromatic components. For each vertex v, $\mathcal{A}(G, P, v)$ checks that the subgraph of G induced by the vertices $u \in B_r(v)$ is in $\mathcal{P}$, for $r = 2D + 1$, and that all monochromatic components of G containing v have weak diameter at most D (this can be clearly done as v has access to the subgraph of G induced by its ball of radius $r = 2D + 1$). If this is the case, then v accepts the instance, and otherwise v rejects the instance.

It follows from the definition of our scheme and the monotonicity of $\mathcal{P}$ that for any $G \in \mathcal{P}$, the local verifier $\mathcal{A}(G, P, v)$ of each vertex v of G accepts the instance. Consider now a graph G and a proof P' such that for each vertex v, $\mathcal{A}(G, P', v) = 1$. The proof P' assigns a subset $I(v)$ of indices of $\{1, \ldots, s\}$ to each vertex v of G, such that $|I(v)| \geq (1-\delta)s$. For each $1 \leq i \leq s$, let X_i be the subset of vertices v of G such that $i \in I(v)$. Let C be a connected component of some $G[X_i]$ (that is, C is a monochromatic component of color i), for some $1 \leq i \leq s$. Since all vertices of C accept the instance, C has weak diameter at most D in G. It follows that C is contained in some ball of radius D in G, and thus (since $\mathcal{P}$ is monotone, and each ball of radius $r = 2D + 1 \geq D$ induces a graph of $\mathcal{P}$), C lies in $\mathcal{P}$. As $\mathcal{P}$ is summable, $G[X_i]$ also lies in $\mathcal{P}$.

It follows from the proof of Corollary 9, that if we set $F_i = E(G) \setminus E(G[X_i])$ for any $1 \leq i \leq s$, then the property that every vertex $v \in V(G)$ belongs to at least $(1-\delta)s$ of sets X_i implies that there is an index $1 \leq i \leq s$ such that $|F_i| \leq \varepsilon\, \mathsf{e}(G)$. By the paragraph above $G \setminus F_i$ satisfies $\mathcal{P}$, and thus G is ε-close from $\mathcal{P}$ (we say that a graph is *ε-close* from $\mathcal{P}$ if it is not ε-far from $\mathcal{P}$). In the contrapositive, we have proved that if G is ε-far from $\mathcal{P}$, then there is at least one vertex v such that $\mathcal{A}(G, P', v) = 0$, as desired. ◀

Using the fact that proper minor-closed classes have asymptotic dimension at most 2 [9], we immediately obtain Theorem 4 as a corollary. Note that for the same purpose we could also use an earlier (and simpler) result of Ostrovskii and Rosenthal [38], who proved that for every integer t, the class of K_t-minor free graphs has asymptotic dimension at most 4^t. We could also use Theorem 8 without any reference to asymptotic dimension, as Theorem 8 implies Theorem 13 for proper minor-closed classes (see the discussion before the proof of Theorem 5).

5.2 Non necessarily summable properties

We now consider proof labelling schemes of complexity $O(\log n)$, rather than $O(1)$. To prove Theorem 6, we will need the following recent result of Bousquet, Feuilloley and Pierron [11].

▶ **Theorem 14** ([11]). *For every integer $d \geq 1$ and every first-order sentence φ, the class of graphs of tree-depth at most d satisfying φ has a 1-round proof labelling scheme of complexity $O(\log n)$.*

Although we will not need it, it is worth noting that the complexity in their result is of order $O(d \log n + f(d, \varphi))$. Observe also that checking whether a graph is $\mathcal{H}$-free for some fixed finite family $\mathcal{H}$ can be expressed by a first-order formula. In particular, it follows directly from Lemma 10 that checking whether $\mathrm{td}(G) \leq d$ can be expressed by a first-order formula (and thus certified with local horizon 1 with labels of size $O(\log n)$ per vertex).

The final ingredient that we will need is the ability to certify a rooted spanning tree, together with the children/parent relationship in this tree, with certificates of $O(\log n)$ bits per vertex (see [1, 6, 34] for the origins of this classical scheme). The prover gives the identifier $\mathrm{id}(r)$ of the root r of T to each vertex v of G, as well as $d_T(v, r)$, its distance to r in T, and each vertex v distinct from the root is also given the identifier of its parent $p(v)$ in T. The local verifier at v starts by checking that v agrees with all its neighbors in G on the identity of the root r of T. If so, if $v \neq r$, v checks that $d_T(v, r) = d_T(p(v), r) + 1$. It can be checked that all vertices accept the instance if and only T is a rooted spanning tree of G. Moreover, once the rooted spanning tree T has been certified, each vertex of G knows its parent and children (if any) in T.

We are now ready to prove Theorem 6.

Proof of Theorem 6. The beginning of the proof proceeds exactly as in the proof of Theorem 4. Fix any real number $\varepsilon > 0$. Let $\mathcal{G}$ be a proper minor-closed class, and let $\mathcal{P}$ be a monotone (not necessarily summable) property of $\mathcal{G}$. Let $\delta = \frac{\varepsilon}{2}$. By Theorem 8, there exist $d = d_8(\mathcal{G}, \delta) \in \mathbb{N}$ and $s = s_8(\mathcal{G}, \delta) \in \mathbb{N}$ satisfying the following. For every $G \in \mathcal{G}$ there exist $X_1, X_2, \ldots, X_s \subseteq V(G)$ such that

- for any $1 \leq i \leq s$, $\mathrm{td}(G[X_i]) \leq d$, and
- every $v \in V(G)$ belongs to at least $(1 - \delta)s$ of the sets X_i.

By Lemma 10, there exists a constant $N = N_{10}(d, \mathcal{P})$ such that if H is minimally not in $\mathcal{P}$ and $\mathrm{td}(H) \leq d$ then $\mathsf{v}(H) \leq N$. Let $\mathcal{H}$ be the (finite) set of all graphs of at most N vertices that are not in $\mathcal{P}$.

For any $v \in V(G)$, the proof $P(v)$ contains (a binary representation of) the set of indices $I(v) \subseteq \{1, \ldots, s\}$ such that $v \in X_i$. This part of the proof has constant size (depending only of $\mathcal{P}$ and ε). As in the proof of Theorem 4, the local verifier at each vertex v checks that $|I(v)| \geq (1 - \delta)s$, and rejects the instance if this does not hold.

For each $1 \leq i \leq s$, we do the following. In each connected component C of $G[X_i]$, we consider a rooted spanning tree T_C of C, with root r_C, and certify it using certificates of $O(\log n)$ bits per vertex. It follows from Theorem 14 that any first-order property of $G[C]$ can be certified with certificates of size $O(\log n)$ bits per vertex (as all the components C are vertex-disjoint, combining all these certificates and schemes still results in a scheme with labels of $O(\log n)$ bits per vertex). In particular we can certify that $\mathrm{td}(G[C]) \leq d$ (this is a first-order property). Let $\mathcal{H}'$ be the class of all (non-empty) graphs obtained from a graph $H \in \mathcal{H}$ by deleting an arbitrary subset of connected components of H (note that if all the graphs of $\mathcal{H}$ are connected, $\mathcal{H} = \mathcal{H}'$). Observe that all the graphs of $\mathcal{H}'$ have size at most N (which is a constant independent of the size of G). Then, for any $H' \in \mathcal{H}'$, we certify that $G[C]$ is H'-free or contains a copy of H' using Theorem 14, and store this information at the vertex r_C in a constant-size binary array $b(r_C)$, whose entries are indexed by all the graphs of $\mathcal{H}'$ (where the entry of $b(r_C)$ corresponding to some $H' \in \mathcal{H}'$ is equal to 1 if and only if C contains a copy of H' as a subgraph).

It remains to aggregate this information along some rooted spanning tree T of G (which can itself be certified with certificates of $O(\log n)$ bits per vertex). We do this as follows, for every $1 \leq i \leq s$. For a vertex v of the rooted tree T, the subtree of T rooted in v is denoted by T_v. For each vertex v of G, let $\mathcal{C}_v$ be the set of components C of $G[X_i]$ such that r_C lies in T_v. Then the proof $P(v)$ contains a binary array $c(v)$, whose entries are indexed by the graphs H' of $\mathcal{H}'$. The array $c(v)$ is defined as follows: for any $H' \in \mathcal{H}'$, the entry of $c(v)$ corresponding to H' is equal to 1 if and only if H' is a disjoint union of (non necessarily connected) graphs $H'_1, H'_2, \ldots, H'_k \in \mathcal{H}'$ such that each H'_i appears in a different component of $\mathcal{C}_v$. The consistency of the binary arrays $c(v)$ is verified locally as follows. For each vertex v of G, the local verifier at v considers the binary arrays $c(u)$, for all children u of v (and the binary array $b(v)$, if v is equal to some root r_C). For any $H' \in \mathcal{H}'$, the local verifier at v checks whether H' can be written as a disjoint union of graphs $H'_1, H'_2, \ldots, H'_k \in \mathcal{H}'$ such that each H'_i appears in a different array among the children of v (plus in $b(v)$, if v is a root of some component C). The local verifier at v then checks whether this is consistent with the entry corresponding to H' in $c(v)$. Clearly, all the vertices accept if and only if the information is consistent along the spanning tree, and it follows that the local verifier at the root r can check for each $H \in \mathcal{H} \subseteq \mathcal{H}'$, whether the entry of $c(r)$ corresponding to H is equal to 0 or 1. It follows that the local verifier at r can check whether $G[X_i]$ is $\mathcal{H}$-free (and accept the instance if and only if this is the case).

It follows from the definition of our scheme that for any $G \in \mathcal{P}$, the local verifier of each vertex of G accepts the instance.

Consider now some graph G together with some proof P' such that the local verifier $\mathcal{A}(G, P', v)$ at each vertex v of G accepts the instance. For any $1 \leq i \leq s$, let X_i be the set of vertices v such that $i \in I(v)$ (where $I(v)$ is given by the proof $P'(v)$), and let $F_i = E(G) \setminus E(G[X_i])$. As in the proof of Theorem 4, the property that every vertex $v \in V(G)$ belongs to at least $(1-\delta)s$ of sets X_i implies that there is an index $1 \leq i \leq s$ such that $|F_i| \leq \varepsilon \mathsf{e}(G)$.

By the properties of the local certificates, each component of $G \setminus F_i = G[X_i]$ has tree-depth at most d, and thus $G \setminus F_i$ has tree-depth at most d. Moreover, our local certificates imply that $G \setminus F_i$ is $\mathcal{H}$-free. Since $\mathcal{P}$ is monotone, $G \setminus F_i$ is in $\mathcal{P}$. It follows that G is ε-close from $\mathcal{P}$. Taking the contrapositive, this shows that if a graph is ε-far from $\mathcal{P}$, then at least one local verifier will reject the instance. This concludes the proof of Theorem 6. ◀

6 Conclusion

In this paper we proved that for any proper minor-closed class $\mathcal{G}$, using constantly many queries to the random neighbor oracle, a tester can decide with good probability whether an input graph $G \in \mathcal{G}$ satisfies some fixed monotone property $\mathcal{P}$, or is ε-far from $\mathcal{P}$. Given the level of generality of the result it is to be expected that no explicit bounds on the query complexity are given. However, we can give explicit estimates on the query complexity for specific properties. For instance, it follows from the bounds of [20, Corollary 35], combined with Lemma 11 and our proof of Theorem 2, that 3-colorability can be tested in planar graphs with $\mathsf{twr}(\mathrm{poly}(1/\varepsilon))$ queries to the random neighbor oracle. This can be extended to testing k-colorability in K_t-minor free graphs, for any k and t, at the expense of a significant increase in the height of the tower function, by combining the results of [20] with the main result of [17] (the bounds there are not explicit as a function of t, but can be made explicit using results from the Graph Minor series). This is to be compared with the main result of [14], that 2-colorability can be tested with $2^{2^{\mathrm{poly}(1/\varepsilon)}}$ queries in planar graphs. It is a

natural problem to understand whether these properties can be tested with poly$(1/\varepsilon)$ queries to the random neighbor oracle, and more generally to develop techniques for proving finer lower bounds on the query complexity of monotone properties in this model (see [7] for recent results in this direction in the bounded degree model).

References

1. Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theoretical Computer Science*, 186(1-2):199–229, 1997. doi:10.1016/S0304-3975(96)00286-1.
2. Noga Alon, Richard A. Duke, Hanno Lefmann, Vojtech Rödl, and Raphael Yuster. The algorithmic aspects of the regularity lemma. *Journal of Algorithms*, 16(1):80–109, 1994. doi:10.1006/jagm.1994.1005.
3. Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: It's all about regularity. *SIAM Journal on Computing*, 39(1):143–167, 2009. doi:10.1137/060667177.
4. Noga Alon and Asaf Shapira. A characterization of the (natural) graph properties testable with one-sided error. *SIAM Journal on Computing*, 37(6):1703–1727, 2008. doi:10.1137/06064888X.
5. Noga Alon and Asaf Shapira. Every monotone graph property is testable. *SIAM Journal on Computing*, 38(2):505–522, 2008. doi:10.1137/050633445.
6. Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 268–277. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185378.
7. Sabyasachi Basu, Akash Kumar, and C. Seshadhri. The complexity of testing all properties of planar graphs, and the role of isomorphism. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1702–1714. SIAM, 2022. doi:10.1137/1.9781611977073.69.
8. Itai Benjamini, Oded Schramm, and Asaf Shapira. Every minor-closed property of sparse graphs is testable. *Advances in Mathematics*, 223(6):2200–2218, 2010.
9. Marthe Bonamy, Nicolas Bousquet, Louis Esperet, Carla Groenland, Chun-Hung Liu, François Pirot, and Alex D. Scott. Asymptotic dimension of minor-closed families and Assouad-Nagata dimension of surfaces. *Journal of the European Mathematical Society*, to appear. arXiv:2012.02435.
10. Nicolas Bousquet, Laurent Feuilloley, and Théo Pierron. Local certification of graph decompositions and applications to minor-free classes. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, volume 217 of *LIPIcs*, pages 22:1–22:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.OPODIS.2021.22.
11. Nicolas Bousquet, Laurent Feuilloley, and Théo Pierron. What can be certified compactly? *CoRR*, abs/2202.06065, 2022. arXiv:2202.06065.
12. Nikolay Brodskiy, Jerzy Dydak, Michael Levin, and Atish Mitra. A Hurewicz theorem for the Assouad–Nagata dimension. *Journal of the London Mathematical Society*, 77(3):741–756, 2008.
13. Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theoretical Computer Science*, 811:112–124, 2020. doi:10.1016/j.tcs.2018.08.020.
14. Artur Czumaj, Morteza Monemizadeh, Krzysztof Onak, and Christian Sohler. Planar graphs: Random walks and bipartiteness testing. *Random Structures & Algorithms*, 55(1):104–124, 2019. doi:10.1002/rsa.20826.

15 Artur Czumaj, Asaf Shapira, and Christian Sohler. Testing hereditary properties of non-expanding bounded-degree graphs. *SIAM Journal on Computing*, 38(6):2499–2510, 2009. doi:10.1137/070681831.

16 Artur Czumaj and Christian Sohler. A characterization of graph properties testable for general planar graphs with one-sided error (it's all about forbidden subgraphs). In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1525–1548. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00089.

17 Matt DeVos, Guoli Ding, Bogdan Oporowski, Daniel P. Sanders, Bruce A. Reed, Paul D. Seymour, and Dirk Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *Journal of Combinatorial Theory, Series B*, 91(1):25–41, 2004. doi:10.1016/j.jctb.2003.09.001.

18 Vida Dujmović, Pat Morin, and David R. Wood. Layered separators in minor-closed graph classes with applications. *Journal of Combinatorial Theory, Series B*, 127:111–147, 2017. doi:10.1016/j.jctb.2017.05.006.

19 Vida Dujmović, Pat Morin, and David R. Wood. Graph product structure for non-minor-closed classes. *CoRR*, abs/1907.05168, 2019. arXiv:1907.05168.

20 Zdeněk Dvořák and Jean-Sébastien Sereni. On fractional fragility rates of graph classes. *Electronic Journal of Combinatorics*, 27(4):P4.9, 2020. URL: https://www.combinatorics.org/ojs/index.php/eljc/article/view/v27i4p9.

21 Gábor Elek. Planarity can be verified by an approximate proof labeling scheme in constant time. *CoRR*, abs/2006.11869, 2020. arXiv:2006.11869.

22 Louis Esperet and Benjamin Lévêque. Local certification of graphs on surfaces. *Theoretical Computer Science*, 909:68–75, 2022. doi:10.1016/j.tcs.2022.01.023.

23 Laurent Feuilloley. Introduction to local certification. *Discrete Mathematics & Theoretical Computer Science*, 23(3), 2021. arXiv:1910.12747.

24 Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Eric Rémila, and Ioan Todinca. Local certification of graphs with bounded genus. *CoRR*, abs/2007.08084, 2020. arXiv:2007.08084.

25 Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Éric Rémila, and Ioan Todinca. Compact distributed certification of planar graphs. *Algorithmica*, 83(7):2215–2244, 2021. doi:10.1007/s00453-021-00823-w.

26 Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. A meta-theorem for distributed certification. *CoRR*, abs/2112.03195, 2021. arXiv:2112.03195.

27 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

28 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

29 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/s00453-001-0078-7.

30 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016. doi:10.4086/toc.2016.v012a019.

31 Mikhael Gromov. *Asymptotic invariants of infinite groups, in Geometric Group Theory, Volume 2*, volume 182 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1993.

32 Avinatan Hassidim, Jonathan A. Kelner, Huy N. Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 22–31. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.77.

33 D Hume. A continuum of expanders. *Fundamenta Mathematicae*, 238:143–152, 2017.

34 Gene Itkis and Leonid A. Levin. Fast and lean self-stabilizing asynchronous protocols. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 226–239. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365691.

35 Wolfgang Mader. Homomorphieeigenschaften und mittlere Kantendichte von Graphen. *Mathematische Annalen*, 174(4):265–268, 1967.

36 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

37 Ilan Newman and Christian Sohler. Every property of hyperfinite graphs is testable. *SIAM Journal on Computing*, 42(3):1095–1112, 2013. doi:10.1137/120890946.

38 Mikhail I. Ostrovskii and David Rosenthal. Metric dimensions of minor excluded graphs and minor exclusion in groups. *International Journal of Algebra and Computation*, 25(4):541–554, 2015.

39 Farhad Shahrokhi. New representation results for planar graphs. In *29th European Workshop on Computational Geometry EuroCG*, pages 177–180, 2013. arXiv:1502.06175.

Streaming Submodular Maximization Under Matroid Constraints

Moran Feldman
University of Haifa, Israel

Paul Liu
Stanford University, CA, USA

Ashkan Norouzi-Fard
Google Research, Zurich, Switzerland

Ola Svensson
EPFL, Lausanne, Switzerland

Rico Zenklusen
ETH Zürich, Switzerland

Abstract

Recent progress in (semi-)streaming algorithms for monotone submodular function maximization has led to tight results for a simple cardinality constraint. However, current techniques fail to give a similar understanding for natural generalizations, including matroid constraints. This paper aims at closing this gap. For a single matroid of rank k (i.e., any solution has cardinality at most k), our main results are:

- A single-pass streaming algorithm that uses $\widetilde{O}(k)$ memory and achieves an approximation guarantee of 0.3178.
- A multi-pass streaming algorithm that uses $\widetilde{O}(k)$ memory and achieves an approximation guarantee of $(1 - 1/e - \varepsilon)$ by taking a constant (depending on ε) number of passes over the stream.

This improves on the previously best approximation guarantees of 1/4 and 1/2 for single-pass and multi-pass streaming algorithms, respectively. In fact, our multi-pass streaming algorithm is *tight* in that any algorithm with a better guarantee than 1/2 must make several passes through the stream and any algorithm that beats our guarantee of $1 - 1/e$ must make linearly many passes (as well as an exponential number of value oracle queries).

Moreover, we show how the approach we use for multi-pass streaming can be further strengthened if the elements of the stream arrive in uniformly random order, implying an improved result for p-matchoid constraints.

2012 ACM Subject Classification Mathematics of computing → Submodular optimization and polymatroids; Theory of computation → Streaming models

Keywords and phrases Submodular maximization, streaming, matroid, random order

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.59

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2107.07183v2

Funding *Moran Feldman*: Research supported in part by the Israel Science Foundation (ISF) grants no. 1357/16 and 459/20.
Ola Svensson: Research supported by the Swiss National Science Foundation project 200021-184656 "Randomness in Problem Instances and Randomized Algorithms."
Rico Zenklusen: Research supported in part by Swiss National Science Foundation grant number 200021_184622. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 817750).

Acknowledgements The authors are very grateful to Jan Vondrák. It is safe to say that this paper would not be nearly as good without Jan's many interesting discussions and comments.

1 Introduction

Submodular function optimization is a classic topic in combinatorial optimization (see, e.g., the book [28]). Already in 1978, Nemhauser, Wolsey, and Fisher [26] analyzed a simple greedy algorithm for selecting the most valuable set $S \subseteq V$ of cardinality at most k. This algorithm starts with the empty set S, and then, for k steps, adds to S the element u with the largest marginal value. Assuming the submodular objective function f is also non-negative and monotone, they showed that the greedy algorithm returns a $(1 - 1/e)$-approximate solution. Moreover, the approximation guarantee of $1 - 1/e$ is known to be tight [10, 25].

A natural generalization of a cardinality constraint is that of a matroid constraint. While a matroid constraint is much more expressive than a cardinality constraint, it has often been the case that further algorithmic developments have led to the same or similar guarantees for both types of constraints. Indeed, for the problem of maximizing a monotone submodular function subject to a matroid constraint, Călinescu, Chekuri, Pál, and Vondrák [5] developed the more advanced continuous greedy method, and showed that it recovers the guarantee $1 - 1/e$ in this more general setting. Since then, other methods, such as local search [14], have been developed to recover the same optimal approximation guarantee.

More recently, applications in data science and machine learning [19], with huge problem instances, have motivated the need for space-efficient algorithms, i.e., (semi-)streaming algorithms for (monotone) submodular function maximization. This is now a very active research area, and recent progress has resulted in a tight understanding of streaming algorithms for maximizing monotone submodular functions with a single cardinality constraint: the optimal approximation guarantee is $1/2$ for single-pass streaming algorithms, and it is possible to recover the guarantee $1 - 1/e - \varepsilon$ in $O_\varepsilon(1)$ passes. That it is impossible to improve upon $1/2$ in a single pass is due to [11], and the first single-pass streaming algorithm to achieve this guarantee is a simple "threshold" based algorithm [2] that, intuitively, selects elements with marginal value at least $\mathrm{OPT}/(2k)$. The $(1-1/e-\varepsilon)$ guarantee in $O_\varepsilon(1)$ passes can be obtained using smart implementations of the greedy approach [3, 17, 22, 23, 27].

It is interesting to note that simple greedy and threshold-based algorithms have led to tight results for maximizing a monotone submodular function subject to a cardinality constraint in both the "offline" RAM and data stream models. However, in contrast to the RAM model, where more advanced algorithmic techniques have generalized these guarantees to much more general constraint families, current techniques fail to give a similar understanding in the data stream model, both for single-pass and multi-pass streaming algorithms. Closing this gap is the motivation for our work. In particular, current results leave open the intriguing possibility to obtain the same guarantees for a matroid constraint as for a cardinality constraint. Our results make significant progress on this question for single-pass streaming algorithms and completely close the gap for multi-pass streaming algorithms.

▶ **Theorem 1.** *There is a single-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k)$ elements, requires $\widetilde{O}(k)$ additional memory, and achieves an approximation guarantee of* 0.3178*.*

The last theorem improves upon the previous best approximation guarantee of $1/4 = 0.25$ [6]. Moreover, the techniques are versatile and also yield a single-pass streaming algorithm with an improved approximation guarantee for non-monotone functions (improving from 0.1715 [12] to 0.1921).

Our next result is a tight multi-pass guarantee of $1 - 1/e - \varepsilon$, improving upon the previously best guarantee of $1/2 - \varepsilon$ [18].

▶ **Theorem 2.** *For every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k/\varepsilon)$ elements, makes $O(1/\varepsilon^3)$ many passes, and achieves an approximation guarantee of $1 - 1/e - \varepsilon$.*

The result is tight (up to the exact dependency on ε) in the following strong sense: any streaming algorithm with a better approximation guarantee than $1/2$ must make more than one pass [11], and any algorithm with a better guarantee than $1 - 1/e$ must make linearly (in the length of the stream) many passes [22] (see Section 6 for more detail).

The way we obtain Theorem 2 is through a rather general and versatile framework based on the "Accelerated Continuous Greedy" algorithm of [3], which was designed for the classic (non-streaming) setting. This allows us to obtain results with an improved number of passes or more general constraints in specific settings. First, if the elements of the stream arrive in uniformly random order, then we can improve the number of passes as stated below.

▶ **Theorem 3.** *If the elements arrive in an independently random order in each pass, then for every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k/\varepsilon)$ elements, makes $O(\varepsilon^{-2} \log \varepsilon^{-1})$ many passes, and achieves an approximation guarantee of $1 - 1/e - \varepsilon$.*

Second, also in the uniformly random order model, we can obtain results with even fewer passes, and that also extend to p-matchoid constraints, but at the cost of weaker approximation guarantees.

▶ **Theorem 4.** *If the elements arrive in an independently random order in each pass, then for every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k (any solution has cardinality at most k) that stores $O(k)$ elements, makes $O(\log \varepsilon^{-1})$ many passes, and achieves an approximation guarantee of $1/2 - \varepsilon$.*

Moreover, if the matroid constraint is replaced with a more general p-matchoid constraint, the above still holds except that now the approximation guarantee is $1/(p+1) - \varepsilon$ and the number of passes is $O(p^{-1} \log \varepsilon^{-1})$.

The p-matchoid result of Theorem 4 improves, in the random order model, over an algorithm of [18] that achieves the same approximation factor, but needs $O(p/\varepsilon)$ passes, whereas our algorithm requires a number of passes that only logarithmically depends on ε^{-1} and decreases (rather than increases) with p. (However, the procedure in [18] does not require random arrival order, and obtains its guarantees even in the adversarial arrival model.)

1.1 Our Technique

Before getting into the technical details of our approaches, we provide an overview of the main ingredients behind the techniques we employ.

Single pass algorithms

The 4-approximation single pass algorithm due to Chakrabarti and Kale [6] (and later algorithms based on it such as [8, 12]) maintains an integral solution in the following way. Whenever a new element u arrives, the algorithm considers inserting u into the solution at the expense of some element u' that gets removed from the solution; and this swap is performed if it is beneficial enough. Naturally, the decision to make the swap is a binary

decision: we either make the swap or we do not do that. The central new idea in our improved single pass algorithms (Theorem 1) is that we make the swap fractional. In other words, we start inserting fractions of u at the expense of fractions of u' (the identity of u' might be different for different fractions of u), and we continue to do that as long as the swap is beneficial enough. Since "beneficial enough" depends on properties of the current solution, the swapping might stop being beneficial enough before all of u is inserted into the solution, which explains why our fractional swapping does not behave like the integral swapping used by previous algorithms.

While our single pass algorithms are based on the above idea, they are presented in a slightly different way for simplicity of the presentation and analysis. In a nutshell, the differences can be summarized by the following two points.

- Instead of maintaining a fractional solution, we maintain multiple sets A_i (for $i \in \mathbb{Z}$). Membership of an element u in each of these sets corresponds to having a fraction of $1/m$ (for a parameter m of the algorithm) of u in the fractional solution.
- We do not remove elements from our fractional solution. Instead, we add new elements to sets A_i with larger and larger i indexes with the implicit view that only fractions corresponding to sets A_i with relatively large indices are considered part of the fractional solution.

To make the above points more concrete, we note that the fractional solution is reconstructed from the sets A_i according to the above principles at the very end of the execution of our algorithms. The reconstructed fractional solution is denoted by s in these algorithms.

Multi-pass algorithms

Badanidiyuru and Vondrák [3] described an algorithm called "Accelerated Continuous Greedy" that obtains $1 - 1/e - O(\varepsilon)$ approximation (for every $\varepsilon \in (0, 1)$) for maximizing a monotone submodular function subject to a matroid constraint. Even though their algorithm is not a data stream algorithm, it accesses the input only in a well-defined restricted way, namely though a procedure called "Decreasing-Threshold Procedure". Originally, this procedure was implemented using a greedy algorithm on an altered objective function. However, we observe that the algorithm of [3] can work even if Decreasing-Threshold Procedure is modified to return any local maximum of the same altered objective function. Therefore, to get a multiple pass data stream algorithm, it suffices to design such an algorithm that produces an (approximate) local maximum (or a solution that is as good as such a local maximum); this algorithm can then be used as the implementation of Decreasing-Threshold Procedure. This is the framework we use to get our $(1 - 1/e - \varepsilon)$-approximation algorithms.

To prove Theorem 2 using the above framework, we show that a known algorithm (a variant of the algorithm of Chakrabarti and Kale [6] due to Huang, Thiery, and Ward [18]) can be repurposed to produce an approximate local maximum using $O(\varepsilon^{-2})$ passes, which, when used in Accelerated Continuous Greedy, leads to the claimed $O(\varepsilon^{-3})$ many passes. Similarly, by adapting an algorithm of Shadravan [29] working in the random order model, and extending it to multiple passes, we are able to get a solution that is as good as an approximate local maximum in only $O(\varepsilon^{-1} \log \varepsilon^{-1})$ random-order passes, which leads to Theorem 3 when combined with the above framework.

Interestingly, any (approximate) local maximum also has an approximation guarantee of its own (without employing the above framework). This means that the above procedures for producing approximate local maxima can also be viewed as approximation algorithms

in their own right, which leads to Theorem 4.[1] It is important to note that Theorem 4 uses fewer passes than what is used in the proof of Theorem 3 to get a solution which is at least as good as an approximate local maximum. This discrepancy happens because in Theorem 4 we only aim for a solution with some approximation ratio r, where r is an approximation ratio guaranteed by any approximate local maximum in any instance. In contrast, Theorem 3 needs a solution that is as good as some approximate local maximum of the particular instance considered.

1.2 Additional Related Work

As mentioned above, Călinescu et al. [5] proposed a $(1 - 1/e)$-approximation algorithm for maximizing a monotone submodular function subject to a matroid constraint in the offline (RAM) setting, which is known to be tight [10, 25]. The corresponding problem with a non-monotone objective is not as well understood. A long line of work [9, 13, 20] on this problem culminated in a 0.385-approximation due to Buchbinder and Feldman [4] and an upper bound by Oveis Gharan and Vondrák [15] of 0.478 on the best obtainable approximation ratio.

The first semi-streaming algorithm for maximizing a monotone submodular function subject to a matroid constraint was described by Chakrabarti and Kale [6], who obtained an approximation ratio of 1/4 for the problem. This remained state-of-the-art prior to this work. However, Chan, Huang, Jiang, Kang, and Tang [7] managed to get an improved approximation ratio of 0.3178 for the special case of a partition matroid in the related preemptive online model. We note that the last approximation ratio is identical to the approximation ratio stated in Theorem 1, which points to some similarity that exists between the algorithms (in particular, both use fractional swaps). However, the algorithm of [7] is not a semi-streaming algorithm (and moreover, it is tailored to partition matroids). The first semi-streaming algorithm for the non-monotone version of the above problem was obtained by Chekuri, Gupta, and Quanrud [8], and achieved a $(1/(4 + e) - \varepsilon) \approx 0.1488$-approximation. This was later improved to 0.1715-approximation by Feldman, Karbasi, and Kazemi [12].[2]

Outline of the paper

In Section 2, we introduce notations and definitions used throughout this paper. Afterwords, in Section 3, we present and analyze our single-pass algorithms. The framework used to prove Theorems 2 and 3 is presented in detail in Section 4, and in the two sections after it we describe the algorithms for obtaining approximate local maxima (or equally good solutions) necessary for using this framework. Specifically, in Section 5 we show how to get such an algorithm for adversarial order streams (leading to Theorem 2), and in Section 7 we show how to get such an algorithm for random order streams (leading to Theorems 3 and 4). It is worth noting that Section 3 is independent of all the other sections, and therefore, can be skipped by a reader interested in the other parts of this paper.

[1] Technically, we can also get a result for adversarial order streams in this way, but we omit this result since it is weaker than a known result of [18].

[2] Mirzasoleiman et al. [24] claimed another approximation ratio for the problem (weaker than the one given later by [12]), but some problems were found in their analysis (see [16] for details).

2 Preliminaries

Recall that we are interested in the problem of maximizing a submodular function subject to a matroid constraint. In Section 2.1 we give the definitions necessary for formally stating this problem. Then, Section 2.2 defines the data stream model in which we study the problem. Finally, in Section 2.3 we present some additional notation and definitions that we use.

2.1 Problem Statement

Submodular Functions

Given a ground set $\mathcal{N}$, a *set function* $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ is a function that assigns a numerical value to every subset of $\mathcal{N}$. Given a set $S \subseteq \mathcal{N}$ and an element $u \in \mathcal{N}$, it is useful to denote by $f(u \mid S)$ the marginal contribution of u to S with respect to f, i.e., $f(u \mid S) \coloneqq f(S \cup \{u\}) - f(S)$. Similarly, we denote the marginal contribution of a set $T \subseteq \mathcal{N}$ to S with respect to f by $f(T \mid S) \coloneqq f(S \cup T) - f(S)$.

A set function $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ is called *submodular* if for any two sets S and T such that $S \subseteq T \subseteq \mathcal{N}$ and any element $u \in \mathcal{N} \setminus T$ we have $f(u \mid S) \geq f(u \mid T)$. Moreover, we say that f is *monotone* if $f(S_1) \leq f(S_2)$ for any sets $S_1 \subseteq S_2 \subseteq \mathcal{N}$, and f is *non-negative* if $f(S) \geq 0$ for every $S \subseteq \mathcal{N}$.

Matroids

A set system is a pair $M = (\mathcal{N}, \mathcal{I})$, where $\mathcal{N}$ is a finite set called the *ground set*, and $\mathcal{I} \subseteq 2^{\mathcal{N}}$ is a collection of subsets of the ground set. We say that a set $S \subseteq \mathcal{N}$ is *independent* in M if it belongs to $\mathcal{I}$ (otherwise, we say that it is a *dependent* set); and the rank of the set system M is defined as the maximum size of an independent set in it. A set system is a *matroid* if it has three properties: i) The empty set is independent, i.e., $\varnothing \in \mathcal{I}$. ii) Every subset of an independent set is independent, i.e., for any $S \subseteq T \subseteq \mathcal{N}$, if $T \in \mathcal{I}$ then $S \in \mathcal{I}$. iii) If $S \in \mathcal{I}$, $T \in \mathcal{I}$ and $|S| < |T|$, then there exists an element $u \in T \setminus S$ such that $S \cup \{u\} \in \mathcal{I}$.[3]

A matroid constraint is simply a constraint that allows only sets that are independent in a given matroid. Matroid constraints are of interest because they have a rich combinatorial structure and yet are able to capture many constraints of interest such as cardinality, independence of vectors in a vector space, and being a non-cyclic sub-graph.

Matchoids and p-matchoids

The matchoid notion (for the case of $p = 2$) was proposed by Jack Edmonds as a common generalization of matching and matroid intersection. Let $M_1 = (\mathcal{N}_1, \mathcal{I}_1), M_2 = (\mathcal{N}_2, \mathcal{I}_2), \ldots, M_q = (\mathcal{N}_q, \mathcal{I}_q)$ be q matroids, and let $\mathcal{N} = \mathcal{N}_1 \cup \cdots \cup \mathcal{N}_q$ and $\mathcal{I} = \{S \subseteq \mathcal{N} \mid S \cap \mathcal{N}_\ell \in \mathcal{I}_\ell \text{ for every } 1 \leq \ell \leq q\}$. The set system $M = (\mathcal{N}, \mathcal{I})$ is a *p-matchoid* if each element $u \in \mathcal{N}$ is a member of $\mathcal{N}_\ell$ for at most p indices $\ell \in [q]$. Informally, a p-matchoid is an intersection of matroids in which every particular element $u \in \mathcal{N}$ is affected by at most p matroids. It is easy to see that a 1-matchoid is just a matroid, and vice versa. 2-matchoids are often referred to simply as matchoids (without a parameter p).

[3] The last property is often referred to as the *exchange axiom* of matroids.

Problem

In the `Submodular Maximization subject to a Matroid Constraint` problem (`SMMatroid`), we are given a non-negative[4] submodular function $f\colon 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ and a matroid $M = (\mathcal{N}, \mathcal{I})$ over the same ground set. The objective is to find an independent set $S \in \mathcal{I}$ that maximizes f. An important special case of `SMMatroid` is the `Monotone Submodular Maximization subject to a Matroid Constraint` problem (`MSMMatroid`) in which we are guaranteed that the objective function f is monotone (in addition to being non-negative and submodular).

2.2 Data Stream Model

In the data stream model, the input appears in a sequential form known as the *input stream*, and the algorithm is allowed to read it only sequentially. In the context of our problem, the input stream consists of the elements of the ground set sorted in either an adversarially chosen order or a uniformly random order, and the algorithm is allowed to read the elements from the stream only in this order. Often the algorithm is allowed to read the input stream only once (such algorithms are called *single-pass* algorithms), but in other cases it makes sense to allow the algorithm to read the input stream multiple times – each such reading is called a *pass*. The order of the elements in each pass might be different; in particular, when the order is random, we assume that it is chosen independently for each pass.

A trivial way to deal with the restrictions of the data stream model is to store the entire input stream in the memory of the algorithm. However, we are often interested in a stream carrying too much data for this to be possible. Thus, the goal in this model is to find a high quality solution while using significantly less memory than what is necessary for storing the input stream. The gold standard are algorithms that use memory of size nearly linear in the maximum possible size of an output; such algorithms are called *semi-streaming* algorithms.[5] For `SMMatroid` and `MSMMatroid`, this implies that a semi-streaming algorithm is a data stream algorithm that uses $O(k \log^{O(1)} |\mathcal{N}|)$ space, where k is the rank of the matroid constraint.

The description of submodular functions and matroids can be exponential in the size of their ground sets, and therefore, it is important to define the way in which an algorithm may access them. We make the standard assumption that the algorithm has two oracles: a *value oracle* and an *independence oracle* which, given a set $S \subseteq \mathcal{N}$ of elements that are explicitly *stored* in the memory of the algorithm, returns the value of $f(S)$ and an indicator whether $S \in \mathcal{I}$, respectively.

2.3 Additional Notation and Definitions

Multilinear Extension

A set function $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ assigns values only to subsets of $\mathcal{N}$. If we think of a set S as equivalent to its characteristic vector $\mathbf{1}_S$ (a vector in $\{0,1\}^{\mathcal{N}}$ that has a value of 1 in every coordinate $u \in S$ and a value of 0 in the other coordinates), then we can view f as a function over the integral vectors in $[0,1]^{\mathcal{N}}$. It is often useful to extend f to general

[4] The assumption of non-negativity is necessary to allow multiplicative approximation guarantees.

[5] The similar term *streaming* algorithms often refers to algorithms whose space complexity is poly-logarithmic in the parameters of their input. Such algorithms are irrelevant for the problem we consider because they do not have enough space even for storing the output of the algorithm.

vectors in $[0,1]^{\mathcal{N}}$. There are multiple natural ways to do that. However, in this paper, we only need the multilinear extension F. Given a vector $x \in [0,1]^{\mathcal{N}}$, let $\mathcal{R}(x)$ denote a random subset of $\mathcal{N}$ including each element $u \in \mathcal{N}$ with probability x_u, independently. Then, $F(x) = \mathbb{E}[f(\mathcal{R}(x))] = \sum_{S \subseteq \mathcal{N}} [f(S) \cdot \prod_{u \in S} x_u \cdot \prod_{u \notin S} (1 - x_u)]$. One can observe that, as is implied by its name, the multilinear extension is a multilinear function. Thus, for every vector $x \in [0,1]^{\mathcal{N}}$, the partial derivative $\frac{\partial F}{\partial x_u}(x)$ is equal to $F(x + (1 - x_u) \cdot \mathbf{1}_u) - F(x - x_u \cdot \mathbf{1}_u)$. Note that in the last expression we have used $\mathbf{1}_u$ as a shorthand for $\mathbf{1}_{\{u\}}$. We often also use $\partial_u F(x)$ as a shorthand for $\frac{\partial F}{\partial x_u}(x)$. When f is submodular, its multilinear extension F is known to be concave along non-negative directions [5].

General Notation

Given a set $S \subseteq \mathcal{N}$ and an element $u \in \mathcal{N}$, we denote by $S + u$ and $S - u$ the expressions $S \cup \{u\}$ and $S \setminus \{u\}$, respectively. Additionally, given two vectors $x, y \in [0,1]^{\mathcal{N}}$, we denote by $x \vee y$ and $x \wedge y$ the coordinate-wise maximum and minimum operations, respectively.

Additional Definitions from Matroid Theory

Matroid theory is extensive, and we refer the reader to [28] for a more complete coverage of it. Here, we give only a few basic definitions from this theory that we employ below. Given a matroid $M = (\mathcal{N}, \mathcal{I})$, a set $S \subseteq \mathcal{N}$ is called *base* if it is an independent set that is maximal with respect to inclusion (i.e., every super-set of S is dependent), and it is called *cycle* if it is a dependent set that is minimal with respect to inclusion (i.e., every subset of S is independent). An element $u \in \mathcal{N}$ is called a loop if $\{u\}$ is a cycle. Notice that such elements cannot appear in any feasible solution for either `SMMatroid` or `MSMMatroid`, and therefore, one can assume without loss of generality that there are no loops in the ground set.

The rank of a set $S \subseteq \mathcal{N}$, denoted by $\text{rank}_M(S)$, is the maximum size of an independent set $T \in \mathcal{I}$ which is a subset of S. The subscript M is omitted when it is clear from the context. We also note that $\text{rank}_M(\mathcal{N})$ is exactly the rank of the matroid M (i.e., the maximum size of an independent set in M), and therefore, it is customary to define $\text{rank}(M) = \text{rank}_M(\mathcal{N})$. We say that a set $S \subseteq \mathcal{N}$ spans an element $u \in \mathcal{N}$ if adding u to S does not increase the rank of the set S, i.e., $\text{rank}(S) = \text{rank}(S + u)$ – observe the analogy between this definition and being spanned in a vector space. Furthermore, we denote by $\text{span}_M(S) \coloneqq \{u \in \mathcal{N} \mid \text{rank}(S) = \text{rank}(S + u)\}$ the set of elements that are spanned by S. Again, the subscript M is dropped when it is clear from the context.

3 Single-Pass Algorithm

In this section, we present our single-pass algorithm for the `Monotone Submodular Maximization subject to a Matroid Constraint` problem (`MSMMatroid`). The properties of the algorithm we present are given by the following theorem.

▶ **Theorem 1.** *There is a single-pass semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k that stores $O(k)$ elements, requires $\widetilde{O}(k)$ additional memory, and achieves an approximation guarantee of* 0.3178.

Our algorithm can be extended to the case in which the objective function is non-monotone (i.e., the `SMMatroid` problem) at the cost of obtaining a lower approximation factor, yielding the following theorem. However, for the sake of concentrating on our main new ideas, we devote this section to the algorithm of Theorem 1 and defer the proof of Theorem 5 to the full version.

▶ **Theorem 5.** *There is a single-pass semi-streaming algorithm for maximizing a non-negative (not necessarily monotone) submodular function subject to a matroid constraint of rank k that stores $O(k)$ elements, requires $\widetilde{O}(k)$ additional memory, and achieves an approximation guarantee of* 0.1921*.*

Throughout this section, we denote by $P_M \coloneqq \{x \in \mathbb{R}^{\mathcal{N}}_{\geq 0} \colon x(S) \leq \operatorname{rank}(S)\ \forall S \subseteq \mathcal{N}\}$ the matroid polytope of M. The algorithm we use to prove Theorem 1 appears as Algorithm 1. This algorithm gets a parameter $\varepsilon > 0$ and starts by initializing a constant α to be approximately the single positive value obeying $\alpha + 2 = e^{\alpha}$. We later prove that the approximation ratio guaranteed by the algorithm is at least $\frac{1}{\alpha+2} - \varepsilon$, which is better than the approximation ratio stated in Theorem 1 for a small enough ε. After setting the value of α, Algorithm 1 defines some additional constants m, c, and L using ε and α. We leave these variables representing different constants as such in the procedure and analysis, which allows for obtaining a better understanding later on of why these values are optimal for our analysis. We also note that, as stated, Algorithm 1 is efficient (i.e., runs in polynomial time) only if the multilinear extension and its partial derivatives can be efficiently evaluated. If that cannot be done, then one has to approximate F and its derivatives using Monte-Carlo simulation, which is standard practice (see, for example, [5]). We omit the details to keep the presentation simple, but we note that, as in other applications of this standard technique, the incurred error can easily be kept negligible, and therefore, does not affect the guarantee stated in Theorem 1.

Algorithm 1 uses sets A_i and vectors $a_i \in [0,1]^{\mathcal{N}}$ for certain indices $i \in \mathbb{Z}$. Throughout the algorithm, we only consider finitely many indices $i \in \mathbb{Z}$. However, we do not know upfront which indices within $\mathbb{Z}$ we will use. To simplify the presentation, we therefore use the convention that whenever the algorithm uses for the first time a set A_i or vector a_i, then A_i is initialized to be $\varnothing$ and a_i is initialized to be the zero vector. The largest index ever used in the algorithm is q, which is computed toward the end of the algorithm at Line 13.

For each $i \in \mathbb{Z}$, the set A_i is an independent set consisting of elements u that already arrived and for which the marginal increase with respect to a reference vector a (at the moment when u arrives) is at least c^i. More precisely, whenever a new element $u \in \mathcal{N}$ arrives and its marginal return $\partial_u F(a)$ exceeds c^i for an index $i \in \mathbb{Z}$ in a relevant range, then we add u to A_i if $A_i + u$ remains independent. When adding u to A_i, we also increase the u-entry of the vector a_i by $\frac{c^i}{m \cdot \partial_u F(a)}$. The vector a built up during the algorithm has two key properties. First, its multilinear value approximates $f(\mathrm{OPT})$ up to a constant factor. Second, one can derive from the sets A_i a vector s (see Algorithm 1) such that $F(s)$ is close to $F(a)$ and s is contained in the matroid polytope $P_{\mathcal{M}}$.

Whenever an element $u \in \mathcal{N}$ arrives, the algorithm first computes the largest index $i(u) \in \mathbb{Z}$ fulfilling $c^{i(u)} \leq \partial_u F(a)$. It then updates sets A_i and vectors a_i for indices $i \leq i(u)$. Purely conceptually, the output of the algorithm would have the desired guarantees even if all infinitely many indices below $i(u)$ where updated. However, to obtain an algorithm running in finite (even polynomial) time and linear memory, we do not consider indices below $\max\{b, i(u) - \operatorname{rank}(M) - L\}$ in the update step. Capping the considered indices like this has only a minor impact in the analysis since the contribution of the vectors a_i to the multilinear extension value of the vector a is geometrically decreasing with decreasing index i.

In the algorithm, and also in its analysis, we sometimes use sums over indices that go up to ∞. However, whenever this happens, beyond some finite index, all terms are zero. Hence, such sums are well defined.

Finally, we provide details on the return statement in Line 17 of the algorithm. This statement is based on a fact stated in [5], namely that a point in the matroid polytope can be rounded losslessly to an independent set. More formally, given any point $y \in P_{\mathcal{M}}$ in the

Algorithm 1 Single-Pass Semi-Streaming Algorithm for `MSMMatroid`.

Set $\alpha = 1.1462$, $m = \left\lceil \frac{3\alpha}{\varepsilon} \right\rceil$, $c = \frac{m}{m-\alpha}$, and $L = \left\lceil \log_c\left(\frac{2c}{\varepsilon(c-1)}\right) \right\rceil$.
Set $a = 0 \in [0,1]^{\mathcal{N}}$ to be the zero vector, and let $b = -\infty$.
for every element arriving $u \in \mathcal{N}$, if $\partial_u F(a) > 0$ **do**
    Let $i(u) = \lfloor \log_c(\partial_u F(a)) \rfloor$. ▷ Thus, $i(u)$ is largest index $i \in \mathbb{Z}$ with $c^i \leq \partial_u F(a)$.
    for $i = \max\{b, i(u) - \operatorname{rank}(M) - L\}$ **to** $i(u)$ **do**
        if $A_i + u \in \mathcal{I}$ **then**
            $A_i \leftarrow A_i + u$.
            $a_i \leftarrow a_i + \frac{c^i}{m \cdot \partial_u F(a)} \mathbf{1}_u$.
    Set $b \leftarrow h - L$, where h is largest index $i \in \mathbb{Z}$ satisfying $\sum_{j=i}^{\infty} |A_j| \geq \operatorname{rank}(M)$.
    $a \leftarrow \sum_{i=b}^{\infty} a_i$.
    Delete from memory all sets A_i and vectors a_i with $i \in \mathbb{Z}_{<b}$.
Set $S_k \leftarrow \varnothing$ for $k \in \{0, \ldots, m-1\}$.
Let q be largest index $i \in \mathbb{Z}$ with $A_i \neq \varnothing$.
for $i = q$ **to** b (stepping down by 1 at each iteration) **do**
    while $\exists u \in A_i \setminus S_{(i \bmod m)}$ with $S_{(i \bmod m)} + u \in \mathcal{I}$ **do**
        $S_{(i \bmod m)} \leftarrow S_{(i \bmod m)} + u$.
return a rounding $R \in \mathcal{I}$ of the fractional solution $s := \frac{1}{m} \sum_{k=0}^{m-1} \mathbf{1}_{S_k}$ with $f(R) \geq F(s)$.

matroid polytope, there is an independent set $I \in \mathcal{I}$ with $f(I) \geq F(y)$. Moreover, assuming that the multilinear extension F can be evaluated efficiently, such an independent set I can be computed efficiently. As before, if one is only given a value oracle for f, then the exact evaluation of F can be replaced by a strong estimate obtained through Monte-Carlo sampling, leading to a randomized algorithm to round y to an independent set I with $f(I) \geq (1-\delta)F(y)$ for an arbitrarily small constant $\delta > 0$.

Due to space constraints, the analysis of Algorithm 1 is deferred to the full version.

4 Framework for Multi-pass Algorithms

In this section we present the details of the framework used to prove our $(1 - 1/e)$-approximation results (Theorems 2 and 3). We remind the reader that the proofs of these theorems (using the framework) can be found in Sections 5 and 7, respectively. Badanidiyuru and Vondrák [3] described an algorithm called "Accelerated Continuous Greedy" that obtains an approximation guarantee of $1 - 1/e - O(\varepsilon)$ for `MSMMatroid` for every $\varepsilon \in (0, 1)$. Their algorithm is not a data stream algorithm, but it enjoys the following nice properties.

- The algorithm includes a procedure called "Decreasing-Threshold Procedure". This procedure is the only part of the algorithm that directly accesses the input.
- The Decreasing-Threshold Procedure is called $O(\varepsilon^{-1})$ times during the execution of the algorithm.
- In addition to the space used by this procedure, Accelerated Continuous Greedy uses only space that is linear in the space necessary to store the outputs of the various executions of the Decreasing-Threshold Procedure.
- The Decreasing-Threshold Procedure returns a base D of M after every execution, and this base is guaranteed to obey Equation (1) stated below. The analysis of the approximation ratio of Accelerated Continuous Greedy treats Decreasing-Threshold Procedure as a black box except for the fact that its output is a base D of M obeying Equation (1),

and therefore, this analysis will remain valid even if Decreasing-Threshold Procedure is replaced by any other algorithm with the same guarantee. Furthermore, one can verify that the analysis continues to work (with only minor technical changes) even if the output D of the replacing algorithm obeys Equation (1) only in expectation.

Let us now formally state the property that the output base of Decreasing-Threshold Procedure obeys. Let P_M be the matroid polytope of M, and let F be the multilinear extension of f. Decreasing-Threshold Procedure gets as input a point $x \in (1-\varepsilon) \cdot P_M$, and its output base D is guaranteed to obey

$$F(x') - F(x) \geq \varepsilon[(1-3\varepsilon) \cdot f(\mathrm{OPT}) - F(x')] \ , \tag{1}$$

where $x' = x + \varepsilon \cdot \mathbf{1}_D$ and OPT denotes an optimal solution.

Our objective in Sections 5 and 7 is to describe semi-streaming algorithms that can function as replacements for the offline procedure Decreasing-Threshold Procedure. The next proposition states that plugging such a replacement into Accelerated Continuous Greedy yields a roughly $(1-1/e)$-approximation semi-streaming algorithm.

▶ **Proposition 6.** *Assume there exists a semi-streaming algorithm that given a point $x \in (1-\varepsilon) \cdot P_M$ makes p passes over the input stream, stores $O(k/\varepsilon)$ elements, and outputs a base D obeying Equation (1) in expectation. Then, there exists a semi-streaming algorithm for maximizing a non-negative monotone submodular function subject to a matroid constraint of rank k that stores $O(k/\varepsilon)$ elements, makes $O(p/\varepsilon)$ many passes and achieves an approximation guarantee of $1-1/e-\varepsilon$.*

Proof. Observe that the proposition is trivial when $\varepsilon \geq 1-1/e$, and therefore, we assume below that $\varepsilon < 1-1/e$. Furthermore, for simplicity, we describe an algorithm with an approximation ratio of $1-1/e-O(\varepsilon)$ rather than a clean ratio of $1-1/e-\varepsilon$. However, one can switch between the two ratios by scaling ε by an appropriate constant.

Let us denote by ALG the algorithm whose existence is promised by the statement of the proposition, and consider an algorithm called `Data Stream Continuous Greedy` (`DSCG`) obtained from the Accelerated Continuous Greedy algorithm of [3] when every execution of the Decreasing-Threshold Procedure by the last algorithm is replaced with an execution of ALG. We explain below why `DSCG` has all the properties guaranteed by the proposition. We begin by recalling that since the approximation ratio analysis of Accelerated Continuous Greedy in [3] treats the Decreasing-Threshold Procedure as a black box that in expectation has the guarantee stated in Equation (1), and ALG also has this guarantee, this analysis can be applied as is also to `DSCG`, and therefore, `DSCG` is a $(1-1/e-O(\varepsilon))$-approximation algorithm.

Recall now that Accelerated Continuous Greedy accesses its input only through the Decreasing-Threshold Procedure, which implies that `DSCG` is a data stream algorithm just like ALG. Furthermore, since Accelerated Continuous Greedy accesses the Decreasing-Threshold Procedure $O(\varepsilon^{-1})$ times, the number of passes used by `DSCG` is larger by a factor of $O(\varepsilon^{-1})$ compared to the number of passes used by ALG (which is denoted by p). Hence, `DSCG` uses $O(p/\varepsilon)$ passes.

It remains to analyze the space complexity of `DSCG`. Since Accelerated Continuous Greedy uses space of size linear in the space necessary to keep the $O(\varepsilon^{-1})$ bases that it receives from the Decreasing-Threshold Procedure, the space complexity of `DSCG` is larger than the space complexity of the semi-streaming algorithm ALG only by an additive term of $\widetilde{O}(k/\varepsilon)$. As this term is nearly linear in k for any constant ε, we get that `DSCG` has a low enough space

complexity to be a semi-streaming algorithm. Furthermore, since the $O(\varepsilon^{-1})$ bases that DSCG gets from ALG can include only $O(k/\varepsilon)$ elements, this expression upper bounds the number of elements stored by DSCG in addition to the $O(k/\varepsilon)$ elements stored by ALG itself. ◀

It turns out that one natural way to get a base D obeying Equation (1) is to output a local maximum with respect to the objective function $g(S) = F(x + \varepsilon \cdot \mathbf{1}_S)$ (i.e., a base D whose value with respect to this objective cannot be improved by replacing an element of D with an element of $\mathcal{N} \setminus D$). Getting such a maximum using a semi-streaming algorithm with a reasonable number of passes is challenging; however, one can define weaker properties that still allow us to get Equation (1). Specifically, for any $\varepsilon \in (0, 1)$, we say that a set D is an *ε-approximate local maximum* with respect to g if

$$g(D \mid \varnothing) \geq g(B \mid D) + \sum_{u \in B \cap D} g(u \mid D - u) - \varepsilon \cdot g(\mathrm{OPT}_g \mid \varnothing)$$

for every base B of M, where OPT_g is a base maximizing g. (Intuitively, one should think of B as being the optimal solution with respect to f.)

One property of an approximate local maximum is that its value (with respect to g) is an approximation to $g(\mathrm{OPT}_g)$.

▶ **Observation 7.** *For every $\varepsilon \in (0, 1)$, if D is an ε-approximate local maximum with respect to g, then $g(D) \geq \frac{1-\varepsilon}{2} \cdot g(OPT_g)$.*

Proof. One can verify that the non-negativeity, monotonicity and submodularity of f implies that g also has these properties. Therefore,

$$\begin{aligned} g(D) &\geq g(D \mid \varnothing) \geq g(OPT_g \mid D) + \sum_{u \in \mathrm{OPT}_g \cap D} g(u \mid D - u) - \varepsilon \cdot g(\mathrm{OPT}_g \mid \varnothing) \\ &\geq g(\mathrm{OPT}_g \mid D) - \varepsilon \cdot g(OPT_g \mid \varnothing) \geq (1 - \varepsilon) \cdot g(\mathrm{OPT}_g) - g(D) \ , \end{aligned}$$

where the first inequality holds by the non-negativity of g, the second inequality follows from the fact that D is an ε-approximate local maximum (for $B = \mathrm{OPT}_g$), the third inequalities follow from the monotonicity of g, and the last inequality hold by g's non-negativity and monotonicity. Rearranging the above inequality now yields the observation. ◀

Using the last observation we can prove that any approximate local maximum with respect to g obeys Equation (1), and the same holds also for any solution that is almost as good as some approximate local maximum.

▶ **Lemma 8.** *For every $\varepsilon \in (0, 1)$, if D' is an ε-approximate local maximum with respect to g, then any (possibly randomized) set D such that $\mathbb{E}[g(D \mid \varnothing)] \geq (1 - \varepsilon) \cdot g(D' \mid \varnothing)$ obeys Equation (1) in expectation. In particular, this is the case for $D = D'$ since the monotonicity of f implies that g is non-negative.*

Proof. We need to consider two cases. The simpler case is when $g(\mathrm{OPT}_g \mid \varnothing) \geq 2\varepsilon \cdot f(\mathrm{OPT})$, where we recall that OPT is an optimal base with respect to f. Since $x' = x + \varepsilon \cdot \mathbf{1}_D$ by definition, we get in this case

$$\begin{aligned} \mathbb{E}[F(x')] - F(x) &= \mathbb{E}[F(x + \varepsilon \cdot \mathbf{1}_D)] - F(x) = \mathbb{E}[g(D \mid \varnothing)] \geq (1 - \varepsilon) \cdot g(D' \mid \varnothing) \\ &\geq \tfrac{(1-\varepsilon)^2}{2} g(\mathrm{OPT}_g \mid \varnothing) \geq \varepsilon(1 - 2\varepsilon) \cdot f(\mathrm{OPT}) \geq \varepsilon((1 - 3\varepsilon) \cdot f(\mathrm{OPT}) - \mathbb{E}[F(x')]) \ , \end{aligned}$$

where the second inequality holds by Observation 7.

In the rest of the proof we consider the case of $g(\mathrm{OPT}_g \mid \varnothing) \leq 2\varepsilon \cdot f(\mathrm{OPT})$. We note that, in this case,

$$\begin{aligned}\frac{\mathbb{E}[F(x')] - F(x)}{1-\varepsilon} = \frac{\mathbb{E}[F(x+\varepsilon\cdot\mathbf{1}_D)] - F(x)}{1-\varepsilon} &= \frac{\mathbb{E}[g(D \mid \varnothing)]}{1-\varepsilon} \geq g(D' \mid \varnothing)\\ &\geq g(\mathrm{OPT} \mid D') + \sum_{u\in \mathrm{OPT}\cap D'} g(u \mid D'-u) - \varepsilon\cdot g(\mathrm{OPT}_g \mid \varnothing)\\ &\geq g(\mathrm{OPT} \mid D') + \sum_{u\in \mathrm{OPT}\cap D'} g(u \mid D'-u) - 2\varepsilon^2\cdot f(\mathrm{OPT})\ ,\end{aligned}$$

where the second inequality holds since D' is an ε-approximate local maximum (by plugging $B = \mathrm{OPT}$ into the definition of such maxima). Let us now further develop the first two terms on the rightmost side of the last inequality. By the submodularity and monotonicity of f, if we denote $y = x + \varepsilon\cdot\mathbf{1}_{D'}$, then

$$\begin{aligned}&g(\mathrm{OPT} \mid D') + \sum_{u\in \mathrm{OPT}\cap D'} g(u \mid D'-u)\\ &\quad= F(x+\varepsilon\cdot\mathbf{1}_{\mathrm{OPT}\cup D'}) - F(x+\varepsilon\cdot\mathbf{1}_{D'}) + \sum_{u\in \mathrm{OPT}\cap D'} [F(x+\varepsilon\cdot\mathbf{1}_{D'}) - F(x+\varepsilon\cdot\mathbf{1}_{D'-u})]\\ &\quad\geq F(y+\varepsilon\cdot\mathbf{1}_{\mathrm{OPT}\setminus D'}) - F(y) + \sum_{u\in \mathrm{OPT}\cap D'} [F((y+\varepsilon\cdot\mathbf{1}_{\{u\}})\wedge\mathbf{1}_{\mathcal{N}}) - F(y)]\\ &\quad\geq F((y+\varepsilon\cdot\mathbf{1}_{\mathrm{OPT}})\wedge\mathbf{1}_{\mathcal{N}}) - F(y)\ .\end{aligned}$$

Combining the last two inequalities yields

$$\begin{aligned}\mathbb{E}[F(x')] - F(x) &\geq (1-\varepsilon)[F((y+\varepsilon\cdot\mathbf{1}_{\mathrm{OPT}})\wedge\mathbf{1}_{\mathcal{N}}) - F(y)] - 2\varepsilon^2\cdot f(\mathrm{OPT})\\ &\geq (1-\varepsilon)[F(y+\varepsilon((\mathbf{1}_{\mathcal{N}}-y)\wedge\mathbf{1}_{\mathrm{OPT}})) - F(y)] - 2\varepsilon^2\cdot f(\mathrm{OPT})\\ &\geq \varepsilon(1-\varepsilon)[F(y\vee\mathbf{1}_{\mathrm{OPT}}) - F(y)] - 2\varepsilon^2\cdot f(\mathrm{OPT})\\ &\geq \varepsilon((1-\varepsilon)f(\mathrm{OPT}) - \mathbb{E}[F(x')]) - 2\varepsilon^2\cdot f(\mathrm{OPT})\\ &= \varepsilon\cdot((1-3\varepsilon)f(\mathrm{OPT}) - \mathbb{E}[F(x')])\ ,\end{aligned}$$

where the second inequality holds by the monotonicity of f, the third inequality holds because the submodularity of f guarantees that F is concave along non-negative directions (such as $(\mathbf{1}_{\mathcal{N}} - y)\wedge\mathbf{1}_{\mathrm{OPT}}$) and the last inequality holds by the motonicity of f and the observation that

$$F(y) = g(D') = g(\varnothing) + g(D' \mid \varnothing) \leq g(\varnothing) + \frac{\mathbb{E}[g(D \mid \varnothing)]}{1-\varepsilon} \leq \frac{\mathbb{E}[g(D)]}{1-\varepsilon} = \frac{\mathbb{E}[F(x')]}{1-\varepsilon}\ . \qquad \blacktriangleleft$$

In Section 5 we describe a semi-streaming algorithm that can be used to find an ε-approximate local maximum of a non-negative monotone submodular function. By applying this algorithms to g, we get (via Lemma 8) an algorithm having all the properties assumed by Proposition 6; which proves Theorem 2. In Section 7 we attempt to use the same approach to get a result for random order streams. However, in this setting we are not able to guarantee an ε-approximate local maximum. Instead, we design an algorithm whose output has in expectation a value that is almost as good as the value of the worst approximate local maximum. This leads to a proof of Theorem 3.

5 Approximate Local Maximum for Adversarial Streams

In this section we prove following proposition, which guarantees the existence of a semi-streaming multi-pass algorithm for finding an ε-approximate local maximum in adversarial streams, i.e., when the order of the elements in the input stream is arbitrary. We note that this section highly depends on Section 4, and should not be read before that section.

▶ **Proposition 9.** *For every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm that given an instance of* `MSMMatroid` *with a matroid of rank k stores $O(k)$ elements, makes $O(\varepsilon^{-2})$ many passes, and outputs an ε-approximate local maximum.*

By Lemma 8 and Proposition 6, the last proposition implies Theorem 2. Therefore, we concentrate in this section on proving Proposition 9. The first data stream algorithm for `MSMMatroid` was described by Chakrabarti and Kale [6]. The first step towards proving Proposition 9 is a re-analysis of a variant of this algorithm that was described by Huang, Thiery and Ward [18] (based on ideas of Chekuri et al. [8]). The following proposition summarizes the properties of this variant that we prove in this re-analysis. Due to space constraints, the proof of this proposition is deferred to the full version.

▶ **Proposition 10.** *There exists a single-pass semi-streaming algorithm that given a base S_0 of M and value $c > 1$ outputs a base S_n that obeys $(c-1) \cdot f(S_n \mid \varnothing) + \frac{3c-2}{c-1}[f(S_n) - f(S_0)] \geq f(B \mid S_0 \setminus B) - f(S_0 \mid \varnothing) \geq f(B \mid S_0) + \sum_{u \in B \cap S_0} f(u \mid S_0 - u) - f(S_0 \mid \varnothing)$ for every base B of M. Furthermore, this algorithm stores $O(k)$ elements at any point during its execution.*

Below we refer to the algorithm whose existence is guaranteed by Proposition 10 as `SinglePass`. Next, we would like to show that `SinglePass` can be used to get an ε-approximate local maximum. The algorithm we use to do that is given as Algorithm 2, and it gets $\varepsilon \in (0, 1)$ as a parameter.

Algorithm 2 Multiple Local Search Passes (ε).

1: Find a base T_0 of M using a single pass (by simply initializing T_0 to be the empty set, and then adding to it any elements that arrives and can be added to T_0 without violating independence in M).
2: Let T_1 be the output of `SinglePass` when given $S_0 = T_0$ and $c = 2$.
3: **for** $i = 2$ **to** $2 + \lceil 40\varepsilon^{-2} \rceil$ **do**
4: Let T_i be the output of `SinglePass` when given $S_0 = T_{i-1}$ and $c = 1 + \varepsilon/2$.
5: **if** $f(T_i) - f(T_{i-1}) \leq \varepsilon^2/10 \cdot f(T_1 \mid \varnothing)$ **then**
6: **return** T_{i-1}.
7: Indicate failure if the execution of the algorithm has arrived to this point.

Intuitively, Algorithm 2 works by employing the fact that every execution of `SinglePass` increases the value of its input base T_{i-1} significantly, unless this input base is close to being a local maximum, and therefore, if the execution produces a base T_i which is not much better than T_{i-1}, then we know that T_{i-1} is an ε-approximate local maximum. The following lemma proves this formally.

▶ **Lemma 11.** *If Algorithm 2 does not indicate a failure, then its output set T obeys $f(B \mid T) + \sum_{u \in B \cap T} f(u \mid T - u) - f(T \mid \varnothing) < \varepsilon \cdot f(\mathrm{OPT} \mid \varnothing)$ for every base B of $\mathcal{M}$. Note that the last inequality implies that T is an ε-approximate local maximum with result to f.*

Proof. Since T_1 is a base of M, $f(T_1 \mid \varnothing) = f(T_1) - f(\varnothing) \leq f(\mathrm{OPT}) - f(\varnothing) = f(\mathrm{OPT} \mid \varnothing)$. This implies that when Algorithm 2 returns a set T_{i-1}, then

$$f(T_i) - f(T_{i-1}) \leq (\varepsilon^2/10) \cdot f(\mathrm{OPT} \mid \varnothing) \ .$$

Plugging this inequality and the fact that $f(T_i \mid \varnothing) \leq f(\mathrm{OPT} \mid \varnothing)$ (because T_i is a base of M) into the guarantee of Proposition 10 for the execution of `SinglePass` that has created T_i yields

$$\begin{aligned}\varepsilon \cdot f(\mathrm{OPT} \mid \varnothing) &\geq (\varepsilon/2) \cdot f(\mathrm{OPT} \mid \varnothing) + \varepsilon(3\varepsilon/2+1)/5 \cdot f(\mathrm{OPT} \mid \varnothing) \\ &\geq (\varepsilon/2) \cdot f(T_i \mid \varnothing) + \tfrac{3\varepsilon/2+1}{\varepsilon/2} \cdot [f(T_i) - f(T_{i-1})] \\ &\geq f(B \mid T_{i-1}) + \sum_{u \in B \cap T_{i-1}} f(u \mid T_{i-1} - u) - f(T_{i-1} \mid \varnothing) \ . \end{aligned}$$ ◀

One could imagine that it is possible for the value of the solution maintained by Algorithm 2 to increase significantly following every iteration of the loop starting on Line 3, which will result in the algorithm indicating failure rather than ever returning a solution. However, it turns out that this cannot happen because the value of the solution of Algorithm 2 cannot exceed $f(\mathrm{OPT})$, which implies a bound on the number of times this value can be increased significantly. This idea is formalized by the next two claims.

▶ **Observation 12.** $f(T_1 \mid \varnothing) \geq \frac{1}{5} f(\mathrm{OPT} \mid \varnothing)$.

Proof. If we set $B = \mathrm{OPT}$, then by applying Proposition 10 to the execution of `SinglePass` on Line 2 of Algorithm 2, we get

$$\begin{aligned} f(T_1 \mid \varnothing) + 4[f(T_1) - f(T_0)] &\geq f(\mathrm{OPT} \mid T_0) + \sum_{u \in \mathrm{OPT} \cap T_0} f(u \mid T_0 - u) - f(T_0 \mid \varnothing) \\ &\geq f(\mathrm{OPT} \mid T_0) - f(T_0 \mid \varnothing) \ , \end{aligned}$$

where the second inequality follows from the monotonicity of f. Since the leftmost side the last inequality is equal to $5f(T_1 \mid \varnothing) - 4f(T_0 \mid \varnothing)$, this inequality implies

$$\begin{aligned} 5f(T_1 \mid \varnothing) &\geq f(\mathrm{OPT} \mid T_0) + 3f(T_0 \mid \varnothing) = f(\mathrm{OPT} \cup T_0) + 2f(T_0) - 3f(\varnothing) \\ &\geq f(\mathrm{OPT}) - f(\varnothing) = f(\mathrm{OPT} \mid \varnothing) \ , \end{aligned}$$

where the second inequality follows again from the monotonicity of f. The observation now follows by dividing the last inequality by 5. ◀

▶ **Lemma 13.** *Algorithm 2 never indicates failure.*

Proof. If $f(\mathrm{OPT} \mid \varnothing) = 0$, then the value of every base of M according to f is $f(\varnothing)$, which guarantees that Algorithm 2 returns T_1 during the first iteration of the loop starting on its Algorithm 2. Therefore, we assume below that $f(\mathrm{OPT} \mid \varnothing) > 0$. Furthermore, assume towards a contradiction that Algorithm 2 indicates failure. By Observation 12, this assumption implies that the value of the solution maintained by Algorithm 2 increases by at least $(\varepsilon^2/10) \cdot f(T_1 \mid \varnothing) \geq \frac{\varepsilon^2}{50} f(\mathrm{OPT} \mid \varnothing)$ after every iteration of the loop starting on Line 3. Therefore, after all the $1 + \lceil 40\varepsilon^{-2} \rceil$ iterations of this loop, the value of the solution of Algorithm 2 is at least

$$f(T_1) + (1 + \lceil 40\varepsilon^{-2} \rceil) \cdot \tfrac{\varepsilon^2}{50} f(\mathrm{OPT} \mid \varnothing) > f(\varnothing) + \tfrac{1}{5} f(\mathrm{OPT} \mid \varnothing) + \tfrac{4}{5} f(\mathrm{OPT} \mid \varnothing) = f(\mathrm{OPT}) \ ,$$

which is a contradiction since the solution of Algorithm 2 is always kept as a base of M. ◀

We now observe that Algorithm 2 has all the properties guaranteed by Proposition 9. In particular, we note that Algorithm 2 can be implemented as a semi-streaming algorithm storing $O(k)$ elements because it needs to store at most two solutions at any given time in addition to the elements and space required by `SinglePass`.

6 Discussion of a Lower Bound by McGregor and Vu [22]

McGregor and Vu [22] showed that any data stream algorithm for the Maximum k-Coverage Problem (which is a special case of `MSMMatroid` in which f is a coverage function and M is a uniform matroid of rank k) that makes a constant number of passes must use $\Omega(m/k^2)$ memory to achieve $(1+\varepsilon)\cdot(1-(1-1/k)^k)$-approximation with probability at least 0.99, where m is the number of sets in the input, and it is assumed that these sets are defined over a ground set of size $n = \Omega(\varepsilon^{-2}k\log m)$. Understanding the implications of this lower bound for `MSMMatroid` requires us to handle two questions.

- The first question is how the lower bound changes as a function of the number of passes. It turns out that when the number of passes is not dropped from the asymptotic expressions because it is considered to be a constant, the lower bound of McGregor and Vu [22] on the space complexity becomes $\Omega(m/(pk^2))$, where p is the number of passes done by the algorithm.
- The second question is about the modifications that have to be done to the lower bound when it is transferred from the Maximum k-Coverage Problem to `MSMMatroid`. Such modifications might be necessary because of input representation issues. However, as it turns out, the proof of the lower bound given by [22] can be applied to `MSMMatroid` directly, yielding the same lower bound (except for the need to replace m with the corresponding value in `MSMMatroid`, namely, $|\mathcal{N}|$). Furthermore, McGregor and Vu [22] had to use a very large ground set so that random sets will behave as one expects with high probability. When the objective function is a general submodular function, rather than a coverage function, it can be chosen to display the above-mentioned behavior of random sets, and therefore, ε can be set to 0.

We summarize the above discussion in the following corollary.

▶ **Corollary 14** (Corollary of McGregor and Vu [22]). *For any $k \geq 1$, any p-pass data stream algorithm for `MSMMatroid` that achieves an approximation guarantee of $1-(1-1/k)^k \leq 1-1/e+1/k$ with probability at least 0.99 must use $\Omega(|\mathcal{N}|/(pk^2))$ memory, and this is the case even when the matroid M is restricted to be a uniform matroid of rank k.*

7 Approximate Local Maximum for Random Streams

In this section we study `MSMMatroid` in random order streams by building on ideas from the analysis of Liu et al. [21] for optimizing f under a cardinality constraint. We begin with simplifying and reanalyzing the single-pass local search algorithm of Shadravan [29]. By applying this algorithm multiple times (in multiple passes), we are able to prove the following proposition. Proposition 15 implies Theorem 3 by Lemma 8 and Proposition 6.

▶ **Proposition 15.** *For every constant $\varepsilon > 0$, there is a multi-pass semi-streaming algorithm that given an instance of `MSMMatroid` with a matroid of rank k stores $O(k/\varepsilon)$ elements and makes $O(\varepsilon^{-1}\log\varepsilon^{-1})$ many passes. Assuming the order of the elements in the input stream is chosen uniformly at random in each pass, this algorithm outputs a solution D such that $\mathbb{E}[f(D \mid \varnothing)] \geq (1-\varepsilon)\cdot f(D' \mid \varnothing)$, where D' is the ε-approximate local maximum whose value with respect to f is the smallest.*

In the full versionwe show that our single-pass algorithm can naturally be extented to p-matchoids. Then, we create a multi-pass algorithm based on this extended single-pass algorithm, which proves Theorem 4.

Intuitively, a local search algorithm should make a swap in its solution whenever this is beneficial. In the adversarial setting, one has to make a swap only when it is beneficial enough to avoid making too many negligible swaps. However, in the random order setting there is a better solution for this problem. Specifically, we (randomly) partition the input stream into *windows* (αk contiguous chunks of the stream with expected size $n/(\alpha k)$ each for some parameter $\alpha > 1$), and then make the best swap within each window. Formally, our random partition is generated according to Algorithm 3.

Algorithm 3 Partitioning of the input stream (α).

Draw $|\mathcal{N}|$ integers uniformly and independently from $1, 2, \ldots, \alpha k$.
for $i = 1$ to αk **do**
    Let $n_i \leftarrow$ # of integers equal to i.
    Let $t_i \leftarrow \sum_{j=1}^{i-1} n_i$.
    Let $w_i \leftarrow$ elements $t_i + 1$ to $t_i + n_i$ in $\mathcal{N}$.
return $\{w_1, w_2, \ldots, w_{\alpha k}\}$.

Our full single pass algorithm, which uses the partition defined by Algorithm 3, is given as Algorithm 4. The input for the algorithm includes the parameter α and some base L_0 of the matroid $\mathcal{M}$. Additionally, during the execution of the algorithm, the set L_i represents the current solution, and H is the set of all elements that were added to this solution at some point. When processing window w_i, Algorithm 4 constructs a set C_i of elements that can potentially be swapped into the solution. This set contains all the elements of the window plus some historical elements (the set R_i). The idea of using a set H to store previously valuable elements is inspired from [1, 21]. Reintroducing previously seen elements allows us to give any element not in the solution a chance of being introduced into the solution in the future, which helps us avoid issues that result from the dependence that exists between the current solution and the set of elements in the current window.

Algorithm 4 MatroidStream(α, L_0).

Partition $\mathcal{N}$ into windows $w_1, w_2, \ldots, w_{\alpha k}$.
Let $H \leftarrow \varnothing$.
for $i = 1$ to αk **do**
    Let R_i be a random subset of H including every $u \in H$ with probability $\frac{1}{\alpha k}$, independently.
    Let $C_i \leftarrow w_i \cup R_i$
    Let $u^\star$ and $u_r^\star$ be elements maximizing $f(L_i - u_r^\star + u^\star)$ subject to the constraints: $u^\star \in C_i$, $u_r^\star \in L_i$ and $L_i - u_r^\star + u^\star \in \mathcal{I}$.
    if $f(L_i) < f(L_i - u_r^\star + u^\star)$ **then**
        Update $H \leftarrow H + u^\star$.
        Let $L_{i+1} \leftarrow L_i - u_r^\star + u^\star$.
return $L_{\alpha k}$.

Note that the number of elements stored by Algorithm 4 is $O(\alpha k)$, as this number is dominated by the size of the set H. For the same reason Algorithm 4 is a semi-streaming algorithm whenever α is constant.

▶ **Definition 16.** *Let H_i denote the state of the set H maintained by Algorithm 4 immediately after processing window i. We define $\mathcal{H}_i$ to be the set of all pairs (u, j) such that element $u \in H_i$ was added to the solution while window j was processed (i.e., $u \in H_i \cap w_j$). For convenience, sometimes we treat $\mathcal{H}_i$ as a set of elements, and say that $u \in \mathcal{H}_i$ if $u \in H_i$.*

One can observe that $\mathcal{H}_i$ encodes all the changes that the algorithm made to its state while processing the first i windows because the element removed from the solution when u is added is deterministic. Additionally, we note that different random permutations of the input and random coins in Line 4 of Algorithm 4 may produce the same history, and we average over all of them in the analysis.

The next lemma is from [21]. It captures the intuition that any element not selected by the algorithm still appears uniformly distributed in future windows, and bounds the probability with which this happens. The proof of this lemma can be found in the full version.

▶ **Lemma 17.** *Fix a history $\mathcal{H}_{i-1}$ for some $i \in [\alpha k]$. For any element $u \in \mathcal{N} \setminus \mathcal{H}_{i-1}$, and any $i \leq j \leq \alpha k$, we have $Pr[u \in w_j \mid \mathcal{H}_{i-1}] \geq 1/(\alpha k)$.*

Let B an arbitrary base of $\mathcal{M}$ (one can think of B as an optimal solution because the monotonicity of f guarantees that some optimal solution is a base, but we sometimes need to consider other bases as B). We now define "active" windows, which are windows for which we can show a definite gain in our solution. Specifically, we show below that in any active window the value of the current solution L increases roughly by $\frac{1}{k}(f(B) - 2f(L))$ in expectation, which yields an approximation ratio of $\frac{1}{2}(1 - 1/e^2)$ after αk windows have been processed in one pass because we expect roughly one in every α windows to be active.

▶ **Definition 18.** *For window w_i, let p_u^i be the probability that $u \in w_i$ conditioned $\mathcal{H}_{i-1}$. Define the active set A_i of w_i to be the union of R_i and a set obtained by sampling each element $u \in w_i$ with probability $1/(\alpha k p_u^i)$. We call w_i an **active window** if $|B \cap A_i| \geq 1$.*

Note that the construction of active sets in Definition 18 is valid as Lemma 17 guarantees that $1/(\alpha k p_e^i)$ is a valid probability (i.e., it is not more than 1). More importantly, the active set A_i includes every element of $\mathcal{N}$ with probability exactly $1/(\alpha k)$, even conditioned on the history $\mathcal{H}_{i-1}$; which implies that, since each element appears in A_i independently, a window is active with probability $(1 - 1/(\alpha k))^k \geq 1 - e^{-1/\alpha} \approx 1/\alpha$ conditioned on any such history. Let $\mathcal{A}_i$ denote the event that window i is active. The following lemma lower bounds the increase in the value of the solution of Algorithm 4 in an active window.

▶ **Lemma 19.** *For every integer $0 \leq i < \alpha k$,*

$$\begin{aligned}\mathbb{E}[f(L_{i+1}) - f(L_i) \mid \mathcal{H}_i, \mathcal{A}_{i+1}] &\geq \tfrac{1}{k}\mathbb{E}\Big[f(B \mid L_i) + \sum\nolimits_{u \in B \cap L_i} f(u \mid L_i - u) - f(L_i \mid \varnothing) \;\Big|\; \mathcal{H}_i\Big] \\ &\geq \tfrac{1}{k}\mathbb{E}[f(B) - 2f(L_i) \mid \mathcal{H}_i] \ .\end{aligned}$$

Moreover, the above inequality holds even when B is a random base as long as it is deterministic when conditioned on any given $\mathcal{H}_i$.

Lemma 19 completes the statement of the properties of Algorithm 4 that we need to prove our results. Specifically, the first inequality of the lemma is used to prove that multiple "concatenated" executions of Algorithm 4 output, in expectation, a solution which is almost as good as some ε-approximation local maximum (i.e., Proposition 15), and the rightmost side of the lemma is used to prove Theorem 4. Due to space constraints, the proof of Lemma 19, and the use of this lemma to prove Proposition 15 are deferred to the full version.

References

1 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 1:1–1:19, 2019. doi:10.4230/LIPIcs.ITCS.2019.1.

2 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proc. of the 20th ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 671–680, 2014.

3 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SODA '14, pages 1497–1514, 2014.

4 Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.

5 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

6 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015.

7 T-H. Hubert Chan, Zhiyi Huang, Shaofeng H.-C. Jiang, Ning Kang, and Zhihao Gavin Tang. Online submodular maximization with free disposal: Randomization beats 1/4 for partition matroids. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1204–1223, 2017.

8 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming*, pages 318–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

9 Alina Ene and Huy L. Nguyen. Constrained submodular maximization: Beyond 1/e. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–257, 2016.

10 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

11 M. Feldman, A. Norouzi-Fard, O. Svensson, and R. Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory on Computing (STOC)*, pages 1363–1374, 2020.

12 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, pages 730–740, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/d1f255a373a3cef72e03aa9d980c7eca-Abstract.html.

13 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 570–579, 2011.

14 Yuval Filmus and Justin Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM Journal on Computing*, 43(2):514–542, 2014.

15 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1098–1116, 2011.

16 Ran Haba, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. Streaming submodular maximization under a k-set system constraint. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 3939–3949, 2020.

17 Chien-Chung Huang and Naonori Kakimura. Multi-pass streaming algorithms for monotone submodular function maximization. *CoRR*, abs/1802.06212, 2018.

18 Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints, 2021. arXiv:2102.09679.

19 Andreas Krause. Submodularity in machine learning. http://submodularity.org/.

20 Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 323–332. ACM, 2009. doi:10.1145/1536414.1536459.

21 Paul Liu, Aviad Rubinstein, Jan Vondrák, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. arXiv:2111.07217.

22 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, 63(7):1595–1619, 2019. doi:10.1007/s00224-018-9878-x.

23 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pages 1358–1367, 2016.

24 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 1379–1386, 2018.

25 George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978.

26 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Math. Program.*, 14(1):265–294, 1978.

27 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 3826–3835. PMLR, 2018. URL: http://proceedings.mlr.press/v80/norouzi-fard18a.html.

28 Alexander Schrijver. *Combinatorial Optimization : Polyhedra and Efficiency*. Springer-Verlag Berlin Heidelberg, 2003.

29 Mohammad Shadravan. Submodular Matroid Secretary Problem with Shortlists, January 2020. arXiv:2001.00894.

(Re)packing Equal Disks into Rectangle

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway

Petr A. Golovach
Department of Informatics, University of Bergen, Norway

Tanmay Inamdar
Department of Informatics, University of Bergen, Norway

Meirav Zehavi
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

The problem of packing of equal disks (or circles) into a rectangle is a fundamental geometric problem. (By a packing here we mean an arrangement of disks in a rectangle without overlapping.) We consider the following algorithmic generalization of the equal disk packing problem. In this problem, for a given packing of equal disks into a rectangle, the question is whether by changing positions of a small number of disks, we can allocate space for packing more disks. More formally, in the repacking problem, for a given set of n equal disks packed into a rectangle and integers k and h, we ask whether it is possible by changing positions of at most h disks to pack $n + k$ disks. Thus the problem of packing equal disks is the special case of our problem with $n = h = 0$.

While the computational complexity of packing equal disks into a rectangle remains open, we prove that the repacking problem is NP-hard already for $h = 0$. Our main algorithmic contribution is an algorithm that solves the repacking problem in time $(h + k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the input size. That is, the problem is fixed-parameter tractable parameterized by k and h.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases circle packing, unit disks, parameterized complexity, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.60

Category Track A: Algorithms, Complexity and Games

Funding *Fedor V. Fomin*: Supported by the Research Council of Norway via the project BWCA, grant no. 314528.
Petr A. Golovach: Supported by the Research Council of Norway via the project BWCA, grant no. 314528.
Tanmay Inamdar: Supported by the European Research Council (ERC) via grant LOPPRE, reference 819416.
Meirav Zehavi: Supported by the Israel Science Foundation (ISF) grant no. 1176/18.

1 Introduction

Packing of equal circles inside a rectangle or a square is one of the oldest packing problems. In addition to many common-life applications, like packing bottles or cans in a box [16], packings of circles have a variety of industrial applications, including circular cutting problems, communication networks, facility location, and dashboard layout. We refer to the survey of Castillo, Kampas, and Pintér [6] for an interesting overview of industrial applications of circle packings.

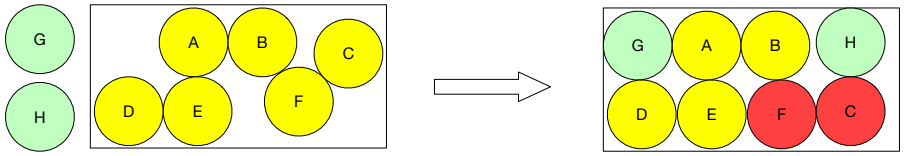


Figure 1 For a packing $\mathcal{P}$ of disks A–F, integers $h = 2$, and $k = 2$, the repacking $\mathcal{P}^*$ of $\mathcal{P}$ is obtained by relocating disks C and F, and by adding disks G and H.

The mathematical study of packing equal circles can be traced to Kepler [20]. Packing of circles also poses exciting mathematical and algorithmic challenges. After the significant efforts spent on packing for several decades [26, 29, 22, 24, 21, 28, 25, 12], optimal packings of equal circles inside a square are known only for instances of up to tens of circles [9, 27]. The computational complexity of packing of equal circles (NP-hardness or membership in NP) remains elusive. For packing circles with different radii, Demaine, Fekete, and Lang claimed NP-hardness [11]. See also the work of Abrahamsen, Miltzow, and Seiferth [1] for a generic framework for establishing $\exists\mathbb{R}$-completeness for packing problems.

Our paper establishes several results on computational and parameterized complexity of a natural generalization of packing equal circles inside a rectangle. A remark in the terminology is in order. In the literature on packing, both terms, circles and disks, could be found. While the term circle is much more popular than disk, we decided to use disks for the following reason: In our hardness proof, it is more convenient to operate with open disks. Thus all disks we consider are open and unit (that is of radius one). Let us remind, that a family of disks forms a *packing* if they are pairwise nonintersecting.[1] In our problem, we have a packing of disks in a rectangle, and the question is whether we can allocate some space for more disks by relocating a small amount of disks. More precisely, we consider the following problem. See Figure 1 for an example.

Disk Repacking

Input:	A packing $\mathcal{P}$ of n unit disks inside a rectangle R and two integers $h, k \geq 0$.
Task:	Decide whether there is a packing $\mathcal{P}^*$ of $n + k$ unit disks inside R obtained from $\mathcal{P}$ by adding k new disks and relocating at most h disks of $\mathcal{P}$ to new positions.

Thus when $n = 0$, that is, initially there are no disks inside the rectangle, this is the classical problem of packing equal circles inside a rectangle.

Related Work on Geometric Packing. Packing problems have received significant attention from the viewpoint of approximation algorithms. For the sake of illustration, let us mention a few examples. In 2D Geometric Bin Packing, which is a variant of classical Bin Packing, the goal is to pack a given collection of rectangles into the minimum number of unit square bins. Typically, it is required that the rectangles be packed in an axis-parallel manner. There has been a long series of results on this problem, culminating in the currently known best approximation given by Bansal and Khan [4]. A related problem is that of 2D Strip

[1] In the literature, it is often required for geometric packings that a packing should be maximal. In particular, for disk packing, every disk should touch either the bounding rectangle or another disk. However, in our problem, the task is to add a specified number of new disks to a given family and this makes the maximality condition in our case very artificial.

Packing problem, where the task is to pack a given set of rectangles into an infinite strip of the given width, so as to minimize the height of packing. This problem has been studied from the context of approximation [17, 19] as well as parameterized [2] algorithms. Finally, we mention the Geometric Knapsack problem, which is also closely related to Geometric Bin Packing. In Geometric Knapsack, we are given a collection of rectangles, where each rectangle has an associated profit. The goal is to pack a subset of the given rectangles (without rotation) in an axis-aligned square knapsack, so as to maximize the total profit of the packed rectangles. Currently, the best approximation is given by Galvez et al. [14]. A detailed survey of the literature on the results of these problems is beyond the scope of this work – we direct an interested reader to the cited works and references therein and the survey paper of Christensen et al. [7]. However, we would like to highlight an important difficulty in Disk Repacking– which is the focus of this work – as compared to the aforementioned geometric packing problems, namely, that packing disks in a rectangle requires the use of intricate geometric arguments as compared to packing rectilinear objects (such as rectangles) in a rectilinear container (such as a unit square, or an infinite strip).

Our Results. We show that Disk Repacking is NP-hard even if the parameter $h = 0$ – we call this special case of problem Disk Appending.

▶ **Theorem 1.** *Disk Appending is* NP*-hard when constrained to the instances* $(R, \mathcal{P}, k)$ *where* $R = [0, a] \times [0, b]$ *for positive integers* a *and* b *and the centers of all disks in* $\mathcal{P}$ *have rational coordinates. Furthermore, the problem remains* NP*-hard when it is only allowed to add new disks to* $\mathcal{P}$ *with rational coordinates of their centers.*

From the positive side, we show that Disk Repacking is FPT when parameterized by k and h. As it is common in Computational Geometry, we assume the *real RAM* computational model, that is, we are working with real numbers and assume that basic operations can be executed in unit time. We use $|I|$ to denote the input size.

▶ **Theorem 2.** *The Disk Repacking problem is* FPT *when parameterized by* $k + h$*. Specifically, it is solvable in time* $(h + k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$.

Theorem 2 also appears to be handy for approximating the maximum number of disks that can be added to a packing. In the optimization variant of Disk Repacking, called Max Disk Repacking, we are given a packing $\mathcal{P}$ of n disks in a rectangle R and an integer h, and the task is to maximize the number of new disks that can be added to the packing if we are allowed to relocate at most h disks of $\mathcal{P}$. By combining Theorem 2 with the approach of Hochbaum and Maass [18], we prove that the optimization variant of Disk Repacking admits the parameterized analog of EPTAS for the parameterization by h. More precisely, we prove the following theorem.

▶ **Theorem 3.** *For any* $0 < \varepsilon < 1$*, there exists an algorithm that, given an instance* $(\mathcal{P}, R, h)$ *of Max Disk Repacking, returns a packing* $\mathcal{P}^*$ *into* R *with at least* $n + (1 - \varepsilon) \cdot \mathsf{OPT}_h$ *disks in time* $\left(\frac{h+1}{\varepsilon}\right)^{\mathcal{O}(h/\varepsilon + 1/\varepsilon^2)} \cdot |I|^{\mathcal{O}(1)}$*, where* OPT_h *is the maximum number of disks that can be added to the input packing if we can relocate at most* h *disks.*

2 Preliminaries

Disks and rectangles. For two points A and B on the plane, we use AB to denote the line segment with endpoints in A and B. The *distance* between $A = (x_1, y_1)$ and $B = (x_2, y_2)$ or the *length* of AB, is $|AB| = \|A - B\|_2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. The *(open unit) disk*

with a *center* $C = (c_1, c_2)$ on the plane is the set of points (x, y) satisfying the inequality $(x - c_1)^2 + (y - c_2)^2 < 1$. Whenever we write "disk" we mean an open unit disk. Throughout the paper, we assume that considered input rectangles $R = [0, a] \times [0, b]$ for some $a, b > 0$.

Parameterized Complexity. We refer to the book of Cygan et al. [10] for introduction to the area and undefined notions. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is a set of strings over a finite alphabet Σ. An input of a parameterized problem is a pair (x, k), where $x \in \Sigma^*$ and $k \in \mathbb{N}$ is a *parameter*. A parameterized problem is *fixed-parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ for some computable function f.

Systems of Polynomial Inequalities. In our algorithms, we will need to find suitable locations for new disks that need to be added such that the locations are compatible with an existing packing. We will achieve this by solving systems of polynomial inequalities. We refer to the book of Basu, Pollack, and Roy [5] for basic tools. We use the following result.

▶ **Proposition 4** (Theorem 13.13 in [5]). *Let R be a real closed field, and let $\mathcal{P} \subseteq R[X_1, \ldots, X_\ell]$ be a finite set of s polynomials, each of degree at most d, and let*

$$(\exists X_1)(\exists X_2) \ldots (\exists X_\ell) F(X_1, X_2, \ldots, X_\ell)$$

be a sentence, where $F(X_1, \ldots, X_\ell)$ be a quantifier-free boolean formula involving $\mathcal{P}$-atoms of type $P \odot 0$, where $\odot \in \{=, \neq, >, <\}$, and P is a polynomial in $\mathcal{P}$. Then, there exists an algorithm to decide the truth of the sentence with complexity $s^{\ell+1} d^{\mathcal{O}(\ell)}$ in D, where D is the ring generated by the coefficients of the polynomials in $\mathcal{P}$.

Furthermore, a point $(X_1^*, \ldots, X_\ell^*)$ satisfying $F(X_1, \ldots, X_\ell)$ can be computed in the same time by Algorithm 13.2 (sampling algorithm) of [5] (see Theorem 13.11 of [5]). Note that because we are using the real RAM model in our algorithms, the complexity is stated with respect to the natural parameters.

3 Hardness of Disk Appending

In this section, we prove that Disk Appending is NP-hard. Due to space constraints, we only sketch the proof.

▶ **Theorem 1.** *Disk Appending is* NP*-hard when constrained to the instances $(R, \mathcal{P}, k)$ where $R = [0, a] \times [0, b]$ for positive integers a and b and the centers of all disks in $\mathcal{P}$ have rational coordinates. Furthermore, the problem remains* NP*-hard when it is only allowed to add new disks to $\mathcal{P}$ with rational coordinates of their centers.*

Sketch of proof. We reduce from the Independent Set problem. In this problem, for a given graph G and a positive integer k, the task is to decide whether G contains an independent set, that is a set of pairwise nonadjacent vertices, of size at least k. It is well-known that Independent Set is NP-complete on cubic planar graphs [15].

We only outline the main ideas of the reduction. Let G be a graph and assume that ℓ_e are positive integers given for all $e \in E(G)$. Suppose that G' is obtained from G by subdividing each edge e by $2\ell_e$ times. Then it can be shown that G has an independent set of size k if and only if G' has an independent set of size $k + \sum_{e \in E(G)} \ell_e$. We exploit this observation. Given a rectilinear embedding of a cubic planar graph G, for each vertex of G, we create a *node* area formed by surrounding disks. We can place an additional disk in such an area and this encodes the inclusion of the corresponding vertex to an independent set. Then we

join the areas created for vertices by *channels* corresponding to subdivided edges. Similarly to node areas, channels are formed by surrounding disks. Each channel contains an even number of positions where new disks can be placed, and these positions are divided into "odd" and "even" in such a way that we can put disks in either all odd or all even positions but no disks could be placed in adjacent even and odd positions. Thus node areas and channels are used to encode a graph, and then we fill the space around them by *filler* disks that prevent placing any new disk outside node areas and channels. Then placing new disks corresponds to the choice of an independent set in a subdivided graph.

To construct channels, consider four touching disks with centers A, B, C, and D shown in Figure 2 (a). Note that $h = 2 + \sqrt{3}$, $\ell = |AC| = |BC| = 2\sqrt{2 + \sqrt{3}}$, and the angle $\alpha = \pi/12$. Given disks with centers in A, B and C, every disk with its center in the triangle ABC has its center in D. Extending this, we make the following observation about the configuration of disks shown in Figure 2 (b). We call such a configuration of disks a *basic channel*.

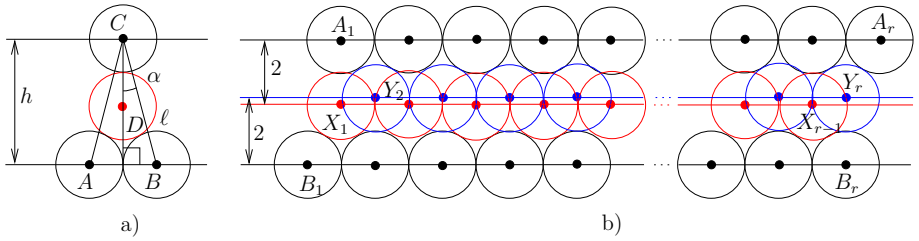


Figure 2 The basic channel; the disks shown in red and blue are not parts of the channel – they show places where new disks can be inserted.

▶ **Observation 5.** *Given disks with centers in $A_1, \ldots, A_r$ and $B_1, \ldots, B_r$ as it is shown in Figure 2 (b), any additional disk with its center properly inside the quadrilateral $A_1B_1B_rA_r$ has its center in one of the points $X_1, \ldots, X_{r-1}$ or $Y_2, \ldots, Y_r$. Furthermore, if a disk with its center in X_i (Y_i, respectively) is placed in the quadrilateral, then no other disk can have its center in Y_i or Y_{i+1} (X_{i-1} or X_i, respectively).*

It can be noted that the construction of basic channels is sufficiently robust to allow us to insert gaps between disks to adjust distances and parities. Furthermore, we can "bend" basic channels.

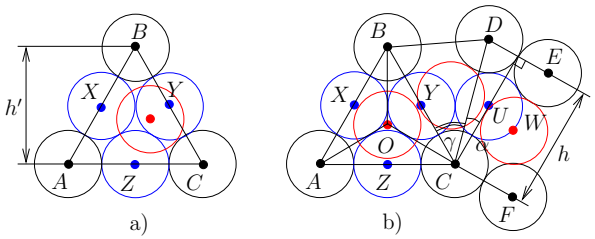


Figure 3 Node area.

For construction of node areas, consider an equilateral triangle ABC with sides of length two as shown in Figure 3 (a), $h' = 2\sqrt{3}$. Suppose that there are disks with centers in A, B and C. Then it is possible to place at most three disks with centers in the triangle ABC, and if exactly three disks are placed, then they have their centers in X, Y and Z and touch each other. Furthermore, if a disk having its center properly inside ABC is placed, then no

other disk with its center inside the triangle can be added. We exploit this property and add a basic channel as it is shown in Figure 3 (b). The point O is the center of ABC, that is, $|OA| = |OB| = |OC|$. Recall that $h = 2 + \sqrt{3}$ and $\alpha = \pi/12$. We set $\gamma = \pi/3 - \pi/12 = \pi/4$. This gives us the configuration of disks with the following properties summarized in the next observation.

▶ **Observation 6.** *Given disks with centers in A, B, C, D, E and F as it is shown in Figure 3 (b), the following is fulfilled:*

- **(i)** *at most one disk with its center in BCD can be added,*
- **(ii)** *if there is a disk with its center either in Y or U, then no other disk can have its center properly in BCD,*
- **(iii)** *if there are disks with their centers in O and W, then a disk with its center in BCD can be added,*
- **(iv)** *if there is a disk having its center properly inside ABC, then no other disk with its center inside ABC can be added.*

The node areas are connected by channels attached as it is shown in Figure 3 (b).

Note that in the described reduction, we used disks with algebraic coordinates of their centers. In particular, the crucial parameters $h = 2 + \sqrt{3}$ and $h' = 3\sqrt{3}$ are algebraic. However, we can observe that our construction is robust to allow rounding of coordinates. More precisely, we can choose a sufficiently small fixed constant $\delta > 0$ and use rational parameters $\hat{h}$ and $\hat{h}'$ such that $2 + \sqrt{3} = h < \hat{h} \leq h + \delta$ and $2\sqrt{3} = h' < \hat{h}' \leq h' + \delta$ in the construction of the channels (see Figure 2) and the node areas (see Figure 3) instead of h and h', respectively. ◀

4 An FPT algorithm for Disk Repacking

In this section, we prove that Disk Repacking is FPT when parameterized by $k + h$.

▶ **Theorem 2.** *The Disk Repacking problem is* FPT *when parameterized by $k + h$. Specifically, it is solvable in time $(h + k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$.*

Proof of Theorem 2: Overview. On a high-level, the idea behind the algorithm is as follows. We first perform a greedy procedure to ensure that all "free" areas to place disks can be intersected by a set $\mathcal{H}$ of at most k disks. Afterwards, we make use of a coloring function of $\mathcal{P}$ with the objective to color all disks in $\mathcal{P}$ that are repacked by a solution (if one exists) blue, and all disks in $\mathcal{P}$ that "closely surround" them by red. We need to ensure that, while relying on the initial greedy procedure, it would suffice to correctly color only $\mathcal{O}(h + k)$ disks. Indeed, this gives rise to the usage of a universal set, which is a "small" family of coloring functions ensured to contain, if there exists a solution, at least one coloring function that correctly colors all $\mathcal{O}(h + k)$ disks we care about.

Considering some coloring function (which expected to be "compatible" with some solution), we identify "slots" and, more generally, "containers" in its coloring pattern. In simple words, a slot is just a disk in R that does not intersect any red disk (from $\mathcal{P}$), and a container is a maximally connected region consisting of slots. We are able to prove that, if the coloring is compatible with some solution, then, for any container, either all or none of the disks in $\mathcal{P}$ that are contained in the container are repacked. This gives rise to a reduction from the problem of finding a solution compatible with a given coloring to the Knapsack problem (more precisely, an extended version of it), where each container corresponds to an item whose weight is the number of disks in $\mathcal{P}$ that it contains, and whose value is the number of disks that can be packed within it.

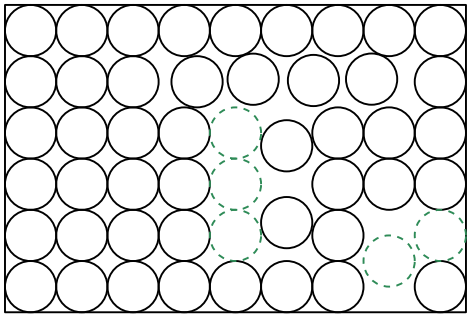


Figure 4 An instance $(\mathcal{P}, R, h = 2, k = 7)$ of Disk Repacking. The disks in $\mathcal{P}$ are colored black. The disks in some hole cover $\mathcal{H}$ are colored green (using dashed lines).

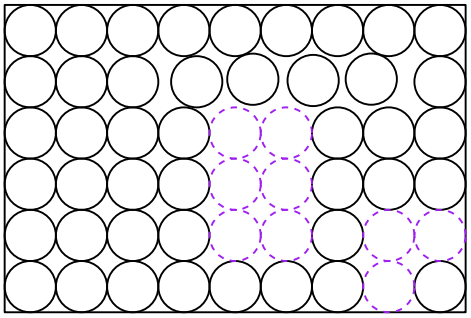


Figure 5 A solution $\mathcal{P}^*$ for the instance on the left. The disks in $\mathcal{P}^* \setminus \mathcal{P}$ are drawn in purple (using dashed lines).The set of $(\mathcal{H}, \mathcal{P}^*)$-critical disks is the set of green disks from the figure on the left and the purple disks from the figure on the right.

To execute the reduction described above, we need to be able to compute the value of each container. For this purpose, we first prove that a container can be "described" by only $\mathcal{O}(h+k)$ many disks from $\mathcal{P} \cup \mathcal{H}$; more precisely, we show that each container is the union of disks contained in R that intersect at least one out of $\mathcal{O}(h+k)$ disks in $\mathcal{P} \cup \mathcal{H}$, from which we subtract the union of some other $\mathcal{O}(h+k)$ disks from $\mathcal{P}$. Having this at hand, to compute the value of a container, we first "guess", for each disk packed by a (hypothetical) optimal packing of disks in the container, a disk from $\mathcal{P} \cup \mathcal{H}$ contained in the container (making use of its description) with whom it intersects. After that, we seek the corresponding optimal packing by making use of a system of polynomial equations (inequalities) of degree 2, $\mathcal{O}(h+k)$ variables, and $\mathcal{O}((h+k)^2)$ equations.

Proof of Theorem 2: Free areas. To execute the plan above, we start with the task of handling the "free" areas. For this, we have the following definition and immediate observation.

▶ **Definition 7** (Holes and Hole Cover). *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. The* set of holes*, denoted by* Holes*, is the set of all disks contained in R that are disjoint from all disks in $\mathcal{P}$. A set $\mathcal{H}$ of disks contained in R such that the set of holes of $(\mathcal{P} \cup \mathcal{H}, R, h, k)$ is empty is called a* hole cover*.*

▶ **Observation 8.** *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $\mathcal{H}$ be a hole cover. Then, every disk contained in R intersects at least one disk in $\mathcal{P} \cup \mathcal{H}$.*

Next, we present a definition and a lemma that will allow us to assume that we have a hole cover of small size at hand.

▶ **Definition 9** (Dense instance). *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. We say that the instance is* dense *if it has a hole cover of size smaller than k.*

▶ **Lemma 10.** *There exists a polynomial-time algorithm that, given an instance $(\mathcal{P}, R, h, k)$ of Disk Repacking, either correctly determines that $(\mathcal{P}, R, h, k)$ is a yes-instance or correctly determines that $(\mathcal{P}, R, h, k)$ is dense and returns a hole cover of size smaller than k.*

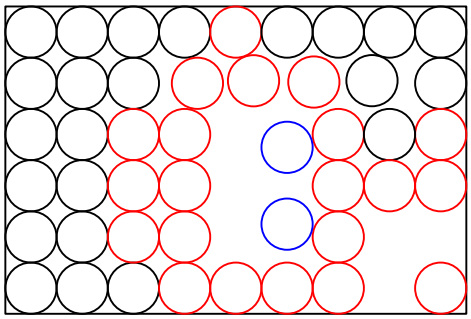


Figure 6 With respect to the instance and solution described in Figures 4 and 5, the disks $(\mathcal{H}, \mathcal{P}^*)$-forced to be blue are colored blue, and the disks $(\mathcal{H}, \mathcal{P}^*)$-forced to be red are colored red. Note that each of the disks colored black can be colored either blue or red by an $(\mathcal{H}, \mathcal{P}^*)$-compatible coloring.

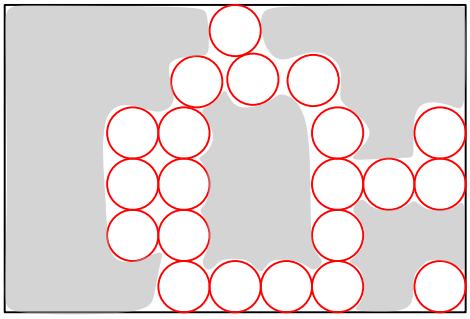


Figure 7 Consider an $(\mathcal{H}, \mathcal{P}^*)$-compatible coloring that colors blue all of the disks colored black in Figure 6. Then, we have four c-Containers, which roughly correspond to the areas colored by grey.

Proof. We perform a simple greedy procedure. Initially, $\mathcal{H} = \emptyset$. Then, as long as there exists a disk D contained in R that is disjoint from all disks in $\mathcal{P} \cup \mathcal{H}$, we add such a disk D to $\mathcal{H}$. The test for the existence of such a D can be performed by using a system of polynomial equations of degree 2 with two variables denoting the x- and y-coordinates of the center of D. For each disk in $\mathcal{P} \cup \mathcal{H}$, we have an equation enforcing that the distance between its center and the center of D is at least 2, and additionally we have two linear equations to enforce that D is contained in R. By Proposition 4, testing whether this system has a solution (which corresponds to the sought disk D) can be done is polynomial time.[2] Once the process terminates, the algorithm checks whether $|\mathcal{H}| \geq k$. If the answer is positive, then adding $\mathcal{H}$ (or, more precisely, any subset of size k of it) to $\mathcal{P}$ is a solution, and so the algorithm answers yes, and otherwise the instance is dense and the algorithm returns $\mathcal{H}$ (which witnesses that). ◀

In the two following definitions, we identify the coloring functions that will be useful.

▶ **Definition 11** ($(\mathcal{H}, \mathcal{P}^*)$-Critical Disks). *Let $(\mathcal{P}, R, h, k)$ be a yes-instance of* Disk Repacking. *Let $\mathcal{H}$ be a hole cover. Let $\mathcal{P}^*$ be a solution to $(\mathcal{P}, R, h, k)$. The set of* $(\mathcal{H}, \mathcal{P}^*)$-critical disks, *denoted by* $\mathsf{Crit}_{\mathcal{H},\mathcal{P}^*}$, *is* $(\mathcal{P}^* \setminus \mathcal{P}) \cup \mathcal{H}$.

▶ **Definition 12** ($(\mathcal{H}, \mathcal{P}^*)$-Compatible Colorings). *Let $(\mathcal{P}, R, h, k)$ be a yes-instance of* Disk Repacking. *Let $\mathcal{H}$ be a hole cover. Let $\mathcal{P}^*$ be a solution to $(\mathcal{P}, R, h, k)$. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. We say that c is* $(\mathcal{H}, \mathcal{P}^*)$-compatible *if:*

1. *For every $D \in \mathcal{P} \setminus \mathcal{P}^*$, we have that $c(D) = \mathsf{blue}$. We say that the disks in $\mathcal{P} \setminus \mathcal{P}^*$ are* $(\mathcal{H}, \mathcal{P}^*)$-forced to be blue.
2. *For every $D \in \mathcal{P} \cap \mathcal{P}^*$ whose center is at distance at most 4 from the center of some disk in $\mathsf{Crit}_{\mathcal{H},\mathcal{P}^*}$, we have that $c(D) = \mathsf{red}$. We say that the disks in $\mathcal{P} \cap \mathcal{P}^*$ whose center is at distance at most 4 from the center of some disk in $\mathsf{Crit}_{\mathcal{H},\mathcal{P}^*}$ are* $(\mathcal{H}, \mathcal{P}^*)$-forced to be red.

We proceed to show that the number of disks in $\mathcal{P}$ that should be colored "correctly" is only $\mathcal{O}(h + k)$. This is done using the following easy observation, in the following lemma.

[2] Additional details on the precise set of equations mentioned here and in other locations in this section are omitted from this extended abstract due to space constraints.

▶ **Observation 13.** *The number of pairwise disjoint disks inside a circle of radius r is at most πr^2.*

▶ **Lemma 14.** *Let $(\mathcal{P}, R, h, k)$ be a dense yes-instance of Disk Repacking. Let $\mathcal{H}$ be a hole cover of size smaller than k. Let $\mathcal{P}^*$ be a solution to $(\mathcal{P}, R, h, k)$. Then, the number of disks $(\mathcal{H}, \mathcal{P}^*)$-forced to be either blue or red is altogether bounded by $\mathcal{O}(h+k)$.*

Proof. Because $\mathcal{P}^*$ is a solution and $|\mathcal{H}| < k$, we have that $|\mathcal{P} \setminus \mathcal{P}^*| \leq h$. So, at most h disks are $(\mathcal{H}, \mathcal{P}^*)$-forced to be blue. Further, $|\mathsf{Crit}_{\mathcal{H},\mathcal{P}^*}| = |(\mathcal{P}^* \setminus \mathcal{P}) \cup \mathcal{H}| < h + 2k$. Observe that every disk in $\mathcal{P} \cap \mathcal{P}^*$ whose center is at distance at most 4 from the center of some disk in $\mathsf{Crit}_{\mathcal{H},\mathcal{P}^*}$ is contained inside a circle of radius 5 whose center is the center of some disk in $\mathsf{Crit}_{\mathcal{H},\mathcal{P}^*}$. So, due to Observation 13 and since the disks in $\mathcal{P} \cap \mathcal{P}^*$ are pairwise disjoint, there exist at most $\pi \cdot 5^2 \cdot (h + 2k) = \mathcal{O}(h+k)$ disks in $\mathcal{P} \cap \mathcal{P}^*$ whose center is at distance at most 4 from the center of some disk in $\mathsf{Crit}_{\mathcal{H},\mathcal{P}^*}$. In particular, this means that at most $\mathcal{O}(h+k)$ disks are $(\mathcal{H}, \mathcal{P}^*)$-forced to be red. This completes the proof. ◀

Proof of Theorem 2: Values of containers. Next, we present the definition of slots and containers, in which we will aim to (re)pack disks. The definition is followed by an observation and a lemma, which, in particular, state that if we try to repack at least one disk in a container, we can just repack all disks in that container.

▶ **Definition 15** (c-Slots and c-Containers)**.** *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. The set of* c-slots*, denoted by Slots_c, is the set of disks contained in R that are disjoint from all disks in $\mathcal{P}$ that are colored red by c. The set of* c-containers*, denoted by $\mathsf{Containers}_c$, is the set of maximally connected regions in the union of all disks in Slots_c.*

▶ **Observation 16.** *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. Then, the regions in $\mathsf{Containers}_c$ are pairwise disjoint.*

▶ **Lemma 17.** *Let $(\mathcal{P}, R, h, k)$ be a yes-instance of Disk Repacking. Let $\mathcal{H}$ be a hole cover. Let $\mathcal{P}^*$ be a solution to $(\mathcal{P}, R, h, k)$. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$ be $(\mathcal{H}, \mathcal{P}^*)$-compatible. Then, for every region $X \in \mathsf{Containers}_c$, either all disks in $\mathcal{P}$ contained in X belong to $\mathcal{P} \setminus \mathcal{P}^*$ or none of the disks in $\mathcal{P} \cup \mathcal{P}^*$ contained in X belongs to $(\mathcal{P} \setminus \mathcal{P}^*) \cup (\mathcal{P}^* \setminus \mathcal{P})$.*

Proof. Targeting a contradiction, suppose that there exists a disk D contained in X that belongs to $(\mathcal{P} \setminus \mathcal{P}^*) \cup (\mathcal{P}^* \setminus \mathcal{P})$ and a disk D' contained in X that belongs to $\mathcal{P} \cap \mathcal{P}^*$. Let γ be a curve connecting the centers of these disks that lies entirely inside X. By the definition of a c-container and due to Observation 8, every point of this curve contained in a disk that belongs to X and intersects a disk in $\mathcal{P}$ colored blue by c or a disk in $\mathcal{H}$. So, there must exist a point on γ that is the center of a disk D^* that intersects both a disk A contained in X that belongs to $(\mathcal{P} \setminus \mathcal{P}^*) \cup \mathcal{H}$ and a disk A' contained in X that belongs to $\mathcal{P} \cap \mathcal{P}^*$. From the definition of a c-container, A' is colored blue by c. Moreover, note that the center of A' is at distance at most 4 from the center of A, since each of the centers of A and A' is at distance at most 2 from the center of D^*. However, since c is $(\mathcal{H}, \mathcal{P}^*)$-compatible, A' is $(\mathcal{H}, \mathcal{P}^*)$-forced to be red and hence it is colored red by c. Since c cannot color a disk both blue and red, we have reached a contradiction. This completes the proof. ◀

We proceed to define the weight and value of a c-container, which will be required for the reduction of our problem to Knapsack.

▶ **Definition 18** (Weight, Validity and Value of Containers). *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. Let $X \in \mathsf{Containers}_c$. The* weight *of X is the number of disks in $\mathcal{P}$ that it contains. We say that X is* valid *if its weight is at most h. The* value *of X is the maximum number of disks that can be packed inside X.*

The following is a corollary of Lemma 17.

▶ **Corollary 19.** *Let $(\mathcal{P}, R, h, k)$ be a yes-instance of Disk Repacking. Let $\mathcal{P}^*$ be a solution to $(\mathcal{P}, R, h, k)$. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$ be $(\mathcal{H}, \mathcal{P}^*)$-compatible. Then, every disk in $(\mathcal{P} \setminus \mathcal{P}^*) \cup (\mathcal{P}^* \setminus \mathcal{P})$ is a c-slot, and it is contained in a valid c-container.*

Now, we define a way in which we can "easily" describe a container, and then prove that this way can be encoded compactly.

▶ **Definition 20** (Descriptions of Containers). *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $\mathcal{H}$ be a hole cover. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. An* $\mathcal{H}$-description *(or, for short,* description*) of a region $X \in \mathsf{Containers}_c$ is a pair $(\mathcal{D}_1, \mathcal{D}_2)$ of a subset $\mathcal{D}_1 \subseteq \mathcal{P} \cup \mathcal{H}$ and a minimal subset $\mathcal{D}_2 \subseteq \mathcal{P}$ such that X equals the set of all points in R at distance less than 2 from at least one disk in $\mathcal{D}_1$ and at least 2 from all disks in $\mathcal{D}_2$.*

▶ **Lemma 21.** *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $\mathcal{H}$ be a hole cover. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. Let $X \in \mathsf{Containers}_c$. Then, X has at least one description $(\mathcal{D}_1, \mathcal{D}_2)$. Moreover, every description $(\mathcal{D}_1, \mathcal{D}_2)$ of X satisfies $|\mathcal{D}_1| + |\mathcal{D}_2| = \mathcal{O}(h' + k')$ where h' is the weight of X, and k' is the number of disks in $\mathcal{H}$ contained in X.*

Proof. By Observation 8, every c-slot intersects at least one disk in $\{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$ and is disjoint from all disks in $\{D \in \mathcal{P} : c(D) = \mathsf{red}\}$. Further, every point in every disk in $\{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$ is contained in a c-slot. So, it is immediate that X has a description $(\mathcal{D}_1, \mathcal{D}_2)$, and that $|\mathcal{D}_1| = \mathcal{O}(h' + k')$. Due to Observation 13 and since the disks in $\mathcal{P} \cup \mathcal{H}$ are pairwise disjoint, any circle of radius 5 whose center is a center of some disk in $\{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$ can contain inside at most $\pi \cdot 5^2$ disks from $\{D \in \mathcal{P} : c(D) = \mathsf{red}\}$. Due to the minimality of $\mathcal{D}_2$ (which is a subset of $\{D \in \mathcal{P} : c(D) = \mathsf{red}\}$), every disk in it must be contained inside a circle of radius 5 whose center is a center of some disk in $\{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$. Hence, $|\mathcal{D}_2| \leq |\mathcal{D}_1| \cdot \pi \cdot 5^2 = \mathcal{O}(h' + k')$. ◀

Next, we use a description in order to efficiently compute the value of a c-container.

▶ **Lemma 22.** *There is an $(h+k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$-time algorithm that, given a dense instance $I = (\mathcal{P}, R, h, k)$ of Disk Repacking, a hole cover $\mathcal{H}$ of size smaller than k, $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$ and a valid region X with a description $(\mathcal{D}_1, \mathcal{D}_2)$, computes the value of X.*

Proof. Given $I = (\mathcal{P}, R, h, k), \mathcal{H}, c, X$ and $(\mathcal{D}_1, \mathcal{D}_2)$, the algorithm works as follows. For $\ell = h+k, h+k-1, \ldots, 1$, and for every vector $(D_1, D_2, \ldots, D_\ell) \in \mathcal{D}_1 \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_1$, it tests whether there exist ℓ disks $S_1, S_2, \ldots, S_\ell$ such that, for every $i \in \{1, 2, \ldots, \ell\}$, S_i intersects D_i, is contained in R and is disjoint from all disks in $\mathcal{D}_2$. The test is done by constructing a system of polynomial equations of degree 2 with 2ℓ variables and $\ell \cdot (|\mathcal{D}_2| + 2)$ equations as follows. For every $i \in \{1, 2, \ldots, \ell\}$, we have two variables, denoting the x- and y-coordinates of the center of S_i, one equation enforcing that the distance between the center of S_i and the center of D_i is smaller than 2, $|\mathcal{D}_2|$ equations enforcing that the distance between the center of S_i and the center of each of the disks in $\mathcal{D}_2$ is at least 2, and two linear equations enforcing that S_i is contained inside R. If the answer is positive, then the algorithm returns that the value of X is ℓ and terminates; else, it proceeds to the next iteration. Observe that, when $\ell = 1$, the algorithm necessarily terminates (since X contains at least one c-slot).

The correctness of the algorithm is immediate from the definition of a description and the exhaustive search that it performs. As for the running time, first observe that, by Lemma 21 and since X is valid and $|\mathcal{H}| < k$, $|\mathcal{D}_1| + |\mathcal{D}_2| \leq \mathcal{O}(h+k)$. So, for a given ℓ, we have $|\mathcal{D}_1|^{\mathcal{O}(\ell)} = (h+k)^{\mathcal{O}(h+k)}$ choices of vectors. Now, consider the iteration corresponding to some ℓ and some vector. Then, we solve a system of polynomial equations of degree 2 with $\mathcal{O}(h+k)$ variables and $\mathcal{O}((h+k)^2)$ equations. By Proposition 4, this can be done in time $(h+k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$. Thus, the algorithm indeed runs in time $(h+k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$. ◀

The following definition captures the set of all descriptions.

▶ **Definition 23** (Blueprint). *Let $(\mathcal{P}, R, h, k)$ be an instance of* Disk Repacking. *Let $\mathcal{H}$ be a hole cover. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. An $(\mathcal{H}, c)$-*blueprint *is a collection of pairs of sets $\mathsf{Blueprint} \subseteq 2^{\mathcal{P} \cup \mathcal{H}} \times 2^{\mathcal{P}}$, where the first elements of the pair are pairwise-disjoint subsets of $\mathcal{P} \cup \mathcal{H}$, such that each region in $\mathsf{Containers}_c$ has exactly one description in $\mathsf{Blueprint}$, and every pair in $\mathsf{Blueprint}$ is a description of a region in $\mathsf{Containers}_c$.*

Next, we show how to compute blueprints.

▶ **Lemma 24.** *There exists a polynomial-time algorithm that, given an instance $(\mathcal{P}, R, h, k)$ of* Disk Repacking, *a hole cover $\mathcal{H}$, and $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$, outputs an $(\mathcal{H}, c)$-blueprint.*

Proof. We will perform a simple greedy procedure to identify, for each disk in $\{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$, the description of the region that contains it. Observe that every c-container contains at least one disk in $\{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$ (due to Observation 8 and the definition of a c-container). So, if for every disk $D \in \{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$ we will take exactly one description $(\mathcal{D}_1, \mathcal{D}_2)$ among the descriptions we identified such that D is contained in $\mathcal{D}_1$, we will obtain an $(\mathcal{H}, c)$-blueprint.

To describe the greedy procedure, consider some $D \in \{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}$. Let us first show how to attain $\mathcal{D}_1$. For this purpose, we initialize $\mathcal{D}_1 = \{D\}$. Then, for every pair of disks $A \in \mathcal{D}_1$ and $B \in (\{D \in \mathcal{P} : c(D) = \mathsf{blue}\} \cup \mathcal{H}) \setminus \mathcal{D}_1$, we test whether there exists a pair of disks C and C' that are contained in R, intersect each other, are disjoint from all disks in $\{D \in \mathcal{P} : c(D) = \mathsf{red}\}$, and such that C intersects A and C' intersects B. The test for the existence of such a C is performed by using a system of polynomial equations of degree 2 with four variables denoting the x- and y-coordinates of the centers of C and C'. For each disk in $\{D \in \mathcal{P} : c(D) = \mathsf{red}\}$, we have two equations enforcing that the distances between its center and the centers of C and C' are each at least 2. Additionally, we have three equations to enforce that the distance between the centers of C and C' is smaller than 2, the distance between the centers of C and A is smaller than 2, and the distance between the centers of C' and B is smaller than 2, as well as four linear equations to enforce that C and C' are contained in R. By Proposition 4, testing whether this system has a solution (which corresponds to the sought disks C and C') can be done is polynomial time. If the answer is positive, then we add B to $\mathcal{D}_1$. In case at least one pair (A, B) resulted in the addition of B to $\mathcal{D}_1$, then we repeat the entire loop, iterating again over all pairs (A, B) (where the domain from which they are taken is updated as a new disk was added to $\mathcal{D}_1$). Notice that we can perform at most $|\mathcal{P}|$ repetitions, and that each repetition results in at most $|\mathcal{P} \cup \mathcal{H}|^2$ many iterations, each taking polynomial time. Hence, the procedure, so far, runs in polynomial time.

Now, let us show how to attain $\mathcal{D}_2$. For this purpose, we initialize $\mathcal{D}_2 = \{D \in \mathcal{P} : c(D) = \mathsf{red}\}$. Now, for every $A \in \{D \in \mathcal{P} : c(D) = \mathsf{red}\}$, we test whether there exists a disk C that is contained in R and intersects both A and at least one disk in $\mathcal{D}_1$, and is disjoint from all disks in $\mathcal{D}_2 \setminus \{A\}$. The test can be performed by iterating over every disk $B \in \mathcal{D}_1$, and

using a system of polynomial equations of degree 2 with two variables denoting the x- and y-coordinates of the center of C. For each disk in $\mathcal{D}_2 \setminus \{A\}$, we have an equation enforcing that the distance between its center and the center of C is at least 2, and additionally we have two equations to enforce that the distance between the center of C and each of the centers of A and B is smaller than 2, as well as two linear equations to enforce that C is contained in R. By Proposition 4, testing whether this system has a solution (which corresponds to the sought disk C) can be done is polynomial time. If the answer is positive, then we remove A from $\mathcal{D}_2$. Notice that this phase of the procedure also runs in polynomial time. Moreover, the correctness of the entire procedure directly follows from the definitions of a c-container and a description. ◀

We proceed to define the (extended version of the) Knapsack problem and the instances of this problem that our reduction produces.

▶ **Definition 25** ((Extended) Knapsack). *In the* (Extended) Knapsack *problem, we are given a collection of n items U, where each item $u \in U$ has a weight $w(u) \in \mathbb{N}_0$ and a value $v(u) \in \mathbb{N}_0$, and an integer $W \in \mathbb{N}_0$. The objective is to find, for every $W' \in \{0, 1, \ldots, W\}$, the maximum $V_{W'} \in \mathbb{N}_0$ for which there exists a subset of items $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} w(u) \leq W'$ and $\sum_{i \in S} v(u) \geq V_{W'}$.*

▶ **Definition 26** ($(\mathcal{H}, c)$-Knapsack instance). *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $\mathcal{H}$ be a hole cover. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. The* $(\mathcal{H}, c)$-Knapsack instance *is the instance (U, w, v, W, V) of* Knapsack *defined as follows: U is the set of all valid regions in $\mathsf{Containers}_c$; for each $X \in U$, $w(X)$ and $v(X)$ are the weight and value of X (see Definition 18); $W = h$; $V = h + k$.*

▶ **Proposition 27** ([8]). *The* (Extended) Knapsack *problem is solvable in time $\mathcal{O}(|U| \cdot W)$.*

We now to prove the correspondence between our problem when we restrict the solution set to solutions compatible with a given coloring and the Knapsack problem.

▶ **Lemma 28.** *Let $(\mathcal{P}, R, h, k)$ be an instance of Disk Repacking. Let $\mathcal{H}$ be a hole cover. Let $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$. Then, there exists a solution $\mathcal{P}^*$ to $(\mathcal{P}, R, h, k)$ such that c is compatible with $\mathcal{P}^*$ if and only if for the $(\mathcal{H}, c)$-Knapsack instance (U, w, v, W, V), there exists $W' \in \{0, 1, \ldots, W\}$ such that $V_{W'} \geq W' + k$.*

Proof. In one direction, suppose that there exists a solution $\mathcal{P}^*$ to $(\mathcal{P}, R, h, k)$ such that c is compatible with $\mathcal{P}^*$. Let $X_1, X_2, \ldots, X_\ell$ be the c-containers that contain at least one disk from $(\mathcal{P} \setminus \mathcal{P}^*) \cup (\mathcal{P}^* \setminus \mathcal{P})$. By Observation 16, these c-containers are pairwise disjoint, by Lemma 17 and since c is compatible with $\mathcal{P}^*$, all disks in $\mathcal{P}$ contained in $X_1 \cup X_2 \cup \cdots \cup X_\ell$ belong to $\mathcal{P} \setminus \mathcal{P}^*$, and by Corollary 19 and since c is compatible with $\mathcal{P}^*$, all disks in $(\mathcal{P} \setminus \mathcal{P}^*) \cup (\mathcal{P}^* \setminus \mathcal{P})$ are contained in $X_1 \cup X_2 \cup \cdots \cup X_\ell$ and all of these c-containers are valid. So, because $\mathcal{P}^*$ can repack h disks from $\mathcal{P}$, the total weight of these c-containers must be some $W' \in \{0, 1, \ldots, h\} = \{0, 1, \ldots, W\}$, and since $\mathcal{P}^*$ also packs k additional disks, the total value of these c-containers must be at least $W' + k$ (to accommodate all of the repacked and k newly packed disks). Thus, $V_{W'} \geq W' + k$.

In the other direction, suppose that there exists $W' \in \{0, 1, \ldots, W\}$ such that $V_{W'} \geq W' + k$. This means that there exist c-containers $X_1, X_2, \ldots, X_\ell$ whose total weight is $W' \in \{0, 1, \ldots, h\}$ and whose total value is at least $W' + k$. However, because these c-containers are pairwise disjoint (by Observation 16), this means that we can construct a solution $\mathcal{P}^*$ such that c is compatible with $\mathcal{P}^*$ by repacking all the disks in $\mathcal{P}$ that are contained in $X_1, X_2, \ldots, X_\ell$ (there are at most h such disks) and, additionally, inserting k new disks, within $X_1, X_2, \ldots, X_\ell$. This completes the proof. ◀

The following is a corollary of Lemmas 22 and 24.

▶ **Corollary 29.** *There exists an $(h+k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$-time algorithm that, given a dense instance $I = (\mathcal{P}, R, h, k)$ of* Disk Repacking*, a hole cover $\mathcal{H}$ of size smaller than k and $c : \mathcal{P} \to \{\mathsf{blue}, \mathsf{red}\}$, computes the $(\mathcal{H}, c)$-*Knapsack *instance.*

To compute coloring functions, we will use the following definition and proposition.

▶ **Definition 30** ((U,k)-Universal Set)**.** *For a universe U and $k \in \mathbb{N}$, a (U,k)-*universal set *is a collection $\mathcal{C}$ of functions $f : U \to \{\mathsf{blue}, \mathsf{red}\}$ such that for every pair of disjoint sets $B, R \subseteq U$ whose union has size at most k, there exists $c \in \mathcal{C}$ that colors all integers in B blue and all integers in R red.*

▶ **Proposition 31** ([23])**.** *There exists an algorithm that, given a universe U of size n and $k \in \mathbb{N}$, constructs a (U,k)-universal set of size $2^{k+\mathcal{O}(\log^2 k)} \log n$ in time $2^{k+\mathcal{O}(\log^2 k)} n \log n$.*

Based on the definition of a universal set, we define the collection of Knapsack instances relevant to our reduction.

▶ **Definition 32** ($(\mathcal{H},\mathcal{C})$-Knapsack Collection)**.** *Let $(\mathcal{P}, R, h, k)$ be an instance of* Disk Repacking*. Let $\mathcal{H}$ be a hole cover. Let $\mathcal{C}$ be a $(\mathcal{P}, q(h+k))$-universal set, where q is the constant hidden in the $\mathcal{O}$-notation in Lemma 14. Then, the* $(\mathcal{H},\mathcal{C})$-Knapsack collection *is the collection of* Knapsack *instances that includes, for every $c \in \mathcal{C}$, the* $(\mathcal{H}, c)$-Knapsack *instance.*

The following is a corollary of Corollary 29.

▶ **Corollary 33.** *There exists an $(h+k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$-time algorithm that, given a dense instance $I = (\mathcal{P}, R, h, k)$ of* Disk Repacking*, a hole cover $\mathcal{H}$ of size smaller than k and a $(\mathcal{P}, q(h+k))$-universal set $\mathcal{C}$, computes the* $(\mathcal{H},\mathcal{C})$-Knapsack *collection.*

Next, we prove the correspondence between our problem and the collection of Knapsack instances we have just defined.

▶ **Lemma 34.** *Let $(\mathcal{P}, R, h, k)$ be an instance of* Disk Repacking*. Let $\mathcal{H}$ be a hole cover. Let $\mathcal{C}$ be a $(\mathcal{P}, q(h+k))$-universal set. Then, $(\mathcal{P}, R, h, k)$ is a yes-instance of* Disk Repacking *if and only if the* $(\mathcal{H},\mathcal{C})$-Knapsack *collection contains an instance (U, w, v, W, V) for which there exists $W' \in \{0, 1, \ldots, W\}$ such that $V_{W'} \geq W' + k$.*

Proof. In one direction, suppose that $(\mathcal{P}, R, h, k)$ is a yes-instance. By the definition of a $(\mathcal{P}, q(h+k))$-universal set and due to Lemma 14, there exists $c \in \mathcal{C}$ that is compatible with $\mathcal{P}^*$. So, the $(\mathcal{H}, c)$-Knapsack instance is contained in the $(\mathcal{H},\mathcal{C})$-Knapsack collection (U, w, v, W, V), and by Lemma 28, for this instance there exists $W' \in \{0, 1, \ldots, W\}$ such that $V_{W'} \geq W' + k$.

In the other direction, suppose that the $(\mathcal{H},\mathcal{C})$-Knapsack collection contains an instance (U, w, v, W, V) for which there exists $W' \in \{0, 1, \ldots, W\}$ such that $V_{W'} \geq W' + k$. This instance is a $(\mathcal{H}, c)$-Knapsack instance for some $c \in \mathcal{C}$. So, by Lemma 28, $(\mathcal{P}, R, h, k)$ is, in particular, a yes-instance of Disk Repacking. ◀

Proof of Theorem 2: Putting it all together. We are now ready to make the final step of the proof of Theorem 2.

The algorithm works as follows. Given an instance $(\mathcal{P}, R, h, k)$ of Disk Repacking, it calls the algorithm in Lemma 10 to either correctly determine that $(\mathcal{P}, R, h, k)$ is a yes-instance or correctly determine that $(\mathcal{P}, R, h, k)$ is dense and obtain a hole cover $\mathcal{H}$

of size smaller than k. In the first case, the algorithm is done. In the second case, the algorithm proceeds as follows. First, it calls the algorithm in Proposition 31 to obtain a $(\mathcal{P}, q(h+k))$-universal set $\mathcal{C}$. Then, it calls the algorithm in Corollary 33 to obtain the $(\mathcal{H}, \mathcal{C})$-Knapsack collection. Afterwards, it uses the algorithm of Proposition 27 to determine whether the $(\mathcal{H}, \mathcal{C})$-Knapsack collection contains an instance (U, w, v, W, V) for which there exists $W' \in \{0, 1, \ldots, W\}$ such that $V_{W'} \geq W' + k$.

The correctness of the algorithm follows from Lemma 34. The runtime bound of $(h+k)^{\mathcal{O}(h+k)} \cdot |I|^{\mathcal{O}(1)}$ follows from the runtimes bounds of the algorithms that the algorithm calls, stated in Lemma 10, Proposition 31, Corollary 33, and Proposition 27.

This concludes the proof of Theorem 2.

5 An FPT approximation for Maximum Disk Repacking

In this section, we use Theorem 2 to show that the optimization variant of Disk Repacking, where we maximize the number of added disks, admits an FPT-AS (i.e., *Fixed Parameter Tractable Approximation Scheme*, a parameterized analog of EPTAS) when parameterized by h. Let us remind that in the optimization problem, called Max Disk Repacking, we are given a packing $\mathcal{P}$ of n disks in a rectangle R and an integer h, and the task is to maximize the number of new disks that can be added to the packing if we are allowed to relocate at most h disks of $\mathcal{P}$.

We need an algorithm for the special case $h = 0$, that is, for the optimization version of Disk Appending. The algorithm is based on the shifting technique, originally introduced by Hochbaum and Maass [18] (also related to Baker's technique [3]). We use OPT for the maximum number of disks that can be added in a rectangle to complement a given packing $\mathcal{P}$.

▶ **Lemma 35.** *For any $0 < \varepsilon < 1$, there exists an algorithm that for a packing of n disks in a rectangle, returns a packing with at least $n + (1-\varepsilon) \cdot \mathsf{OPT}$ disks in time $\left(\frac{1}{\varepsilon}\right)^{\mathcal{O}(1/\varepsilon^2)} \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the input size.*

Proof. Let $\mathcal{S}^*$, $|\mathcal{S}^*| = \mathsf{OPT}$, be the set of newly added disks in an optimal solution. Let $\ell \geq 1$ be a fixed positive integer. Recall that the instance is contained inside a bounding rectangle R. Let us assume that the bottom-left corner of R has Cartesian coordinates $(0, 0)$. For every $1 \leq i, j \leq 2\ell$, let $G_{i,j}$ be a grid of side-length $\ell \times \ell$, with origin at $(-i, -j)$. Note that there exists a pair (i, j) such that the number of disks of $\mathcal{S}^*$ that do not intersect with the boundary of the grid cells in $G_{i,j}$ is at least $(1 - \frac{1}{\ell})^2 \cdot \mathsf{OPT}$.

For any $1 \leq i, j \leq n$, and a grid cell C in $G_{i,j}$, let $\Pi(C)$ be the following subproblem. Let $\mathcal{P}(C) \subseteq \mathcal{P}$ denote the packing of the original disks that are completely contained in C, or partially intersect with C. The goal is to add the maximum number of new disks to obtain a packing $\mathcal{P}^*(C)$. Note that the number of original disks in $\mathcal{P}$, as well as the new disks that can be added inside C, is upper bounded by ℓ^2, which is a constant. Therefore, an optimal solution to $\Pi(C)$ can be found by solving a system of polynomial equations. Let $OPT_{i,j}$ denote the sum of the optimal values for the subproblems $\Pi(C)$, over all grid cells C in $G_{i,j}$.

Let $\mathcal{P}(C)$ denote the packing of the original disks that are completely contained in the cell C, or partially intersect with C. Recall that C is a square of size $\ell \times \ell$, and since $\mathcal{P}(C)$ is a packing, $|\mathcal{P}(C)| = \mathcal{O}(\ell^2)$. Furthermore, the number of new disks that can be added to C to obtain a new packing is also upper bounded by $p = \mathcal{O}(\ell^2)$. We first "guess" the number of new disks, by trying all possible values q between 1 and $p = \mathcal{O}(\ell^2)$. Now, we construct a system of polynomial equations with $2q$ variables and $q(|\mathcal{P}| + 4)$ equations, as follows.

For every new disk D_i for $1 \leq i \leq q$, we have two variables corresponding to the x and y coordinates of its center in the new packing. For every new disk D_i, we also add 4 linear equations that restrict the center to lie at a horizontal/vertical distance of at least 1 from the perimeter of the cell, so that the disk D_i lies completely within the cell C. Finally, for every disk D'_j in the original packing $\mathcal{P}$, we have an equation that enforces that the distance between the center of D_i and that of D'_j must be at least 2. Now, we solve this system of $\mathcal{O}(\ell^2)$ variables and $\mathcal{O}(\ell^4)$ equations in time $\mathcal{O}(\ell)^{\mathcal{O}(\ell^2)}$ time, using Proposition 4.

Note that since the diameter of a unit disk is 2, by an averaging argument, there exists an index $1 \leq i \leq \ell$, such that $OPT_{i,j} \geq \left(1 - \frac{1}{\ell}\right)^2 \cdot \mathsf{OPT}$. This is because, there exists an index i such that at most $\frac{1}{\ell}$ disks from $\mathcal{S}^*$ intersect the vertical lines $x = a\ell + i$ for integers a. Then, for this value of i, there exists an index j, such that at most $\frac{1}{\ell}$ fraction of the disks that are completely contained within the lines $x = a\ell + i$ intersect horizontal lines $b\ell + j$ for integers b. We direct the reader to [18] for a formal argument of this type.

Therefore, for every $1 \leq i, j \leq 2\ell$, and for every grid cell C in $G_{i,j}$, we solve the subproblem $\Pi(C)$, and return the best solution. Note that if we are looking for an $(1-\varepsilon)$-approximation to the number of newly added disks, then $(1-\varepsilon) \leq \left(1 - \frac{1}{\ell}\right)^2 \leq 1 - \frac{1}{\ell}$ That is, $\ell = 1/\varepsilon$. Thus, the running time of this algorithm is $\left(\frac{1}{\varepsilon}\right)^{\mathcal{O}(1/\varepsilon^2)} \cdot |I|^{\mathcal{O}(1)}$. ◀

Now we construct an algorithm for Max Disk Repacking in Theorem 3.

▶ **Theorem 3.** *For any $0 < \varepsilon < 1$, there exists an algorithm that, given an instance $(\mathcal{P}, R, h)$ of Max Disk Repacking, returns a packing $\mathcal{P}^*$ into R with at least $n + (1-\varepsilon) \cdot \mathsf{OPT}_h$ disks in time $\left(\frac{h+1}{\varepsilon}\right)^{\mathcal{O}(h/\varepsilon + 1/\varepsilon^2)} \cdot |I|^{\mathcal{O}(1)}$, where OPT_h is the maximum number of disks that can be added to the input packing if we can relocate at most h disks.*

Proof. Let $0 < \varepsilon < 1$. Consider an instance $(\mathcal{P}, R, h)$ of Max Disk Repacking. We find the maximum nonnegative integer $k \leq 10h/\varepsilon$ such that $(\mathcal{P}, R, h, k)$ is a yes-instance of Disk Repacking using the algorithm from Theorem 2. This can be done in $\left(\frac{h+1}{\varepsilon}\right)^{\mathcal{O}(h/\varepsilon)} \cdot |I|^{\mathcal{O}(1)}$ time. Next, we run the algorithm from Lemma 35 for (G, R) for $\varepsilon' = \frac{1}{2}\varepsilon$, i.e., assuming that relocations of disks are not allowed. The algorithm runs in $\left(\frac{1}{\varepsilon}\right)^{\mathcal{O}(1/\varepsilon^2)} \cdot |I|^{\mathcal{O}(1)}$ time and returns a solution of size k'. We set $k^* = \max\{k, k'\}$. We claim that $(1-\varepsilon)\mathsf{OPT}_h \leq k^* \leq \mathsf{OPT}_h$. The second inequality is trivial. To show that $(1-\varepsilon)\mathsf{OPT}_h \leq k^*$, we consider two cases.

Suppose that $\mathsf{OPT}_h \leq 10h/\varepsilon$. Then $\mathsf{OPT}_h = k$ as the algorithm from Theorem 2 is exact and $(1-\varepsilon)\mathsf{OPT}_h \leq \mathsf{OPT}_h = k \leq k^*$.

Assume that $\mathsf{OPT}_h > 10h/\varepsilon$. Let $\mathcal{S}$ be the set of added disks in an optimum solution for $(\mathcal{P}, R, h)$ and let $\mathcal{L} \subseteq \mathcal{P}$ be the set of relocated disks. Denote by OPT' the maximum number of disks that can be added to $\mathcal{P}$ without relocations. Observe that every disk in $\mathcal{L}$ intersects at most 5 disks of $\mathcal{S}$. Therefore, $\mathsf{OPT}' \geq |\mathcal{S}| - 5|\mathcal{L}| \geq \mathsf{OPT}_h - 5h$. By Lemma 35, $(1-\varepsilon/2)\mathsf{OPT}' \leq k'$. We obtain that $(1-\varepsilon/2)(\mathsf{OPT}_h - 5h) \leq k' \leq k^*$. Because $\mathsf{OPT}_h > 10h/\varepsilon$, $k^* \geq (1-\varepsilon/2)(\mathsf{OPT}_h - \varepsilon\mathsf{OPT}_h/2) = (1-\varepsilon/2)^2\mathsf{OPT}_h \geq (1-\varepsilon)\mathsf{OPT}_h$. This proves the claim.

We conclude that k^* is the required approximation of OPT_h. To conclude the proof, note that the algorithms from Theorem 2 and Lemma 35 can be adapted to return solutions, that is, the sets of added and relocated disks. ◀

6 Conclusion and open questions

We have shown in Theorem 1 that Disk Repacking problem is NP-hard even if $h = 0$. On the other hand, by Theorem 2, Disk Repacking is FPT when parameterized by k and h. Both theorems naturally lead to the question about parameterization by k only. The difficulty here is that even for adding one disk, one has to relocate many disks. Already for $k = 1$, we do not know, whether the problem is in P or is NP-hard.

Another natural question stemming from Theorem 2 is about kernelization of Disk Repacking. Does Disk Repacking admit a polynomial kernel with parameters k and h? (We refer to books [10, 13] for an introduction to kernelization).

Finally, approximation of Disk Repacking is an interesting research direction. In Theorem 3 we demonstrated that our FPT algorithm can be used to construct an FPT-AS with respect to h for Max Disk Repacking. We leave open the question about polynomial approximation. Another open question concerns the approximability of the minimum number of relocations h for a given k. Already for $k = 1$ finding a good approximation of h is a challenging problem.

References

1 Mikkel Abrahamsen, Tillmann Miltzow, and Nadja Seiferth. Framework for er-completeness of two-dimensional packing problems. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1014–1021. IEEE, 2020. doi:10.1109/FOCS46700.2020.00098.

2 Pradeesha Ashok, Sudeshna Kolay, Syed Mohammad Meesum, and Saket Saurabh. Parameterized complexity of strip packing and minimum volume packing. *Theor. Comput. Sci.*, 661:56–64, 2017. doi:10.1016/j.tcs.2016.11.034.

3 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.

4 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 13–25. SIAM, 2014. doi:10.1137/1.9781611973402.2.

5 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Springer, Berlin, Heidelberg, 2009.

6 Ignacio Castillo, Frank J Kampas, and János D Pintér. Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research*, 191(3):786–802, 2008.

7 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Comput. Sci. Rev.*, 24:63–79, 2017. doi:10.1016/j.cosrev.2016.12.001.

8 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2006.

9 Hallard T Croft, Kenneth Falconer, and Richard K Guy. *Unsolved problems in geometry: unsolved problems in intuitive mathematics*, volume 2. Springer Science & Business Media, 2012.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Erik D. Demaine, Sándor P. Fekete, and Robert J. Lang. Circle packing for origami design is hard. *CoRR*, abs/1008.1224, 2010. arXiv:1008.1224.

12 Sándor P. Fekete, Sebastian Morr, and Christian Scheffer. Split packing: Algorithms for packing circles with optimal worst-case density. *Discret. Comput. Geom.*, 61(3):562–594, 2019. doi:10.1007/s00454-018-0020-2.

13 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization*. Cambridge University Press, Cambridge, 2019. Theory of parameterized preprocessing.

14 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. *ACM Trans. Algorithms*, 17(4):33:1–33:67, 2021. doi:10.1145/3473713.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Michael Goldberg. The packing of equal circles in a square. *Mathematics Magazine*, 43(1):24–30, 1970. doi:10.1080/0025570X.1970.11975991.

17 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A $(5/3 + \epsilon)$-approximation for strip packing. *Comput. Geom.*, 47(2):248–267, 2014. doi:10.1016/j.comgeo.2013.08.008.

18 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

19 Klaus Jansen and Malin Rau. Closing the gap for pseudo-polynomial strip packing. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 62:1–62:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.62.

20 Johannes Kepler. *Strena seu de nive sexangula*. Frankfurt: Godefrid Tampach, 1611.

21 Marco Locatelli and Ulrich Raber. Packing equal circles in a square: a deterministic global optimization approach. *Discrete Applied Mathematics*, 122(1-3):139–166, 2002.

22 Costas D Maranas, Christodoulos A Floudas, and Panos M Pardalos. New results in the packing of equal circles in a square. *Discrete Mathematics*, 142(1-3):287–293, 1995.

23 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191, 1995.

24 Kari J Nurmela et al. More optimal packings of equal circles in a square. *Discrete & Computational Geometry*, 22(3):439–457, 1999.

25 Kari J Nurmela and Patric RJ Östergård. Packing up to 50 equal circles in a square. *Discrete & Computational Geometry*, 18(1):111–120, 1997.

26 J Schaer. The densest packing of 9 circles in a square. *Canadian Mathematical Bulletin*, 8(3):273–277, 1965.

27 E Specht. The best known packings of equal circles in a square (up to N= 10000), 2015. URL: http://hydra.nat.uni-magdeburg.de/packing/csq/csq.html.

28 Péter Gábor Szabó, Mihaly Csaba Markót, Tibor Csendes, Eckard Specht, Leocadio G Casado, and Inmaculada García. *New approaches to circle packing in a square: with program codes*, volume 6. Springer Science & Business Media, 2007.

29 L Fejes Tóth. *Lagerungen in der Ebene auf der Kugel und im Raum*. Springer, 1953.

Faster Cut Sparsification of Weighted Graphs

Sebastian Forster
Universität Salzburg, Austria

Tijn de Vos
Universität Salzburg, Austria

Abstract

A cut sparsifier is a reweighted subgraph that maintains the weights of the cuts of the original graph up to a multiplicative factor of $(1 \pm \epsilon)$. This paper considers computing cut sparsifiers of weighted graphs of size $O(n \log(n)/\epsilon^2)$. Our algorithm computes such a sparsifier in time $O(m \cdot \min(\alpha(n) \log(m/n), \log(n)))$, both for graphs with polynomially bounded and unbounded integer weights, where $\alpha(\cdot)$ is the functional inverse of Ackermann's function. This improves upon the state of the art by Benczúr and Karger (SICOMP 2015), which takes $O(m \log^2(n))$ time. For unbounded weights, this directly gives the best known result for cut sparsification. Together with preprocessing by an algorithm of Fung et al. (SICOMP 2019), this also gives the best known result for polynomially-weighted graphs. Consequently, this implies the fastest approximate min-cut algorithm, both for graphs with polynomial and unbounded weights. In particular, we show that it is possible to adapt the state of the art algorithm of Fung et al. for unweighted graphs to weighted graphs, by letting the partial maximum spanning forest (MSF) packing take the place of the Nagamochi-Ibaraki (NI) forest packing. MSF packings have previously been used by Abraham at al. (FOCS 2016) in the dynamic setting, and are defined as follows: an M-partial MSF packing of G is a set $\mathcal{F} = \{F_1, \ldots, F_M\}$, where F_i is a maximum spanning forest in $G \setminus \bigcup_{j=1}^{i-1} F_j$. Our method for computing (a sufficient estimation of) the MSF packing is the bottleneck in the running time of our sparsification algorithm.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Cut Sparsification, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.61

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2112.03120`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702) and is supported by the Austrian Science Fund (FWF): P 32863-N.

1 Introduction

In many applications, graphs become increasingly large, hence storing and working with such graphs becomes a challenging problem. One strategy to deal with this issue is graph sparsification, where we model the graph by a sparse set of (reweighted) edges that preserve certain properties. Especially because the aim is to work with large input graphs, this process should be efficient with respect to the graph size. Among the different types of graph sparsifiers, there are spanners (preserving distances, see e.g. [26, 2, 4, 11]), resistance sparsifiers (preserving effective resistances, see e.g. [10]), cut sparsifiers (preserving cuts, see e.g. [6, 7, 13]), and spectral sparsifiers (preserving Laplacian quadratic forms, see e.g. [28, 27, 21, 23]). This paper focuses on cut sparsifiers, as first introduced by Benczúr and Karger in [6]. We say that a (reweighted) subgraph $H \subseteq G$ is a $(1 \pm \epsilon)$-*cut sparsifier* for a weighted graph G if for every cut C, the total weight $w_H(C)$ of the edges of the cut in H is within a multiplicative factor of $1 \pm \epsilon$ of the total weight $w_G(C)$ of the edges of the cut in G.

The main approach to compute cut sparsifiers uses the process of *edge compression*: each edge $e \in E$ is part of the sparsifier with some probability p_e, and if selected obtains weight $w(e)/p(e)$. It is immediate that such a scheme gives a sparsifier in expectation, but it has to be shown that the result is also a sparsifier with high probability. The main line of research has been to select good *connectivity estimators* λ_e for each edge such that sampling with $p_e \sim 1/\lambda_e$ yields a good sparsifier. The simplest such result is by Karger [18], where we sample uniformly with each λ_e equal to the weight of the min cut. Continuing along these lines are parameters as: edge connectivity [13], strong connectivity [6, 7], electrical conductance [27], and Nagamochi-Ibaraki (NI) indices [24, 25, 13]. The challenge within the approach of edge compression is to find a connectivity estimator that results in a sparse graph, but can be computed fast.

For weighted graphs, there are roughly three regimes for sparsification. The first regime consists of cut sparsifiers of size $O(n \log^2(n)/\epsilon^2)$. Fung, Hariharan, Harvey, and Panigrahi [12, 13] show that sparsifiers of this asymptotic size can be computed in linear time for polynomially-weighted graphs. For this they introduce a general framework of cut sparsification with a connectivity estimator, see Section 2.1. For unbounded weights, Hariharan and Panigrahi [16] give an algorithm to compute a sparsifier of size $O(n \log^2(n)/\epsilon^2)$ in time $O(m \log^2(n)/\epsilon^2)$.

The second regime consists of cut sparsifiers of size $O(n \log(n)/\epsilon^2)$. Benczúr and Karger [6, 7] show that these can be computed in time $O(m \log^2(n))$ for polynomially-weighted graphs, and in time $O(m \log^3(n))$ for graphs with unbounded weights. Note that these results can be optimized by preprocessing with the algorithms for the first regime.

A third regime, consists of sparsifiers of size $O(n/\epsilon^2)$. The known constructions in this regime yield *spectral* sparsifiers, which are more general than cut sparsifiers. Spectral sparsification was first introduced by Spielman and Teng in [28]. It considers subgraphs that preserve Laplacian quadratic forms. Lee and Sun [23] give an algorithm for finding $(1 \pm \epsilon)$-spectral sparsifiers of size $O(n/\epsilon^2)$ in time $O(m \cdot \text{poly}(\log(n), 1/\epsilon))$. Analyzing their results, we believe that the poly-logarithmic factor contributes at least a factor of $\log^{10}(n)$. While this is optimal in size, both for spectral sparsifiers [5] and cut sparsifiers [3], it is not in time.

In this paper, we improve on the results in the second regime, both for graphs with polynomially bounded and unbounded weights[1]. For an overview of the previous best running times and our results, see Figure 1. We present our sparsification algorithm in Section 4, for the special treatment of unbounded weights we refer to the full version of the paper. Our algorithm improves on the algorithm of Benczúr and Karger [6, 7] for bounded weights, which has been unchallenged for the last 25 years. It also improves on the algorithm of Hariharan and Panigrahi [16] for unbounded weights, which has been unchallenged for the last 10 years. We obtain the following theorem, where $\alpha(\cdot)$ refers to the functional inverse of Ackermann's function, for a definition see e.g. [29]. For any realistic value x, we have $\alpha(x) \leq 4$.

▶ **Theorem 1.** *There exists an algorithm that, given a weighted graph G and a freely chosen parameter $\epsilon \in (0, 1)$, computes a graph G_ϵ, which is a $(1 \pm \epsilon)$-cut sparsifier for G with high probability. The running time of the algorithm is $O(m \cdot \min(\alpha(n) \log(m/n), \log(n)))$ and the number of edges of G_ϵ is $O\big(n \log(n)/\epsilon^2\big)$.*

Using preprocessing with a result from Fung et al. [13] (see Theorem 23), we obtain the following corollary for polynomially-weighted graphs.

[1] See Section 2.2 for our assumptions on the computational model in case of unbounded weights.

Algorithm	Size	Running time
Unweighted		
Fung et al. [13]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\right)$
Polynomial weights		
Benczúr and Karger [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\log^2(n)\right)$
Fung et al. [13]	$O\left(n\log^2(n)/\epsilon^2\right)$	$O\left(m\right)$
[13] + [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m+n\log^4(n)/\epsilon^2\right)$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\log(n))$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\alpha(n)\log(m/n))$
[13] + this paper	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m+n\left(\log^2(n)/\epsilon^2\right)\alpha(n)\log(\log(n)/\epsilon)\right)$
Unbounded weights		
Hariharan and Panigrahi [16]	$O\left(n\log^2(n)/\epsilon^2\right)$	$O\left(m\log^2(n)/\epsilon^2\right)$
Benczúr and Karger [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\log^3(n)\right)$
[16] + [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\log^2(n)/\epsilon^2+n\log^5(n)/\epsilon^2\right)$
Lee and Sun [23]	$O\left(n/\epsilon^2\right)$	$O\left(m\cdot\mathrm{poly}(\log(n),1/\epsilon)\right)$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\log(n))$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\alpha(n)\log(m/n))$

Figure 1 An overview of the state of the art algorithms for computing cut sparsifiers for undirected graphs with integer weights. Algorithm $A+B$ indicates that algorithm B is preprocessed with algorithm A.

▶ **Corollary 2.** *There exists an algorithm that, given a polynomially-weighted graph G and a freely chosen parameter $\epsilon\in(0,1)$, computes a graph G_ϵ, which is a $(1\pm\epsilon)$-cut sparsifier for G with high probability. The running time of the algorithm is $O(m+n\left(\log^2(n)/\epsilon^2\right)\alpha(n)\log(\log(n)/\epsilon))$ and the number of edges of G_ϵ is $O(n\log(n)/\epsilon^2)$.*

Following Benczúr and Karger [7], the computation of cut sparsifiers of graphs with fractional or even real weights can be reduced to integer weights. For the reduction see Appendix C. Thus our algorithm also gives a speedup for such graphs. Since the integer case is the essential one, we follow prior works and only formulate our results for this particular case.

As a direct application of the cut sparsifier, we can use Theorem 1 and Corollary 2 to replace m by $n\log(n)/\epsilon^2$ in the time complexity of algorithms solving cut problems, at the cost of a $(1\pm\epsilon)$-approximation. We detail the effects for the minimum cut problem. Recently, Gawrychowski, Mozes, and Weiman [14] showed that one can compute the minimum cut of a weighted graph in $O(m\log^2(n))$ time. Using the existing sparsification techniques [7, 13] for preprocessing, the state of the art for $(1+\epsilon)$-approximate min-cut is $O(m+n\log^4(n)/\epsilon^2)$. When we use our new sparsification results, we obtain faster $(1+\epsilon)$-approximate min-cut algorithms when $m=\Omega(n\log(n)/\epsilon^2)$.

▶ **Corollary 3.** *There exists an algorithm that, given a polynomially-weighted graph G and a freely chosen parameter $\epsilon\in(0,1)$, with high probability computes an $(1+\epsilon)$-approximation of the minimum cut in time $O(m+n\log^3(n)/\epsilon^2)$.*

There exists an algorithm that, given a weighted graph G and a freely chosen parameter $\epsilon\in(0,1)$, with high probability computes an $(1+\epsilon)$-approximation of the minimum cut in time $O(m\cdot\min(\alpha(n)\log(m/n),\log(n))+n\log^3(n)/\epsilon^2)$.

For unweighted graphs, even faster minimum cut algorithms exist: Ghaffari, Nowicki, and Thorup [15] show that we can find the minimum cut in $O(\min\{m+n\log^3(n), m\log(n)\})$ time. Combining this with the linear time cut sparsifier of Fung et al. [13], we get $(1+\epsilon)$-approximate minimum cut in unweighted graphs in $O(m + n\log(n)\min\{1/\epsilon + \log^2(n), \log(n)/\epsilon\})$ time.

The remainder of this article is organized as follows. The rest of the introduction consists of a technical overview of our algorithms. Section 2 contains a review of the general sparsification framework from Fung et al. [13] tailored to our needs, and can be skipped by readers that are already familiar with this work. We present our algorithm to compute the MSF indices in Section 3. This is used as a black box in our algorithm, which is presented and analyzed in Section 4.

Technical Overview

The high-level set-up of our sparsification algorithm is similar to the algorithm for unweighted graphs of Fung et al. [13]. Our main contribution consists of showing how to generalize this technique to weighted graphs, by using maximum spanning forest (MSF) indices instead of Nagamochi-Ibaraki (NI) indices. On a less significant note, we prove that by a tightening of the analysis one can show that the size and time bounds hold with high probability, and not only in expectation.

NI indices are defined by means of an NI forest packing: view graphs with integer weights as unweighted multigraphs, and repeatedly compute a spanning forest. The NI index is the (last) forest in which an edge appears (for details see Definition 22). The MSF index is also defined by a forest packing, but in this case the MSF packing: we say $\mathcal{F} = \{F_1, \ldots, F_M\}$ is an *M-partial maximum spanning forest packing* of G if for all $i = 1, \ldots, M$, F_i is a maximum spanning forest in $G \setminus \bigcup_{j=1}^{i-1} F_j$. Now, we say that an edge e has MSF index i (w.r.t. to some (partial) MSF packing $\mathcal{F}$) if e appears in the i-th forest F_i of the (partial) MSF packing $\mathcal{F}$. The MSF index has been used previously in the context of dynamic graph sparsifiers (see Abraham et al. [1]). However, there it was only used because it rendered a faster running time, but using NI indices in the corresponding static construction would have been possible as well. In this paper, we use distinctive properties of the MSF index, and the NI index would not suffice. We show that using the MSF index, we can generalize the sparsification algorithm for unweighted graphs to an algorithm for weighted graphs, thereby demonstrating that the MSF index is a natural analogue for the NI index in the weighted setting. We provide an algorithm to compute an M-partial MSF packing in time $O(m \cdot \min(\alpha(n)\log(M), \log(n)))$ for polynomially-weighted graphs. We show that for unbounded weights we can compute a sufficient estimation, also in time $O(m \cdot \min(\alpha(n)\log(M), \log(n)))$.

An important distinction between the unweighted algorithm of Fung et al. and our weighted algorithm, is that the use of contractions to keep running times low throughout the algorithm is no longer possible: edges of different weights have to be treated differently, hence cannot be contracted. By using multiple iterations with an exponentially decreasing precision parameter we can overcome this problem.

In the case of a polynomially-weighted input graph, the algorithm consists of two main phases. In the first phase, we compute sets $F_0, F_1, \ldots, F_\Gamma \subseteq E$, where edges satisfy some lower bound on the weight of any cut separating their endpoints. In the second phase, we sample edges from each set F_i with a corresponding probability.

We set a parameter $\rho = \Theta\left(\frac{\ln(n)}{\epsilon^2}\right)$ and start by computing a 2ρ-partial maximum spanning forest packing for G. We define F_0 to be the union of these 2ρ forests. We add the edges of F_0 to G_ϵ, which will become our sparsifier. We sample each of the remaining edges $E \setminus F_0$ with probability $1/2$ to construct X_1. To counterbalance for the sampling, we will boost

the weight of each sampled edge with a factor 2. Now we continue along these lines, but in each iteration we let F_i consist of an exponentially growing number of spanning forests: F_i is defined as the union of the forests in a $(2^{i+1} \cdot \rho)$-partial MSF packing packing of X_i. Then, X_{i+1} is sampled from the remaining edges $X_i \setminus F_i$, where again each edge is included with probability $1/2$. We continue this process until there are sufficiently few edges left in X_{i+1}. We add these remaining edges to G_ϵ.

The second phase of the algorithm is to sample edges from the sets F_i and add these sampled edges to G_ϵ. Hereto, note that an edge e of F_i (for $i \geq 1$) was not part of F_{i-1}, meaning it was not part of any spanning forest in a $(2^i \cdot \rho)$-partial MSF packing of X_{i-1}. This implies that for an edge $e \in F_i$ the weight of any cut C in X_{i-1} containing e is at least $2^i \cdot \rho \cdot w(e)$. Now we use the general framework for cut sparsification of Fung et al. [13], which boils down to the fact that this guarantee on the weights of cuts implies that we can sample edges from F_i with probability proportional to $1/(2^i w(e))$. We show that this results in a sufficiently sparse graph.

Intuitively, it might seem redundant to sample edges from $X_i \setminus F_i$ to form X_{i+1}. This is indeed not necessary to guarantee that the resulting graph is a sparsifier. However, it ensures that the number of iterations is limited, which leads to better bounds on the size of the sparsifier and the running time. Since we sample edges with probability $1/2$ in each phase, we need to repeat the sampling $O(\log(m/(m_0))$ times to get the size of X_i down to $O(m_0)$. As this number of steps depends on the initial number of edges m, we get better bounds for size and running time if m is already small. We will exploit this by preprocessing the graph with an algorithm from Fung et al. [13] that gives a cut sparsifier of size $O(n \log^2(n)/\epsilon^2)$ in linear time. Moreover, we can show that repeatedly calling our algorithm has no worse asymptotic time bound than calling it once, since the input graph becomes sparser very quickly. By doing so, we obtain a sparsifier of size $O(n \log(n)/\epsilon^2)$.

Since we only use that the MSF index gives a guaranteed lower bound on the connectivity of an edge, one might wonder why the NI index does not work here. After all, the NI indices of a graph can be computed in linear time, which would result in a significant speed-up. However, when computing the NI index, the weight of an edge influences the number of forests necessary, while computing the MSF index only requires the comparison of weights. Moreover, the number of trees in a MSF packing is always bounded by n. We can use this to bound the number of edges in the created sparsifier. The same technique with NI indices would make the size of the sparsifier depend on the maximum weight in the original graph.

To show that the algorithm outputs a cut sparsifier, it needs to be proven that both the sampling in the first and the second phase preserve cuts. We follow the lines of the analysis of Fung et al. [13], which makes use of cut projections and Chernoff bounds. We show that by partitioning the edge sets according to their weight this method extends to weighted graphs.

One part of the algorithm has remained unaddressed: the computation of the maximum spanning forests. The approach we use here is related to Kruskal's algorithm for computing minimum spanning trees [22]. We start by sketching the M-partial MSF packing algorithm for polynomial weights. We sort the edges according to their weights using radix sort in $O(m)$ time. We create M empty forests on n vertices. Starting with the heaviest edge, we add each edge e to the first forest in which it does not create a cycle. We can find this forest using a binary search in $\log(M)$ steps. By using a disjoint-forest representation for the union-find data structure necessary to carry out these steps, we achieve a total time of $O(m\alpha(n) \log(M))$.

When working with unbounded weights, the bottleneck is the initial sorting of the edges. Radix sort does not guarantee to be efficient for unbounded weights. Instead we could use a comparison-based algorithm, such as merge sort, which takes time $O(m \log(n))$. By employing

a different data structure than before, we can guarantee total running time $O(m \log(n))$. However, we do not need the exact MSF indices for our sampling procedure, an estimate suffices. We can adapt a "windowing" technique from Benczúr and Karger [7] to split the graph into subgraphs, where we can rescale the weights to polynomial weights and apply our previously mentioned algorithm. We then achieve a total running time of $O(m\alpha(n) \log(M))$, as before. For more details on this, we refer to the full version. So in total we have running time $O(m \cdot \min(\alpha(n) \log(m/n), \log(n)))$.

2 Notation and Review

Throughout this paper, we consider $G = (V, E)$ to be an undirected, integer weighted graph on $|V| = n$ vertices with $|E| = m$ edges. We define a set of edges $C \subseteq E$ to be a *cut* if there exists a partition of the vertices V in two non-empty subsets A and B, such that C consists of all edges with one endpoint in A and the other endpoint in B. The weight of the cut is the sum of the weights of the edges of the cut: $w_G(C) = \sum_{e \in C} w_G(e)$. The *minimum cut* is defined as the cut with minimum weight. We say that a (reweighted) subgraph $H \subseteq G$ is a $(1 \pm \epsilon)$-*cut sparsifier* for a weighted graph G if for every cut C in H, its weight $w_H(C)$ is within a multiplicative factor of $1 \pm \epsilon$ of its weight $w_G(C)$ in G. A key concept in the realm of cut sparsification is the connectivity of an edge.

▶ **Definition 4.** *Let $G = (V, E)$ be a graph, possibly weighted. We define the* connectivity *of an edge $e = (u, v) \in E$ to be the minimal weight of any cut separating u and v. We say that e is k-*heavy *if it has connectivity at least k. For a cut C, we define the k-*projection *of C to be the k-heavy edges of the cut C.*

The following theorem from Fung et al. [13] bounds the number of distinct k-projections of a graph, it is a generalization of a preceding theorem by Karger, see [17, 20]. This result can be useful when showing that cuts are preserved by a sampling scheme. This is due to the fact that while there may be exponentially many different cuts, this theorem shows that there are only polynomially many cut projections. Hence if one can reduce a claim for cuts to their k-projections, a high probability bound can be obtained through the application of a Chernoff bound.

▶ **Theorem 5.** *For any $k \geq \lambda$ and any $\eta \geq 1$, the number of distinct k-projections in cuts of weight at most ηk in a graph G is at most $n^{2\eta}$, where λ is the weight of a minimum cut in G.*

Throughout this paper, we say a statement holds *with high probability* (w.h.p.) if it holds with probability at least $1 - n^c$, for some constant c. This constant can be modified by adjusting the constants hidden in asymptotic notation.

2.1 A General Framework for Cut Sparsification

We review the general framework for cut sparsification as presented in [13]. This section does not contain new results, and can be skipped by readers that are only interested in our contribution.

The framework shows that edges can be sampled using different notions of connectivity estimators. Although this scheme provides one proof for the validity of multiple parameters, it might be worth noting that an analysis tailored to the used connectivity estimator might provide a better result. For example, when the framework is applied with "edge strengths", it produces a sparsifier of size $O(n \log^2(n)/\epsilon^2)$, a $\log(n)$ factor denser than the edge strength-based sparsifier of Benczúr and Karger [7].

Let $G = (V, E)$ be a graph with integer weights, and let $\epsilon \in (0, 1)$, $c \geq 1$ be parameters. Given a parameter γ (possibly depending on n) and an integer-valued parameter λ_e for each $e \in E$. We obtain G_ϵ from G by independently *compressing* each edge e with parameter

$$p_e = \min\left(1, \frac{16(c+7)\gamma \ln(n)}{0.38\lambda_e \epsilon^2}\right).$$

Compressing an edge e with weight $w(e)$ consists of sampling r_e from a binomial distribution with parameters $w(e)$ and p_e. If $r_e > 0$, we include the edge in G_ϵ with weight r_e/p_e.

In the following we describe a sufficient condition on the parameters γ and λ_e such that G_ϵ is a $(1 \pm \epsilon)$-cut sparsifier for G with probability at least $1 - 4/n^c$. Hereto we partition the edges according to their value λ_e:

$$\Lambda := \left\lfloor \log\left(\max_{e \in E}\{\lambda_e\}\right)\right\rfloor;$$
$$R_i := \{e \in E : 2^i \leq \lambda_e \leq 2^{i+1} - 1\}.$$

Let $\mathcal{G} = \{G_i = (V, E_i) : 1 \leq i \leq \Lambda\}$ be a set of integer-weighted subgraphs such that $R_i \subseteq G_i$. Moreover suppose that $w_{G_i}(e) \geq w_G(e)$ for each $e \in R_i$. For a given set of parameters $\Pi = \{\pi_1, \ldots, \pi_\Lambda\} \subseteq \mathbb{R}^\Lambda$, we define

- Π-*connectivity*: each edge $e \in R_i$ is π_i-heavy in G_i;
- γ-*overlap*: for any cut C,

$$\sum_{i=0}^{\Lambda} \frac{e_i^{(C)} 2^{i-1}}{\pi_i} \leq \gamma \cdot e^{(C)},$$

where $e^{(C)} = \sum_{e \in C} w_G(e)$ and $e_i^{(C)} = \sum_{e \in C \cap E_i} w_{G_i}(e)$.

The following theorem shows that compressing with parameters adhering to these conditions gives a cut sparsifier with high probability.

▶ **Theorem 6** (See [13, Theorem 1.14]). *Fix the parameters γ and λ_e for each edge e. If there exists $\mathcal{G}$ satisfying Π-connectivity and γ-overlap for some Π, then G_ϵ is a $(1 \pm \epsilon)$-cut sparsifier for G, with probability at least $1 - 4/n^c$, where G_ϵ is obtained by edge compression using parameters γ and λ_e's.*

2.2 The Computational Model

If we have an input graph $G = (V, E)$ with weights $w \colon E \to \{1, \ldots, W\}$, we assume our computational model has word size $\Theta(\log(W) + \log(n))$. Note that for polynomial weights, this comes down to a word size of $\Theta(\log(n))$. Moreover, we assume that basic operations on such words have uniform cost, i.e., they can be performed in constant time. In particular, these basic operations are addition, multiplication, inversion, logarithm, and sampling a random bit string of word size precision. Such assumptions are in line with previous work [7, 13], where they are made implicitly.

3 A Maximum Spanning Forest Packing

An important primitive in our algorithm is the use of the maximum spanning forest (MSF) index. The concept is similar to the Nagamochi-Ibaraki index, the important difference is that an edge e with weight $w(e)$ appears in $w(e)$ different NI forests. This means that the

number of NI forests depends on the numerical values of the edge weights, and thus can grow far beyond $O(n)$. On the other hand, the number of maximum spanning forests in a MSF packing is bounded by the maximum degree in the graph, hence also by n. While this already has noteworthy implications for polynomially-weighted graphs, it is even more significant for superpolynomially-weighted graphs. We believe that this property might make them suitable for applications other than presented here.

▶ **Definition 7.** *Let $G = (V, E)$ be a weighted graph. We say $\mathcal{F} = \{F_1, \dots, F_M\}$ is an* M-partial maximum spanning forest packing *of G if for all $i = 1, \dots, M$, F_i is a maximum spanning forest in $G \setminus \bigcup_{j=1}^{i-1} F_j$. If we have that $\bigcup_{i=1}^{M} F_i = G$, then we call $\mathcal{F}$ a* (complete) maximum spanning forest packing *of G. Moreover, for $e \in E$ we denote the* MSF index *of e (w.r.t. $\mathcal{F}$) by f_e, i.e., f_e is the unique index such that $e \in F_{f_e}$.*

Note that we do not demand the $F_i \in \mathcal{F}$ to be non-empty, as this suits notation bests in our applications. Also note that a (partial) MSF packing is fully determined by the MSF indices.

The following theorem states that computing the MSF indices up to M takes $O(m\alpha(n)\log(M))$ time for polynomially-weighted graphs.

▶ **Theorem 8.** *Let $G = (V, E)$ be a polynomially weighted graph, where we allow parallel edges but no self-loops, and we suppose $m \leq n^2$. Then, for any $M > 0$, there exists an algorithm that computes an M-partial MSF packing in $O(m \cdot \min(\alpha(n)\log(M), \log(n)))$ time.*

The outline of the algorithm is as follows, for a complete proof see the full version.

1. Sort the edges by weight in descending order using radix sort in base n.[2]
2. Create empty forests $F_1, \dots, F_M$.
3. Iterate over the edges in descending order and for each edge $e = (u, v)$ do the following:
   a. Find the smallest index i such that u and v are not connected in F_i.
   b. Store i as the MSF index f_e of e. If u and v are connected in every F_i, store $f_e > M$.
   c. Add e to F_i.

We need at most M trees, since we only compute an M-partial MSF packing. By using radix sort, the initial sorting takes time $O(m)$ time (for a time bound of radix sort, see e.g. [9]). In the full version, we show that the remainder of the algorithm can be executed in $O(m\alpha(n)\log(M))$ time or $O(m\log(n))$, depending on the data-structure used. There we also consider an algorithm for *sparse* graphs with unbounded weights.

4 Cut Sparsification for Weighted Graphs

In this section, we present our algorithm for computing a $(1 \pm \epsilon)$-cut sparsifier G_ϵ for a weighted graph G. This makes use of the framework as presented in Section 2.1 and the maximum spanning forest packing as treated in Section 3. This section works towards proving the following theorem for polynomially-weighted graphs. In the full version, we generalize the techniques of this section to graphs with unbounded weights.

▶ **Theorem 9.** *There exists an algorithm that, given a weighted graph $G = (V, E)$, and freely chosen parameter $\epsilon > 0$, computes a graph G_ϵ, which is a $(1 \pm \epsilon)$-cut sparsifier for G with high probability. The algorithm runs in time $O(m \cdot \min(\alpha(n)\log(m/n), \log(n)))$ and the number of edges of G_ϵ is $O\big(n\left(\log(n)/\epsilon^2\right)\log\left(m/(n\log(n)/\epsilon^2)\right)\big)$.*

[2] Note that conversion to base n takes time $O(\log_n(w(e))) \leq O(\log_n(n^c)) = O(c)$ for each edge if the weights are bounded by n^c, so total time $O(mc)$.

To be precise, we give an algorithm where the given bounds on both running time and size of the sparsifier hold with high probability. By simply halting when the running time exceeds the bound, and outputting an empty graph if we exceed the size bound, this gives the result above.

To achieve a better bound on the size of the sparsifier, we repeatedly apply this theorem to the input graph, with an exponentially decreasing precision parameter. The proof of this can be found in the full version.

▶ **Theorem 1** (Restated). *There exists an algorithm that, given a weighted graph $G = (V, E)$, and freely chosen parameter $\epsilon \in (0, 1)$, computes a graph G_ϵ, which is a $(1 \pm \epsilon)$-cut sparsifier for G with high probability. The algorithm runs in time $O(m \cdot \min(\alpha(n) \log(m/n), \log(n)))$ and the number of edges of G_ϵ is $O\big(n \log(n)/\epsilon^2\big)$.*

4.1 The Algorithm

To sparsify the graph, two methods of sampling are used. One of which is the framework presented in Section 2.1. However, instead of applying the framework to the graph directly, there is another sampling process that precedes it.

To simplify equations, let us set $\rho := \frac{(7+c)1352 \ln(n)}{0.38\epsilon^2}$. If $|E| \leq 4\rho n \log\big(m/(n \log(n)/\epsilon^2)\big)$, we do nothing. That is, we return $G_\epsilon = G$. If not, we start by an initialization step and continue with an iterative process, which ends when the remaining graph becomes sufficiently small.

In the initialization step, we define $X_0 := E$. We compute an $\lfloor 2\rho \rfloor$-partial maximum spanning forest packing $T_1, \ldots, T_{\lfloor 2\rho \rfloor}$ and we define $F_0 := \bigcup_{j=1}^{\lfloor 2\rho \rfloor} T_j$. The remaining edges $Y_0 := X_0 \setminus F_0$ move on to the next phase.

In iteration i, we create X_{i+1} from Y_i by sampling each edge with probability $1/2$. Next, we compute $k_i := \rho \cdot 2^{i+1}$ maximum spanning forests $T_1, \ldots, T_{k_i}$. We define $F_i := \bigcup_{j=1}^{k_i} T_j$, and $Y_i := X_i \setminus F_i$.

We continue until Y_i has at most $2\rho n$ edges, and set Γ to be the number of iterations. We retain all edges in F_0. In other words: add each edge $e \in F_0$ to G_ϵ with weight $w(e)$. The edges of Y_Γ are also retained, but they need to be scaled to counterbalance the $\Gamma - 1$ sampling steps: add each edge $e \in Y_\Gamma$ to G_ϵ with weight $2^{\Gamma-1} w(e)$.

Any other edge $e \in F_i$ is at least $k_i w(e)$-heavy in X_{i-1}, as $e \notin F_{i-1}$. We exploit this heavyness to sample from these edges using the framework. For each $e \in F_i$ we:

- Define $n_e := 2^i w(e)$ and $p_e := \min\left(1, \frac{384}{169} \frac{1}{4^i w(e)}\right)$;
- Generate r_e from the binomial distribution with parameters n_e and p_e;
- If r_e is positive, add e to G_ϵ with weight r_e/p_e.

The factor 2^i in calling upon the binomial distribution can be seen as boosting the weight of the edge by a factor 2^i, which is needed to counterbalance the i sampling steps in creating F_i.

Pseudocode of this algorithm can be found in Algorithm 1. Up to the computation method of the MSF packing, the presented algorithm is the same for polynomially and superpolynomially-weighted graphs. For the unbounded case, we use the MSF index estimator as presented in the full version. There we also detail how this influences the correctness of the algorithm, and the bounds on size and running time.

Algorithm 1 SPARSIFY(V, E, w, ϵ, c).

Input: An undirected graph $G = (V, E)$, with integer weights $w\colon E \to \mathbb{N}^+$, and parameters $\epsilon \in (0, 1)$, $c \geq 1$.
Output: An undirected weighted graph $G_\epsilon = (V, E_\epsilon)$.

Set $\rho \leftarrow \frac{(7+c)1352\ln(n)}{0.38\epsilon^2}$.
if $|E| \leq 4\rho n \log\left(m/(n\log(n)/\epsilon^2)\right)$ **then**
  return $G_\epsilon = G$.
end
Compute an $\lfloor 2\rho \rfloor$-partial maximum spanning forest packing $T_1, T_2, \ldots, T_{\lfloor 2\rho \rfloor}$ for G.
Set $i \leftarrow 0$.
Set $X_0 \leftarrow E$.
Set $F_0 \leftarrow \bigcup_{j=1}^{\lfloor 2\rho \rfloor} T_j$.
Set $Y_0 \leftarrow X_0 \setminus F_0$.
while $|Y_i| > 2\rho n$ **do**
  Sample each edge in Y_i with probability $1/2$ to construct X_{i+1}.
  $i \leftarrow i + 1$.
  Set $k_i \leftarrow \rho \cdot 2^{i+1}$.
  Compute an k_i-partial maximum spanning forest packing $T_1, T_2, \ldots, T_{k_i}$ for the graph $G_i := (V, X_i)$.
  Set $F_i \leftarrow \bigcup_{j=1}^{k_i} T_j$
  Set $Y_i \leftarrow X_i \setminus F_i$.
end
Set $\Gamma \leftarrow i$. // Γ is the number of elapsed iteration in the previous while-loop.
Add each edge $e \in Y_\Gamma$ to G_ϵ with weight $2^{\Gamma-1}w(e)$.
Add each edge $e \in F_0$ to G_ϵ with weight $w(e)$.
for $j = 1, \ldots, \Gamma$ **do**
  foreach $e \in F_j$ **do**
    Set $p_e \leftarrow \min\left(1, \frac{384}{169}\frac{1}{4^j w(e)}\right)$.
    Generate r_e from Binom$(2^j w(e), p_e)$.
    if $r_e > 0$ **then**
      Add e to G_ϵ with weight r_e/p_e.
    end
  end
end
return $G_\epsilon = (V, E_\epsilon)$.

4.2 Correctness

We will prove that G_ϵ constructed in SPARSIFY$(V, E, w, \epsilon$,c) is a $(1 \pm \epsilon)$-cut sparsifier for G with probability at least $1 - 8/n^c$. Following the proof structure of [13], we first define

$$S := \left(\bigcup_{i=0}^{\Gamma} 2^i F_i\right) \cup 2^\Gamma Y_\Gamma,$$

where Γ is the maximum number such that $F_i \neq \emptyset$. We define $G_S := (V, S)$. And we prove the following two lemmas, that together yield the desired result.

▶ **Lemma 10.** *G_S is a $(1 \pm \epsilon/3)$-cut sparsifier for G with probability at least $1 - 4/n^c$.*

▶ **Lemma 11.** *G_ϵ is a $(1 \pm \epsilon/3)$-cut sparsifier for G_S with probability at least $1 - 4/n^c$.*

Let us start by proving Lemma 10. The omitted proofs of the lemmas and corollaries used can all be found in the full version. In creating the sets F_i, we repeatedly makes use of the MSF indices. The MSF index of an edge immediately ensures a certain connectivity of that edge. The following lemma makes this precise.

▶ **Lemma 12.** *Let $i \geq 0$ and $e \in Y_i$ be an edge, and set $k_i := \rho \cdot 2^{i+1}$. Then e is $w(e)k_i$-heavy in $G'_{i,e} = (V, X'_{i,e})$, where $X'_{i,e} := \{e' \in X_i : w(e') \geq w(e)\}$. Consequently, e is also $w(e)k_i$-heavy in $G_i = (V, X_i)$.*

Next, we show in a general setting that certain ways of sampling preserve cuts. The following lemma is a generalization of Lemma 5.5 in [13].

▶ **Lemma 13.** *Let $R \subseteq Q$ be subsets of weighted edges on some set of vertices V, satisfying $0 < w(e) \leq 1$ for all $e \in Q$. Moreover, assume that each edge in R is π-heavy in (V, Q). Suppose that each edge $e \in R$ is sampled with probability $p \in (0, 1]$, and if selected, given a weight of $w(e)/p$ to form a set of edges $\widehat{R}$. We denote, for every cut C:*

$$r^{(C)} := \sum_{e \in R \cap C} w(e), \qquad q^{(C)} := \sum_{e \in Q \cap C} w(e), \qquad \widehat{r}^{(C)} := \sum_{e \in = \widehat{R} \cap C} w(e)/p.$$

Let $\zeta \in \mathbb{N}_{\geq 5}$, and $\delta \in (0, 1]$ such that $\delta^2 p\pi \geq \frac{\zeta \ln(n)}{0.38}$, then

$$\left| r^{(C)} - \widehat{r}^{(C)} \right| \leq \delta q^{(C)}$$

for all cuts C, with probability at least $1 - 4/n^{\zeta - 4}$.

We want to apply this lemma to our sampling procedure. We do this by considering different weight classes separately. We define $X_{i,k} := \{e \in X_i : 2^k \leq w(e) \leq 2^{k+1} - 1\}$, and $x^{(C)}_{i,k} = \sum_{e \in X_{i,k} \cap C} w(e)$. We define $Y_{i,k}$ and $y^{(C)}_{i,k}$ analogously. Some rescaling is necessary to ensure that all weights lie in $(0, 1]$, as Lemma 13 requires. For $A \subseteq E$ and $\beta > 0$, we write βA to indicate we multiply the weight of the edges by a factor of β.

▶ **Lemma 14.** *With probability at least $1 - 4/n^{4+c}$, for every cut C in G_i,*

$$\left| 2^{-k} x^{(C)}_{i+1,k} - 2^{-k-1} y^{(C)}_{i,k} \right| \leq \frac{\epsilon/13}{2^{i/2+1}} \sum_{k'=k}^{\infty} 2^{-k'-1} x^{(C)}_{i,k'}.$$

Now we look at the general case, for which we sum all weight classes. Hereto, we define $x^{(C)}_i = \sum_{e \in X_i \cap C} w(e)$, $x^{(C)}_{i+1} = \sum_{e \in X_{i+1} \cap C} w(e)$, and $y^{(C)}_i = \sum_{e \in Y_i \cap C} w(e)$.

▶ **Corollary 15.** *With probability at least $1 - 4/n^{1+c}$, for every cut C in G_i,*

$$\left| 2x^{(C)}_{i+1} - y^{(C)}_i \right| \leq \frac{\epsilon/13}{2^{i/2}} \cdot x^{(C)}_i.$$

We will repeatedly apply this lemma. To show that the accumulated error does not grow beyond $\epsilon/3$, we use the following fact. For a proof we refer to [13].

▶ **Lemma 16.** *Let $x \in (0, 1]$ be a parameter. Then for any $k \geq 0$,*

$$\prod_{i=0}^{k} \left(1 + \frac{x/13}{2^{i/2}}\right) \leq 1 + x/3,$$

$$\prod_{i=0}^{k} \left(1 - \frac{x/13}{2^{i/2}}\right) \geq 1 - x/3.$$

As a final step towards proving Lemma 10, we prove a lemma that focusses on the sparsification occurring in the last $\Gamma - j + 1$ iterative steps of our algorithm.

▶ **Lemma 17.** *Let*

$$S_j = \left(\bigcup_{i=j}^{\Gamma} 2^{i-j} F_i \right) \cup 2^{\Gamma - j} Y_\Gamma$$

for any $j \geq 0$. Then, S_j is a $\left(1 \pm (\epsilon/3)2^{-j/2}\right)$-cut sparsifier for $G_j = (V, X_j)$, with probability at least $1 - 4/n^c$.

Note that setting $j = 0$ gives us Lemma 10.

To prove Lemma 11, we will invoke the framework from [13], as given in Section 2.1. More specifically, we will apply Theorem 6. We set the parameter $\gamma := 64/3$, and for each $e \in F_i$ we set $\lambda_e := \rho \cdot 4^i w(e)$. This is in line with our choice for p_e:

$$\min\left(1, \frac{16(c+7)\gamma \ln(n)}{0.38\lambda_e \epsilon^2}\right) = \min\left(1, \frac{16(c+7)\gamma \ln(n)}{0.38\rho \cdot 4^i w(e)_e \epsilon^2}\right) = \min\left(1, \frac{384}{169}\frac{1}{4^i w(e)}\right) = p_e.$$

We have to provide a set of subgraphs $\mathcal{G}$ and a set of parameters Π such that Π-connectivity and γ-overlap are satisfied.

To explore the connectivity of edges in $R_i := \{e \in E : 2^i \leq \lambda_e \leq 2^{i+1} - 1\}$ we partition these sets as follows:

$$R_{j,k} := \{e \in F_j : 2^k \leq \rho w(e) \leq 2^{k+1} - 1\}.$$

We will view these edges in the subgraph:

$$E_{j,k} := \bigcup_{j'=j-1}^{\Gamma} \bigcup_{k'=k}^{\infty} \rho \cdot 4^{\Gamma - j' + 1} 2^{\Lambda - k' + j'} R_{j',k'}.$$

▶ **Lemma 18.** *Each edge $e \in R_{j,k}$ is $\pi := \rho \cdot 4^\Gamma 2^\Lambda$-heavy in $(V, E_{j,k})$.*

Now we take all weight classes together to find the set of subgraphs $\mathcal{G}$ for which Π-connectivity is satisfied.

▶ **Corollary 19.** *Each edge in $e \in R_i$ is $\rho \cdot 4^\Gamma 2^\Lambda$-heavy in $G_i = (V, E_i)$, with $E_i := \bigcup_{j=1}^{\min(\lfloor i/2 \rfloor, \Gamma)} E_{j,i-2j}$.*

It remains to show that γ-overlap is satisfied.

▶ **Lemma 20.** *For any cut C,*

$$\sum_{i=0}^{\Lambda} \frac{e_i^{(C)} 2^{i-1}}{\rho \cdot 4^\Gamma 2^\lambda} \leq 64/3 \cdot e^{(C)},$$

where $e^{(C)} = \sum_{e \in C} w_{G_S}(e)$ and $e_i^{(C)} = \sum_{e \in C \cap E_i} w_{G_i}(e)$.

Together Corollary 19 and Lemma 20 show that the conditions of Theorem 6 are met with the given parameters. This proves Lemma 11, and then Theorem 9 follows.

4.3 Size of the Sparsifier

The sparsifier G_ϵ consists of F_0, Y_Γ, and F', where $F' = \cup_{i=1}^{\Gamma} F'_i$, with F'_i the sampled edges of F_i. First of all, note that $|F_0| = O(cn\ln(n)/\epsilon^2)$ and $|Y_\Gamma| = O(cn\ln(n)/\epsilon^2)$. Now take $e \in F_i$. This edge results to an edge in G_ϵ if the sample from the binomial distribution with parameters $n_e = 2^i w(e)$ and $p_e = \min\left(1, \frac{384}{169}\frac{1}{4^i w(e)}\right)$ is positive. The probability that this happens is

$$\begin{aligned}
\mathbb{P}[\text{Binom}(n_e, p_e) > 0] &= \sum_{k=1}^{n_e} \mathbb{P}[\text{Binom}(n_e, p_e) = k] && \leq \sum_{k=1}^{n_e} k\,\mathbb{P}[\text{Binom}(n_e, p_e) = k] \\
&= \sum_{k=0}^{n_e} k\,\mathbb{P}[\text{Binom}(n_e, p_e) = k] && = \mathbb{E}[\text{Binom}(n_e, p_e)] \\
&= n_e p_e && \leq \frac{384}{169} 2^{-i}.
\end{aligned}$$

Note that this probability is equal for all $e \in F_i$. Since F_i is the union of $k_i = \rho \cdot 2^{i+1}$ spanning forests, we know that $|F_i| \leq \rho 2^{i+1} n$. Hence the expected size of F'_i, the sampled edges in F_i, equals

$$\begin{aligned}
\mathbb{E}[|F'_i|] &= \sum_{e \in F_i} \mathbb{P}[\text{Binom}(n_e, p_e) > 0] \leq \sum_{e \in F_i} \frac{384}{169} 2^{-i} = |F_i| \frac{384}{169} 2^{-i} \leq \rho 2^{i+1} n \frac{384}{169} 2^{-i} \\
&= \rho \frac{768}{169} n.
\end{aligned}$$

We have that the total number of sampled edges equals

$$\mathbb{E}[|F'|] = \sum_{i=1}^{\Gamma} \mathbb{E}[|F'_i|] \leq \Gamma \rho \frac{768}{169} n,$$

so it remains to bound Γ, i.e., the number of F_i's. Hereto, note that the while loop of lines 10–17 ends if $|Y_i| \leq 2\rho n$. We bound the number of edges in Y_i by bounding the number of edges of X_i, of which Y_i is a subset. Each edge in $Y_{i-1} \subseteq X_{i-1}$ is sampled with probability $1/2$ to form X_i. So $\mathbb{E}[|X_i|] \leq |X_{i-1}|/2$. Now by a Chernoff bound (see Theorem 26) we obtain:

$$\mathbb{P}\left[|X_i| > \frac{2}{3}|X_{i-1}|\right] \leq \exp\left(-\frac{0.38}{36}|X_{i-1}|\right) > \exp\left(-\frac{cn\ln(n)}{36}\right) = n^{-cn/36},$$

since $|X_{i-1}| \geq |Y_{i-1}| \geq 2\rho n = 2 \cdot \frac{(7+c)1352\ln(n)}{0.38\epsilon^2} n \geq \frac{cn\ln(n)}{0.38}$. We have at most n^2 sets X_i, so we can conclude that with high probability $|X_i| \leq \frac{2}{3}|X_{i-1}|$ in each step, and by induction $|Y_i| < |X_i| \leq \left(\frac{2}{3}\right)^i m$. We see that

$$m\left(\frac{2}{3}\right)^{\Gamma} \leq 2\rho n = \frac{21632}{0.38\epsilon^2} cn\ln(n),$$

which is equivalent to

$$\Gamma \geq \log\left(\frac{m}{\frac{21632}{0.38\epsilon^2} cn\ln(n)}\right) / \log(3/2).$$

So, we can conclude $\Gamma = O\left(\log\left(\frac{m}{cn\log(n)/\epsilon^2}\right)\right)$. This gives that the total number of sampled edges is, in expectation,

$$\mathbb{E}[|F'|] \leq \Gamma \rho \frac{768}{169} n = O(cn\log(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2).$$

This compression process can also be seen as the sum of m independent random variables that take values in $\{1, 0\}$.[3] We have just calculated that the expected value μ is at most $Bcn\ln(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2$, for some $B > 0$. Using this, we apply a Chernoff bound (Theorem 26) to get an upper limit for the number of sampled edges:

$$\begin{aligned}
&\mathbb{P}\left[|F'| > 2Bcn\ln n\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2\right] \\
&\qquad \leq \exp\left(-0.38Bcn\ln(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2\right) \\
&\qquad = n^{-0.38cnB\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2}.
\end{aligned}$$

We conclude that, with high probability, the number of sampled edges is

$$O(2Bcn\ln(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2) = O(cn\log(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2).$$

And finally, we conclude that with high probability the number of edges of G_ϵ is bounded by $|E(G_\epsilon)| = |F_0| + |Y_\Gamma| + |F'| = O(cn\log(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2)$.

4.4 Time Complexity

First off, if $m \leq 4\rho n\log\left(m/(n\log(n)/\epsilon^2)\right) = O(cn\log(n)/\epsilon^2\log\left(m/(n\log(n)/\epsilon^2)\right))$, the algorithm does nothing and returns the original graph. So for this analysis we can assume $m > 4\rho n\log\left(m/(n\log(n)/\epsilon^2)\right)$. We analyze the time complexity of the algorithm in two phases. The first phase consists of computing the probabilities p_e for all $e \in E$. The second one is compressing edges, given these probabilities.

The first phase contains i iterations of the while loop (lines 10–17). In each iteration we sample edges from $Y_i \subseteq X_i$ with probability $1/2$ to form X_{i+1}. This takes time at most $O(|X_i|)$. Next, we compute a maximum spanning forest packing of the graph $G_{i+1} = (V, X_{i+1})$. We know that we can compute a M-partial maximum spanning forest packing of a polynomially-weighted graph with n vertices and m_0 edges in $O(m_0 \cdot \min(\alpha(n)\log(M), \log(n)))$ time (see Theorem 8). So this iteration takes at most $O(|X_{i+1}| \cdot (\min(\alpha(n)\log(k_{i+1}), \log(n))))$ time. As noted earlier, we have with high probability that $|X_i| \leq \left(\frac{2}{3}\right)^i m$. If $m\alpha(n)\log(m/n) \leq m\log(n)$, we conclude w.h.p. that the first phase takes total time at most

$$\begin{aligned}
\sum_{i=0}^{\Gamma} O(|X_i|) + O(|X_{i+1}|\alpha(n)\log(k_{i+1})) &= \sum_{i=0}^{\Gamma}\left(\frac{2}{3}\right)^i O(m) + \left(\frac{2}{3}\right)^{i+1} O(m\alpha(n)\log(\rho 2^{i+2})) \\
&\leq 3O(m) + 3O(m\alpha(n)\log(\rho 2^{\Gamma})) \\
&= O(m\alpha(n)\log(m/n)).
\end{aligned}$$

And if $m\log(n) < m\alpha(n)\log(m/n)$, we have that w.h.p. the first phase takes total time at most

$$\begin{aligned}
\sum_{i=0}^{\Gamma} O(|X_i|) + O(|X_{i+1}|\log(n)) &= \sum_{i=0}^{\Gamma}\left(\frac{2}{3}\right)^i O(m) + \left(\frac{2}{3}\right)^{i+1} O(m\log(n)) \\
&\leq 3O(m) + 3O(m\log(n)) \\
&= O(m\log n).
\end{aligned}$$

In the second phase, we sample each edge e from the binomial distribution with parameters n_e and p_e. We will show this can be done with a process that takes $T = O(m)$ time with high probability.

[3] To be precise, we set the probability of an edge $e \notin \bigcup_i F_i$ to exist to 0.

▶ **Lemma 21.** *With high probability, the sampling phase of Algorithm 1 takes $O(m)$ time.*

For the proof, see the full version. Concluding, the algorithm takes

$$O(m \cdot \min(\alpha(n)\log(m/n), \log(n)) + O(m) = O(m \cdot \min(\alpha(n)\log(m/n), \log(n))$$

time in total for polynomially-weighted graphs.

5 Conclusion

In this paper, we presented a faster $(1 \pm \epsilon)$-cut sparsification algorithm for weighted graphs. We have shown how to compute sparsifiers of size $O(n\log(n)/\epsilon^2)$ in $O(m \cdot \min(\alpha(n)\log(m/n), \log(n)))$ time, for integer weighted graphs. Both algorithms apply a sampling technique where the MSF index is used as a connectivity estimator.

We have shown that we can compute an M-partial MSF packing in $O(m\alpha(m)\log(M))$ time for polynomially-weighted graphs. For graphs with unbounded integer weights, we have shown that we can compute a complete MSF packing in $O(m\log(n))$ time, and a sufficient estimation of an M-partial MSF packing can be computed in time $O(m\alpha(m)\log(M))$. An open question is whether a more efficient computation is possible. This would improve on our sparsification algorithm, but might also be advantageous in other applications. The NI index has shown to be useful in various applications. We believe to have shown that the MSF index is a natural analogue.

To develop an algorithm to compute an MSF packing, one might be inclined to build upon one of the algorithms that compute a minimum spanning tree faster than Kruskal's algorithm, such as the celebrated linear-time algorithm of Karger, Klein, and Tarjan [19]. However, this algorithm and many other fast minimum spanning tree algorithms make use of edge contractions. It is far from obvious how to generalize this to a packing: in that case, we need to work simultaneously on multiple trees, hence we cannot simply contract the input graph in favor of any single one. To make this work, a more meticulous use of data structures seems necessary.

Computation of the MSF indices in linear time would be an ultimate goal. However, for our application a slightly looser bound suffices. If we can reduce the running time to compute the MSF indices to $O(m + n\log(n))$, then we obtain a time bound of $O(m)$ for cut sparsification. Moreover, we do not need the exact MSF index, an estimate suffices. This can either be a constant-factor approximation of the MSF index for each edge, or an estimate in the weights used in the forests, as done for graphs with unbounded weights in the full version.

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *Proc. of the Symposium on Foundations of Computer Science (FOCS)*, pages 335–344, 2016.

2 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

3 Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 311–319, 2016.

4 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007. doi:10.1002/rsa.20130.

5 Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

6 András A Benczúr and David R Karger. Approximating st minimum cuts in $\tilde{O}(n^2)$ time. In *Proc. of the Symposium on Theory of Computing (STOC)*, pages 47–55, 1996.

7 András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

8 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.

9 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

10 Michael Dinitz, Robert Krauthgamer, and Tal Wagner. Towards resistance sparsifiers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 40 of *LIPIcs*, pages 738–755, 2015.

11 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Transactions on Algorithms*, 15, July 2016. doi:10.1145/3274651.

12 Wai Shing Fung, Ramesh Hariharan, Nicholas J A Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proc. of the Symposium on Theory of Computing (STOC)*, pages 71–80, New York, NY, USA, 2011. doi:10.1145/1993636.1993647.

13 Wai-Shing Fung, Ramesh Hariharan, Nicholas J A Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM Journal on Computing*, 48(4):1196–1223, 2019.

14 Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Minimum Cut in $O(m \log^2 n)$ Time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. of the International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.57.

15 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proc. of the Symposium on Discrete Algorithms (SODA)*, pages 1260–1279, 2020.

16 Ramesh Hariharan and Debmalya Panigrahi. A general framework for graph sparsification, 2010. arXiv:1004.4080.

17 David R Karger. Global min-cuts in $\mathcal{RNC}$, and other ramifications of a simple min-cut algorithm. In *Proc. of the Symposium on Discrete Algorithms (SODA)*, volume 93, pages 21–30, 1993.

18 David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

19 David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

20 David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.

21 Ioannis Koutis and Shen Chen Xu. Simple parallel and distributed algorithms for spectral graph sparsification. *ACM Trans. Parallel Comput.*, 3(2):14:1–14:14, 2016. doi:10.1145/2948062.

22 Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. of the American Mathematical Society*, 7(1):48–50, 1956.

23 Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proc. of the Symposium on Theory of Computing (STOC)*, pages 678–687, 2017.

24 Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

25 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.

26 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

27 Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

28 Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

29 Robert E Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975. doi:10.1145/321879.321884.

A Review: A First Application of the Sparsification Framework

In this section, we review the application of the framework of Section 2.1 with *Nagamochi-Ibaraki (NI) indices* as parameters, as presented by Fung et al. [13]. As the name suggests, NI indices were first introduced by Nagamochi and Ibaraki [24, 25]. The algorithm they provide gives a graph partitioning into forests, and subsequently a corresponding index for each edge, called the NI index.

▶ **Definition 22.** *Let $G = (V, E)$ be a graph, possibly weighted. We say an edge-disjoint sequence $F_1, F_2, \ldots$ of forests is a* Nagamochi-Ibaraki forest packing *for G if F_i is a spanning forest for $G \setminus \bigcup_{j=1}^{i-1} F_i$, where the weights of $\bigcup_{j=1}^{i-1} F_i$ are subtracted of G. If G is a weighted graph, each edge e must be contained in $w(e)$ contiguous forests. We define the* NI index*, denoted by l_e, to be the index of the (last if weighted) forest in which e appears.*

Nagamochi and Ibaraki show that the NI indices can be computed in linear time for unweighted graphs and in $O(m + n\log(n))$ time for weighted graphs, see [25, 24]. As is shown in [13], we can use the NI index as the connectivity estimator in the sparsification framework to obtain the following result.

▶ **Theorem 23.** *Let $G = (V, E)$ be a weighted graph, and let $\epsilon > 0$ be a constant. Let G_ϵ be obtained by independently compressing each edge with parameter $p_e = \min(1, \rho/l_e)$, where $\rho = \frac{224}{0.38}\ln(n)/\epsilon^2$. Then G_ϵ is a $(1 \pm \epsilon)$-cut sparsifier for G with high probability.*

The sampling itself takes at most $O(m)$ time, as explained in Lemma 21. As the NI indices can be computed in $O(m + n\log(n))$ time, this implies that the total running time is $O(m+n\log(n))$. As a graph with $m \leq n\log(n)$ is already sparse, we can assume $m > n\log(n)$. Thus, for our purposes, the total running time is simply $O(m)$.

Next we provide a bound for the number of edges in the sparsifier G_ϵ. Fung et al. [13] prove this same bound in expectation, we provide a proof for this bound "with high probability".

▶ **Lemma 24.** *With high probability, the size of the graph G_ϵ in Theorem 23 is $O(n\log^2(n)/\epsilon^2)$.*

Proof. Let $v \in V$ be a vertex with degree $d_v \geq O(\log^2(n)/\epsilon^2)$ in G. We denote the degree of v in G_ϵ by d'_v and we write $d' := \max_{v\in V} d'_v$. For each neighbor u of v in G, we compress the edge $e = (u, v)$ with parameter $p_e = \min\left(1, \frac{224\ln(n)}{0.38\epsilon^2 l_e}\right)$, where l_e is the NI index of e. For each edge, the probability that it remains after compression is $1 - (1 - p_e)^{w_e}$. From Bernoulli's inequality we see $1 - (1 - p_e)^{w_e} \leq w_e p_e$. Let Y_e be the random variable that is 1 if e remains, and 0 else. We note that $\mathbb{E}\left[\sum_{e:v\in e} Y_e\right] \leq \frac{224}{0.38}\ln^2(n)/\epsilon^2$. Now we apply a Chernoff bound (Theorem 26) to obtain

$$\mathbb{P}\left[d'_v \geq \delta\frac{224}{0.38}\ln^2(n)/\epsilon^2\right] \leq \exp\left(-0.38\delta\frac{224}{0.38}\ln^2(n)/\epsilon^2\right) = n^{-224\delta\ln(n)/\epsilon^2}.$$

Using a union bound we get the desired result

$$\mathbb{P}\left[d' \leq \delta \frac{224}{0.38} \ln^2(n)/\epsilon^2\right] \geq 1 - n^{1-224\delta \ln(n)/\epsilon^2}.$$

Consequently, we obtain that with high probability the number of edges of the sparsifier is at most $O(n \log^2(n)/\epsilon^2)$. ◀

The state of the art for polynomially-weighted graphs is achieved by postprocessing this result with the algorithm by Benczúr and Karger [7]. Thus our improvement on Benczúr and Karger leads to an overall improved result.

B Tail bounds

To analyze the sampling methods used in Section 4, we make use of the well-known Chernoff bound to get a grasp on the tail of various distributions [8].

▶ **Theorem 25.** *Let $Y_1, \ldots, Y_n$ be n independent random variables such that each Y_i takes values in $[0, 1]$. Let $\mu = \sum_{i=1}^{n} \mathbb{E}[Y_i]$ and $\xi = 2\ln(2) > 0.38$. Then for all $\epsilon > 0$*

$$\mathbb{P}\left[\left|\sum_{i=1}^{n} Y_i - \mu\right| > \epsilon\mu\right] \leq 2\exp\left(-\xi \min(\epsilon, \epsilon^2)\mu\right).$$

At times, the expected value μ itself is not known. Fortunately an upper bound on the expected value also suffices.

▶ **Theorem 26.** *Let $Y_1, \ldots, Y_n$ be n independent random variables such that Y_i takes values in $[0, 1]$. Let $\mu = \sum_{i=1}^{n} \mathbb{E}[Y_i]$ and $\xi = 2\ln(2) > 0.38$. Suppose $\mu' \geq \mu$. Then for all $\delta \geq 2$*

$$\mathbb{P}\left[\sum_{i=1}^{n} Y_i > \delta\mu'\right] \leq 2\exp\left(-\xi(\delta - 1)\mu'\right).$$

Proof. Let $\epsilon := (\delta - 1)\frac{\mu'}{\mu}$. We have $\epsilon \geq 1$, so $\min(\epsilon, \epsilon^2) = \epsilon$. The statement now follows directly from Theorem 25. ◀

C Reduction from Real to Integer Weights

In this section, we show how to reduce the computation of a cut sparsifier of a graph with non-negative real weights to integer weights, formalizing the procedure sketched by Benczúr and Karger [7]. Let $G = (V, E, w)$ be a weighted graph, where $w\colon E \to \mathbb{R}$. Denote $W_{\max} := \max_{e \in E} w(e)$ and $W_{\min} := \min\left\{1, \min_{e \in E} w(e)\right\}$. Then the reduction consists of the following steps:

1. Compute $W_{\min}$ and $r := -\lfloor \log(\frac{\epsilon}{2} W_{\min}) \rfloor$.
2. Create $w'\colon E \to \mathbb{R}$ by rounding the weights $w(e)$ to the closest multiple of 2^{-r}, and define $G' := (V, E, w')$.
3. Create $\hat{w}\colon E \to \mathbb{R}$ by $\hat{w}(e) := 2^r w'(e)$.
4. Compute a $(1 \pm \epsilon/3)$-cut sparsifier $\hat{H} = (V, E_H, \hat{w}_H)$ of $\hat{G} = (V, E, \hat{w})$.
5. Output $H = (V, E_H, w_H)$ where $w_H(e) := 2^{-r}\hat{w}_H(e)$.

First, we show that the graph H is indeed a $(1+\epsilon)$-cut sparsifier of G. Hereto, we note that for any cut C we have

$$w_H(C) = 2^{-r} w_{\hat{H}}(C) \leq 2^{-r}(1+\epsilon/3) w_{\hat{G}}(C) = (1+\epsilon/3) w_{G'}(C) \leq (1+\epsilon) w_G(C),$$

where the last inequality holds as each weight $w'(e)$ has at most an additive error of $2^{-r} \leq \frac{\epsilon}{2} W_{\min} \leq \frac{\epsilon}{2}$ with respect to $w(e)$, hence at most an multiplicative error of $\frac{\epsilon}{2}$. Analogously we obtain $w_H(C) \geq (1-\epsilon) w_G(C)$.

By construction, $\hat{G}$ has integer weights, which are bounded by $O(\frac{W_{\max}}{\epsilon W_{\min}})$. Steps 1, 2, 3, and 5 can be implemented in $O(m)$ time. So indeed we have reduced the problem to finding a cut sparsifier of a graph with integer weights. Moreover, note that if G has polynomially bounded real weights, in the sense that $W_{\max} = O(\text{poly}(n))$ and $W_{\min} = \Omega(1/\text{poly}(n))$, then the graph $\hat{G}$ has polynomially bounded integer weights. We can state this independent of ϵ, since for $\epsilon \leq 1/m$ we can always output the entire input graph as a cut sparsifier of optimal size $O(n/\epsilon^2)$ [3].

Social Distancing Network Creation

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany

Hans Gawendowicz
Hasso Plattner Institute, University of Potsdam, Germany

Pascal Lenzner
Hasso Plattner Institute, University of Potsdam, Germany

Anna Melnichenko
Hasso Plattner Institute, University of Potsdam, Germany

Abstract

During a pandemic people have to find a trade-off between meeting others and staying safely at home. While meeting others is pleasant, it also increases the risk of infection. We consider this dilemma by introducing a game-theoretic network creation model in which selfish agents can form bilateral connections. They benefit from network neighbors, but at the same time, they want to maximize their distance to all other agents. This models the inherent conflict that social distancing rules impose on the behavior of selfish agents in a social network. Besides addressing this familiar issue, our model can be seen as the inverse to the well-studied Network Creation Game by Fabrikant et al. [PODC 2003] where agents aim at being as central as possible in the created network. Thus, our work is in-line with studies that compare minimization problems with their maximization versions.

We look at two variants of network creation governed by social distancing. In the first variant, there are no restrictions on the connections being formed. We characterize optimal and equilibrium networks, and we derive asymptotically tight bounds on the Price of Anarchy and Price of Stability. The second variant is the model's generalization that allows restrictions on the connections that can be formed. As our main result, we prove that Swap-Maximal Routing-Cost Spanning Trees, an efficiently computable weaker variant of Maximum Routing-Cost Spanning Trees, actually resemble equilibria for a significant range of the parameter space. Moreover, we give almost tight bounds on the Price of Anarchy and Price of Stability. These results imply that, compared the well-studied inverse models, under social distancing the agents' selfish behavior has a significantly stronger impact on the quality of the equilibria, i.e., allowing socially much worse stable states.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Mathematics of computing → Graph algorithms

Keywords and phrases Algorithmic Game Theory, Equilibrium Existence, Price of Anarchy, Network Creation Game, Social Distancing, Maximization vs. Minimization Problems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.62

Category Track A: Algorithms, Complexity and Games

Related Version All omitted details of this paper can be found in the full version.
Full Version: `https://arxiv.org/abs/2204.10423` [25]

Funding This work was supported by the DFG project GEONET under grant DFG 442003138.

1 Introduction

Network Design is a core topic in Theoretical Computer Science and Operations Research. Many classical combinatorial optimization problems, inspired by real world applications, have been formulated and analyzed, such as the Minimum Spanning Tree problem [29], the Network Design problem [35, 41] and finding geometric spanners [14, 46]. Typically, a network having certain properties must be found by a centralized algorithm. However, in

many settings, the desired network is not created by a central authority but by individually acting agents, e.g., people or institutions, controlling a local part of the network. Prominent examples are the Internet, road networks, and, most relevant for our work, social networks.

Especially in settings with little coordination, these individual agents tend to selfishly optimize their own utility without taking the impact of their actions on the efficiency of the whole network into account. To better understand the dynamics arising in these decentralized settings and the network structures resulting from them, many influential game-theoretic network formation models have been introduced in the last decades [33, 9, 23, 7, 8]. The main research questions are: Do equilibrium networks, i.e., stable networks where no agent can improve by performing a local change, exist? What properties do these networks have? And how efficient are they compared to centrally computed optimal solutions?

All of the above mentioned influential game-theoretic network formation models assume that the creation of an edge is costly but the agents benefit from having small distances to other agents in the network. However, departing from this standard assumption in the field, there are real-world settings that should better be modeled via an inverted utility function: neighbors yield benefit but being close to many agents is costly as it yields an increased risk. One example for this choice are financial networks. There, financial institutions benefit from working together but suffer from risks arising from one of them failing[1]. Another example, that is the main motivation of our work, came up with the current COVID-19 pandemic and is described by the now commonly used term *social distancing*. It refers to reducing social contacts in order to contain the spread of a contagious virus in the population. While often mandated by the government, social distancing was performed by many people voluntarily. One of the main reasons is quite simple: While reducing social contacts is a restriction of the quality of life, it also reduces the probability of getting infected. Hence, the network of social interactions between people was sparsified by individual strategic decisions.

In this work we introduce a novel game-theoretic network formation model in which selfish agents strategically form a social network under the influence of social distancing. Agents benefit from direct connections to other agents, modeling the positive effects of social contacts on their social life. However, at the same time they want to maximize their distances to all other agents in the network in order to reduce their risk of getting infected via an increased reaction time in case a contagious disease starts spreading in the network. Here we assume that a random network node becomes infected and that it is beneficial to be far away from the source of infection in order to gain valuable time for setting up counter-measures.

The agents in our model act according to an inverted utility function, compared to the famous models by Jackson and Wolinsky [33] and Fabrikant et al. [23]. Thus, to the best of our knowledge, this is one of the rare cases of a game-theoretic model where both minimizing and maximizing the utility function has a natural interpretation. Another similar well-known example is the contrast between the Network Design Game with fair cost sharing by Anshelevich et al. [7] and the Selfish Routing model by Roughgarden and Tardos [47]. In both models the agents select paths in a given network but in the former sharing an edge is beneficial for the involved agents whereas in the latter edge sharing is detrimental. This difference yields vastly different behavior in terms of the quality of the equilibria. However, this is not obvious, as can also be seen by comparing classical minimization and maximization variants of optimization problems, e.g., Minimum Spanning Tree versus Maximum Spanning Tree or Shortest Path versus Longest Path. Sometimes, as with spanning trees, the inverse problems are almost identical, whereas sometimes, as with the

[1] The financial crisis in the late 2000s was mainly driven by contagious network effects of failing banks.

path problems, the inverse problems may have completely opposite behavior. We set out to explore this comparison for the natural inverse counter-part to the well-known Network Creation Game by Fabrikant et al. [23]. Along the way, we will uncover a connection to the Maximum Routing-Cost Spanning Tree problem that is inverse to the well-studied Minimum Routing-Cost Spanning Tree problem [31][2].

1.1 Model and Notation

Before we start with the model definition, we introduce some notation regarding networks. A *network* is a tuple $G \coloneqq (V, E)$ where V is the set of *nodes* and E is the set of *edges*. An *edge* is represented by a set containing both incident nodes. If we do not give the tuple defining G explicitly, we denote the set of nodes of G as V_G and the set edges of G as E_G. We only consider unweighted undirected networks. For addition and removal of a single edge e, we write $G + e \coloneqq (V, E \cup \{e\})$ and $G - e \coloneqq (V, E \setminus \{e\})$. A network G' with $V_{G'} \subseteq V$ and $E_{G'} \subseteq E$ is called a *subnetwork* of G and denoted as $G' \leq G$. If G' is connected and $V_{G'} = V$, G' is a *spanning subnetwork* of G. Let $n \in \mathbb{N}$ denote the number of nodes. The set of all connected networks containing exactly n nodes will be referred to as $\mathcal{G}_n$.

For two nodes $v, x \in V$, we define $d_G(v, x)$ as the *distance* between v and x in network G, that is, the number of edges on a shortest path from v to x in G. For convenience, we extend the definition of d_G to sets of nodes: Let $v \in V$ be a node and $M, N \subseteq V$ be sets of nodes. Then $d_G(v, M) \coloneqq \sum_{x \in M} d_G(v, x)$ and $d_G(M, N) \coloneqq \sum_{x \in M, y \in N} d_G(x, y)$. We call the special case $d_G(v, V)$ the *distances from/for* v and $d_G(V, V)$ the *total/summed distances* or *routing costs* of G. The *degree* of v in the network G is the number of edges that are incident to v and is denoted as $\deg_G(v)$. We call a tree which is a spanning subnetwork of G a *spanning tree* of G. A spanning tree of G with routing costs at least as high as the routing costs of any other spanning tree of G will be called a *Maximum Routing-Cost Spanning Tree* (MRCST). A spanning tree of G with routing costs that cannot be increased by swapping one edge is a *Swap-Maximal Routing-Cost Spanning Tree* (SMRCST).

Now, we can define the game-theoretic model. Let $H = (V, E)$ be a connected network. We call H the *host network* and its nodes *agents*. A *state* of the game $G \leq H$ is a spanning subnetwork of H. We only consider connected networks as host networks and states.

Each agent $v \in V$ selfishly tries to maximize its utility in state G given by

$$u_v(G) \coloneqq \alpha \deg_G(v) + d_G(v, V)$$

where $\alpha \in \mathbb{R}_{>0}$ is a global parameter. We will call $\alpha \deg_G(v)$ the *edge utility* and $d_G(v, V)$ the *distance utility* of v. Note that α is a parameter of the game, i.e., equal for all agents, that allows to adjust the agents' trade-offs between edge utility and distance utility. Here α is the benefit of a single edge, i.e., the benefit for each direct neighbor in the network.

For measuring the efficiency of the network G, we use the *social welfare* defined as $\mathrm{SW}(G) \coloneqq \sum_{v \in V_G} u_v(G) = 2\alpha|E_G| + d_G(V, V)$. This quantifies the well-being of the society of all agents. We call a network maximizing the social welfare for the host network H a *social optimum* and denote it as OPT_H.

Agents are allowed to form connections bilaterally. More specifically, each agent can unilaterally remove any incident edge if it does not disconnect the network, and two agents together can form an edge between them if it is contained in the host network. If removing an edge strictly increases the utility of one of its incident nodes or adding an edge strictly

[2] This problem is also known as the Optimum Communication Spanning Tree problem.

increases the utility of both incident nodes, we call this an *improving move*. A network without improving moves is referred to as *pairwise stable* [33] or *stable* for short[3]. If there are no improving edge additions or removals we call the network *stable against edge addition* and *stable against edge removal*, respectively.

For a host network H, we define $\mathcal{S}(H)$ as the set of all pairwise stable states. For measuring the efficiency lost by letting agents form the network selfishly, we use the *Price of Anarchy* (PoA) [38] and *Price of Stability* (PoS) [7] defined as

$$\mathrm{PoA}_n \coloneqq \max_{H\in\mathcal{G}_n}\max_{G\in\mathcal{S}(H)}\frac{\mathrm{SW}(\mathrm{OPT}_H)}{\mathrm{SW}(G)} \quad \text{and} \quad \mathrm{PoS}_n \coloneqq \max_{H\in\mathcal{G}_n}\min_{G\in\mathcal{S}(H)}\frac{\mathrm{SW}(\mathrm{OPT}_H)}{\mathrm{SW}(G)}.$$

We will call this model *Social Distancing Network Creation Game* (SDNCG). In Section 2 we will restrict the host networks to complete networks K_n. We will call this restricted variant *complete Social Distancing Network Creation Game* (K-SDNCG).

1.2 Related Work

Variants of game-theoretic network formation models have been studied extensively for decades and we refer to Jackson [32] for an overview.

Closest to our work is the literature on the Network Creation Game (NCG) by Fabrikant et al. [23]. This influential model can be seen as the unilateral inverted variant of the K-SDNCG. There, an agent can buy any incident edge without the consent of the other endpoint for the price of $\alpha > 0$. Each agent aims at minimizing its cost, which is defined as the sum of α times the number of bought edges and the sum of hop-distances to all agents. The authors of [23] show that Nash equilibria always exist, i.e., complete networks are stable for $\alpha \leq 2$ and stars are stable for $\alpha \geq 2$. However, besides these generic examples finding Nash equilibria is challenging since the NCG and many of its variants do not belong to the class of potential games [39, 37]. Besides finding equilibria, also computing a best possible strategy is challenging, since this problem was shown to be NP-hard in [23]. However, such strategies can be efficiently approximated with greedy strategy changes [40]. Regarding the quality of equilibrium states the authors of [23] show that the PoA is in $\mathcal{O}(\sqrt{\alpha})$, that the PoA for tree Nash equilibria is constant, and that the PoS is at most $\frac{4}{3}$. Later, a series of papers [2, 20, 43, 42, 4, 12, 5] improved the PoA bounds, with the best general upper bound of $2^{\mathcal{O}\sqrt{\log n}}$ by Demaine et al. [20]. The latter also proved that the PoA is constant for $\alpha \in \mathcal{O}(n^{1-\varepsilon})$ for any fixed $\varepsilon > \frac{1}{\log n}$. For large α, it was shown by Bilò and Lenzner [12] that for $\alpha > 4n - 13$ all Nash equilibria must be trees and this bound was recently improved by Dippel and Vetta [21] to $\alpha > 3n - 3$. This implies a constant PoA for $\alpha > 3n - 3$. Finally, Álvarez and Messegué [5] established a constant PoA for $\alpha > n(1 + \varepsilon)$, for any $\varepsilon > 0$.

The NCG was generalized by Demaine et al. [19] by introducing a host network that specifies which edges can be bought. They show that the PoA deteriorates by providing a lower bound of $\Omega(\min\{\alpha/n, n^2/\alpha\})$ and an upper bound of $\mathcal{O}(\sqrt{\alpha})$, for $\alpha < n$, and $\mathcal{O}(\min\{\sqrt{n}, n^2/\alpha\})$, for $\alpha \geq n$. Interestingly, no results on the existence of equilibria are known. Recently, a further generalization that allows weighted host networks was proposed by Bilò et al. [11]. This variant has a tight PoA of $(\alpha + 2)/2$ for metric weights. Later a

[3] As shown by Corbo and Parkes [18] for bilateral Network Creation Games, pairwise stability is equivalent to pairwise Nash stability, which is a refinement of the Nash equilibrium: it must be stable against unilateral deviations and it must be stable against joint strategy changes by coalitions of agents of size two. The strategy space of any agent $i \in V$ is the power set of $V \setminus i$. An edge $\{u, v\}$ is formed if and only if v is in agent u's strategy and u is in agent v's strategy.

tight bound of $\Theta(\alpha)$ was shown for arbitrary weights [24]. Also a bilateral variant of the NCG was studied by Corbo and Parkes [18]. There, similar to our model, edges can only be established by bilateral consent of the involved nodes and both nodes have to pay α. The authors of [18] prove existence of pairwise stable networks, i.e., complete networks are stable for $\alpha \leq 1$ and stars are stable for $\alpha \geq 1$, they give a tight PoA bound of $\Theta(\min\{\sqrt{\alpha}, n/\sqrt{\alpha}\})$, and they show that the PoS is 1. To the best of our knowledge, the bilateral variant with a given host network has not yet been studied. Recently, also a bilateral variant modeling the formation of social networks was introduced [10].

The idea of a game-theoretic model of network formation in a context of spreading risk is not new. Goyal et al. [28] study a setting where a node is attacked and this attack spreads to all vulnerable neighbors. Agents strategically create edges and immunize themselves to maximize their connected component post attack. For this model, also the efficient computation of best strategies [26] and a variant with probabilistic spread [17] was studied. Moreover, there has been much research in the context of financial contagion, where agents benefit from collaborating, but also suffer from the risk of cascading failure arising with the collaboration [3, 30, 15, 1]. In particular, Blume et al. [13] developed an elegant model where nodes form a network and then some randomly chosen nodes fail and this failure then spreads with some probability via the edges. The utility is a linear combination of the node degree and the risk of failing in the second phase. The virtue of this model is that utilities are based on a random process that realistically models the spread of a contagious infection. However, the major downside of this model is that the computation of the random process is #P-complete. Thus, this model does not yield a realistic prediction of real-world behavior.

While analyzing our model for general host networks, we consider Maximum Routing-Cost Spanning Trees. Routing costs have been studied much in mathematics, mostly under the name of the *Wiener index* [48]. Trees were of special interest and there has been much research on the Wiener index of trees with different properties. But although spanning trees minimizing the Wiener index were studied extensively, the concept of spanning trees maximizing the Wiener index received little attention [22, 49]. However, it was shown that finding or even approximating a tree maximizing the Wiener index is NP-hard [16, 27].

1.3 Our Contribution

We introduce the Social Distancing Network Creation Game (SDNCG), a game-theoretic model in which selfish agents try to maximize their utility by strategically connecting to other agents and thereby creating a network. Each agent values direct connections to other agents but at the same time wants to maximize the distances to all other agents in order to lower their exposure and increase their reaction time to risks appearing in the network. In contrast to the similar model by Blume et al. [13], our model, while not modeling a perfectly realistic spread of the infection, has the advantage of an efficiently computable utility function. By using the distance to the other agents as part of the utility, it also accounts for reaction time: If an infection breaks out far away, an agent has more time to prepare or react to it. Another virtue of our model is that it is the inverse to the well-known Network Creation Game [23] and its bilateral variant [18]. Hence, we can study and compare the game-theoretic properties of the inverted models. To the best of our knowledge, this is one of the rare cases where both the minimization and the maximization of a utility function have a natural interpretation.

Our results and the comparison with the inverted models are summarized in Table 1. We analyze two variants of the SDNCG. For the K-SDNCG, where we assume a complete host network, we characterize optimal and several stable networks and show that the PoS is 1. We provide an improving response cycle, which implies that equilibrium existence

Table 1 An overview of our results (yellow) and a comparison with the results for the inverted models (white). BNCG abbreviates the bilateral NCG by Corbo and Parkes [18] whereas H-NCG denotes the NCG on a host network by Demaine et al. [19]. $N_2 := \frac{(n-1)^2}{4}$, $N_3 := \frac{(n-2)n(n+2)}{24}$, H denotes the host network, P_n, K_n, S_n are the path, clique, and star networks on n nodes, respectively.

	Optimum	Equilibria	PoA	PoS
NCG [23]	$\alpha \leq 2$: K_n [23] $\alpha \geq 2$: S_n [23]	$\alpha \leq 1$: K_n [23] $\alpha \geq 1$: S_n [23]	$2^{\mathcal{O}\left(\sqrt{\log n}\right)}$ [20] $\alpha \in \mathcal{O}\left(n^{1-\varepsilon}\right)$: $\Theta(1)$ [20] $\alpha > n(1+\varepsilon)$: $\Theta(1)$ [5]	$\alpha \leq 1$: 1 [23] $1<\alpha<2$: $\leq \frac{4}{3}$ [23] $\alpha \geq 2$: 1 [23]
BNCG [18]	$\alpha < 1$: K_n [18] $\alpha > 1$: S_n [18]	$\alpha < 1$: K_n [18] $\alpha > 1$: $S_n, \dots$ [18]	$\Theta\left(\min\left\{\sqrt{\alpha}, \frac{n}{\sqrt{\alpha}}\right\}\right)$ [18] $\alpha < 1$: 1 [18]	1 [18]
K-SDNCG	$\alpha < \frac{n}{3}$: P_n [T. 1] $\alpha > \frac{n}{3}$: K_n [T. 1]	$\alpha \leq 1$: trees [T. 2] $1 \leq \alpha \leq \frac{n}{2}$: $P_n, K_n, \dots$ [T. 2] $\alpha \geq \frac{n}{2}$: K_n [T. 2]	$\mathcal{O}(n)$ [T. 5] $\alpha \leq \sqrt{n}$: $\Theta(n)$ [T. 5] $\alpha \leq \frac{n}{6} - 3$: $\Omega\left(\frac{n}{\log n}\right)$ [T. 5] $\alpha \leq \lfloor \frac{n}{2} \rfloor - 2$: $\Omega\left(\sqrt{n}\right)$ [T. 5] $\alpha \geq \frac{n}{2}$: 1 [T. 5]	1 [T. 6]
H-NCG [19]	open	open	$\alpha < n$: $\mathcal{O}\left(\sqrt{\alpha}\right)$ [19] $\alpha \geq n$: $\min\left\{\mathcal{O}\left(\sqrt{n}, \frac{n^2}{\alpha}\right)\right\}$ [19] $\Omega\left(\min\left\{\frac{\alpha}{n}, \frac{n^2}{\alpha}\right\}\right)$ [19]	open
SDNCG	$\alpha \leq 1$: MRCST [T. 7] $\alpha > N_3$: H [T. 7]	$\alpha \leq 1$: trees [T. 9] $1 \leq \alpha \leq \frac{n}{3}$: SMRCST [T. 10] $\alpha \geq N_2$: H [T. 9]	$\mathcal{O}(n)$ [C. 14] $\alpha \leq n$: $\Theta(n)$ [T.14] $\alpha \leq N_2$: $\Omega\left(\frac{n^2}{\alpha}\right)$ [T. 14] $N_2 < \alpha \leq N_3$: $\Theta(1)$ [T. 14] $\alpha \geq N_3$: 1 [T. 14]	$\alpha \leq 1$: 1 [T. 15] $\alpha < \frac{n}{3}$: $\mathcal{O}\left(\sqrt{n}\right)$ [T. 15] $N_2 < \alpha \leq N_3$: $\Theta(1)$ [T. 15] $\alpha \geq N_3$: 1 [T. 15]

for the (K-)SDNCG cannot be derived from potential function arguments. Finally, derive several bounds for the PoA which are tight for $\alpha \geq \frac{n}{2}$, asymptotically tight for $\alpha \leq \sqrt{n}$, and asymptotically tight up to a log-factor for $\alpha \leq \frac{n}{6} - 3$.

For the SDNCG on arbitrary host networks we utilize Maximum Routing-Cost Spanning Trees for characterizing optimal networks for $\alpha \leq 1$. As our main result, we show that their locally optimal variant, the Swap-Maximal Routing-Cost Spanning Trees, and hence also Maximum Routing-Cost Spanning Trees, are pairwise stable for $\alpha \leq \frac{n}{3}$. We prove that computing the MRCST is NP-hard, while the SMRCST can be constructed efficiently. Thus, for the significant range of $1 \leq \alpha \leq \frac{n}{3}$, we not only have guaranteed equilibrium existence on any host graph, but we can compute stable states efficiently. This is in stark contrast to what is known for the inverse model studied by Demaine et al. [19]. Additionally, we approximate optimal networks and we derive several (tight) bounds on the PoA and the PoS.

Compared with the NCG [23] and the bilateral NCG [18], we find that the results for the K-SDNCG regarding optimal and stable networks are analogous but reversed, with the spanning path taking over the role of the spanning star. Moreover, our PoA results for both the K-SDNCG and the SDNCG show that our maximization variant has a significantly worse PoA that is linear or almost linear in n, compared to the PoA upper bounds of $o(n^{\varepsilon})$ and $\mathcal{O}(\sqrt{\alpha}, n/\sqrt{\alpha})$ for the NCG and the bilateral NCG, respectively. As main take away from our paper, this implies that under social distancing the agents' selfish behavior has significantly more impact on the quality of the equilibria. This calls for strong coordination mechanisms governing the network formation to avoid detrimental stable states.

2 Complete Host Networks

We analyze the properties of the K-SDNCG, i.e., the SDNCG on complete host networks. First, we characterize optimal networks and give some examples for stable networks, dependent on the relation between n and α. After that, we show several bounds on the PoA and PoS.

2.1 Stable and Optimal Networks

Intuitively, for small α the distance utility dominates the social welfare. Hence, the path should be the optimum since it maximizes the total distances. For large α, the edge utility dominates, which leads to the clique being optimal since it maximizes the number of edges. Now we show that this intuition is indeed true. Moreover, the optimal construction is unique.

▶ **Theorem 1.** *For $\alpha < \frac{n}{3}$, the unique social optimum is the path. For $\alpha > \frac{n}{3}$, the unique optimum is the clique. For $\alpha = \frac{n}{3}$, the clique and the path are the only social optima.*

Proof Sketch. [See [25] for the full proof.] Šoltés and Ľubomír [50] showed that for a fixed number of nodes and edges, the network maximizing the summed distances is unique and contains a clique and a path with at least two edges between one endpoint of the path and the clique. We call this a *PathClique*. (Note that the clique can be empty, resulting in just a path) For a visualization, we refer to Figure 1. Note, that the social optimum has to be such

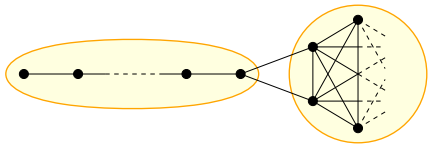


Figure 1 This figure shows a PathClique. It consists of a path (left) and a clique (right), which are connected by at least two edges between one endpoint of the path and some nodes of the clique.

a network, since for every other network, there is a PathClique with the same number of edges but larger summed distances and therefore a larger social welfare.

Let G be a PathClique but neither a clique nor a path and let v be the endpoint of the path that is connected to the clique. Observe that removing an edge between v and the clique results in a PathClique. (This is still true if there are only two edges connecting v to the clique: Removing one of these edges makes the remaining neighbor of v in the clique the new endpoint of the path and reduces the size of the clique by 1.) Similarly, adding an edge between v and the clique (or between the neighbor of v on the path and the clique if v is fully connected to the clique) yields a PathClique, too. Now, it is easy to calculate that either adding or removing edges improves the social welfare.

Thus, only the path and the clique are possibly optimal. ◀

Next, we have a look at the existence of pairwise stable networks. Similar to the social optimum, for small α, agents prefer large distances over many incident edges and therefore should remove as many edges as possible, leading to only trees being stable. Interestingly, the restrictions of pairwise stability lead to all trees being stable for small α, even if the distances are very small (like in a star). This is shown by the next theorem.

▶ **Theorem 2** (Stable Networks)**.**
(1) *For $\alpha \leq 1$, every tree is pairwise stable. For $\alpha < 1$, any pairwise stable network is a tree.*
(2) *For $\alpha \geq 1$, the clique is pairwise stable.*
(3) *For $\alpha \leq \frac{n-1}{2}$, the path is pairwise stable.*
(4) *For $\alpha > \frac{n}{2}$, the clique is the only pairwise stable network.*

Proof of (4). Let V be a set of n agents and $G = (V, E)$ be a stable network. Let $v \in V$ be a node having minimum total distances, i.e., for all $v' \in V$, we have $d_G(v, V) \leq d_G(v', V)$. Let $N_G[v]$ denote the closed neighborhood of v.

Now suppose, the network induced by $N_G[v]$ is not a clique. Then there are two neighbors x, y of v with $\{x, y\} \notin E$. We observe that for each node $z \in V$, the distances $d_G(v, z)$ and $d_G(x, z)$ can only differ by 1, since v and x are neighbors. By choice of v, there are at least as many nodes that are closer to v than nodes that are closer to x. Therefore, there are at most $\frac{n}{2}$ many nodes that are closer to x. Adding an edge between x and y can, for node y, only shorten distances to nodes which are closer to x than to v. Thus, this edge shortens the distances from y by at most $\frac{n}{2}$. The same holds for node x. Therefore, for $\alpha > \frac{n}{2}$, this edge would improve the utility of agents x and y and, thus, G would not be stable. This contradicts our assumption. Thus, $N_G[v]$ must induce a clique.

Now let x be a neighbor of v. Since x is connected to all neighbors of v, we have $d_G(x, V) \leq d_G(v, V)$, i.e., also x minimizes pairwise distances. Hence, $N_G[x]$ also induces a clique, leading to $N_G[v] = N_G[x]$. By induction, since G is connected, it must be a clique. ◀

Theorem 2 implies that socially optimal networks are also stable. In fact, they are stable for a wide range of α-values. The clique is stable for $\alpha \geq 1$, meeting the bound below which only trees are stable. Similarly, the path is stable for $\alpha \leq \frac{n-1}{2}$, almost meeting the lower bound for only the clique being stable. Additionally, we observe that we only need two networks (path and clique) to provide pairwise stable networks for all possible values of α.

For further constructions, we need the following definition. Let G be a network. We call G' a *clique network* of G, if it can be obtained by replacing each node of G by a clique of size at least 2 and for each edge of G connect the two corresponding cliques fully bipartite. By using only constant-size cliques, some properties of G (density, length of shortest paths) are preserved while the network is more stable against edge removal.

▶ **Theorem 3.** *Let G be a clique network. For $\alpha \geq 1$, G is stable against edge removal.*

Finally, we show that stable states may not be found by simply letting agents iteratively play improving moves, i.e., via a sequential process of improving strategy changes. Figure 2 provides an example of a cyclic sequence of improving moves. This also implies that both the K-SDNCG and the SDNCG do not belong to the class of potential games [45], i.e., the existence of equilibria cannot be proven via potential function arguments.

▶ **Theorem 4.** *The Social Distancing Network Creation Game is not a potential game.*

Proof. This is shown by the existence of improving cycles. See Figure 2 for an example. ◀

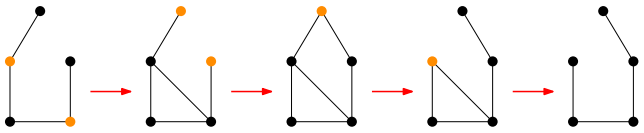


Figure 2 This figure shows a cyclic sequence of improving moves performed by $n = 5$ agents for $\alpha = 2.5$. In each step, the nodes responsible for the next change are highlighted in orange. Note that the last step is isomorphic to the first step.

2.2 Price of Anarchy and Price of Stability

In this section, we give a series of bounds for the Price of Anarchy and the Price of Stability.

▶ **Theorem 5** (Price of Anarchy).

(1) *The Price of Anarchy is in* $\mathcal{O}(n)$.

(2) *For* $\alpha \leq 1$, *the Price of Anarchy is in* $\Theta(n)$.

(3) *For* $1 < \alpha \leq \sqrt{n}$, *the Price of Anarchy is in* $\Theta(n)$.

(4) *For* $\sqrt{n} \leq \alpha \leq \frac{n}{6} - 3$, *the Price of Anarchy is in* $\Omega\left(\frac{n}{\log n}\right)$.

(5) *For* $\frac{n}{6} - 3 < \alpha \leq \lfloor \frac{n}{2} \rfloor - 2$, *the Price of Anarchy is in* $\Omega\left(\sqrt{n}\right)$.

(6) *For* $\alpha \geq \frac{n}{2}$, *the Price of Anarchy is 1.*

Proof of (3). We construct a star-like network with cliques as leaves in the following way. Let $c \coloneqq \lceil \alpha \rceil + 2$. Additionally, let $K_1, \ldots, K_d$ be $d \coloneqq \lfloor \frac{n-2}{c} \rfloor$ cliques containing $c-2$ nodes and $v_1, v_1', v_2, v_2', \ldots, v_d, v_d'$ be $2d$ nodes. Let furthermore M be a clique of size $n - cd$. We now define our network G as

$$V_G \coloneqq \bigcup_{i=1}^{d} V_{K_i} \cup \bigcup_{i=1}^{d} \{v_i, v_i'\} \cup V_M$$

$$E_G \coloneqq \bigcup_{i=1}^{d} E_{K_i} \cup E_M \cup \bigcup_{i=1}^{d} \{\{v_i, v_i'\}\} \cup \bigcup_{i=1}^{d} \bigcup_{v \in K_i} \{\{v, v_i\}, \{v, v_i'\}\} \cup \bigcup_{i=1}^{d} \bigcup_{v \in M} \{\{v, v_i\}, \{v, v_i'\}\}.$$

We essentially connect the outer cliques $K_1, \ldots, K_d$ to the center clique M via d 2-cliques and each connection is fully bipartite (see Figure 3). Since $n = |V_G|$, G is a network of the desired size.

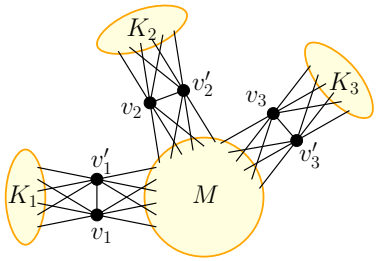


Figure 3 The figure shows a star-like clique network, where the center is formed by a clique M and each ray consists of two nodes v_i, v_i' and a clique K_i.

We now show that G is pairwise stable. We see that G is a clique network. Because of Theorem 3 and $\alpha > 1$, G is stable against edge removal. On the other hand, adding an edge shortens distances to at least $|K_i| = c - 2 \geq \alpha$ nodes which means a distance decrease of at least α for the two incident nodes. This also does not increase their utility. Therefore, G is pairwise stable.

For the center clique M, we see that $|V_M| = n - cd = n - c\left\lfloor \frac{n-2}{c} \right\rfloor$ and therefore $2 \leq |V_M| < n - (n-2-c) = c+2$. With this and $1 < \alpha \leq \sqrt{n}$, we obtain

$$\begin{aligned}|E_G| &= d\binom{c-2}{2} + \binom{|V_M|}{2} + 2d + d(c-2)2 + d|V_M|2 \\ &= \left\lfloor \frac{n-2}{\lceil\alpha\rceil+2} \right\rfloor \left(\frac{\lceil\alpha\rceil(\lceil\alpha\rceil-1)}{2} + 2 + 2\lceil\alpha\rceil + 2|V_M| \right) + \frac{|V_M|(|V_M|-1)}{2} \\ &\in \Theta(\alpha n)\end{aligned}$$

and $d_G(V_G, V_G) \in \Theta(n^2)$.

For $\alpha < \sqrt{n}$, the socially optimal network is the path. With the previous calculations, we can now bound the Price of Anarchy as

$$\text{PoA} \geq \frac{2\alpha(n-1) + \Theta(n^3)}{2\alpha\Theta(\alpha n) + \Theta(n^2)} = \frac{\Theta(n^3)}{\Theta(n^2)} \in \Omega(n).$$

From (1), we have $\text{PoA} \in \mathcal{O}(n)$ and therefore $\text{PoA} \in \Theta(n)$. ◀

Proof of (4). Let $d = \lfloor \log n \rfloor - 1$. Then, the d-dimensional hypercube is represented by G_H with $V_{G_H} = \{0,1\}^d$ and $E_{G_H} = \{\{v,x\} \mid v,x \in V \wedge d_H(v,x) = 1\}$ where $d_H(v,x)$ denotes the Hamming Distance between v and x. Let G be a clique network for G_H with $|V_G| = n$ such that the sizes of the cliques replacing the nodes of G_H differ by at most 1. Observe, that each clique is of size 2 or 3 if $2 \cdot 2^d \leq n < 3 \cdot 2^d$ and of size 3 or 4 if $3 \cdot 2^d \leq n < 4 \cdot 2^d$. By Theorem 3 and since $\alpha \geq 1$, we know that G is stable against edge removal. We now show that adding an edge shortens the total distances for the incident nodes by at least $\frac{n}{6} - 3$.

Let $v, x \in V_G$ such that $e \coloneqq \{v,x\} \notin E_G$ and let $v', x' \in V_{G_H}$ be the nodes corresponding to the cliques that contain v and x, respectively. Therefore, $e' \coloneqq \{v', x'\} \notin E_{G_H}$, which implies $d_H(v', x') \geq 2$. By symmetry of the hypercube, we can assume w.l.o.g. that

$$v' = \underbrace{00\ldots0}_{d_H(v,x)}\ \underbrace{0\ldots00}_{d-d_H(v,x)} \qquad \text{and} \qquad x' = \underbrace{11\ldots1}_{d_H(v,x)}\ \underbrace{0\ldots00}_{d-d_H(v,x)}\ .$$

Adding e' to G_H decreases the distances from v' to another node $y' \in V_{G_H}$ if and only if $d_H(v', y') \geq d_H(x,y) + 2$. The difference in distance can only come from the first $d_H(v', x')$ bits of the label since the remaining bits are equal for v' and x'. Let ℓ be the number of the first $d_H(v', x')$ bits of y' equal to 1. Then, $d_H(v', x') - \ell$ is the number of the first $d_H(v', x')$ bits of y' equal to 0. We obtain $d_H(v', y') - d_H(x', y') = \ell - (d_H(v', x') - \ell) = 2\ell - d_H(v', x')$. Thus, adding e' to G_H shortens the distance from v' to y' by $2\ell - d_H(v', x') - 1$.

The number of nodes where exactly ℓ of the first $d_H(v', x')$ bits are equal to 1 is $\binom{d_H(v',x')}{\ell} \cdot 2^{d-d_H(v',x')}$. Therefore, we get a distance decrease for v' of

$$\begin{aligned}&d_{G_H}(v', V_{G_H}) - d_{G_H+e'}(v', V_{G_H}) \\ &= \sum_{\ell=\left\lceil \frac{d_H(v',x')}{2} \right\rceil + 1}^{d_H(v',x')} \binom{d_H(v',x')}{\ell} \cdot 2^{d-d_H(v',x')} \cdot (2\ell - d_H(v',x') - 1) \overset{\text{steps omitted}}{\geq} \frac{2^d}{4}.\end{aligned}$$

Observe that the distances from v to all nodes in other cliques in G are exactly the same as the distances from v' to all other nodes in G_H. The same holds for $G + e$ and $G_H + e'$, with the exception of the distances from v to the (at most 3) nodes in the same clique as x. We distinguish two cases:

If $2 \cdot 2^d \leq n < 3 \cdot 2^d$, each clique consists of 2 or 3 nodes. Therefore, we have a distance decrease of at least

$$d_G(v, V_G) - d_{G+e}(v, V_G) \geq 2(d_{G_H}(v', V_{G_H}) - d_{G_H+e'}(v', V_{G_H})) - 3 \geq \frac{2^d}{2} - 3 \geq \frac{n}{6} - 3.$$

If $3 \cdot 2^d \leq n < 4 \cdot 2^d$, each clique consists of 3 or 4 nodes. This means, we have a distance decrease of at least

$$d_G(v, V_G) - d_{G+e}(v, V_G) \geq 3(d_{G_H}(v', V_{G_H}) - d_{G_H+e'}(v', V_{G_H})) - 3 \geq 3\frac{2^d}{4} - 3 \geq \frac{n}{6} - 3.$$

Thus, edge additions are not beneficial for the incident agents if $\alpha \leq \frac{n}{6} - 3$ and we conclude that the constructed network is stable for $1 \leq \alpha \leq \frac{n}{6} - 3$. We also see that the number of edges m is in $\Theta(n \log n)$ and the distance $d(V_G, V_G)$ is in $\Theta(n^2 \log n)$. Since the social optimum for $1 \leq \alpha \leq \frac{n}{6} - 3$ is the path P_n, we get for the Price of Anarchy:

$$\text{PoA}_n \geq \frac{SW(P_n)}{SW(G)} = \frac{\alpha(n-1) + \Theta(n^3)}{\alpha\Theta(n \log n) + \Theta(n^2 \log n)} \in \Omega\left(\frac{n}{\log n}\right). \quad \blacktriangleleft$$

Proof of (5). We construct a path of cliques in the following way. Let $2 \leq d \leq \frac{n-6}{2}$ be some even number and $c = \left\lfloor \frac{n-6}{d} \right\rfloor$. Furthermore, let $K_1, \ldots, K_d$ be d cliques consisting of c or $c+1$ nodes, such that $\sum_{i=1}^{d} |V_{K_i}| = n - 6$ and $\sum_{i=1}^{\frac{d}{2}} |V_{K_i}| = \left\lceil \frac{n-6}{2} \right\rceil$ and $\sum_{i=\frac{d}{2}+1}^{d} |V_{K_i}| = \left\lfloor \frac{n-6}{2} \right\rfloor$, and $v_1, v_1', v_2, v_2', v_3, v_3'$ be 6 more nodes. We now define the network G as

$$V_G \coloneqq \bigcup_{i=1}^{d} V_{K_i} \cup \{v_1, v_1', v_2, v_2', v_3, v_3'\},$$

$$E_G \coloneqq \bigcup_{i=1}^{d} E_{K_i} \cup \{\{v_1, v_1'\}, \{v_2, v_2'\}, \{v_3, v_3'\}\} \cup \{\{v, x\} \mid v \in \{v_2, v_2'\} \wedge x \in \{v_1, v_1', v_3, v_3'\}\}$$
$$\cup \bigcup_{i=1}^{\frac{d}{2}-1} \{\{v, x\} \mid v \in K_i \wedge x \in K_{i+1}\} \cup \bigcup_{v \in K_{\frac{d}{2}}} \{\{v, v_1\}, \{v, v_1'\}\}$$
$$\cup \bigcup_{i=\frac{d}{2}+1}^{d-1} \{\{v, x\} \mid v \in K_i \wedge x \in K_{i+1}\} \cup \bigcup_{v \in K_{\frac{d}{2}+1}} \{\{v, v_3\}, \{v, v_3'\}\}.$$

Figure 4 shows a sketch of G.

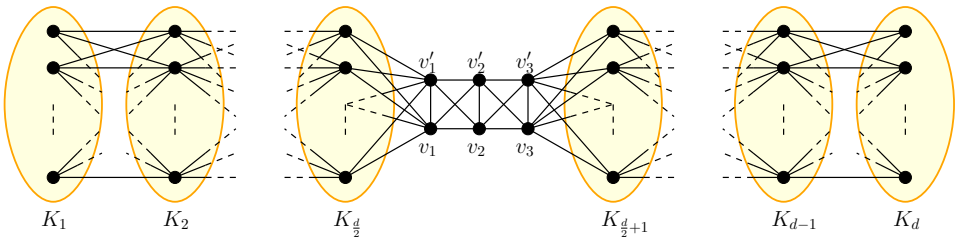


Figure 4 The figure shows clique network for a path consisting of d cliques $K_1, \ldots, K_d$ highlighted in yellow with 6 additional nodes in the middle. Note, that edges inside the cliques are not shown in this figure.

We observe that G is stable against edge removal because of Theorem 3, since $\alpha > \frac{n}{6} - 3 \geq 1$ and G being a clique network for the path. We now show that adding an edge is also not an improving move.

We quickly see that, for a node v, adding an edge into the 2-neighborhood always shortens distances the least. We therefore only have to consider these edges. We observe that adding an edge between v_1 and v_3 (or because of symmetry, v_1' or v_3') decreases distances from v_1 to v_3 and all nodes in $K_{\frac{d}{2}+1}, \dots, K_d$ and decreases distances from v_3 to v_1 and all nodes in $K_1, \dots, K_{\frac{d}{2}}$ by exactly 1. This means, we get

$$d_G(v_1, V) - d_{G+\{v_1,v_3\}}(v_1, V) = 1 + \sum_{i=\frac{d}{2}+1}^{d} |V_{K_i}| = \left\lfloor \frac{n}{2} \right\rfloor - 2 \quad \text{and}$$

$$d_G(v_3, V) - d_{G+\{v_1,v_3\}}(v_3, V) = 1 + \sum_{i=1}^{\frac{d}{2}} |V_{K_i}| = \left\lceil \frac{n}{2} \right\rceil - 2.$$

Every other edge we could add decreases distances to all the cliques of one side of the path, resulting in larger distance decreases. This means that adding an edge is not an improving move for $\alpha \leq \left\lfloor \frac{n}{2} \right\rfloor - 2$. Therefore, G is pairwise stable for the desired values of α.

We now evaluate the number of edges. We have $|E_{K_i}| \in \Theta(c^2)$. The number of edges between two neighboring cliques is also in $\Theta(c^2)$. This means that the total number of edges is $|E_G| \in \Theta(dc^2)$. We also see that the diameter of G is d and therefore $d_G(V, V) \in \mathcal{O}(dn^2)$. If we choose $d = 2\left\lfloor \frac{\sqrt{n}}{2} \right\rfloor$, we have $d \in \Theta(\sqrt{n})$ and $c \in \Theta(\sqrt{n})$. Since $\alpha \in \Theta(n)$, we get for the Price of Anarchy

$$\text{PoA}_n \geq \frac{\alpha(n-1) + \Theta(n^3)}{\alpha\Theta(dc^2) + \mathcal{O}(dn^2)} \in \Omega\left(\frac{n^3}{n^{\frac{5}{2}} + n^{\frac{5}{2}}}\right) = \Omega\left(\sqrt{n}\right). \qquad \blacktriangleleft$$

We have established that the Price of Anarchy is relatively high for $\alpha \leq \frac{n}{2}$. It even meets the trivial upper bound of $\mathcal{O}(n)$ for a large range of α. In contrast to the high PoA values, we observe that the Price of Stability is independent of α and best possible.

▶ **Theorem 6.** *The Price of Stability is 1.*

From an efficiency point-of-view, the huge gap between the PoA and the PoS suggests that having an outside force assigning an initial strategy to all players is beneficial. That way, stability and optimal social welfare can be guaranteed. Without such coordination, the players could end up in socially bad equilibria or in a cyclic sequence of improving moves.

3 General Host Networks

We now analyze the SDNCG on arbitrary connected but not necessarily complete host networks. First, we analyze socially optimal networks and then we investigate the existence of pairwise stable networks. We prove our main result that establishes equilibrium existence on any connected host network for a wide parameter range of α. Finally, we derive bounds on the Price of Anarchy and the Price of Stability. Additionally, we show that computing the social optimum and the Maximum Routing-Cost Spanning Tree is NP-hard while computing a Swap-Maximal Routing-Cost Spanning Tree can be done in polynomial time.

3.1 Stable and Optimal Networks

While for the K-SDNCG, we only have two possible social optima (dependent on α), this gets more complicated for general host networks. Of course, if they exist on general host networks, then the optima for the K-SDNCG are still the most efficient networks. Intuitively,

if the host network does not contain a Hamilton path, then the social optimum should be a tree if α is small enough. Since all trees have the same number of edges, the social welfare of a tree is only influenced by the total distances. Remember that the spanning tree maximizing the total distances is by definition the Maximum Routing-Cost Spanning Tree (MRCST). We now show, that this intuition is indeed correct.

▶ **Theorem 7** (Social Optimum). *Let H be a connected host network containing n nodes.*

(1) *If H contains a Hamilton path, then this path is the social optimum for $\alpha \leq \frac{n}{3}$. The Hamilton path is the unique social optimum if $\alpha < \frac{n}{3}$.*

(2) *For $\alpha \leq 1$, the MRCST of H is socially optimal.*

(3) *For $\alpha > \frac{1}{24}(n-2)n(n+2)$, H itself is the unique social optimum.*

Contrasting statement (3) from Theorem 7, we observe that for $\alpha < \frac{1}{24}(n-2)n(n+2)$, the host network is not necessarily the social optimum. Consider the host network $H \coloneqq C_n$ for even n, i.e., an even cycle with n nodes. In the proof of (3), we see that $\mathrm{SW}(P_n) > \mathrm{SW}(C_n)$, implying that C_n cannot be the social optimum. In fact, in this example, P_n is the optimum since there are only two possible states (up to isomorphism): P_n and C_n itself. This is in stark contrast to the K-SDNCG, where the host network is optimal for $\alpha \geq \frac{n}{3}$.

Since finding a MRCST is NP-hard [16], finding the social optimum for a given host network must also be NP-hard.

▶ **Theorem 8** (Computational Hardness). *Computing the social optimum for a connected host network H is NP-hard.*

Next, we discuss stable networks. In contrast to the K-SDCNG, it is not obvious that pairwise stable networks are guaranteed to exist for any connected host network. However, we can directly transfer the result that spanning trees are stable for small α. For large α, similar to the clique being the unique stable network for $\alpha > \frac{n}{2}$ for complete host networks, as shown in Theorem 2, we show that the whole host network is pairwise stable. However, in contrast to the K-SDNCG, this is true only for much larger values of α.

▶ **Theorem 9** (Stable Networks). *Let H be a connected host network containing n nodes.*

(1) *For $\alpha \leq 1$, every spanning tree of H is pairwise stable. For $\alpha < 1$, spanning trees are the only pairwise stable networks.*

(2) *For $\alpha > \frac{1}{4}(n-1)^2$, H is the only pairwise stable network.*

Contrasting statement (2) of Theorem 9, using $H \coloneqq C_n$ for odd n and $\alpha < \frac{1}{4}(n-1)^2$ shows that the host network is not necessarily pairwise stable. This example also shows that the optimum is not necessarily stable: For $\alpha \geq \frac{1}{4}(n-1)^2$ and $H \coloneqq C_n$ as the host network, C_n is the only pairwise stable network but it is not the optimum for $\alpha < \frac{1}{24}(n-2)n(n+2)$. This is another significant difference to the K-SDNCG.

Now that we characterized stable networks for extreme α-values, the question remains whether stable states also exist for in-between values. For the K-SDNCG, the path is stable up to $\alpha < \frac{n-1}{2}$. This is, of course, still true for non-complete host networks if they contain a Hamilton path. Since a Hamilton path (if it exists) is the MRCST, it is natural to suspect that the MRCST properties at least partially ensure stability for some $\alpha \geq 1$. However, even if true, the MRCST is still NP-hard to compute. Hence, in quest of an efficiently computable stable network, we introduce a less strict variant of MRCSTs which is only locally optimal: Swap-Maximal Routing-Cost Spanning Trees. Remember, a SMRCST is a spanning tree whose summed distances cannot be increased by removing one edge and adding another edge.

As our main result, we now show that SMRCSTs (and therefore MRCSTs, too) are indeed stable beyond $\alpha \leq 1$. Note, that for the inverse model of the NCG on an arbitrary host network [19], so far no equilibrium existence statement is known.

▶ **Theorem 10.** *Let H be a connected host network containing n nodes. Then for $\alpha \leq \frac{n}{3}$, any Swap-Maximal Routing-Cost Spanning Tree is pairwise stable.*

Proof Sketch. [See [25] for the full proof.]
Let G with $V_G = V_H$ and $E_G \subseteq E_H$ be a SMRCST. Since G is a tree, we only have to consider edge additions. We show that adding any edge decreases the summed distances for at least one of the endpoints of the edge by at least $\frac{n}{3}$. This is sufficient to show the claim.

Let $e_1 \in E_H \setminus E_G$ be an edge not contained in the SMRCST. Adding e_1 would form a cycle of length $d \in \mathbb{N}$ consisting of nodes $v_1, \ldots, v_d \in V$, with v_1 and v_d being the nodes incident to e_1. Let E_C be the set of all edges on this cycle. Removing all edges in E_C from G would create d trees rooted in $v_1, \ldots, v_d$, respectively. Furthermore, let $x_1, \ldots, x_d$ be the number of nodes in each of the d trees. See Figure 5 for an illustration.

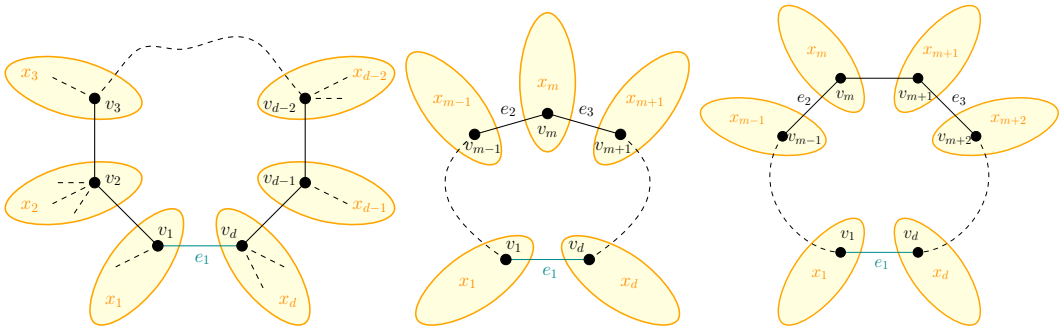


Figure 5 This figure shows the cycle formed by adding e_1 to the SMRCST. The cycle is of length d and contains the nodes $v_1, \ldots, v_d$. Every other node is contained in one of the subtrees rooted in one of the nodes on the cycle. These subtrees are represented in yellow. The number of nodes contained in the subtree rooted in v_i is x_i. Middle and right: the cycle for d being odd or even, respectively, and the two special edges e_2 and e_3.

Since G is a tree, there is exactly one path between each pair of nodes (which is also the shortest). For each edge $e \in E_G$, we define $d_G(e)$ as the number of paths between two nodes in G which include e. We then can express the total distances as

$$d_G(V, V) = 2 \sum_{e \in E_G} d_G(e).$$

Note, that each path between two nodes contributes twice to the total distances (one for each node), which leads to the factor of 2.

Let $x \coloneqq (x_1, \ldots, x_d)$. We now define for each edge $e \in E_C$ on the cycle

$$c_e(x) \coloneqq \sum_{e' \in E_C} d_{G+e_1-e}(e') = \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} x_i x_j d_{G+e_1-e}(v_i, v_j).$$

This is the contribution of all the edges on the cycle to the total distances, if we add e_1 to it and instead remove e from it. Note, that c_{e_1} is the value for the original network since $G + e_1 - e_1 = G$. We see that c_e does not depend on the structure of the subtrees rooted in the v_i but only on the number of nodes in each subtree. Since the number of paths going over an edge that is not on the cycle does not change when we add e_1 and remove e, we have

$$\begin{aligned} d_G(V,V) - d_{G+e-e_1}(V,V) &= 2\sum_{e' \in E}(d_G(e') - d_{G+e-e_1}(e')) \\ &= 2\sum_{e' \in E_C}(d_G(e') - d_{G+e-e_1}(e')) + 2\sum_{e' \in E_G \setminus E_C}(d_G(e') - d_{G+e-e_1}(e')) \\ &= 2c_{e_1}(x) - 2c_e(x). \end{aligned}$$

We know that G is a SMRCST of H. This implies $d_G(V,V) \geq d_{G'}(V,V)$ for any other spanning tree G' that can be obtained from G by an edge swap. We therefore also have

$$\forall e \in E_C \colon c_{e_1}(x) \geq c_e(x). \tag{1}$$

Now, we use the previous observations to formulate and solve a minimization problem that yields the desired bound. We start with some definitions.

We call $x = (x_1, \ldots, x_d) \in \mathbb{N}^d$, with $x_i \geq 1$ and $\sum_{i=1}^{d} x_i = n$, a *node distribution*. For each edge $e \in E_C$, we call $c_e(x)$ (defined above) the *cost* of e. And lastly, we define the *distance decrease* Δd as

$$\Delta d(x) \coloneqq \max \left\{ \sum_{i=1}^{\lfloor \frac{d-1}{2} \rfloor} (d-2i)x_i, \sum_{i=1}^{\lfloor \frac{d-1}{2} \rfloor} (d-2i)x_{d-i+1} \right\}. \tag{2}$$

The goal then is: Find a node distribution x that fulfills (1) and minimizes $\Delta d(x)$. Observe that this indeed yields a lower bound for the distance decrease when adding e to G. If we show that this is at least $\frac{n}{3}$, we have proved the statement.

Let $x = (x_1, \ldots, x_d) \in \mathbb{N}^d$ be a node distribution minimizing $\Delta d(x)$. For $d \leq 4$, it is easy to show that $\Delta d(x) \geq \frac{n}{3}$. For further steps, we allow $x \in \mathbb{R}^d_{\geq 1}$. Note, that this only allows for smaller minima and therefore still yields a lower bound for the original problem.

The high level idea of the following steps is that we can redistribute weights of the node distribution x without changing $\Delta(x)$ or violating 1 and thereby reducing the number of variables contained in x by setting most x_i to 1. We now make a case distinction.

Case d odd: Let $m = \frac{d+1}{2}$ and $e_2 \coloneqq \{v_{m-1}, v_m\}$ and $e_3 \coloneqq \{v_m, v_{m+1}\}$. Thus, v_m is the node equidistant from v_1 and v_d in C and e_2 and e_3 are the edges on C incident to v_m. (see Figure 5 (middle)) We will only consider the two constraints

$$c_{e_1}(x) \geq c_{e_2}(x) \quad \text{and} \quad c_{e_1}(x) \geq c_{e_3}(x). \tag{3}$$

Again, this still yields a lower bound for the original problem.

We define the node distributions $x^{(1)}, x^{(2)}, x^{(3)}$ such that for all $1 \leq i \leq d$

$$x_i^{(1)} \coloneqq \begin{cases} x_1 + \sum_{p=2}^{m-2} \frac{m-1-p}{m-2}(x_p - 1) & \text{if } i = 1, \\ x_{m-1} + \sum_{p=2}^{m-2} \frac{p-1}{m-2}(x_p - 1) & \text{if } i = m-1, \\ x_m & \text{if } i = m, \\ x_{m+1} + \sum_{p=m+2}^{d-1} \frac{d-p}{m-2}(x_p - 1) & \text{if } i = m+1, \\ x_d + \sum_{p=m+2}^{d-1} \frac{p-m-1}{m-2}(x_p - 1) & \text{if } i = d, \\ 1 & \text{else,} \end{cases}$$

(w.l.o.g. we assume $x_1^{(1)} \geq x_d^{(1)}$ and $(2m-3)x_1^{(1)} + x_{m-1}^{(1)} = (2m-3)x_d^{(1)} + x_{m+1}^{(1)}$)

$$x_i^{(2)} \coloneqq \begin{cases} x_1^{(1)} & \text{if } i = 1 \text{ or } i = d, \\ x_{m-1}^{(1)} & \text{if } i = m-1 \text{ or } i = m+1, \\ x_m^{(1)} + (2m-4)\left(x_1^{(1)} - x_d^{(1)}\right) & \text{if } i = m, \\ 1 & \text{else,} \end{cases}$$

$$x_i^{(3)} \coloneqq \begin{cases} x_m^{(2)} + 2\frac{2m-4}{2m-3}\left(x_{m-1}^{(2)} - 1\right) & \text{if } i = m, \\ x_1^{(2)} + \frac{1}{2m-3}\left(x_{m-1}^{(2)} - 1\right) & \text{if } i = 1 \text{ or } i = d, \\ 1 & \text{else.} \end{cases}.$$

It can be shown iteratively that $\Delta d(x) = \Delta d\left(x^{(1)}\right) = \Delta d\left(x^{(2)}\right) = \Delta d\left(x^{(3)}\right)$ and that $c_{e_1}\left(x^{(i)}\right) \geq c_{e_2}\left(x^{(i)}\right)$ and $c_{e_1}\left(x^{(i)}\right) \geq c_{e_3}\left(x^{(i)}\right)$ for $1 \leq i \leq 3$. This means, $x^{(3)}$ is also a solution of the minimization problem. Because of $x^{(3)}$ only having 2 variables left $\left(x_1^{(3)} = x_d^{(3)} \text{ and } x_m^{(3)}\right)$, it is easy to show $\Delta d\left(x^{(3)}\right) \geq \frac{n}{3}$.

Case d even: Let $m = \frac{d}{2}$ and $e_2 \coloneqq \{v_{m-1}, v_m\}$ and $e_3 \coloneqq \{v_{m+1}, v_{m+2}\}$. (see Figure 5 (right)) Again, we will only consider the two constraints

$$c_{e_1}(x) \geq c_{e_2}(x) \text{ and } c_{e_1}(x) \geq c_{e_3}(x). \tag{4}$$

The rest of the reasoning is analogous to the odd case. ◀

Next, we show that we can find a SMRCST in polynomial time via Algorithm 1 and even guarantee some bounds on the social welfare of the resulting network. Our algorithm employs

Algorithm 1 Computes a SMRCST for a given connected host network.

Input: connected host network H
Output: SMRCST T
1 $\mathsf{P} \leftarrow$ `greedyLongPath` (H);
2 $\mathsf{T} \leftarrow$ *extend* P *to form a spanning tree of* H;
3 **while** $\exists e \in E_\mathsf{T}, e' \in E_\mathsf{H} \setminus E_\mathsf{T} : d_{\mathsf{T}-e+e'}(V_\mathsf{H}, V_\mathsf{H}) > d_\mathsf{T}(V_\mathsf{H}, V_\mathsf{H})$ **do**
4 $\quad \mathsf{T} \leftarrow \mathsf{T} - e + e'$
5 **end**

a greedy algorithm developed by Karger et al. [36] which can find a path of length at least $\frac{|E_H|}{|V_H|}$ in $\mathcal{O}(|E_H|)$ as a subroutine for initialization. We call this subroutine *greedyLongPath*. This will help us to derive bounds on the total distances of the computed SMRCST later. For extending the path to a spanning tree in line 2 of Algorithm 1, we can simply iterate over all edges and add them to the network if they do not close a cycle.

▶ **Theorem 11.** *Let H be a connected network containing n nodes and m edges. Then Algorithm 1 finds a Swap-Maximal Routing-Cost Spanning Tree of H in runtime $\mathcal{O}(n^5 m)$.*

Proof. It is easy to see that, by construction, T is always a spanning tree of H. The condition in the while-loop ensures that all possible swaps are tried. This means that the while-loop ends if and only if T is a SMRCST. Therefore if the while-loop stops, the result is correct.

In every iteration in which the while-loop does not stop, the total distances of T increase by at least 1. Since the tree maximizing the total distances is the path, we get its total distances $\frac{1}{3}(n-1)n(n+1) \in \mathcal{O}(n^3)$ as an upper bound for the number of iterations.

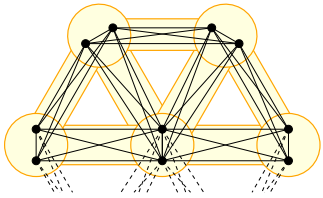


Figure 6 This figure shows a clique network (black) for a wheel network (yellow).

The runtime of Algorithm 1 is clearly dominated by the while-loop. Since T has $n-1$ edges which can be removed and $E_H \setminus E_T$ has $\mathcal{O}(m)$ possible edges to add, the number of possible swaps is in $\mathcal{O}(nm)$. For each swap, the total distances can be computed in $\mathcal{O}(n)$ [44]. Therefore computing the condition can be done in $\mathcal{O}(n^2m)$. Altering the current solution in the body of the while-loop only takes $\mathcal{O}(n)$ when using adjacency lists. Since there are at most $\mathcal{O}(n^3)$ iterations of the while-loop, the overall runtime is in $\mathcal{O}(n^5m)$. ◀

For the K-SDNCG, the social optima were also stable. For general host networks, this is not necessarily the case. However, we can show that for $\alpha \leq \frac{n}{3}$ there are stable states which approximate the social welfare better than with the trivial factor of $\mathcal{O}(n)$.

▶ **Theorem 12** (OPT-Approximation via the MRCST). *Let H be a connected host network containing n nodes and m edges and T be the MRCST of H.*

(1) *We have* $\frac{\mathrm{SW}(\mathrm{OPT}_H)}{\mathrm{SW}(T)} \in \mathcal{O}\left(\frac{m}{n}\right)$.

(2) *For* $\alpha \in \mathcal{O}\left(\frac{n^2}{m}\right)$, *we have* $\frac{\mathrm{SW}(\mathrm{OPT}_H)}{\mathrm{SW}(T)} \in \mathcal{O}(1)$.

(3) *For* $\alpha \in \omega\left(\frac{n^2}{m}\right)$, *we have* $\frac{\mathrm{SW}(\mathrm{OPT}_H)}{\mathrm{SW}(T)} \in \mathcal{O}\left(\min\left\{\frac{m}{n}, \alpha\frac{n}{m}\right\}+1\right)$.

Since finding the MRCST is NP-hard [16], these are only existence results. However, the next theorem yields a bound for dense networks and the computed SMRCST from Algorithm 1.

▶ **Theorem 13.** *Let H be a connected host network containing n nodes and m edges and T be the SMRCST obtained by Algorithm 1. Then for $\alpha \in \mathcal{O}(n)$, we have* $\frac{\mathrm{SW}(\mathrm{OPT}_H)}{\mathrm{SW}(T)} \in \mathcal{O}\left(\frac{n^4}{m^2}\right)$.

This means, for $\alpha \leq \frac{n}{3}$ and a dense host network, we can compute a state which is pairwise stable and also has a favorable social welfare.

3.2 Price of Anarchy and Price of Stability

We derive several bounds on the PoA and the PoS for the SDNCG. For the K-SDNCG, the PoA is already quite high for small α. The next Theorem shows that this gets even worse for general host networks since the PoA is linear up to $\alpha \leq n$ and super-constant for $\alpha \in o(n^2)$.

▶ **Theorem 14** (Price of Anarchy).

(1) *The Price of Anarchy is in $\mathcal{O}(n)$.*

(2) *For $\alpha < 1$, the Price of Anarchy is in $\Theta(n)$.*

(3) *For $1 \leq \alpha \leq n$, the Price of Anarchy is in $\Theta(n)$.*

(4) *For $n < \alpha \leq n^2$, the Price of Anarchy is in $\Omega\left(\frac{n^2}{\alpha}\right)$.*

(5) *For $\frac{1}{4}(n-1)^2 < \alpha \leq \frac{1}{24}(n-2)n(n+2)$, the Price of Anarchy is in $\Theta(1)$.*

(6) *For $\alpha > \frac{1}{24}(n-2)n(n+2)$, the Price of Anarchy is 1.*

Proof of (3) and (4). Let $W = (V_W, E_W)$ be a wheel network on $n' \coloneqq \lfloor \frac{n}{2} \rfloor$ nodes, i.e.,

$$V_W \coloneqq \{v_1, \dots, v_{n'}\} \quad \text{and}$$
$$E_W \coloneqq \{\{v_1, v_i\} \mid 2 \leq i \leq n'\} \cup \{\{v_i, v_{i+1}\} \mid 2 \leq i \leq n'\} \cup \{\{v_2, v_{n'}\}\}.$$

We then define the host network H as the clique network obtained by replacing every node of W by a clique of size 2. (See Figure 6 for an illustration.) For odd n, we instead replace the central node by a clique of size 3. We see that H contains n nodes, $\Theta(n)$ edges, and most importantly a Hamilton path. We also know that H is stable because of Theorem 3 and since no edge can be added. This yields the following lower bound for the Price of Anarchy

$$\text{PoA} \geq \frac{SW(P)}{SW(H)} = \frac{\alpha(n-1) + \Theta(n^3)}{\alpha\Theta(n) + \Theta(n^2)} \in \Omega\left(\min\left\{n, \frac{n^2}{\alpha}\right\}\right),$$

which proves the claim. ◀

▶ **Theorem 15** (Price of Stability)**.**
(1) *The Price of Stability is in* $\mathcal{O}(n)$*.*
(2) *For* $\alpha \leq 1$*, the Price of Stability is 1.*
(3) *For* $1 < \alpha \leq \frac{n}{3}$*, the Price of Stability is in* $\mathcal{O}(\sqrt{n})$*.*
(4) *For* $\frac{1}{4}(n-1)^2 < \alpha \leq \frac{1}{24}(n-2)n(n+2)$*, the Price of Stability is in* $\Theta(1)$*.*
(5) *For* $\alpha > \frac{1}{24}(n-2)n(n+2)$*, the Price of Stability is 1.*

4 Conclusion

We introduced and analyzed a natural game-theoretic model for network formation governed by social distancing. Besides modeling this timely issue, our model resembles the inverse compared to the well-known (bilateral) Network Creation Game [23, 18]. Thus, via our analysis we could explore the impact of inverting the utility function in a non-trivial strategic game. We find that this inverts some of the properties, like the rough structure of optimum states, while it also yields non-obvious insights. First of all, for the variant with non-complete host networks we could show a strong equilibrium existence result, whereas no such result is known for the inverse model. Moreover, we established that the PoA is significantly higher in the (K-)SDCNG compared to the (bilateral) NCG. This demonstrates that the impact of the agents' selfishness is higher under social distancing, which calls for external coordination.

The most obvious open question for future work is to settle the equilibrium existence. Do pairwise stable states exist for all connected host networks H and α? Another research direction would be to consider the unilateral variant of the SDNCG. While this no longer realistically models the formation of social networks, it might still yield interesting insights and it allows for studying stronger solution concepts like the Nash equilibrium or strong Nash equilibria, similar to [34, 6]. Also, altering the utility function, e.g., to using the maximum distance instead of the summed distances, or the probability of infection, similar to [13], seems promising. Finally, also considering weighted host networks, as in [11], where the edge weight models the benefit of the social interaction, would be an interesting generalization.

References

1 Daron Acemoglu, Asuman Ozdaglar, and Alireza Tahbaz-Salehi. Systemic risk and stability in financial networks. *Am. Econ. Review*, 105(2):564–608, 2015. doi:10.1257/aer.20130456.
2 Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On nash equilibria for a network creation game. *ACM TEAC*, 2(1):1–27, 2014. doi:10.1145/2560767.

3 Franklin Allen and Douglas Gale. Financial contagion. *Journal of Political Economy*, 108(1):1–33, 2000. doi:10.1086/262109.

4 Carme Àlvarez and Arnau Messegué. Network creation games: Structure vs anarchy. *CoRR*, abs/1706.09132, 2017. doi:10.48550/arXiv.1706.09132.

5 Carme Àlvarez and Arnau Messegué. On the price of anarchy for high-price links. In *WINE 2019*, pages 316–329, 2019. doi:10.1007/978-3-030-35389-6_23.

6 Nir Andelman, Michal Feldman, and Yishay Mansour. Strong price of anarchy. *Games and Economic Behavior*, 65(2):289–317, 2009. doi:10.1016/j.geb.2008.03.005.

7 Elliot Anshelevich, Anirban Dasgupta, Jon M. Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008. doi:10.1137/070680096.

8 Elliot Anshelevich, Anirban Dasgupta, Éva Tardos, and Tom Wexler. Near-optimal network design with selfish agents. *Theory of Computing*, 4(1):77–109, 2008. doi:10.4086/toc.2008.v004a004.

9 Venkatesh Bala and Sanjeev Goyal. A noncooperative model of network formation. *Econometrica*, 68(5):1181–1229, 2000. doi:10.1111/1468-0262.00155.

10 Davide Bilò, Tobias Friedrich, Pascal Lenzner, Stefanie Lowski, and Anna Melnichenko. Selfish creation of social networks. In *AAAI 2021*, pages 5185–5193, 2021.

11 Davide Bilò, Tobias Friedrich, Pascal Lenzner, and Anna Melnichenko. Geometric network creation games. In *SPAA 2019*, pages 323–332, 2019. doi:10.1145/3323165.3323199.

12 Davide Bilò and Pascal Lenzner. On the tree conjecture for the network creation game. *Theory of Computing Systems*, 64(3):422–443, 2020. doi:10.1007/s00224-019-09945-9.

13 Larry Blume, David A. Easley, Jon M. Kleinberg, Robert D. Kleinberg, and Éva Tardos. Network formation in the presence of contagious risk. In *EC 2011*, pages 1–10, 2011. doi:10.1145/1993574.1993576.

14 Prosenjit Bose and Michiel Smid. On plane geometric spanners: A survey and open problems. *Computational Geometry*, 46(7):818–830, 2013. doi:10.1016/j.comgeo.2013.04.002.

15 Ricardo Caballero and Alp Simsek. Fire sales in a model of complexity. *The Journal of Finance*, 68, 2009. doi:10.2139/ssrn.1496592.

16 P.M. Camerini, G. Galbiati, and F. Maffioli. On the complexity of finding multi-constrained spanning trees. *Discrete Applied Mathematics*, 5(1):39–50, 1983. doi:10.1016/0166-218X(83)90014-8.

17 Yu Chen, Shahin Jabbari, Michael J. Kearns, Sanjeev Khanna, and Jamie Morgenstern. Network formation under random attack and probabilistic spread. In *IJCAI 2019*, pages 180–186, 2019. doi:10.24963/ijcai.2019/26.

18 Jacomo Corbo and David C. Parkes. The price of selfish behavior in bilateral network formation. In *PODC 2005*, pages 99–107, 2005. doi:10.1145/1073814.1073833.

19 Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. The price of anarchy in cooperative network creation games. *SIGecom Exchanges*, 8(2):2:1–2:20, 2009. doi:10.1145/1980522.1980524.

20 Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. The price of anarchy in network creation games. *ACM Transactions on Algorithms*, 8(2):13, 2012. doi:10.1145/2151171.2151176.

21 Jack Dippel and Adrian Vetta. An improved bound for the tree conjecture in network creation games. *CoRR*, abs/2106.05175, 2021. doi:10.48550/arXiv.2106.05175.

22 Andrey A. Dobrynin, Roger C. Entringer, and Ivan Gutman. Wiener index of trees: Theory and applications. *Acta Applicandae Mathematica*, 66:211–249, 2001. doi:10.1023/A:1010767517079.

23 Alex Fabrikant, Ankur Luthra, Elitza N. Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *PODC 2003*, pages 347–351, 2003. doi:10.1145/872035.872088.

24 Wilhelm Friedemann, Tobias Friedrich, Hans Gawendowicz, Pascal Lenzner, Anna Melnichenko, Jannik Peters, Daniel Stephan, and Michael Vaichenker. Efficiency and stability in euclidean network design. In *SPAA 2021*, pages 232–242, 2021. doi:10.1145/3409964.3461807.

25 Tobias Friedrich, Hans Gawendowicz, Pascal Lenzner, and Anna Melnichenko. Social distancing network creation, 2022. arXiv:2204.10423.

26 Tobias Friedrich, Sven Ihde, Christoph Keßler, Pascal Lenzner, Stefan Neubert, and David Schumann. Efficient best response computation for strategic network formation under attack. In *SAGT 2017*, pages 199–211, 2017. doi:10.1007/978-3-319-66700-3_16.

27 Giulia Galbiati, Angelo Morzenti, and Francesco Maffioli. On the approximability of some maximum spanning tree problems. *Theoretical Computer Science*, 181(1):107–118, 1997.

28 Sanjeev Goyal, Shahin Jabbari, Michael J. Kearns, Sanjeev Khanna, and Jamie Morgenstern. Strategic network formation with attack and immunization. In *WINE 2016*, pages 429–443, 2016. doi:10.1007/978-3-662-54110-4_30.

29 Ronald L. Graham and Pavol Hell. On the history of the minimum spanning tree problem. *IEEE Annals of the History of Computing*, 7(1):43–57, 1985. doi:10.1109/MAHC.1985.10011.

30 Andrew G Haldane and Robert M May. Systemic risk in banking ecosystems. *Nature*, 469(7330):351–355, 2011. doi:10.1038/nature09659.

31 T. C. Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, 1974. doi:10.1137/0203015.

32 Matthew O Jackson. *Social and economic networks*. Princeton university Press, 2008. doi:10.2307/j.ctvcm4gh1.

33 Matthew O. Jackson and Asher Wolinsky. A strategic model of social and economic networks. *Journal of Economic Theory*, 71(1):44–74, 1996. doi:10.1006/jeth.1996.0108.

34 Tomasz Janus and Bart de Keijzer. On strong equilibria and improvement dynamics in network creation games. In *WINE 2017*, pages 161–176, 2017. doi:10.1007/978-3-319-71924-5_12.

35 David S. Johnson, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978. doi:10.1002/net.3230080402.

36 David R. Karger, Rajeev Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997. doi:10.1007/BF02523689.

37 Bernd Kawald and Pascal Lenzner. On dynamics in selfish network creation. In *SPAA 2013*, pages 83–92, 2013. doi:10.1145/2486159.2486185.

38 Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In *STACS 1999*, pages 404–413, 1999. doi:10.1007/3-540-49116-3_38.

39 Pascal Lenzner. On dynamics in basic network creation games. In *SAGT 2011*, pages 254–265, 2011. doi:10.1007/978-3-642-24829-0_23.

40 Pascal Lenzner. Greedy selfish network creation. In *WINE 2012*, pages 142–155, 2012. doi:10.1007/978-3-642-35311-6_11.

41 Thomas L Magnanti and Richard T Wong. Network design and transportation planning: Models and algorithms. *Transportation science*, 18(1):1–55, 1984. doi:10.1287/trsc.18.1.1.

42 Akaki Mamageishvili, Matús Mihalák, and Dominik Müller. Tree nash equilibria in the network creation game. *Internet Mathematics*, 11(4-5):472–486, 2015. doi:10.1080/15427951.2015.1016248.

43 Matúš Mihalák and Jan Christoph Schlegel. The price of anarchy in network creation games is (mostly) constant. *TCS*, 53(1):53–72, 2013. doi:10.1007/s00224-013-9459-y.

44 Bojan Mohar and Tomaž Pisanski. How to compute the wiener index of a graph. *Journal of Mathematical Chemistry*, 2(3):267–277, 1988. doi:10.1007/BF01167206.

45 Dov Monderer and Lloyd S. Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, 1996. doi:10.1006/game.1996.0044.

46 Giri Narasimhan and Michiel Smid. *Geometric spanner networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

47 Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002. doi:10.1145/506147.506153.

48 Hans Wiener. Structural determination of paraffin boiling points. *Journal of the American Chemical Society*, 69 1:17–20, 1947. doi:10.1021/ja01193a005.

49 Kexiang Xu, Muhuo Liu, Kinkar Das, Ivan Gutman, and Boris Furtula. A survey on graphs extremal with respect to distance–based topological indices. *Match (Mulheim an der Ruhr, Germany)*, 71:461–508, 2014.

50 Ľubomír Šoltés. Transmission in graphs: A bound and vertex removing. *Mathematica Slovaca*, 41(1):11–16, 1991. URL: http://hdl.handle.net/10338.dmlcz/132387.

Approximating Observables Is as Hard as Counting

Andreas Galanis
Department of Computer Science, University of Oxford, UK

Daniel Štefankovič
Department of Computer Science, Univerity of Rochester, NY, USA

Eric Vigoda
Department of Computer Science, University of California Santa Barbara, CA, USA

Abstract

We study the computational complexity of estimating local observables for Gibbs distributions. A simple combinatorial example is the average size of an independent set in a graph. A recent work of Galanis et al (2021) established NP-hardness of approximating the average size of an independent set utilizing hardness of the corresponding optimization problem and the related phase transition behavior. We instead consider settings where the underlying optimization problem is easily solvable. Our main contribution is to classify the complexity of approximating a wide class of observables via a generic reduction from approximate counting to the problem of estimating local observables. The key idea is to use the observables to interpolate the counting problem.

Using this new approach, we are able to study observables on bipartite graphs where the underlying optimization problem is easy but the counting problem is believed to be hard. The most-well studied class of graphs that was excluded from previous hardness results were bipartite graphs. We establish hardness for estimating the average size of the independent set in bipartite graphs of maximum degree 6; more generally, we show tight hardness results for general vertex-edge observables for antiferromagnetic 2-spin systems on bipartite graphs. Our techniques go beyond 2-spin systems, and for the ferromagnetic Potts model we establish hardness of approximating the number of monochromatic edges in the same region as known hardness of approximate counting results.

2012 ACM Subject Classification Theory of computation → Generating random combinatorial structures; Mathematics of computing → Probabilistic algorithms

Keywords and phrases Approximate Counting, Averages, Phase Transitions, Random Structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.63

Category Track A: Algorithms, Complexity and Games

Funding *Daniel Štefankovič*: supported by NSF grant CCF-2007287.
Eric Vigoda: supported by NSF CCF-2007022.

1 Introduction

Can we efficiently estimate the average size of an independent set in an input graph $G = (V, E)$? Moreover, can we do so without utilizing a sampling algorithm for generating a random independent set?

In this paper, for a broad class of problems captured by Gibbs distributions, we address the relationship between the computational complexity of approximating local observables (such as estimating the average size of an independent set) and the computational complexity of approximating the partition function (such as estimating the total number of independent sets). It is a standard technique in the area to reduce estimating observables to approximate counting, by first implementing an approximate sampler and then using an unbiased estimator of the desired observable. The focus of this paper is the converse, where there is no previously


known technique to answer the following question: does an algorithm for local observables yield an algorithm for the partition function? We prove, in a broad setting, that these two genres of problems are computationally equivalent.

Previous work of [9] only achieved this indirectly; they showed hardness of approximating local observables (in fact, only for a certain observable, called magnetization, see below for definitions) utilizing the hardness of MaxCut. Here, we show a direct reduction from the observable problem to the partition-function problem, relating therefore more crisply the two problems. This allows us to obtain hardness results in several new regimes (in particular,not covered by [9]) where the counting problem is hard but there is no underlying hard optimization problem.

An interesting setting to highlight the usefulness of our reduction is *bipartite* independent sets. In this example there is no corresponding hard optimization problem (as the maximum independent set problem is poly-time solvable in bipartite graphs), and hence to prove hardness we need to utilize hardness of approximate counting results. Another pertinent example for our results are attractive graphical models, these are equivalent to ferromagnetic spin systems in statistical physics. The simplest case is the ferromagnetic Ising model and its generalization known as the Potts model. In the Ising/Potts model on a graph (see Section 1.1 for more precise definitions), the configurations of the model are the collection of labellings σ of the vertices with q spins (colours), each weighted as $\beta^{m(\sigma)}$ where $m(\sigma)$ is the number of monochromatic edges and β is a parameter > 1 (so that labellings with many monochromatic edges are favored). Because of the attractiveness assumption that $\beta > 1$, once again, there is no corresponding hard optimization problem for this problem (contrast this with the case $\beta < 1$ where the largest weight labellings have the smallest number of monochromatic edges). Nevertheless, using our new reduction, we show that hardness of the associated approximate counting problem implies hardness of estimating the (weighted) average of the monochromatic edges in the Potts model.

Our two illustrative examples, the average size of an independent set and the number of monochromatic edges in the Ising/Potts model, are instances of a local observable in statistical physics; specifically they correspond to the magnetization and susceptibility, respectively. The behavior of observables is fundamental to the study of phase transitions, e.g., see [1, 4].

We begin giving more precise definitions for our initial example of *bipartite* independent sets, before considering the ferromagnetic Potts model, and finally generalizing to arbitrary local observables in general 2-spin systems. For a graph $G = (V, E)$ let $\mathcal{I}_G$ denote the set of independent sets (of all sizes) of G, and let $\mu := \mu_G$ denote the uniform distribution over $\mathcal{I}_G$. Denote the average independent set size by $\mathcal{M}(G) = \mathbf{E}_{\sigma\sim\mu}\big[|\sigma|\big]$; this corresponds to the magnetization in statistical physics (and hence the choice of notation $\mathcal{M}$). We say that an algorithm is an FPRAS for the average independent set size if given a graph $G = (V, E)$ and parameters $\epsilon, \delta > 0$, the algorithm outputs an estimate EST which is within a multiplicative factor $(1 \pm \epsilon)$ of the desired quantity $\mathcal{M}(G)$, with probability $\geq 1 - \delta$, and runs in time $\mathrm{poly}(|V|, 1/\epsilon, \log(1/\delta))$. One can also consider an FPRAS for estimating $|\Omega|$, the number of independent sets of the input graph G; we refer to this as an efficient approximate counting algorithm.

It is a classical result [14] that an efficient approximate counting algorithm is polynomial-time interreducible with an efficient algorithm for approximate sampling from μ. In turn, efficiently estimating the average independent set size of a graph G is easily reduced to approximate sampling from the uniform distribution μ_G. The challenging aspect, and the focus of this paper, is the *reverse* implication. Can we estimate the typical size of an independent set without utilizing an approximate sampling algorithm? We will show it is *not* possible, i.e., hardness of approximate counting (and hence approximate sampling) implies hardness of estimating the average independent set size.

For graphs of maximum degree 5, Weitz [20] presented an FPTAS for approximating the number of independent sets, which yields an efficient approximate sampling scheme; note an FPTAS is the deterministic analog of an FPRAS, i.e., it achieves $\delta = 0$. Very recently, Chen et al. [5] proved that the simple MCMC algorithm known as the Gibbs sampler (or Glauber dynamics) has $O(n \log n)$ mixing time for this same class of graphs of maximum degree 5. Hence, one immediately obtains an FPRAS for the average independent set size $\mathcal{M}(G)$.

On the other side, for graphs of maximum degree 6, Sly [18] proved that approximating the number of independent sets is NP-hard, by a reduction from max-cut. Schulman et al. [16] showed #P-hardness for exact computation of the average independent set size. Moreover, recent work of Galanis et al. [9] shows that approximating the average independent-set size is also NP-hard for graphs of maximum degree 6. The proof of [9] does not directly relate approximate counting and estimating the average independent set size; instead [9] also shows a (more sophisticated) reduction from max-cut and utilizes the associated gadgets used in Sly's inapproximability result [18].

This begets the question: are these problems still intractable when restricted to *bipartite graphs*? For bipartite graphs there is no longer a hard optimization problem, such as max-cut, that one can utilize as a starting point for a hardness reduction. However, approximately counting independent sets is considered to be intractable on bipartite graphs of maximum degree 6; in particular, it is is #BIS-hard [3] where #BIS refers to the problem of approximately counting independent sets on general bipartite graphs (with potentially unbounded degree). There are now a multitude of approximate counting problems which share the same #BIS-hardness status or are even #BIS-equivalent, e.g., see [6, 3, 11, 7].

We present a general approach for reducing approximate counting to approximating averages. This yields hardness for approximating the average independent-set size in *bipartite graphs* of maximum degree 6.

▶ **Theorem 1.** *Let $\Delta \geq 6$ be an integer. There is no* FPRAS *for the average independent-set size on* bipartite *graphs of maximum degree Δ unless* #BIS *admits an* FPRAS*.*

Note that the #BIS-hardness result of Theorem 1 gives a weaker guarantee than those shown in [9] where they obtain in some cases constant-factor inapproximability results (using the constant-factor NP-hardness of the optimization problem). This difference is inherent with the #BIS-hardness assumption, i.e., that there is no FPRAS for #BIS. Moreover, an algorithm which approximates #BIS within any poly(n)-factor implies an FPRAS, and obtaining constant-factor inapproximability results for magnetization on bipartite graphs would require (among other things) hardness of #BIS within an exponential-factor.

Our results extend to the hard-core model on weighted independent sets, and to general 2-spin antiferromagnetic models. These more general results are detailed in Section 1.2.

1.1 Ferromagnetic Potts Model

Ferromagnetic spin systems, which are equivalent to attractive undirected graphical models, are an interesting class of models to illustrate our new proof technique on. In ferromagnetic models there is no hard optimization problem as the maximum likelihood configurations are trivial assignments (setting all vertices to the same spin/label). Consequently, to obtain hardness results for computing averages in ferromagnetic models we need to work directly from hardness of approximate counting results, which we can do using our new approach.

The most well-studied examples of ferromagnetic models are the Ising and Potts models. Given a graph G and an integer $q \geq 2$, configurations of the Ising/Potts model are the collection Ω of assignments $\sigma : V(G) \to [q]$ where $[q] = \{1, \ldots, q\}$ are the labels of the q spins. The case $q = 2$ corresponds to the Ising model and the case $q \geq 3$ is the Potts model.

The models are parameterised by an edge activity[1] $\beta > 0$. The weight of an assignment σ is defined as $w_{G;q,\beta}(\sigma) = \beta^{m_G(\sigma)}$ where $m_G(\sigma) = |\{(u,v) \in E : \sigma(u) = \sigma(v)\}|$ is the number of edges which are monochromatic in σ. Finally, the Gibbs distribution is defined as $\mu_{G;q,\beta}(\sigma) = w_{G;q,\beta}(\sigma)/Z_{G;q,\beta}$ where the normalising factor $Z_{G;q,\beta} := \sum_{\tau:V(G)\to[q]} w(\tau)$ is the partition function. In this paper, we restrict attention to the case $\beta > 1$ which is the ferromagnetic (attractive) case, and hence the most likely configurations are the q monochromatic configurations (all vertices are assigned the same spin).

For the Ising and Potts models, the analog of the average independent set size is the average number of vertices assigned spin 1. This quantity $\mathcal{M}_{q,\beta}(G)$, known as the magnetization, is trivial in these cases since, due to the Ising/Potts models symmetry among spins, it holds that $\mathcal{M}_{q,\beta}(G) = n/q$. The simplest and most natural observable to consider is the average number of monochromatic edges under the Potts distribution, i.e., the quantity

$$\mathcal{S}_{q,\beta}(G) := \mathbf{E}_{\sigma\sim\mu_{G;q,\beta}}[m_G(\sigma)]$$

which is known as the *susceptibility*. Sinclair and Srivastava [17] showed that exact computation of the susceptibility in the ferromagnetic Ising model is #P-hard.

For the Ising model a classical result of Jerrum and Sinclair [13] presents an efficient sampling scheme for all G, all β. This yields an efficient algorithm for approximating averages in the Ising model (this holds for any local observables as defined subsequently in Section 1.2). In contrast for the Potts model (for any $q \geq 3$) approximating the partition function becomes computationally intractable for large β as we detail below.

The Potts model with $q \geq 3$ spins undergoes a computational phase transition on bipartite graphs of maximum degree Δ at the following critical point $\beta_c(q,\Delta) = \frac{q-2}{(q-1)^{1-2/\Delta}-1}$. In [10] it was established that for all $q, \Delta \geq 3$ and $\beta > \beta_c(q,\Delta)$ approximating the partition function of the ferromagnetic Potts model is #BIS-hard on bipartite graphs of maximum degree Δ. Using our general counting-to-observables reduction we show that approximating the average number of monochromatic edges under the Potts distribution is as hard as approximating the partition function for the ferromagnetic Potts model.

▶ **Theorem 2.** *Let $q, \Delta \geq 3$ be integers and $\beta > \beta_c(q,\Delta)$. There is no* FPRAS *for the susceptibility in the q-state Potts model on bipartite graphs of maximum degree Δ, unless* #BIS *admits an* FPRAS.

1.2 General 2-spin systems

Theorem 1 for independent sets is a special case of a general result for arbitrary 2-spin antiferromagnetic systems. Such spin systems are parameterized by three parameters, β, γ and λ; the first two are edge activities and control the strength of the spin interactions between neighboring vertices, and the third is a vertex activity (a.k.a. external field) that favors one spin over the other.

More precisely, for a graph $G = (V, E)$, $\beta, \gamma \geq 0$ which are not both equal to zero and $\lambda > 0$, let $\mu_{G;\beta,\gamma,\lambda}$ denote the Gibbs distribution on G with edge activities β, γ and external field λ, i.e., for $\sigma : V \to \{0,1\}$ we have

$$\mu_{G;\beta,\gamma,\lambda}(\sigma) = \frac{\lambda^{|\sigma|}\beta^{m_0(\sigma)}\gamma^{m_1(\sigma)}}{Z_{G;\beta,\gamma,\lambda}},$$

[1] We remark that β is usually used to denote the so-called inverse temperature of the Potts model; here to have consistent notation with general 2-spin systems presented in Section 1.2 we take β to be the exponent of the inverse temperature.

where $|\sigma|$ is the number of vertices with spin 1, and $m_0(\sigma), m_1(\sigma)$ denote the number of edges in G whose endpoints are assigned under σ the pair of spins $(0,0)$ and $(1,1)$, respectively.

The parameter pair (β,γ) is called *antiferromagnetic* if $\beta\gamma \in [0,1)$ and at least one of β,γ is non-zero, and it is called ferromagnetic, otherwise. Note that the hard-core model on independent sets weighted by $\lambda > 0$ is the case $\beta = 1, \gamma = 0$ (under the convention that $0^0 \equiv 1$). Our earlier example of unweighted independent sets corresponds to the hard-core model with $\lambda = 1$. The antiferromagnetic Ising model is the special case $0 < \beta = \gamma < 1$.

Our results apply to general "vertex-edge observables" defined as follows.

▶ **Definition 3.** *Let (β,γ) be antiferromagnetic and $\lambda > 0$. For real numbers a, b, c, the* (a,b,c) vertex-edge observable *of a graph G in the 2-spin system corresponding to (β,γ,λ) is given by*

$$\mathcal{O}_{\beta,\gamma,\lambda}(G) = \mathbf{E}_{\sigma\sim\mu_{G;\beta,\gamma,\lambda}}\big[o_G(\sigma)\big], \text{ where } o_G(\sigma) = a|\sigma| + bm_0(\sigma) + cm_1(\sigma).$$

The observable is trivial on general graphs *if any of the following hold: (i) $a = b = c = 0$, (ii) $\beta = 0$ and $a = c = 0$, (iii) $\gamma = 0$ and $a = b = 0$, (iv) $\beta = \gamma$, $\lambda = 1$ and $b + c = 0$. We say that the observable is* trivial on bipartite graphs *if either any of the above hold, or $\beta = \gamma$ and $\lambda = 1$. Otherwise, we say that the observable is* non-trivial.

Notice that by setting $(a,b,c) = (1,0,0)$ we obtain the magnetization $\mathcal{M}_{\beta,\gamma,\lambda}(G)$, which in the special case of the hard-core model with $\lambda = 1$ is the average size of an independent set. Furthermore, by setting $(a,b,c) = (0,1,1)$ we obtain the susceptibility, denoted by $\mathcal{S}_{\beta,\gamma,\lambda}(G)$, which is the average number of monochromatic edges.

The terminology "trivial" is applied liberally here and meant to convey that there is an efficient algorithm for the relevant parameters. In particular, while cases (i)-(iii) are degenerate, case (iv) corresponds to the Ising model without an external field. A classical (and highly non-trivial) result of Jerrum and Sinclair [13] presented an FPRAS for the ferromagnetic Ising model on any graph, any $\beta > 1$. Moreover, for bipartite graphs, the subcase $\beta < 1$ (antiferromagnetic) can be reduced to an equivalent $\beta > 1$ (ferromagnetic) system.

We next define the range of parameters (β,γ,λ) where our inapproximability results for vertex-edge observables apply; these are precisely the parameters where the hard-core and the antiferromagnetic Ising models exhibit non-uniqueness on the infinite Δ-regular tree (for general 2-spin systems this threshold corresponds to what is known as up-to-Δ non-uniqueness, which captures the computational phase transition).

▶ **Definition 4.** *Let $\Delta \geq 3$ be an integer. We let $\mathcal{N}_\Delta$ be the set of (β,γ,λ) such that (β,γ) is antiferromagnetic, and the (unique) fixpoint $x^* > 0$ of the function $f(x) = \frac{1}{\lambda}\big(\frac{\beta x+1}{x+\gamma}\big)^{\Delta-1}$ satisfies $|f'(x^*)| > 1$. The region $\mathcal{N}_\Delta$ is known as the* non-uniqueness region.

Note there is an efficient sampling/counting algorithm for graphs of maximum degree Δ, roughly[2], for (β,γ,λ) outside the parameter region $\mathcal{N}_\Delta$ [15, 5]. Inside $\mathcal{N}_\Delta$, it is NP-hard to approximate the partition function on graphs of maximum degree Δ [19] and it is #BIS-hard to approximate the partition function on *bipartite* graphs of maximum degree Δ [3]. We prove that it is also hard to compute any non-trivial vertex-edge observable in exactly this same region where the corresponding counting problem is hard.

[2] More precisely, the (strict) uniqueness region is defined as those $\beta,\gamma,\lambda)$ where the fixpoint x^* in Definition 4 satisfies the (strict) inequality $|f'(x^*)| < 1$. For certain monotonicity reasons, the algorithm for max-degree Δ graphs demands that (β,γ,λ) lie in the intersection of these uniqueness regions for all degrees $d \leq \Delta$, see [15] for details.

▶ **Theorem 5.** *Let $\Delta \geq 3$ be an integer and $(\beta, \gamma, \lambda) \in \mathcal{N}_\Delta$. Then, for any vertex-edge observable that is non-trivial on bipartite graphs, there is no* FPRAS *on bipartite graphs of maximum degree Δ unless* #BIS *admits an* FPRAS.

We stress that the above result holds for bipartite graphs. The previous work of Galanis et al. [9] showed hardness for general antiferromagnetic 2-spin systems in the same non-uniqueness region but on general graphs, only for the magnetization, and only achieved the stronger constant-factor hardness for a dense set of λ.

We begin by establishing Theorem 2 for hardness of approximating the susceptibility for the ferromagnetic Potts model, see Section 2. We then present the refinements to establish Theorems 5 for general 2-spin antiferromagnetic systems in Section 3; Theorem 1 follows as a corollary of Theorem 5.

2 Hardness of Susceptibility for the Ferromagnetic Potts model

Let $q, \Delta \geq 3$ be integers and $\beta > \beta_c(q, \Delta)$. To prove Theorem 5, we will assume the existence of an FPRAS for the susceptibility of Potts with parameters q, β on maximum degree Δ graphs and show how to obtain an FPRAS for the partition function of the Potts model with parameters q, β^* on bipartite graphs of maximum degree 3 for some $\beta^* > \beta_c(q, 3)$; the latter problem is #BIS-hard by [10].

To aid the presentation, it will be convenient to consider the following computational problems and use the notion of AP-reduction between counting problems [6]; roughly, for two problems A, B, the notation $A \leq_{\text{AP}} B$ means that the existence of an FPRAS for B implies the existence of an FPRAS for A. In the first computational problem that will be relevant, the parameters are q, β, Δ as detailed below.

Name #Susc(q, β, Δ).
Instance A bipartite graph G with max degree Δ.
Output The susceptibility on G with parameters q, β, i.e., the value $\mathcal{S}_{q,\beta}(G)$.

In the second, the parameter is going to be just q; note that the problem allows the edge activity to be part of the input.

Name #SuscCubic(q).
Instance A cubic bipartite graph H, and a rational edge activity $\hat{\beta} \geq 1$.
Output The susceptibility on H with parameters $q, \hat{\beta}$, i.e., the value $\mathcal{S}_{q,\hat{\beta}}(H)$.

The key ingredient underpinning our proof approach is captured by the following lemma, whose proof is given in Section 2.3. Roughly, the lemma asserts that, despite the fact that the parameter β is fixed, with appropriate gadget constructions we can "shift" it in a fine-tuned way to any desired $\hat{\beta}$. In turn, this allows us to do an appropriate integration of the observable (viewed as a function of the parameter $\hat{\beta}$) to recover the partition function of a #BIS-hard problem; we will refer loosely to this integration technique as *interpolation*.

▶ **Lemma 6.** *Let $q, \Delta \geq 3$ be integers and $\beta > \beta_c(q, \Delta)$ be an arbitrary real. Then,*

$$\#\text{SuscCubic}(q) \leq_{\text{AP}} \#\text{Susc}(q, \beta, \Delta).$$

Before proceeding with outlining the proof of the key Lemma 6, we first present the interpolation-scheme idea that allows us to conclude Theorem 2 from Lemma 6.

Proof of Theorem 2 (assuming Lemma 6). Let $\beta^* > \beta_c(q,3)$ be an arbitrary rational number and consider the problem #PottsCubic(q,β^*), i.e., the problem of approximating the partition function $Z_{G;q,\beta^*}$ on cubic bipartite graphs G. From [10, Theorem 3], we have that #PottsCubic(q,β^*) is #BIS-hard. From Lemma 6, we have that for $\beta > \beta_p(q,\Delta)$ it holds that #SuscCubic$(q) \leq_{\mathrm{AP}}$ #Susc(q,β,Δ), so to prove the theorem it suffices to show that #PottsCubic$(q,\beta^*) \leq_{\mathrm{AP}}$ #SuscCubic(q).

Let G be an instance of #PottsCubic(q,β^*) with n vertices and m edges, and $\epsilon > 0$ be the desired relative error that we want to approximate $Z_{G;q,\beta^*}$. Since $\frac{\partial \log Z_{G;q,\hat{\beta}}}{\partial \hat{\beta}} = \frac{1}{\hat{\beta}}\mathcal{S}_{q,\hat{\beta}}(G)$, we have

$$\log Z_{G;q,\beta^*} = \int_1^{\beta^*} \tfrac{1}{\hat{\beta}}\mathcal{S}_{q,\hat{\beta}}(G)\mathrm{d}\hat{\beta}. \tag{1}$$

Let $M = \lceil (10q\hat{\beta}m/\epsilon)^4 \rceil$ and for $i = 0,1,\ldots,M$, consider the sequence of edge parameters $\hat{\beta}_i = 1 + i\frac{\beta^*-1}{M}$. It is a standard fact that the function $\log Z_{G;\hat{\beta}}$ is convex with respect to $\hat{\beta}$ (the second derivative is equal to the variance of the number of monochromatic edges) and therefore the function $\frac{1}{\hat{\beta}}\mathcal{S}_{q,\hat{\beta}}(G)$ is increasing. Therefore, from the standard technique of approximating integrals with rectangles, we obtain from (1) that

$$\frac{1}{M}\sum_{i=0}^{M-1}\frac{\mathcal{S}_{G;q,\hat{\beta}_i}}{\hat{\beta}_i} \leq \log Z_{G;\beta^*} \leq \frac{1}{M}\sum_{i=1}^{M}\frac{\mathcal{S}_{q,\hat{\beta}_i}(G)}{\hat{\beta}_i}.$$

Using the bound $m/q \leq \mathcal{S}_{q,\hat{\beta}}(G) \leq m$ that holds for all $\hat{\beta} \geq 1$, we obtain that

$$\log Z_{G;q,\beta^*} = \big(1 \pm \frac{\epsilon}{10}\big)\sum_{i=1}^{M}\frac{\mathcal{S}_{q,\hat{\beta}_i}(G)}{\hat{\beta}_i}.$$

Using the presumed oracle for #SuscCubic(q) we can compute $\hat{S}_i$ such that $\hat{S}_i = (1 \pm \frac{\epsilon}{10Mm})\mathcal{S}_{q,\hat{\beta}_i}(G)$ for $i \in [M]$, and therefore the quantity $\hat{Z} = \exp\big(\sum_{i\in[M]}\frac{\mathcal{S}_{q,\hat{\beta}_i}(G)}{\hat{\beta}_i}\big)$ is a $(1\pm\epsilon)$-factor approximation to $Z_{G;q,\beta^*}$. This completes the AP-reduction, and therefore the proof as well. ◀

In the rest of Section 2, we focus on proving Lemma 6.

2.1 Proof overview of Lemma 6

In this section, we give the proof overview of Lemma 6 which as we saw in the previous section is the key ingredient to carry out the interpolation-scheme idea. We highlight here some of the key ideas (with a non-technical overview), which are also used to prove the analogous Lemma for obtaining our inapproximability results for 2-spin systems.

To prove Lemma 6, we will use three different types of gadgets.

The first type of gadgets, that have also been used in previous inapproximability results, are the so-called "phase gadgets", which are almost Δ-regular bipartite graphs with a relatively small number of degree $\Delta - 1$ vertices (the so-called "ports"). This type of gadget exploits the phase transitions of the model and has q-ary behaviour, in the sense that a typical sample from their Gibbs distribution is in one of the q ordered phases, favoring one spin over the others. Aside from this q-ary behaviour, another feature of these gadgets is that they are convenient to maintain the degree of the vertices in our constructions small, using the ports to make connections between gadgets.

The second type of gadgets are paths; these allow us to interpolate the edge activity β. The key point is that long paths induce some small edge-interaction β between their endpoints (bigger than but close to 1) and by using a big number of them (in parallel-style connections) we can achieve a target edge activity $\hat{\beta}$ with arbitrary good precision; here, the ports of the phase gadgets allow us to do these parallel connections without exceeding the degree bound Δ. This is a crucial ingredient in implementing the new reduction idea.

The final type of gadgets consists of the so-called edge-interaction gadgets. Each such gadget has two vertices, say ρ, ρ', which we also refer to as ports. We are interested in two quantities of these gadgets (cf. Definition 9):

- the effective edge activity, i.e., the relative ratio of the aggregate weight of configurations where $\sigma(u) = \sigma(v)$ versus $\sigma(u) \neq \sigma(v)$. Note that this ratio is always bigger than 1, due to the ferromagnetic interaction.
- the susceptibility gap, i.e., the difference between the expected susceptibility conditioned on $\sigma(u) = \sigma(v)$ and the susceptibility conditioned on $\sigma(u) \neq \sigma(v)$.

We prove the existence of pairs of susceptibility gadgets which have roughly equal induced edge parameters but different susceptibility gaps. The equality between the induced edge parameters allows us to use them as probes (without changing the underlying distribution) for "susceptibility" between two vertices s, t, i.e., the probability that s, t have the same colour, in a graph G. That is, we can invoke a presumed oracle for susceptibility when we use the first gadget (by identifying s, t with the terminals) and get a "reading" for susceptibility, and do the same for the second and get a second "reading"; the difference between the two readings gives us information about the probability that s, t have the same colour in the original graph G.

The reason that these susceptibility gadgets are useful is that analysing the susceptibility of the other two types of gadgets is deeply unpleasant and, in fact, it is not even known how to obtain susceptibility estimates for the phase gadgets (since their analysis in earlier works builds upon second moment methods that give rather crude bounds in our setting). Hence, by the subtraction trick above, we have the required modularity to avoid such refined considerations.

That said, establishing the existence of pairs of susceptibility gadgets with the required properties has various challenges and the proof is based on an elaborate construction which finishes by a contradiction argument via Cauchy's functional equation. Fortunately, this ground work has been largely done in [9], though in our setting we need to consider edge gadgets instead of vertex gadgets, which complicates the underlying functions involved in the proofs. We believe that these constructions can be used to strengthen the results of [9] and obtain inapproximability for multi-spin systems such as colourings or the antiferromagnetic Potts model.

These ideas suitably adapted apply to obtain our inapproximability results for antiferromagnetic 2-spin systems. The difference for 2-spin systems is that the interpolation is quite trickier, since in the setting there it is harder to make vertex or edge activities that are close to 1 and do the interpolation (in contrast to the paths used above which is the fairly natural choice). Instead, to do the interpolation, we use a pair of trees whose induced vertex activities (at the root) are sufficiently close and which are attached (in appropriate numbers) to the ports of the phase gadgets to imitate the effect of an external field close to 1. We are then able to interpolate in terms of λ by a suitable implementation of the subtraction trick idea; we again need to depart from [9] (where 2-spin models were also considered) since the construction there does not yield a suitable interpolation parameter. The final new ingredient

is to account for the general vertex-edge observables, since a key fact used in [9] is that the magnetization is an appropriate derivative of the log-partition function, which is no longer the case for general vertex-edge observables.

We now state more formally the above ingredients and show how to combine these and conclude the proof of Lemma 6.

2.2 The gadgets

2.2.1 Bipartite phase gadgets for the Potts model

For integers t, n, Δ, we let $\mathcal{G}^t_{n,\Delta}$ be the distribution on bipartite graphs where there are n vertices with degree Δ on each side, and t vertices of degree $\Delta-1$ on each side. For a graph $G \in \mathcal{G}^t_{n,\Delta}$, we denote the set of vertices with degree Δ by U and by W those with degree $\Delta-1$, so that $|U| = 2n$ and $|W| = 2t$. We will refer to set W as the *ports* of G. For $\sigma : U \to [q]$, we define the *phase* $\mathcal{Y}(\sigma)$ of the configuration σ as the most frequent color (breaking ties arbitrarily), i.e., which has the most occupied vertices under σ, i.e.,

$$\mathcal{Y}(\sigma) = \arg\max_{i\in[q]} |\sigma^{-1}(i)|.$$

Let $p > 1/q$ be given from $p = \frac{x}{x+q-1}$ where $x > 1$ is the largest solution of $x = \left(\frac{\beta x+q-1}{x+\beta+q-2}\right)^{\Delta-1}$, cf. [10, Footnote 5]. For a colour $i \in [q]$, we define the product measure $Q^i_W(\cdot)$ on configurations $\tau : W \to [q]$, given by

$$Q^i_W(\tau) = p^{|\tau^{-1}(i)|}\left(\tfrac{1-p}{q-1}\right)^{|W|-|\tau^{-1}(i)|}.$$

We will need the following two properties from the phase gadget G for some sufficiently small $\epsilon > 0$. Let $\mu := \mu_{G;q,\beta}$.

1. The q phases appear with roughly equal probability, i.e., $\left|\mu(\mathcal{Y}(\sigma) = i) - \frac{1}{q}\right| \le \epsilon$ for $i \in [q]$.
2. For $i \in [q]$ and any $\tau : W \to [q]$, $\left|\frac{\mu\left(\sigma_W=\tau \mid \mathcal{Y}(\sigma)=i\right)}{Q^i_W(\tau)} - 1\right| \le \epsilon$.

Let $\mathcal{G}^{t,\epsilon}_{n,\Delta}$ denote the set of graphs $G \in \mathcal{G}^t_{n,\Delta}$ satisfying Items 1 and 2. The following lemma is shown in [10].

▶ **Lemma 7** ([10, Lemma 28]). *Let $q, \Delta \ge 3$ be integers and $\beta > \beta_c(q,\Delta)$. Then, there is a randomized algorithm that, on input integer $t \ge 1$ and $\epsilon > 0$, outputs in time $\mathrm{poly}(t, \frac{1}{\epsilon})$ an integer n and a graph G that belongs to $\mathcal{G}^{t,\epsilon}_{n,\Delta}$, with probability $\ge 3/4$.*

2.2.2 Edge-interaction/susceptibility gadgets

▶ **Definition 8.** *An* edge-interaction gadget *is a connected series-parallel graph $\mathcal{E}$ with two distinct vertices ρ, ρ' that have degree one. We will refer to ρ, ρ' as the* ports *of $\mathcal{E}$.*

▶ **Definition 9.** *Let $\mathcal{E}$ be an edge-interaction gadget with ports ρ, ρ', and $\mu = \mu_{\mathcal{E};\beta}$. We denote by $B_{\mathcal{E}} = B_{\mathcal{E}}(\beta)$ the* effective interaction *of the gadget, i.e., $B_{\mathcal{E}} = \frac{\mu(\sigma_\rho=\sigma_{\rho'}=1)}{\mu(\sigma_\rho=1,\sigma_{\rho'}=2)}$ and by $S_{\mathcal{E}} = S_{\mathcal{E}}(\beta)$ the* susceptibility gap *of the gadget, i.e., $S_{\mathcal{E}} = \mathbf{E}_{\sigma\sim\mu}[m_{\mathcal{E}}(\sigma) \mid \sigma_\rho = \sigma_{\rho'}] - \mathbf{E}_{\sigma\sim\mu}[m_{\mathcal{E}}(\sigma) \mid \sigma_\rho \ne \sigma_{\rho'}]$.*

The following "interaction" gadget will allow us to change the edge interaction parameter to any desired value.

▶ **Lemma 10.** *Let $q \ge 2$ be an integer and $\beta > 1$ be a real. There is an algorithm, which, on input a rational $r \in (0, 1/2)$, outputs in time poly$(\mathrm{bits}(r))$ a path $\mathcal{P}$ of length $O(|\log r|)$, such that $0 < B_{\mathcal{P}} - 1 < r$.*

The proof of Lemma 10 is given in Section C.1.2 of the full version. The following lemma gives pairs of edge-interaction gadgets which have almost the same edge interaction but different susceptibility gaps; this difference in the susceptibility gaps while maintaining the edge interaction will be the key to read off the susceptibility by subtraction.

▶ **Lemma 11.** *Let $q \geq 3$ be an integer and $\beta > 1$ be a real. For any arbitrarily small constant $\delta > 0$, there exist constants $S, \Xi > 0$ and $B \in (1, 1+\delta)$ such that the following holds. There is an algorithm, which, on input a rational $r \in (0, 1/2)$, outputs in time poly$(\text{bits}(r))$ a pair of edge-interaction gadgets $\mathcal{E}_1, \mathcal{E}_2$, each of maximum degree 3 and size $O(|\log r|)$, such that*

$$|B_{\mathcal{E}_1} - B|, |B_{\mathcal{E}_2} - B| \leq r, \text{ but } |S_{\mathcal{E}_1} - S_{\mathcal{E}_2}| \geq S.$$

Moreover, the susceptibility gaps $|S_{\mathcal{E}_1}|, |S_{\mathcal{E}_2}|$ are upper-bounded in absolute value by the constant Ξ, i.e., $|S_{\mathcal{E}_1}|, |S_{\mathcal{E}_2}| \leq \Xi$.

The proof of Lemma 11 generalises the techniques from [9], and is given in Section C.3 of the full version.

2.3 The reduction – proof of Lemma 6

Let $q, \Delta \geq 3$ be integers and $\beta > \beta_c(q, \Delta)$. Let H be a cubic bipartite graph which is input to the problem #Susc(q) of Section 2. For integers $n, t \geq 1$ and rational $\epsilon > 0$, let $G \in \mathcal{G}^{t,\epsilon}_{n,\Delta}$ be a bipartite phase gadget satisfying Items 1 and 2 of Section 2.2.1. Let $\mathcal{E}$ be an edge-interaction gadget with effective interaction $B_{\mathcal{E}}$ and susceptibility gap $S = S_{\mathcal{E}}$. Let $\mathcal{P}$ be a path with effective edge interaction $B_{\mathcal{P}}$.

For an integer ℓ satisfying $\ell < t/3$, we define the graph $H^{\ell}_{G,\mathcal{E},\mathcal{P}}$ as follows. For each vertex v of H replace it with a distinct copy of G, denoted by G_v; we also use U_v, W_v to denote the sets corresponding to U, W in G_v. Moreover for each $\{u, v\}$ of H, add a matching of size $\ell + 1$ between W_u and W_v, and replace ℓ edges of the matching by (distinct) copies of the path $\mathcal{P}$ and the last edge of the matching by the gadget $\mathcal{E}$. Since H is bipartite, this constuction can clearly be done so that the final graph $H^{\ell}_{G,\mathcal{E},\mathcal{P}}$ obtained this way is bipartite. Let $H^{\ell}_{G,\mathcal{P}}$ be the graph with the copies of the susceptibility gadget removed.

The lemma below relates the susceptibility $\mathcal{S}_{q,\beta}(H^{\ell}_{G,\mathcal{E},\mathcal{P}})$ with the susceptibility of $\mathcal{S}_{q,\hat{\beta}}(H)$, for some appropriate $\hat{\beta}$ that is a function of the parameters q, Δ, β and $\ell, B_{\mathcal{E}}, B_{\mathcal{P}}$; we expain how the lemma corresponds to the overview of Section 2.1 right after its statement. The following piece of notation will be useful: for a graph J and a subgraph J' of J, given a configuration $\sigma : V(J) \to [q]$, it will be convenient to denote by $m_{J'}(\sigma) = \sum_{e=\{u,v\} \in E(J')} \mathbf{1}\{\sigma(u) = \sigma(v)\}$ the number of monochromatic edges of J' under σ.

▶ **Lemma 12.** *Let $q, \Delta \geq 3$ be integers and $\beta > \beta_c(q, \Delta)$. There are constants $1 > R_0 > R_1 > 0$ so that the following holds for any path $\mathcal{P}$ with edge interaction $B_{\mathcal{P}}$, any edge-interaction gadget $\mathcal{E}$ with effective interaction $B_{\mathcal{E}}$ and susceptibility gap $S_{\mathcal{E}}$, and any integers ℓ, t with $t \geq 3(\ell + 1)$.*

For a cubic bipartite graph H, for any $\epsilon \leq \frac{1}{(5q|V(H)|)^2}$, any integer n and phase gadget $G \in \mathcal{G}^{t,\epsilon}_{n,\Delta}$, for $\mu := \mu_{H^{\ell}_{G,\mathcal{E},\mathcal{P}}}$ and $\epsilon' = 10q|V(H)|\epsilon$, it holds that

$$\mathcal{S}_{q,\beta}(H^{\ell}_{G,\mathcal{E},\mathcal{P}}) = \mathcal{A}_{\mathcal{E}}|E(H)| + \mathbf{E}_{\sigma\sim\mu}[m_{H^{\ell}_{G,\mathcal{P}}}(\sigma)] + (1 \pm \epsilon')S_{\mathcal{E}}\big[(A_0 - A_1)\mathcal{S}_{q,\hat{\beta}}(H) + A_1|E(H)|\big],$$

where $A_{\mathcal{E}} = \mathbf{E}_{\sigma\sim\mu_{\mathcal{E}}}[m_{\mathcal{E}}(\sigma) \mid \sigma_\rho = \sigma_{\rho'}]$ and

$$\hat{\beta} := \big(\tfrac{1+(B_{\mathcal{P}}-1)R_0}{1+(B_{\mathcal{P}}-1)R_1}\big)^{\ell}\big(\tfrac{1+(B_{\mathcal{E}}-1)R_0}{1+(B_{\mathcal{E}}-1)R_1}\big), \qquad A_j := \tfrac{B_{\mathcal{E}}}{B_{\mathcal{E}} + \frac{1-R_j}{R_j}} \text{ for } j \in \{0, 1\}.$$

To give a bit of intuition behind the expression of $\mathcal{S}_{q,\beta}(H^{\ell}_{G,\mathcal{E},\mathcal{P}})$, recall that the vertices of H are replaced with copies of the bipartite phase gadgets G and that for each pair of neighboring vertices in H we connect the corresponding copies of G using the appropriate number of the gadgets $\mathcal{E},\mathcal{P}$. The point here is that the bipartite gadgets are so large that each one of them is with very high probability in one of the q phases (cf. Item 1 in Section 2.2.1) and therefore in the Gibbs distribution of $H^{\ell}_{G,\mathcal{E},\mathcal{P}}$ (with parameters q,β) they behave like meta-vertices which are in one of q states, analogously to a Potts model on H with q spins and a new edge activity $\hat{\beta}$, which is ultimately determined by the $\mathcal{E},\mathcal{P}$-connections and the (induced) probability distributions on the ports of the bipartite phase gadgets (conditioned on the phase, cf. Item 2 in Section 2.2.1). This explains (at an intuitive level) the presence of the quantity $\mathcal{S}_{q,\hat{\beta}}(H)$; the remaining terms are offsets to account for the addition of the various gadgets. Of those, the most complicated is the term $\mathbf{E}_{\sigma\sim\mu}[m_{H^{\ell}_{G,\mathcal{P}}}(\sigma)]$ which involves the contribution to the susceptibility from edges in the graph $\mathbf{E}_{\sigma\sim\mu}[m_{H^{\ell}_{G,\mathcal{P}}}(\sigma)]$ which is hard to get a neat expression since the average is taken over the complicated distribution μ. This is where the idea of having a pair of susceptibility gadgets $(\mathcal{E}_1,\mathcal{E}_2)$ with the same effective interaction but different susceptibility gaps will come into play (in the proof of Lemma 6 below): by subtracting the susceptibilities for the graphs $H^{\ell}_{G,\mathcal{E}_1,\mathcal{P}}$ and $H^{\ell}_{G,\mathcal{E}_2,\mathcal{P}}$ between these, the terms corresponding to $\mathbf{E}_{\sigma\sim\mu}[m_{H^{\ell}_{G,\mathcal{P}}}(\sigma)]$ will cancel (since $\mathcal{E}_1,\mathcal{E}_2$ have roughly the same effective interaction $B_{\mathcal{E}_1},B_{\mathcal{E}_2}$) allowing us to approximate the target quantity $\mathcal{S}_{q,\hat{\beta}}(H)$ (since $\mathcal{E}_1,\mathcal{E}_2$ have substantially different susceptibility gaps $S_{\mathcal{E}_1},S_{\mathcal{E}_2}$). That said, the proof of Lemma 12 is on the technical side and is deferred to Section B of the full version.

To finish the proof of Lemma 6, we need the following crude bound on the change of susceptibility when we slightly change the values of the edge activities on a subset of the edges. To state the lemma, for a graph G with edge-activity vector $\boldsymbol{\beta}=\{\beta_e\}_{e\in E(H)}$, define the weight of an assignment $\sigma:V(G)\to[q]$ as $w(\sigma)=\prod_{e=\{u,v\}\in E(G)}(\beta_e)^{\mathbf{1}\{\sigma(u)=\sigma(v)\}}$, and let $\mu_{G;q,\boldsymbol{\beta}}(\sigma)=w(\sigma)/Z_{G;q,\boldsymbol{\beta}}$ denote the corresponding Gibbs distribution, where $Z_{G;q,\boldsymbol{\beta}}$ is the normalizing constant.

▶ **Lemma 13.** *Let H be a graph and F be a subgraph of H on the same set of vertices. Suppose that $\boldsymbol{\beta}=\{\beta_e\}_{e\in E(H)},\boldsymbol{\beta}'=\{\beta'_e\}_{e\in E(H)}$ are edge activity vectors such that $\beta_e=\beta'_e=\beta$ for $e\in E(F)$, and $\beta_e=\beta_0$, $\beta'_e=\beta_1$ for $e\notin E(F)$. Then, for $\mu:=\mu_{H;q,\boldsymbol{\beta}}$ and $\mu':=\mu'_{H;q,\boldsymbol{\beta}}$, it holds that*

$$\Big|\mathbf{E}_{\sigma\sim\mu}[m_F(\sigma)]-\mathbf{E}_{\sigma\sim\mu'}[m_F(\sigma)]\Big|\leq|E(H)|^2|\beta_0-\beta_1|.$$

Proof. Suppose without loss of generality that $\beta_0\geq\beta_1$; by the monotonicity of the ferromagnetic Potts model we have that $\mathbf{E}_{\sigma\sim\mu}[m_F(\sigma)]\geq\mathbf{E}_{\sigma\sim\mu'}[m_F(\sigma)]$ (see, e.g., [12, Theorems 1.16 & 3.21]). For a configuration $\sigma:V(H)\to[q]$, let $w(\sigma),w'(\sigma)$ denote its weight under the edge activity vectors $\boldsymbol{\beta}$ and $\boldsymbol{\beta}'$, respectively. Consider an edge $e\in E(F)$. Then, using that for reals $a>b>0$ it holds that $|a^k-b^k|\leq k|a-b|a^k$, we obtain that for every σ it holds that

$$0<w(\sigma)-w'(\sigma)=\beta^{m_F(\sigma)}\big(\beta_0^{|E(H)|-m_F(\sigma)}-(\beta_1)^{|E(H)|-m_F(\sigma)}\big)\leq|E(H)|(\beta_0-\beta_1)w(\sigma)$$

By summing over σ, it follows also that $Z_{G;q,\boldsymbol{\beta}'}\leq Z_{G;q,\boldsymbol{\beta}}$, and combining these we obtain that

$$\mathbf{E}_{\sigma\sim\mu}[m_F(\sigma)]-\mathbf{E}_{\sigma\sim\mu'}[m_F(\sigma)]\leq\frac{\sum_\sigma m_F(\sigma)\big(w(\sigma)-w'(\sigma)\big)}{Z_{G;q,\boldsymbol{\beta}}}\leq|E(H)|^2(\beta_0-\beta_1).\qquad\blacktriangleleft$$

We now give the proof of Lemma 6 which we restate here.

▶ **Lemma 6.** *Let $q, \Delta \geq 3$ be integers and $\beta > \beta_c(q, \Delta)$ be an arbitrary real. Then,*

$\#\mathsf{SuscCubic}(q) \leq_{\mathrm{AP}} \#\mathsf{Susc}(q, \beta, \Delta)$.

Proof. Let H be a cubic bipartite graph and $\hat{\beta} > 1$ be the inputs to $\#\mathsf{SuscCubic}(q)$, and let $\eta \in (0,1)$ be the desired relative error that we want to approximate $\mathcal{S}_{q,\hat{\beta}}(H)$. We may assume that $\hat{\beta} \geq \beta_0 = \left(\frac{q-1}{3}\right)^{1/\Delta}$; for $\hat{\beta} < \beta_0$, a fairly standard coupling argument shows that Glauber dynamics converges rapidly to the Gibbs distribution $\mu_{H;q,\hat{\beta}}$, see for example [2, Theorem 1.1], and therefore it can be used to approximate $\mathcal{S}_{q,\hat{\beta}}(H)$ in time $poly\big(V(H), \frac{1}{\eta}, \mathrm{bits}(\hat{\beta})\big)$ using rejection sampling. For some of the bounds below, it will also be convenient to assume that $|V(H)|, |E(H)|$ are bigger than a sufficiently large constant (otherwise, we can just brute-force).

Let $1 > R_0 > R_1 > 0$ be the constants in Lemma 12, and let $\delta \in (0,1)$ be a rational constant such that for all $B \in (1, 1+\delta)$, it holds that $\frac{1+(B-1)R_0}{1+(B-1)R_1} \leq \beta_0 < \hat{\beta}$. Note that the choice of δ is a constant depending on q, Δ but independent of H and $\hat{\beta}$. By Lemma 11, there are constants $B \in (1, 1+\delta)$, $S > 0$ and an algorithm, which, on input a rational $r \in (0, 1/2)$, outputs in time $poly(\mathrm{bits}(r))$ a pair of susceptibility gadgets $\mathcal{E}_1, \mathcal{E}_2$, each of maximum degree 3 and size $O(|\log r|)$, such that

$$|B_{\mathcal{E}_1} - B|, |B_{\mathcal{E}_2} - B| \leq r, \text{ but } |S_{\mathcal{E}_1} - S_{\mathcal{E}_2}| \geq S. \tag{2}$$

Let $\epsilon = \frac{\eta}{|E(H)|^5}$ and $t = \left\lceil \left(\frac{|E(H)| \log \hat{\beta}}{\epsilon\delta(B-1)}\right)^4 \right\rceil$. By Lemma 7, there is an algorithm that outputs in time $poly(t, \frac{1}{\epsilon})$ an integer n and a graph $G \in \mathcal{G}^{t,\epsilon}_{n,\Delta}$ (satisfying Items 1 and 2). Use the algorithm of Lemma 11 to obtain gadgets $\mathcal{E}_1, \mathcal{E}_2$ satisfying (2) for $r = \frac{\epsilon^4}{10\delta(R_0-R_1)\beta_0}$. Moreover, use Lemma 10, to obtain in time $poly(\mathrm{bits}(r))$ an edge interaction gadget with $1 < B_{\mathcal{P}} < 1 + \epsilon$. Let ℓ be the smallest positive integer such that

$$\left(\tfrac{1+(B_{\mathcal{P}}-1)R_0}{1+(B_{\mathcal{P}}-1)R_1}\right)^{\ell} \left(\tfrac{1+(B-1)R_0}{1+(B-1)R_1}\right) > \hat{\beta}$$

and note that such an integer exists by the choice of δ since the l.h.s. for $\ell = 0$ is smaller than $\hat{\beta}$, and each of the fractions is bigger than 1 from $R_0 > R_1$ and $B > 1$. In fact, we have that $\ell = O(\frac{1}{\epsilon} \log \hat{\beta})$, where the implicit constants depend only on q, Δ. It follows in particular that $\ell < t/3$.

For $i \in \{1, 2\}$, consider now the graphs $\widehat{H}_i = H^{\ell}_{G,\mathcal{P},\mathcal{E}_i}$ and let $\mu_i = \mu_{\widehat{H}_i;q,\beta}$. From Lemma 12, we have that

$$\mathcal{S}_{q,\beta}(\widehat{H}_i) = \mathcal{A}_{\mathcal{E}_i}|E(H)| + \mathbf{E}_{\sigma\sim\mu_i}[m_{H^{\ell}_{G,\mathcal{P}}}(\sigma)] + (1 \pm \eta^2) S_{\mathcal{E}_i} \Big[\big(A^{(i)}_0 - A^{(i)}_1\big) \mathcal{S}_{q,\hat{\beta}_i}(H) + A^{(i)}_1 |E(H)| \Big],$$

where $A_{\mathcal{E}_i} = \mathbf{E}_{\sigma\sim\mu_{\mathcal{E}_i}}[m_{\mathcal{E}_i}(\sigma) \mid \sigma_\rho = \sigma_{\rho'}]$ and

$$\hat{\beta}_i := \left(\tfrac{1+(B_{\mathcal{P}}-1)R_0}{1+(B_{\mathcal{P}}-1)R_1}\right)^{\ell} \left(\tfrac{1+(B_{\mathcal{E}_i}-1)R_0}{1+(B_{\mathcal{E}_i}-1)R_1}\right), \qquad A^{(i)}_j := \frac{B_{\mathcal{E}_i}}{B_{\mathcal{E}_i} + \frac{1-R_j}{R_j}} \text{ for } j \in \{0,1\}.$$

From (2), we have that $\hat{\beta}_i = (1 \pm \epsilon^3)\hat{\beta}$, and therefore from Lemma 13, we have that

$$\big|\mathbf{E}_{\sigma\sim\mu_1}[m_{H^{\ell}_{G,\mathcal{P}}}(\sigma)] - \mathbf{E}_{\sigma\sim\mu_2}[m_{H^{\ell}_{G,\mathcal{P}}}(\sigma)]\big| \leq |E(H^{\ell}_{G,\mathcal{E},\mathcal{P}})|\epsilon^3 \leq \epsilon^2,$$
$$|\mathcal{S}_{q,\hat{\beta}_i}(H) - \mathcal{S}_{q,\hat{\beta}}(H)| \leq |E(H)|^2\epsilon^3 \leq \epsilon^2$$

Using (2), we also have that

$$A^{(i)}_0 = (1 \pm \epsilon)A_0, A^{(i)}_1 = (1 \pm \epsilon^2)A_i, \text{ where } A_j := \frac{B}{B + \frac{1-R_j}{R_j}} \text{ for } j \in \{0,1\}.$$

We can invoke the oracle for $\mathcal{S}_{q,\beta}(\widehat{H}_i)$ to compute $\hat{\mathcal{S}}_i$ such that $\hat{\mathcal{S}}_i = (1 \pm \epsilon^2)\mathcal{S}_{q,\beta}(\widehat{H}_i)$. Note also that $\mathcal{E}_i$ has size $poly(\text{bits}(r))$ and therefore we can invoke the oracle for $\#\mathsf{Susc}(q, \Delta, \beta)$ to compute $\hat{A}_{\mathcal{E}_i}, \hat{S}_{\mathcal{E}_1}$ such that $\hat{A}_{\mathcal{E}_i} = (1 \pm \epsilon^2)A_{\mathcal{E}_i}$ and $\hat{S}_{\mathcal{E}_i} = (1 \pm \epsilon^2)S_{\mathcal{E}_i}$.[3] It follows that

$$\hat{S} = \frac{1}{A_0 - A_1}\Big(\frac{(\hat{\mathcal{S}}_1 - \hat{\mathcal{S}}_2) - |E(H)|(\hat{A}_{\mathcal{E}_1} - \hat{A}_{\mathcal{E}_2})}{\hat{S}_{\mathcal{E}_1} - \hat{S}_{\mathcal{E}_2}} - A_1|E(H)|\Big)$$

is within a factor of $(1 \pm \eta)$ of the susceptibility $\mathcal{S}_{q,\hat{\beta}}(H)$, as needed. This finishes the reduction and therefore the proof of Lemma 6. ◀

3 Hardness of vertex-edge observables for 2-spin systems

Throughout this section, we will fix integer $\Delta \geq 3$, and antiferromagnetic $(\beta, \gamma, \lambda) \in \mathcal{N}_\Delta$ in the non-uniqueness region. We will also fix an (a, b, c) vertex-edge observable that is non-trivial on bipartite graphs.

3.1 The interpolation scheme

Analogously to Section 2, it will be convenient to consider the following computational problems.

Name $\#\mathsf{Observable2Spin}(\beta, \gamma, \lambda, a, b, c)$.
Instance A bipartite graph G with max degree Δ.
Output The (a, b, c) vertex-edge observable on G with parameters β,γ,λ, i.e., the value $\mathcal{O}_{\beta,\gamma,\lambda}(G)$.

In the second, the parameter is going to be the edge activity $\alpha < 1$ of an antiferromagnetic Ising model; note that the problem allows the vertex activity to be part of the input.

Name $\#\mathsf{MagnetIsingCubic}(\alpha)$.
Instance A cubic bipartite graph H, and a rational vertex activity $\hat{\lambda} > 0$.
Output The magnetization on H for the Ising model with parameters $\alpha, \hat{\lambda}$, i.e., the value $\mathcal{M}_{\alpha,\alpha,\hat{\lambda}}(H)$.

We now show the following analogue of the interpolation scheme in Lemma 6.

▶ **Lemma 14.** *Let $\Delta \geq 3$ be an integer and $(\beta, \gamma, \lambda) \in \mathcal{N}_\Delta$. Then, there is $\alpha \in (0, 1)$ such that for any (a, b, c) vertex-edge observable that is not trivial on bipartite graphs,*

$$\#\mathsf{MagnetIsingCubic}(\alpha) \leq_{\text{AP}} \#\mathsf{Observable2Spin}(\beta, \gamma, \lambda, a, b, c).$$

Assuming the key Lemma 14, the proof of Theorem 5 can be done analogously to Theorem 2, interpolating now in terms of the vertex activity $\hat{\lambda}$. We defer the proof to Section A of the full version.

[3] For $\hat{A}_{\mathcal{E}_i}$, just invoke the oracle on the graph obtained from $\mathcal{E}_i$ by identifying ρ_i and ρ_i'; this graph has maximum degree at most 3 since ρ_i, ρ_i' both have degree 1. Observe that $S_{\mathcal{E}_i} = 2A_{\mathcal{E}_i} - \mathcal{S}_{q,\beta}(\mathcal{E}_i)$ and therefore we can obtain the desired $\hat{S}_{\mathcal{E}_i}$ by using a further oracle call on $\mathcal{E}_i$ to approximate $\mathcal{S}_{q,\beta}(\mathcal{E}_i)$.

3.2 The gadgets

In this section, we outline the gadgets that will be used to prove Lemma 14. These are analogous to those presented in the case of the Potts model, especially the phase gadgets. To account for general vertex-edge observables, we refine appropriately the field-gadget idea of [9], by now paying attention to the so-called observable gap (cf. Definition 17).

3.2.1 Bipartite phase gadgets for antiferromagnetic 2-spin systems

We follow the same notation as in Section 2.2.1 to denote for integers t, n, Δ the class $\mathcal{G}^t_{n,\Delta}$ of bipartite graphs where there are n vertices with degree Δ on each side, and t vertices of degree $\Delta - 1$ on each side. For a graph $G \in \mathcal{G}^t_{n,\Delta}$, we denote its bipartition by (U^+, U^-) where U^+, U^- are vertex sets with $|U^+| = |U^-| = n$, and we denote by W^+, W^- the sets of vertices with degree $\Delta - 1$ on each side of the bipartition, so that $|W^+| = |W^-| = t$. We will refer to set $W = W^+ \cup W^-$ as the *ports* of G. For $\sigma : U \to \{0, 1\}$, we define the *phase* $\mathcal{Y}(\sigma)$ of the configuration σ as the side of the bipartite graph which has the most occupied vertices under σ, i.e.,

$$\mathcal{Y}(\sigma) = \arg \max_{i \in \{+,-\}} |\sigma^{-1}(1) \cap U^i|.$$

It is known that for $(\beta, \gamma, \lambda) \in \mathcal{N}_\Delta$ the system of equations $x = \frac{1}{\lambda}\left(\frac{\beta y+1}{y+\gamma}\right)^{\Delta-1}, y = \frac{1}{\lambda}\left(\frac{\beta x+1}{x+\gamma}\right)^{\Delta-1}$ has a unique solution with $y > x > 0$, see, e.g., [8, Lemma 7]. Let $q_+ = \frac{1}{1+x}$, $q_- = \frac{1}{1+y}$ and note that q_+, q_- are distinct numbers in the interval $(0, 1)$. Define the product distributions $Q^+_W(\cdot), Q^-_W(\cdot)$ by

$$Q^{\pm}_W(\tau) = (q_{\pm})^{|\tau^{-1}(1) \cap W^+|}(1 - q_{\pm})^{|\tau^{-1}(0) \cap W^+|}(q_{\mp})^{|\tau^{-1}(1) \cap W^-|}(1 - q_{\mp})^{|\tau^{-1}(0) \cap W^-|}. \tag{3}$$

We will need the following two properties from the phase gadget G for some sufficiently small $\epsilon > 0$. Let $\mu := \mu_{G;\beta,\gamma,\lambda}$.

1. The phases appear with roughly equal probability, i.e., $|\mu(\mathcal{Y}(\sigma) = \pm) - \frac{1}{2}| \le \epsilon$.
2. For any $\tau : W \to \{0, 1\}$, $\left|\frac{\mu(\sigma_W = \tau \mid \mathcal{Y}(\sigma) = \pm)}{Q^{\pm}_W(\tau)} - 1\right| \le \epsilon$.

Let $\mathcal{G}^{t,\epsilon}_{n,\Delta}$ denote the set of graphs $G \in \mathcal{G}^t_{n,\Delta}$ satisfying Items 1 and 2. The following lemma is is shown in [3].

▶ **Lemma 15** ([3, Lemma 9]). *Let $\Delta \ge 3$ and $(\beta, \gamma, \lambda) \in \mathcal{N}_\Delta$. Then, there is a randomized algorithm that, on input integer $t \ge 1$ and $\epsilon > 0$, outputs in time $\mathrm{poly}(t, \frac{1}{\epsilon})$ an integer n and a graph G that belongs to $\mathcal{G}^{t,\epsilon}_{n,\Delta}$, with probability $\ge 3/4$.*

3.2.2 Field gadgets with observable gaps

We adopt the following definition of "field" gadgets from [9].

▶ **Definition 16.** *For $\lambda \ne \frac{1-\beta}{1-\gamma}$, a* field gadget *is a rooted tree $\mathcal{T}$ whose root ρ has degree one. When $\lambda = \frac{1-\beta}{1-\gamma}$, a field gadget consists of a rooted bipartite graph obtained from a rooted tree where some of the leaves have been replaced by a distinct cycle of length four (by identifying the leaf with a vertex of the cycle).*

▶ **Definition 17.** *Let $\mathcal{T}$ be a field gadget rooted at ρ, and $\mu = \mu_{\mathcal{T};\beta,\gamma,\lambda}$. We denote by $R_{\mathcal{T}} = R_{\mathcal{T}}(\beta, \gamma, \lambda)$ the* effective field *of the gadget, i.e., $R_{\mathcal{T}} = \frac{1}{\lambda}\frac{\mu(\sigma_\rho=1)}{\mu(\sigma_\rho=0)}$ and by $O_{\mathcal{T}} = O_{\mathcal{T}}(\beta, \gamma, \lambda)$ the* observable gap *of the gadget, i.e., $O_{\mathcal{T}} = \mathbf{E}_{\sigma\sim\mu}[\, o_{\mathcal{T}}(\sigma) \mid \sigma_\rho = 1] - a - \mathbf{E}_{\sigma\sim\mu}[\, o_{\mathcal{T}}(\sigma) \mid \sigma_\rho = 0]$.*

The division by λ in the definition of the effective field of a gadget is to avoid double-counting the contribution of the root later on.

▶ **Lemma 18.** *Let (β,γ,λ) be antiferromagnetic such that at least one of $\beta\neq\gamma$ or $\lambda\neq 1$ holds. There are constants $C,\tilde{R}>0$ with $\tilde{R}\neq 1$ and an algorithm which, on input a rational $r\in(0,1/2)$, outputs in time poly$(\mathrm{bits}(r))$ field gadgets $\mathcal{T}_+,\mathcal{T}_-$, each of maximum degree 3 and size $O(|\log r|)$, such that*

$$R_{\mathcal{T}_+} > R_{\mathcal{T}_-} + r/2 \text{ and } |R_{\mathcal{T}_+}-\tilde{R}|,\ |R_{\mathcal{T}_-}-\tilde{R}|\leq r.$$

▶ **Theorem 19.** *Let (β,γ,λ) be antiferromagnetic, and consider any non-trivial vertex-edge observable on general graphs. There exist constants $\hat{R},\hat{O},\Xi>0$ and an algorithm, which, on input a rational $r\in(0,1/2)$, outputs in time poly$(\mathrm{bits}(r))$ a pair of field gadgets $\mathcal{T}_1,\mathcal{T}_2$, each of maximum degree 3 and size $O(|\log r|)$, such that*

$$|R_{\mathcal{T}_1}-\hat{R}|,\ |R_{\mathcal{T}_2}-\hat{R}|\leq r,\ \text{ but } |O_{\mathcal{T}_1}-O_{\mathcal{T}_2}|\geq\hat{O}.$$

Moreover, the observable gaps $O_{\mathcal{T}_1},O_{\mathcal{T}_2}$ are upper-bounded in absolute value by the constant Ξ.

The proofs of Lemma 18 and Theorem 19 follow closely the approach in [9], and are therefore deferred to Section C.3 of the full version.

3.3 The reduction

Let H be a cubic bipartite graph which is input to the problem #MagnetIsingCubic(α) of Section 3.1, for some constant $\alpha\in(0,1)$ to be specified. For integers $n,t\geq 1$ and rational $\epsilon>0$, let $G\in\mathcal{G}^{t,\epsilon}_{n,\Delta}$ be a bipartite phase gadget satisfying Items 1 and 2 of Section 3.2.1. Let $\mathcal{T}_+,\mathcal{T}_-,\mathcal{T}$ be field gadgets. Note that the gadgets $\mathcal{T}_+,\mathcal{T}_-$ serve a different role to that of $\mathcal{T}$, and in particular they will be used to interpolate over the vertex activity $\hat{\lambda}$.

To achieve this, for integers ℓ_+,ℓ_- satisfying $t\geq 5+\max\{\ell_+,\ell_-\}$, we define the graph $H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-,\mathcal{T}}$ as follows. For each vertex v of H replace it with a distinct copy of G, denoted by G_v; we denote by $U^{\pm}_v,W^{\pm}_v$ the sets corresponding to $U^{\pm},W^{\pm}$ in G_v. Moreover, for each $v\in V(H)$, attach one copy of the gadget $\mathcal{T}$ and ℓ_+ copies of the gadget $\mathcal{T}_+$ on mutually distinct vertices of W^+ by identifying them with the corresponding roots. Similarly, attach ℓ_- copies of the gadget $\mathcal{T}_-$ on mutually distinct vertices of W^-. Let $\mathcal{T}_v$ be the copy of $\mathcal{T}_v$ corresponding to v, and w_v be its root. Let $W_{\mathcal{T}}=\{w_v\mid v\in V(H)\}$ be the set of all these roots. Further, for each edge $\{u,v\}$ of H, add an edge between W^+_u and W^+_v, and an edge between W^-_u and W^-_v.

Let $H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-}$ be the graph without the internal vertices and edges of the copies of gadget $\mathcal{T}$, i.e., we keep only the roots $W_{\mathcal{T}}$ of the gadgets in $H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-}$. The following piece of notation will be useful: for a graph J and a subgraph J' of J, given a configuration $\sigma:V(J)\to[q]$, it will be convenient to denote by $m_{J'}(\sigma)=\sum_{e=\{u,v\}\in E(J')}\mathbf{1}\{\sigma(u)=\sigma(v)\}$ the number of monochromatic edges of J' under σ.

The following lemma relates the value of the observable $\mathcal{O}_{\beta,\gamma,\lambda}(H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-,\mathcal{T}})$ with the magnetization $\mathcal{S}_{\alpha,\hat{\lambda}}(H)$, for some appropriate $\hat{\lambda}$ that is a function of the parameters β,γ,λ and $\ell_+,\ell_-,R_{\mathcal{T}_+},R_{\mathcal{T}_+},R_{\mathcal{T}}$. Analogously to Section 2.3, for a graph J and a subgraph J' of J, given a configuration $\sigma:V(J)\to[q]$, it will be convenient to denote by $o_{J'}(\sigma)=a|\sigma_{V(J')}|+bm_0(\sigma_{V(J')})+cm_1(\sigma_{V(J')})$ the contribution of J' to the value of the observable on J.

▶ **Lemma 20.** *Let $\Delta \geq 3$ be an integer, $(\beta,\gamma,\lambda) \in \mathcal{N}_\Delta$, and (a,b,c) be a vertex-edge observable. Then, there are constants $q_+, q_- \in (0,1)$ with $q_+ > q_-$ and $\alpha \in (0,1)$ so that the following holds for any field gadgets $\mathcal{T}_+, \mathcal{T}_-, \mathcal{T}$ with effective fields R_+, R_-, R and observable gaps O_1, O_2, O, and any positive integers ℓ_+, ℓ_-, t with $t \geq 5 + \max\{\ell_+, \ell_-\}$.*

For a cubic bipartite graph H, for any $\epsilon \leq \frac{1}{(5|V(H)|)^2}$, any integer n and phase gadget $G \in \mathcal{G}^{t,\epsilon}_{n,\Delta}$, for $\mu := \mu_{H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-,\mathcal{T}}}$ and $\epsilon' = 10|V(H)|\epsilon$, it holds that

$$\mathcal{O}_{\beta,\gamma,\lambda}(H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-,\mathcal{T}}) = \mathcal{A}|V(H)| + \mathbf{E}_{\sigma\sim\mu}\big[o_{H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-}}(\sigma)\big] + (1\pm\epsilon')O\big[(q_+ - q_-)\mathcal{M}_{\alpha,\hat{\lambda}}(H) + q_-|V(H)|\big],$$

where $\mathcal{A} = \mathbf{E}_{\sigma\sim\mu_\mathcal{T}}[o_\mathcal{T}(\sigma) \mid \sigma_\rho = 0]$ and $\hat{\lambda} := \big(\frac{q_+R+1-q_+}{q_-R+1-q_-}\big)\big(\frac{q_+R_++1-q_+}{q_-R_++1-q_-}\big)^{\ell_+}/\big(\frac{q_+R_-+1-q_+}{q_-R_-+1-q_-}\big)^{\ell_-}$.

The proof of Lemma 20 builds upon similar ideas to that of Lemma 12 (see in particular the discussion around there for how this blends with the overview of Section 2.1) and is deferred to Section B of the full version.

We will need the following bound on the change of the observable value when we change the vertex activities of a subset of the vertices. Namely, let $G = (V,E)$ be a graph and (β,γ) be antiferromagnetic. For a vertex-activity vector $\boldsymbol{\lambda} = \{\lambda_v\}_{v\in V}$, define the Gibbs distribution $\mu_{G;\beta,\gamma,\boldsymbol{\lambda}}(\sigma) \propto \beta^{m_0(\sigma)}\gamma^{m_1(\sigma)}\prod_{v\in V;\,\sigma(v)=1}\lambda_v$ for $\sigma : V \to \{0,1\}$.

▶ **Lemma 21** (Minor adaptation of [9, Lemma 35])**.** *Let (β,γ) be antiferromagnetic, $\lambda,\lambda_1,\lambda_2 > 0$, and (a,b,c) be a vertex-edge observable. Let $G = (V,E)$ be a graph and $S \subseteq V$. For $i \in \{1,2\}$, let $\boldsymbol{\lambda}_i$ be the field vector on V, where every $v \in S$ has activity λ_i, whereas every $v \in V\backslash S$ has activity λ. Let μ_i be the Gibbs distribution on G with parameters $\beta,\gamma,\boldsymbol{\lambda}_i$. Then, for every subgraph F of G, it holds that*

$$\big|\mathbf{E}_{\sigma\sim\mu_2}[o_F(\sigma)] - \mathbf{E}_{\sigma\sim\mu_1}[o_F(\sigma)]\big| \leq 2\big(|a|+|b|+|c|\big)\big(|V(G)|^2 + |E(G)|^2\big)\left|\frac{\lambda_2}{\lambda_1} - 1\right|.$$

We now have all the ingredients to prove Lemma 14.

▶ **Lemma 14.** *Let $\Delta \geq 3$ be an integer and $(\beta,\gamma,\lambda) \in \mathcal{N}_\Delta$. Then, there is $\alpha \in (0,1)$ such that for any (a,b,c) vertex-edge observable that is not trivial on bipartite graphs,*

$$\#\mathsf{MagnetIsingCubic}(\alpha) \leq_{\mathrm{AP}} \#\mathsf{Observable2Spin}(\beta,\gamma,\lambda,a,b,c).$$

Proof. Let $K = |a|+|b|+|c|$ and q_+, q_-, α be the constants from Lemma 20; recall that $\alpha \in (0,1)$ and $1 > q_+ > q_- > 0$. Let H be a cubic bipartite graph and $\hat{\lambda} > 1$ be the inputs to $\#\mathsf{MagnetIsingCubic}(\alpha)$, and let $\eta \in (0,1)$ be the desired relative error that we want to approximate $\mathcal{M}_{\alpha,\hat{\lambda}}(H)$.

By Lemma 18 and Theorem 19, there exist constants $\tilde{R}, \hat{R}, \hat{O}, C > 0$ with $\tilde{R} \neq 1$ and polynomial time algorithms, which on input rationals $r, r' \in (0,1/2)$, output in time $poly(\mathrm{bits}(r), \mathrm{bits}(r'))$ pairs of field gadgets $(\mathcal{T}_+,\mathcal{T}_-)$ and $(\mathcal{T}_1,\mathcal{T}_2)$ satisfying

$$\begin{aligned} &R_+ > R_- + Cr \text{ and } |R_+ - \tilde{R}|,\ |R_- - \tilde{R}| \leq r,\\ &|R_1 - \hat{R}|,\ |R_2 - \hat{R}| \leq r, \text{ but } |O_1 - O_2| \geq \hat{O}, \end{aligned} \tag{4}$$

where R_+, R_-, R_1, R_2 are the effective fields of $\mathcal{T}_+, \mathcal{T}_-, \mathcal{T}_1, \mathcal{T}_2$ and O_1, O_2 are the observable gaps of $\mathcal{T}_1, \mathcal{T}_2$, respectively.

Let $\epsilon = \frac{\eta}{|V(H)|^8}$ and $t = \big\lceil\big(\frac{1}{\epsilon}|V(H)|^2\log\hat{\lambda}\big)^6\big\rceil$. By Lemma 7, there is an algorithm that outputs in time $poly(t, \frac{1}{\epsilon})$ an integer n and a graph $G \in \mathcal{G}^{t,\epsilon}_{n,\Delta}$ (satisfying Items 1 and 2). Let also $\mathcal{T}_+, \mathcal{T}_-$ be field gadgets satisfying (4) for $r = \frac{|\tilde{R}-1|}{10}\epsilon^4$. Consider also the integers $\ell_+, \ell_- = \ell$,

where ℓ is an integer specified according to whether $\hat{\Lambda} = \hat{\lambda}/\big(\frac{q_+\hat{R}+1-q_+}{q_-\hat{R}+1-q_-}\big)$ is bigger than 1. Suppose first that $\hat{\Lambda} \geq 1$. Since $R_+ > R_-$ and $q_+ > q_-$, we have that $\frac{q_+R_++1-q_+}{q_-R_++1-q_-} > \frac{q_+R_-+1-q_+}{q_-R_-+1-q_-}$, and we pick ℓ to be the smallest positive integer such that

$$\big(\tfrac{q_+R+1-q_+}{q_-R+1-q_-}\big)\big(\tfrac{q_+R_++1-q_+}{q_-R_++1-q_-}\big)^{\ell}/\big(\tfrac{q_+R_-+1-q_+}{q_-R_-+1-q_-}\big)^{\ell} \geq \hat{\lambda}. \tag{5}$$

If $\hat{\Lambda} < 1$, then we pick ℓ to be the small positive integer so that

$$\big(\tfrac{q_+R+1-q_+}{q_-R+1-q_-}\big)\big(\tfrac{q_+R_-+1-q_+}{q_-R_-+1-q_-}\big)^{\ell}/\big(\tfrac{q_+R_++1-q_+}{q_-R_++1-q_-}\big)^{\ell} \leq \hat{\lambda},$$

In either case, using the lower bound $R_+ - R_- > Cr$ from (4), we have that $\ell = O(\frac{1}{r}\log\hat{\lambda})$ where the implicit constant depends only on $\beta, \gamma, \lambda, \Delta$. In particular, we have that $t \geq 5 + \max\{\ell_+, \ell_-\}$. In the argument below, we assume w.l.o.g. that $\hat{\Lambda} \geq 1$; otherwise, just apply the same argument by swapping the roles of the gadgets $\mathcal{T}_+, \mathcal{T}_-$ in the construction below.

For $i \in \{1,2\}$, consider now the graphs $\widehat{H}_i = H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-,\mathcal{T}_i}$ and let $\mu_i = \mu_{\widehat{H}_i;\beta,\gamma,\lambda}$. For convenience, let also F denote the graph $H^{\ell_+,\ell_-}_{G,\mathcal{T}_+,\mathcal{T}_-}$, and note that F is a subgraph of both $\widehat{H}_1, \widehat{H}_2$. From Lemma 20, we have that for $i \in \{1,2\}$, for $\epsilon' = 10|V(H)|\epsilon$, it holds that

$$\mathcal{O}_{\beta,\gamma,\lambda}(\widehat{H}_i) = \mathcal{A}_i|V(H)| + \mathbf{E}_{\sigma\sim\mu_i}\big[o_F(\sigma)\big] + (1\pm\epsilon')O_i\big[(q_+ - q_-)\mathcal{M}_{\alpha,\hat{\lambda}_i}(H) + q_-|V(H)|\big], \tag{6}$$

where $\mathcal{A}_i = \mathbf{E}_{\sigma\sim\mu_{\mathcal{T}_i}}[o_{\mathcal{T}_i}(\sigma) \mid \sigma_{\rho_i} = 0]$ and $\hat{\lambda}_i := \big(\frac{q_+R_i+1-q_+}{q_-R_i+1-q_-}\big)\big(\frac{q_+R_++1-q_+}{q_-R_++1-q_-}\big)^{\ell_+}/\big(\frac{q_+R_-+1-q_+}{q_-R_-+1-q_-}\big)^{\ell_-}$. From (2), we have that $\hat{\lambda}_i = (1\pm\epsilon^3)\hat{\lambda}$, and therefore from Lemma 21, we have that

$$\big|\mathbf{E}_{\sigma\sim\mu_1}[o_F(\sigma)] - \mathbf{E}_{\sigma\sim\mu_2}[o_F(\sigma)]\big| \leq |E(H^{\ell}_{G,\mathcal{E},\mathcal{P}})|\epsilon^3 \leq \epsilon^2,$$
$$|\mathcal{M}_{\alpha,\hat{\lambda}_1}(H) - \mathcal{M}_{\alpha,\hat{\lambda}_2}(H)| \leq 2K\big(|V(H)|^2 + |E(H)|^2\big)\epsilon^3 \leq \epsilon^2.$$

We now invoke the oracle for $\mathcal{M}_{\beta,\gamma,\lambda}(\widehat{H}_i)$ to compute $\hat{\mathcal{M}}_i$ such that $\hat{\mathcal{M}}_i = (1\pm\epsilon^2)\mathcal{M}_{\beta,\gamma,\lambda}(\widehat{H}_i)$. By exploiting the tree structure of the field gadgets $\mathcal{T}_1, \mathcal{T}_2$ (cf. Definition 16), and since they both have size $poly(\text{bits}(r))$, we can compute the values $\mathcal{A}_1, \mathcal{A}_2$ exactly in time $poly(|V(H)|, \frac{1}{\eta})$ by fairly routine dynamic programming techniques. Combining these with (6), it follows that

$$\hat{M} = \frac{1}{q_+ - q_-}\Big(\frac{(\hat{\mathcal{M}}_1 - \hat{\mathcal{M}}_2) - |V(H)|(\mathcal{A}_1 - \mathcal{A}_2)}{O_1 - O_2} - q_-|E(H)|\Big)$$

is within a factor of $(1\pm\eta)$ of the susceptibility $\mathcal{M}_{\alpha,\hat{\lambda}}(H)$, as needed. This finishes the reduction and therefore the proof of Lemma 6. ◀

References

1. R. J. Baxter. Onsager and Kaufman's calculation of the spontaneous magnetization of the Ising model. *Journal of Statistical Physics*, 145(3):518–548, 2011.
2. M. Bordewich, C. Greenhill, and V. Patel. Mixing of the Glauber dynamics for the ferromagnetic Potts model. *Random Structures & Algorithms*, 48(1):21–52, 2016.
3. J.-Y. Cai, A. Galanis, L. A. Goldberg, H. Guo, M. Jerrum, D. Štefankovič, and E. Vigoda. #BIS-hardness for 2-spin systems on bipartite bounded degree graphs in the tree non-uniqueness region. *Journal of Computer and System Sciences*, 82(5):690–711, 2016.
4. D. Chelkak and S. Smirnov. Universality in the 2D Ising model and conformal invariance of fermionic observables. *Inventiones mathematicae*, 189(3):515–580, 2012.
5. Z. Chen, K. Liu, and E. Vigoda. Optimal mixing of Glauber dynamics: Entropy factorization via high-dimensional expansion. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1537–1550, 2021.

6 M. E. Dyer, L. A. Goldberg, C. S. Greenhill, and M. Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

7 A. Galanis, L. A. Goldberg, and M. Jerrum. Approximately counting H-colorings is #BIS-hard. *SIAM Journal on Computing*, 45(3):680–711, 2016.

8 A. Galanis, D. Štefankovič, and E. Vigoda. Inapproximability of the partition function for the antiferromagnetic Ising and hard-core models. *Combinatorics, Probability and Computing*, 25(4):500–559, 2016.

9 A. Galanis, D. Štefankovič, and E. Vigoda. The complexity of approximating averages on bounded-degree graphs. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1345–1355, 2020.

10 A. Galanis, D. Štefankovič, E. Vigoda, and L. Yang. Ferromagnetic Potts model: refined #BIS-hardness and related results. *SIAM Journal on Computing*, 45(6):2004–2065, 2016.

11 L. A. Goldberg and M. Jerrum. Approximating the partition function of the ferromagnetic Potts model. *Journal of the ACM*, 59(5), 2012.

12 G. Grimmett. The Random-Cluster Model. In *Probability on Discrete Structures*, pages 73–123. Springer, 2004.

13 M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22(5):1087–1116, 1993.

14 M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

15 L. Li, P. Lu, and Y. Yin. Correlation decay up to uniqueness in spin systems. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 67–84, 2013.

16 L. J. Schulman, A. Sinclair, and P. Srivastava. Symbolic integration and the complexity of computing averages. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1231–1245, 2015.

17 A. Sinclair and P. Srivastava. Lee-Yang theorems and the complexity of computing averages. *Communications in Mathematical Physics*, 329(3):827–858, 2014.

18 A. Sly. Computational transition at the uniqueness threshold. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 287–296, 2010.

19 A. Sly and N. Sun. Counting in two-spin models on d-regular graphs. *Ann. Probab.*, 42(6):2383–2416, 2014.

20 D. Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 140–149, 2006.

The Decision Problem for Perfect Matchings in Dense Hypergraphs

Luyining Gan
Department of Mathematics and Statistics, University of Nevada, Reno, NV, USA

Jie Han[1]
School of Mathematics and Statistics and Center for Applied Math,
Beijing Institute of Technology, China

Abstract

Given $1 \leq \ell < k$ and $\delta \geq 0$, let $\mathbf{PM}(k, \ell, \delta)$ be the decision problem for the existence of perfect matchings in n-vertex k-uniform hypergraphs with minimum ℓ-degree at least $\delta\binom{n-\ell}{k-\ell}$. For $k \geq 3$, the decision problem in general k-uniform hypergraphs, equivalently $\mathbf{PM}(k, \ell, 0)$, is one of Karp's 21 NP-complete problems. Moreover, for $k \geq 3$, a reduction of Szymańska showed that $\mathbf{PM}(k, \ell, \delta)$ is NP-complete for $\delta < 1 - (1 - 1/k)^{k-\ell}$. A breakthrough by Keevash, Knox and Mycroft [STOC '13] resolved this problem for $\ell = k - 1$ by showing that $\mathbf{PM}(k, k-1, \delta)$ is in P for $\delta > 1/k$. Based on their result for $\ell = k - 1$, Keevash, Knox and Mycroft conjectured that $\mathbf{PM}(k, \ell, \delta)$ is in P for every $\delta > 1 - (1 - 1/k)^{k-\ell}$.

In this paper it is shown that this decision problem for perfect matchings can be reduced to the study of the minimum ℓ-degree condition forcing the existence of fractional perfect matchings. That is, we hopefully solve the "computational complexity" aspect of the problem by reducing it to a well-known extremal problem in hypergraph theory. In particular, together with existing results on fractional perfect matchings, this solves the conjecture of Keevash, Knox and Mycroft for $\ell \geq 0.4k$.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Computational Complexity, Perfect Matching, Hypergraph

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.64

Category Track A: Algorithms, Complexity and Games

1 Introduction

As arguably the most natural extension of graph objects to hypergraphs, matchings have attracted a great deal of attention from both mathematicians and theoretical computer scientists. However, the study of hypergraph matching problems is still a challenging task. One particular reason for this is that finding the maximal matchings in k-uniform hypergraphs for $k \geq 3$ (e.g. 3-partite 3-uniform hypergraphs) is famously NP-complete [14], in contrast to the tractability in the graph case (Edmonds' blossom algorithm [5]).

Hypergraph matchings also find exciting applications in other fields, e.g. the Existence Conjecture of Block Designs [16, 8], Ryser's Conjecture on Latin Squares and Samuels' Conjecture in Probability Theory. For applications on practical problems, one prime example is that Asadpour, Feige and Saberi [2] used hypergraph perfect matchings to study the *Santa Claus problem*.

In this paper we continue the study of the decision problem of perfect matchings in dense hypergraphs, initiated by Karpiński, Ruciński and Szymańska [15]. Given $k \geq 2$, a *k-uniform hypergraph* (or k-graph) H consists of a vertex set $V(H)$ and an edge set $E(H)$, where each edge in $E(H)$ is a set of k vertices of H. A subset $M \subseteq E(H)$ is a *matching* if every two edges from M are vertex-disjoint. A matching in H is called *perfect* if it covers all vertices

[1] Corresponding author.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 64; pp. 64:1–64:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of H. Given a k-graph H with an ℓ-element vertex set S (where $0 \le \ell \le k-1$) we define $\deg_H(S)$ to be the number of edges containing S. The *minimum ℓ-degree* $\delta_\ell(H)$ of H is the minimum of $\deg_H(S)$ over all ℓ-element sets of vertices in H.

The following decision problem was raised by Keevash, Knox and Mycroft [18], generalizing a problem of Karpiński, Ruciński and Szymańska [15] for the case $\ell = k-1$.

▶ **Problem 1.** *Given integers $\ell < k$ and $\delta \in [0,1]$, denote by $\boldsymbol{PM}(k,\ell,\delta)$ the problem of deciding whether there is a perfect matching in a given k-graph H on $n \in k\mathbb{N}$ vertices with $\delta_\ell(H) \ge \delta\binom{n-\ell}{k-\ell}$.*

The motivating fact is that for $k \ge 3$, $\mathbf{PM}(k,\ell,0)$ is equivalent to the problem for general k-graphs, so is NP-complete; on the other hand $\mathbf{PM}(k,\ell,\delta)$ is trivially in P when δ is large (e.g., when $\delta > 1-1/k$ by the result of [9]) because all such k-graphs contain perfect matchings. Therefore, it is natural to ask for the point where the behavior changes. A reduction of Szymańska [22] showed that $\mathbf{PM}(k,\ell,\delta)$ is NP-complete for $k \ge 3$ and $\delta < 1-(1-1/k)^{k-\ell}$. In a breakthrough paper, Keevash, Knox and Mycroft [18] conjectured that $1-(1-1/k)^{k-\ell}$ is the turning point and verified the case $\ell = k-1$.

▶ **Conjecture 2** (Keevash, Knox and Mycroft [18])**.** *For $1 \le \ell < k$, $\boldsymbol{PM}(k,\ell,\delta)$ is in P for every $\delta > 1-(1-1/k)^{k-\ell}$.*

Recently, Han and Treglown [13] showed that the conjecture holds for $0.5k \le \ell \le (1+\ln(2/3))k \approx 0.59k$. In this paper we verify Conjecture 2 for all $\ell \ge 0.4k$. In fact, our main result reduces the conjecture to the study of the minimum-degree-type threshold for the existence of a perfect fractional matching in k-graphs. To illustrate this, we introduce the following definitions.

Given a k-graph $H = (V,E)$, a *fractional matching* in H is a function $\omega : E \to [0,1]$ such that for each $v \in V(H)$ we have that $\sum_{e \ni v} w(e) \le 1$. Then $\sum_{e \in E(H)} w(e)$ is the *size* of w. If the size of w in H is n/k then we say that w is a *perfect fractional matching*. Given $k,\ell \in \mathbb{N}$ such that $\ell \le k-1$, define $c^*_{k,\ell}$ to be the smallest number c such that every k-graph H on n vertices with $\delta_\ell(H) \ge (c+o(1))\binom{n-\ell}{k-\ell}$ contains a perfect fractional matching. The following is our main result.

▶ **Theorem 3.** *Suppose $k,\ell \in \mathbb{N}$ such that $1 \le \ell \le k-1$. Then for any $\delta \in (c^*_{k,\ell},1]$, $\mathbf{PM}(k,\ell,\delta)$ is in P. That is, for any $\delta \in (c^*_{k,\ell},1]$, there exists a constant $c = c(k)$ such that there is an algorithm with running time $O(n^c)$ which given any n-vertex k-graph H with $\delta_\ell(H) \ge \delta\binom{n-\ell}{k-\ell}$, determines whether H contains a perfect matching.*

In fact, in [13] a similar result was proved for $\delta \in (\delta^*,1]$ where $\delta^* = \max\{c^*_{k,\ell},1/3\}$. Comparing with their result, Theorem 3 drops the extra $1/3$ and thus extends the result to large values of ℓ, e.g., for $\ell > (1+\ln(2/3))k$.

For the parameter $c^*_{k,\ell}$, Alon, Frankl, Huang, Rödl, Ruciński, and Sudakov [1] in 2012 made the following conjecture.

▶ **Conjecture 4** ([1])**.** *For all $\ell,k \in \mathbb{N}$, $c^*_{k,\ell} = 1-(1-1/k)^{k-\ell}$.*

They [1] verified the case $k-\ell \le 4$. The conjecture was further validated by Kühn, Osthus and Townsend [19, Theorem 1.7] for $\ell \ge k/2$ and by Han [10, Theorem 1.5] for $\ell = (k-1)/2$. In a recent work, Frankl and Kupavskii [6] verified this conjecture for $\ell \ge 0.4k$. Unfortunately, despite the efforts from experts in the field, Conjecture 4 is still open and appears to be very challenging for small values of ℓ. In fact, Conjecture 4 is also closely related to an old conjecture of Erdős on the size of the largest matching in hypergraphs (in particular, results of [6, 10] are corollaries of the corresponding progresses on the conjecture of Erdős).

Combining Theorem 3 with the current status on $c^*_{k,\ell}$ we get the following corollary.

▶ **Corollary 5.** *Conjecture 2 holds for $\ell \geq 0.4k$ and for $k - \ell \leq 4$.*

Thus, by Theorem 3, Conjecture 2 holds for all cases when $c^*_{k,\ell} = 1-(1-1/k)^{k-\ell}$, that is, whenever Conjecture 4 holds. Indeed, it is not hard to show that if $\delta > c^*_{k,\ell}$, then the k-graph contains a matching that covers all but exactly k vertices (see Theorem 11). Given this, our result can be viewed as the efficient detection of a certain class of *divisibility constructions* that prevents the existence of perfect matchings. As a consequence, we reduce the decision problem to an extremal problem on the existential problem of a perfect fractional matching, which can be recognized as a resolution on the "computational complexity" aspect of this problem.

We now give an overview of the minimum-degree-type conditions as well as the divisibility constructions.

1.1 Minimum degree conditions and divisibility barriers

The minimum degree conditions forcing a perfect matching have been studied extensively over the last two decades. Focusing on the asymptotical thresholds, all known results support the following conjecture raised by Hàn, Person and Schacht [9]. Note that this corresponds to the case when the decision problem is trivially in P (a trivial algorithm that always outputs yes).

▶ **Conjecture 6** (Hàn–Person–Schacht, [9])**.** *Given $1 \leq \ell < k$, if a k-graph H on n vertices satisfies $\delta_\ell(H) \geq (\max\{1/2, c^*_{k,\ell}\} + o(1))\binom{n-\ell}{k-\ell}$, then H contains a perfect matching.*

The conjecture has attracted a great deal of attention and so far has been verified for $\ell \geq 3k/8$ by Frankl and Kupavskii [6] and a handful of pairs (k, ℓ). Note that this conjecture is slightly weaker than our problem, as e.g. for certain values of ℓ, it *suffices* to show that $c^*_{k,\ell} \leq 1/2$, rather than determining the precise value of $c^*_{k,\ell}$ (and this is the reason that the known record on Conjecture 6 by [6] is slightly wider than that for the conjecture on $c^*_{k,\ell}$).

In fact under the assumption $\delta_\ell(H) \geq (c^*_{k,\ell} + o(1))\binom{n-\ell}{k-\ell}$, Chang, Ge, Han and Wang [3] recently proved that one can find a matching in H of size $n/k-1$ (see Theorem 11). However, such H may or may not have a perfect matching, and, prior to this work, it is not clear how to characterize these two types of k-graphs. To understand this, what is interesting to our problem is the divisibility constructions that achieve the bound $1/2$ in the above conjecture. Consider an n-vertex set V with a bipartition $X \dot\cup Y$, where X and Y have almost equal size subject to that $|Y|$ is odd. Now define a k-graph H_0 on V with the edge set consisting of all k-tuples that contain an even number of vertices in Y. It is not hard to see that $\delta_\ell(H) \approx \frac{1}{2}\binom{n-\ell}{k-\ell}$ and H_0 has no perfect matching. To see this, note that any matching in H_0 covers an even number of vertices in Y, so not the entire Y.

One can actually construct such partitions for an arbitrary number of parts. For certain sizes of parts, divisibility conditions similar to the parity issue in the above example prevent the existence of perfect matchings. Thus, our result and algorithm can be viewed as efficient detection of such constructions. Indeed, in the Keevash–Knox–Mycroft proof of Conjecture 2 for $\ell = k-1$, they designed efficient algorithms to exhibit a number of $(O(n^{k+1}))$ such partitions and tested the divisibility (solubility) for each of them. In contrast, we show that one can focus on one partition and prove a sufficient and necessary condition for the existence of a perfect matching solely on that partition. This will be made clear in Section 2.

1.2 Related work

The decision problem for perfect matchings in dense hypergraphs was first raised by Karpiński, Ruciński and Szymańska [15] for the case $\ell = k-1$, where they formulated the problem as $\mathbf{PM}(k,\delta)$ which is equivalent to $\mathbf{PM}(k,k-1,\delta)$ in this paper. They showed that $\mathbf{PM}(k,1/2-\varepsilon)$ is in P for some absolute $\varepsilon > 0$, thus showing that $1/2$ is not the turning point for the change of behavior, while Szymańska's [22] reduction showed that $\mathbf{PM}(k,\delta)$ is NP-complete when $\delta < 1/k$. This leaves a hardness gap for $\delta \in [1/k, 1/2)$. Significant progress was made by Keevash–Knox–Mycroft [17, 18] who showed that $\mathbf{PM}(k,\delta)$ is in P for $\delta > 1/k$. This hardness problem was fully settled by Han [11] who proved that $\mathbf{PM}(k,1/k)$ is in P. Very recently, this result was strengthened by Han and Keevash [12], who showed that the minimum $(k-1)$-degree condition can be weakened to $n/k - c$ for any constant $c > 0$ and their algorithm can actually output the perfect matching provided that one exists.

The similar decision problem for Hamilton cycles (spanning cycles) has also been studied. First, it is well-known that it is NP-complete to determine if a (2-)graph has a Hamilton cycle. A k-graph C is called a tight cycle if its vertices can be listed in a cyclic order so that the edges are all consecutive k-tuples. For tight Hamilton cycles in dense k-graphs, it is shown that there is no such hardness gap as for perfect matchings. That is, it is showed that by Rödl, Ruciński and Szemerédi [21] for n-vertex k-graph H, if $\delta_{k-1}(H) \geq (1/2+o(1))n$ then H contains a tight Hamilton cycle, i.e., the decision problem is trivially in P; on the other hand, Garbe and Mycroft [7] showed that there exists a constant C such that if $\delta_{k-1}(H) \geq n/2 - C$, then the decision problem of tight Hamilton cycles is NP-complete. However, such hardness gap is shown to exist for looser cycles [7].

Han and Treglown [13] considered the similar decision problem for F-factors[2] in graphs and k-graphs. In particular, they determined the turning point for the F-factor problem for graphs and thus disproved a conjecture of Yuster [23].

2 A partition lemma and a structural theorem

To prove Theorem 3, we shall establish a structural theorem (Theorem 9) for perfect matchings, namely, we exhibit a sufficient and necessary condition for the existence of perfect matchings, which, in addition, can be checked in polynomial time. In turn, the heart of the proof of the structural theorem is the lattice-based absorption method developed by Han [11], which features a vertex partition of the given k-graph (Lemma 8).

The first key definition is the *reachability* introduced by Lo and Markström [20] for the absorption property we need for building the perfect matching.

2.1 Reachability

Let H be an n-vertex k-graph. For $i \in \mathbb{N}$ and $\beta \in (0,1)$, we say that two vertices u and v in $V(H)$ are (β,i)-*reachable in* H if there are at least βn^{ik-1} $(ik-1)$-sets S such that both $H[S\cup\{u\}]$ and $H[S\cup\{v\}]$ have perfect matchings. We refer to such a set S as a *reachable* $(ik-1)$-*set for* u *and* v. We say a vertex set $U \subseteq V(H)$ is (β,i)-*closed in* H if any two vertices $u,v \in U$ are (β,i)-reachable in H. Given any $v \in V(H)$, define $\tilde{N}_{\beta,i}(v,H)$ to be the set of vertices in $V(H)$ that are (β,i)-reachable to v in H.

[2] Given k-graphs F and H, an F-factor in H is a set of vertex-disjoint copies of F whose union covers $V(H)$.

2.2 Index vector and robust vector

Given an n-vertex k-graph H and integer $r \geq 0$, let $\mathcal{P} = \{V_0, V_1, \dots, V_r\}$ be a partition of $V(H)$ into disjoint vertex sets, namely, $\dot{\bigcup}_{0\leq i\leq r} V_i = V(H)$. In this paper, every partition has an implicit ordering of its parts.

Next we introduce the index vectors and edge-lattices. Given a k-graph H and a partition $\mathcal{P} = \{V_0, V_1, \dots, V_s, V_{s+1}, \dots, V_r\}$ of $V(H)$, the *index vector* $\mathbf{i}_{\mathcal{P}}(e) \in \mathbb{Z}^r$ of an edge $e \in E(H)$ with respect to $\mathcal{P}$ is the vector whose coordinates are the sizes of the intersections of e with each part of $\mathcal{P}$ *except* V_0, namely, $\mathbf{i}_{\mathcal{P}}(e)|_i = |e \cap V_i|$ for $i \in [r]$, where $\mathbf{v}|_i$ is defined as the ith digit of $\mathbf{v}$. For any $\mathbf{v} = \{v_1, \dots, v_r\} \in \mathbb{Z}^r$, let $|\mathbf{v}| := \sum_{i=1}^r v_i$. Here we say that $\mathbf{v} \in \mathbb{Z}^r$ is a *k-vector* if it has non-negative coordinates and $|\mathbf{v}| = k$. In previous works, for $\mu > 0$, the set of μ-robust vectors (denoted by $I^{\mu}_{\mathcal{P}}(H)$) is defined as the vectors $\mathbf{i} \in \mathbb{Z}^r$ such that H contains at least μn^k edges whose index vectors are equal to $\mathbf{i}$. In this paper we need a more detailed description of robust vectors – where we need to distinguish the roles of two different groups of V_i.

▶ **Definition 7** (μ-robust vectors). *Let $\mathcal{P} = \{V_0, V_1, \dots, V_s, V_{s+1}, \dots, V_r\}$ be a partition of $V(H)$. Given $\mu > 0$, define $I^{\mu}_{\mathcal{P}}(H) := I^{\mu}_{\mathcal{P},1}(H) \cup I^{\mu}_{\mathcal{P},2}(H)$ as the union of the following two sets:*

1. *the set $I^{\mu}_{\mathcal{P},1}(H)$ consists of all k-vectors $\mathbf{i} \in \mathbb{Z}^r$ such that $\mathbf{i}|_i = 0$ for $i \in \{0, 1, \dots, s\}$ and H contains at least μn^k edges e with $\mathbf{i}_{\mathcal{P}}(e) = \mathbf{i}$;*
2. *the set $I^{\mu}_{\mathcal{P},2}(H)$ consists of all k-vectors $\mathbf{i} \in \mathbb{Z}^r$ such that $\mathbf{i}|_i = 1$ for some $i \in [s]$, $\mathbf{i}|_j = 0$ for $j \in \{0, 1, \dots, s\} \setminus \{i\}$ and every vertex $v \in V_i$ is in at least μn^{k-1} edges e with $\mathbf{i}_{\mathcal{P}}(e) = \mathbf{i}$.*

The new ingredient of this definition is the assumption (2), which helps us to classify the vertices which do not enjoy the reachability information.

Now we are ready to state our partition lemma, which outputs a refined partition compared to the partition lemmas in [11, 13]. Throughout the paper, we write $\alpha \ll \beta \ll \gamma$ to mean that it is possible to choose the positive constants α, β, γ from right to left. More precisely, there are increasing functions f and g such that, given γ, whenever we choose some $\beta \leq f(\gamma)$ and $\alpha \leq g(\beta)$, the subsequent statement holds. Hierarchies of other lengths are defined analogously.

▶ **Lemma 8.** *Given integers $k \geq 3$, $C > 0$ and real $\delta > 0$, suppose we have $1/n_0 \ll \mu \ll \beta \ll \delta' \ll \delta, 1/k, 1/C$. Given an n-vertex k-graph H with $n \geq n_0$ and $\delta_\ell(H) \geq \delta\binom{n-\ell}{k-\ell}$, there is a partition $\mathcal{P}$ of $V(H)$ as*

$$\mathcal{P} = \{V_0, V_1, \dots, V_s, V_{s+1}, \dots, V_r\}$$

such that with $c := \lfloor 1/\delta \rfloor$

1. *$s \leq 2^{\binom{c+k-2}{k-1}}$ and $r - s \leq c$,*
2. *$|V_0| \leq k^{2^{\binom{c+k-2}{k-1}}} \left(k\binom{k+2^{\binom{c+k-2}{k-1}}+c-1}{k} + \binom{c+k-2}{k-1}C \right)$ and $|\bigcup_{0\leq i\leq s} V_i| \leq c\delta' n$,*
3. *for $1 \leq i \leq s$, $|V_i| \geq (k-1)|V_0| + k\binom{k+2^{\binom{c+k-2}{k-1}}+c-1}{k} + \binom{c+k-2}{k-1}C$,*
4. *for $1 \leq i \leq s$, there exists $\mathbf{i} \in I^{\mu}_{\mathcal{P},2}(H)$ such that $\mathbf{i}|_i = 1$,*
5. *for $s+1 \leq i \leq r$, $|V_i| \geq \delta' n/2$ and V_i is $(\beta, 2^c)$-closed in $H[\bigcup_{s+1\leq i\leq r} V_i]$.*

In particular, such a partition $\mathcal{P}$ of H can be found in time $O(n^{2^{c-1}k+1})$.

2.3 Lattices, solubility and the structural theorem

Keevash, Knox and Mycroft [18] introduced the following notions, which help us to transfer the divisibility problem to an algebraic setting as follows.

Given a partition $\mathcal{P}$ of m parts, denote by $L^{\mu}_{\mathcal{P}}(H)$ the *lattice* (additive subgroup) in $\mathbb{Z}^m$ generated by $I^{\mu}_{\mathcal{P}}(H)$. We write $L^m_{\max}$ for the lattice generated by all k-vectors, that is, $L^m_{\max} := \{\mathbf{v} \in \mathbb{Z}^m : k \text{ divides } |\mathbf{v}|\}$.

Suppose $L \subset L^{|\mathcal{P}|}_{\max}$ is a lattice in $\mathbb{Z}^{|\mathcal{P}|}$, where $\mathcal{P}$ is a partition of a set V. The *coset group* of $(\mathcal{P}, L)$ is $Q = Q(\mathcal{P}, L) := L^{|\mathcal{P}|}_{\max}/L$. For any $\mathbf{i} \in L^{|\mathcal{P}|}_{\max}$, the *residue* of $\mathbf{i}$ in Q is $R_Q(\mathbf{i}) := \mathbf{i} + L$. For any $A \subseteq V$ of size divisible by k, the *residue* of A in Q is $R_Q(A) := R_Q(\mathbf{i}_{\mathcal{P}}(A))$.

Let $q \in \mathbb{N}$. A (possibly empty) matching M in H of size at most q is a *q-solution* for $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ (in H) if $\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M)) \in L^{\mu}_{\mathcal{P}}(H)$; we say that $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ is *q-soluble* if it has a q-solution. We also need a strengthening of this definition as follows. Given a set $U \subset V(H)$, we define that $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ is *(U, q)-soluble* if there is a matching M in H such that M covers U and M is a $(|U| + q)$-solution.

In our proof, we shall pick a suitable $\mu > 0$ and let q be an upper bound of the order of the coset group $Q = L^{|\mathcal{P}|}_{\max}/L^{\mu}_{\mathcal{P}}(H)$ (then a trivial bound is $q = \binom{r+k-1}{k}$, the number of ee) and U be the part V_0. Then we show that H has a perfect matching if and only if $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ is (V_0, q)-soluble.

▶ **Theorem 9** (Structural Theorem). *Let $k, \ell, q \in \mathbb{N}$ where $\ell \le k-1$ and let $\gamma > 0$ be given. There exist $n_0, C := C(k, q) \in \mathbb{N}$ and $\beta, \mu > 0$ such that*

$$1/n_0 \ll \beta, \mu \ll \delta' \ll \gamma, c^*_{k,\ell}, 1/q, 1/C, 1/k. \tag{1}$$

Let H be an n-vertex k-graph with $\delta_\ell(H) \ge (c^*_{k,\ell} + \gamma)\binom{n-\ell}{k-\ell}$, where $n \ge n_0$ and k divides n. Suppose $\mathcal{P}$ is a partition of $V(H)$ satisfying Lemma 8 (1)-(5) with $\delta = c^*_{k,\ell}$. Moreover, suppose $|Q(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))| \le q$. Then H contains a perfect matching if and only if $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ is (V_0, q)-soluble.

3 Highlights of the proof: a comparison with the Han–Treglown proof

The basic idea for establishing the structural theorem is to distinguish the roles of *robust* and *non-robust* edges: to avoid the divisibility barriers, we may have to use edges with certain (combination of) index vectors. For some index vectors $\mathbf{v}$ there are many edges e with $\mathbf{i}_{\mathcal{P}}(e) = \mathbf{v}$, namely, there are many "replacements" even when we are forbidden from using, say, a small number of such edges. For other index vectors $\mathbf{v}$ there are few edges e with $\mathbf{i}_{\mathcal{P}}(e) = \mathbf{v}$, so we have to be careful when using such edges. In fact, the algebraic setting allows us to show that one can restrict the attention to only a *constant* number of such non-robust edges (using the lattice and coset group arguments), and thus this can be tested by brute force. Then the rest of the proof follows from the lattice-based absorption argument. Roughly speaking, it reserves a small matching which can be used to turn an almost perfect matching to a perfect matching given certain divisibility condition on the leftover vertices.

In [13] Han and Treglown proved our Theorem 3 under the additional assumption that $\delta > 1/3$, which gives a resolution of Conjecture 2 for $0.5k \le \ell \le 0.59k$. Embarrassingly, this does not solve the conjecture for $\ell = k-2$, which might be considered as the easiest case after the resolution of the case $\ell = k-1$. Below we shall first outline the proof in [13], and then explain our innovation compared with their approach and how such an improvement is achieved.

The partition lemma used in [13] is Lemma 12 in this paper (which we use as a building block to establish our partition). The key problem is that when $\ell < k-1$, one can not apply Lemma 12 directly to the k-graph H, as in H there might be a set W of vertices v which are not reachable to many vertices, namely, $|\tilde{N}_{\beta,i}(v,H)|$ is small for any proper choice of $\beta > 0$ and $i \in \mathbb{N}$. However, it is straightforward to show that $|W|$ is small, and (after some work) we can apply Lemma 12 with $S = V(H) \setminus W$ and get a partition of $V(H) \setminus W$. Now we face the following challenge.

▶ **Problem.** *Suppose $|W| = \Omega(n)$. How do we find a matching M covering W so that $H - V(M)$ has a perfect matching (or conclude that none exists)?*

The problem is trivial if $|W|$ is a constant, for which we can do brute force search for a matching M of constant size, which involves $O(n^{|W|})$ possibilities; otherwise it is hopeless without further assumptions.

Furthermore, it was not clear how to deal with the vertices of W by absorption, as $|W|$ might be smaller than the threshold for μ-robustness but still a small linear size, i.e., $\varepsilon n \le |W| < \mu n$, so that every vector touching W will not be recorded as a μ-robust vector. The proof in [13] avoided the "decision" part of the problem by assuming $\delta > 1/3$, so that when W is non-empty $V(H) \setminus W$ is closed, in which case H always contains a perfect matching (so any matching M covering W will work). Therefore, the problem is left open for $\delta < 1/3$ (i.e., for $\ell > (1 + \ln(2/3))k \approx 0.595k$).

We also note that the existence of W is not a problem in the existential results in the literature. For previous works on sufficient (minimum-degree-type) conditions for perfect matchings, those vertices can be put into a small matching of small linear size, whose removal does not affect much the minimum-degree conditions, guaranteeing that the absorption can proceed after the removal of this small matching.

Our new proof can be seen as a considerable refinement to the previous approach, where we strengthen our control on both the partition and the μ-robust vectors. As mentioned earlier, our new proof features a finer partition lemma (Lemma 8) than previous ones, where we classify vertices in W as well. More precisely, we first partition $S := V(H) \setminus W$, the set of vertices which are 1-reachable to $\Omega(n)$ other vertices, by Lemma 12 and denote the partition by $\mathcal{P}_1 = \{W_1, \dots, W_d\}$. Then we classify vertices of W according to their edge distributions in $\mathcal{P}_1$, that is, we obtain a partition of W by collecting vertices with common robust edge vectors together, so that the partition satisfies Definition 7 (2). Next we put the clusters that are too small (smaller than a certain constant) to a trash set V_0 in a recursive manner. This results a trash set V_0 of constant order, and because we have no control on V_0 at all, we will check how to match V_0 by brute force in time $O(n^{|V_0|})$. Now the good point is that all clusters survived from this greedy process have a good (though still constant) size (Lemma 8 (3)), which is enough (and crucial) for a (refined) absorption argument to work in later proofs. Since all the above procedures can be done in polynomial time, we get the desired polynomial-time algorithm for the decision problem $\mathbf{PM}(k, \ell, \delta)$.

4 Proof of Theorem 3

Now we prove Theorem 3. Recall that $c^*_{k,\ell} \ge c^*_{k,k-1} = 1/k$. Then $\lfloor 1/c^*_{k,\ell} \rfloor \le k$. Let $C := C(k, q)$ be given by Theorem 9 and let

$$q := \binom{k + 2^{\binom{2k-2}{k-1}} + k - 1}{k}.$$

Suppose we have constants satisfying the hierarchy (1).

Both Lemma 8 and Theorem 9 require that n is larger than a constant n_0, and by custom k-graphs with less than n_0 vertices can be tested by brute force. By Lemma 8, in time $O(n^{2^{k-1}k+1})$ we can find a partition $\mathcal{P}$ satisfying Lemma 8 (1)-(5). Because of Lemma 8 (1), we know $r \le k + 2^{\binom{2k-2}{k-1}}$ and obtain that $|Q(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))| \le \binom{r+k-1}{k} \le q$ (no matter what $L^{\mu}_{\mathcal{P}}(H)$ actually is). Then by Theorem 9, to determine if H contains a perfect matching it suffices to test if $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ is (V_0, q)-soluble. This can be done by testing whether any matching M of size at most $|V_0| + q$ covering V_0 is a solution of $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$, in time $O(n^{|V_0|+q})$. The overall time is polynomial in n because

$$q = \binom{k + 2^{\binom{2k-2}{k-1}} + k - 1}{k}$$

and

$$|V_0| \le k^{2^{\binom{2k-2}{k-1}}} \left(k \binom{2^{\binom{2k-2}{k-1}} + 2k - 1}{k} + \binom{2k-2}{k-1} C \right),$$

where we recall that $C := C(k, q)$ only depends on k.

Organization. The rest of this paper is organized as follows. Note that it remains to prove Lemma 8 and Theorem 9. We collect and prove a number of auxiliary results in Section 5, and give a proof of Lemma 8 in Section 6. In Section 7, we prove an absorbing lemma, which is an important component of the proof of Theorem 9. The proof of Theorem 9 is presented in Section 8.

5 Useful tools

In this section we collect together some results that will be used in our proof of Theorem 9. When considering ℓ-degree together with ℓ'-degree for some $\ell' \neq \ell$, the following proposition is very useful.

▶ **Proposition 10.** *Let $0 \le \ell \le \ell' < k$ and H be a k-graph. If $\delta_{\ell'}(H) \ge x\binom{n-\ell'}{k-\ell'}$ for some $0 \le x \le 1$, then $\delta_{\ell}(H) \ge x\binom{n-\ell}{k-\ell}$.*

Proof. Since $\ell \le \ell'$, we count $\delta_\ell(H)$ by

$$\begin{aligned}
\delta_\ell(H) &\ge \delta_{\ell'}(H) \binom{n-\ell}{\ell'-\ell} \frac{1}{\binom{k-\ell}{\ell'-\ell}} \\
&\ge x \binom{n-\ell'}{k-\ell'} \binom{n-\ell}{\ell'-\ell} \frac{1}{\binom{k-\ell}{\ell'-\ell}} \\
&\ge x \binom{n-\ell}{k-\ell} \binom{k-\ell}{\ell'-\ell} \frac{1}{\binom{k-\ell}{\ell'-\ell}} = x \binom{n-\ell}{k-\ell}
\end{aligned}$$

where the last inequality is from $\binom{a}{b}\binom{b}{c} = \binom{a}{c}\binom{a-c}{b-c}$. ◀

5.1 Almost perfect matchings

Let $k, \ell \in \mathbb{N}$ where $\ell \le k-1$. Given $D \in \mathbb{N}$, define $\delta(k, \ell, D)$ as the smallest number δ such that every k-graph H on $n \in k\mathbb{N}$ vertices with $\delta_\ell(H) \ge (\delta + o(1))\binom{n-\ell}{k-\ell}$ contains a matching covering all but at most D vertices. It is proved in [13] that $\delta(k, \ell, k) \le \max\{1/3, c^*_{k,\ell}\}$. We need the extra term $1/3$ removed, which was very recently proved by Chang, Ge, Han and Wang [3].

▶ **Theorem 11** ([3]). *Let k, ℓ be integers such that $1 \le \ell \le k-1$ and $\gamma > 0$, then there exists $n_0 \in \mathbb{N}$ such that the following holds for $n \ge n_0$. Suppose H is an n-vertex k-graph with $\delta_\ell(H) \ge (c^*_{k,\ell} + \gamma)\binom{n-\ell}{k-\ell}$, then H contains a matching M that covers all but at most $2k-\ell-1$ vertices. In particular, when $n \in k\mathbb{N}$, M is a perfect matching or covers all but exactly k vertices, namely, $\delta(k,\ell,k) \le c^*_{k,\ell}$.*

To build the partition, we need the following partition lemma from [13].

▶ **Lemma 12** ([13, Lemma 6.3]). *Given $\delta' > 0$, integers $c, k \ge 2$ and $0 < \alpha \ll 1/c, \delta'$, there exists a constant $\beta > 0$ such that the following holds for all sufficiently large n. Assume H is an n-vertex k-graph and $S \subseteq V(H)$ is such that $|\tilde{N}_{\alpha,1}(v,H) \cap S| \ge \delta' n$ for any $v \in S$. Further, suppose every set of $c+1$ vertices in S contains two vertices that are $(\alpha,1)$-reachable in H. Then in time $O(n^{2^{c-1}k+1})$ we can find a partition $\mathcal{P}$ of S into $V_1, \dots, V_r$ with $r \le \min\{c, 1/\delta'\}$ such that for any $i \in [r]$, $|V_i| \ge (\delta' - \alpha)n$ and V_i is $(\beta, 2^{c-1})$-closed in H.*

To deal with the vertices that are reachable to few other vertices, we collect them by the following greedy process. Note that a similar lemma was used in [4].

▶ **Lemma 13.** *Let integers $c, k \ge 2$ be given and suppose $1/n \ll \delta' \ll \alpha, 1/k, 1/c$. Assume that H is a k-graph on n vertices satisfying that every set of $c+1$ vertices contains two vertices that are $(2\alpha,1)$-reachable in H. Then in time $O(cn^{k+1})$ we can find a set of vertices $S \subseteq V(H)$ with $|S| \ge (1-c\delta')n$ such that $|\tilde{N}_{\alpha,1}(v, H[S])| \ge \delta' n$ for any $v \in S$.*

We remark that in the above lemma it is important to obtain the conclusion on $\tilde{N}_{\alpha,1}(v, H[S])$ rather than $\tilde{N}_{\alpha,1}(v) \cap S$. Indeed, in the latter one the reachable sets are still defined in H, so may contain vertices in $V(H) \setminus S$. This is not strong enough in our later proof (see Lemma 14 and its proof).

Proof. Let H be a k-graph on n vertices satisfying the condition of Lemma 13. We greedily identify vertices with few "reachable neighbors" and remove the vertex together with the vertices reachable to it from H. Set $V_0 := V(H)$. First, for every two vertices $u, v \in V(H)$, we determine if they are $(\alpha,1)$-reachable in H, which can be done by testing if any $(k-1)$-set is a reachable set in time $O(n^{k-1})$. Summing over all pairs of vertices, this step can be done in time $O(n^{k+1})$. Then we check if there is a vertex $v_0 \in V_0$ such that $|\tilde{N}_{\alpha,1}(v_0, H)| < \delta' n$ in time $O(n^2)$. If there exists such a vertex v_0, then let $A_0 := \{v_0\} \cup \tilde{N}_{\alpha,1}(v_0, H)$ and let $V_1 := V_0 \setminus A_0$. Next, we check V_1, that is, if there exists a vertex $v_1 \in V_1$ such that $|\tilde{N}_{\alpha,1}(v_1, H[V_1])| < \delta' n$, then let $A_1 := \{v_1\} \cup \tilde{N}_{\alpha,1}(v_1, H[V_1])$ and let $V_2 := V_1 \setminus A_1$ and repeat the procedure until no such v_j exists.

Suppose we stop and obtain a set of vertices $v_0, \dots, v_s$. We claim that $s < c$ and thus $|\bigcup_{0 \le i \le s} A_i| \le c\delta' n$. Indeed, otherwise consider $v_0, \dots, v_c$, the first $c+1$ of them and we shall show that every pair of them is not $(2\alpha,1)$-reachable in H, contradicting our assumption. Given $0 \le i < j \le c$, as $v_j \notin \tilde{N}_{\alpha,1}(v_i, H[V_i])$, v_i and v_j have less than αn^{k-1} 1-reachable sets in $H[V_i]$. Also, because $\delta' \ll \alpha, 1/c$, there are at most $c\delta' n \cdot n^{k-2} \le \alpha n^{k-1}$ 1-reachable sets in $H \setminus H[V_i]$. These two together yield that v_i and v_j are not $(2\alpha,1)$-reachable in H.

This greedy procedure needs to recompute $\tilde{N}_{\alpha,1}(v, H[V_i])$ at each time and can be done in time $O(cn^{k+1})$. Set $S := V(H) \setminus (\bigcup_{0 \le i \le s} A_i)$. We have $|S| \ge (1-c\delta')n$ and $|\tilde{N}_{\alpha,1}(v, H[S])| \ge \delta' n$ for every $v \in S$. ◀

6 Proof of Lemma 8

Let H be an n-vertex k-graph and define

$$1/n_0 \ll \mu \ll \beta \ll \alpha \ll \gamma, \delta' \ll \delta, 1/k, 1/C.$$

Assume $n \geq n_0$ and k divides n. Write $c := \lfloor 1/\delta \rfloor$, then by Proposition 10 we have

$$(c+1)\delta_1(H) > (c+1)\delta\binom{n-1}{k-1} > (1+\gamma)\binom{n-1}{k-1}.$$

Thus every set of $c+1$ vertices of $V(H)$ contains two vertices that are $(2\alpha, 1)$-reachable, as otherwise, by the inclusion-exclusion principle and $\alpha \ll \gamma, \delta$

$$n \geq (c+1)\delta_1(H) - \binom{c+1}{2} \cdot 2\alpha n^{k-1} \geq (1+\gamma)\binom{n-1}{k-1} - (c+1)^2\alpha n^{k-1} > n$$

a contradiction.

By Lemma 13, we find $S \subseteq V(H)$ with $|S| \geq (1-c\delta')n$ such that $|\tilde{N}_{\alpha,1}(v, H[S])| \geq \delta' n$ for any $v \in S$, in time $O(n^{k+1})$. Let $V' := V(H) \setminus S$ and thus $|V'| \leq c\delta' n$. Apply Lemma 12 to $H[S]$, and in time $O(n^{2^{c-1}k+1})$ we find a partition $\mathcal{P}_1$ of S into $W_1, \ldots, W_d$ with $d \leq c$ such that for $i \in [d]$, $|W_i| \geq (\delta' - \alpha)n$ and W_i is $(\beta, 2^{c-1})$-closed in $H[S]$.

Let I_d^{k-1} be the set of all $(k-1)$-vectors on $\mathcal{P}_1$ and note that $|I_d^{k-1}| = \binom{d+k-2}{d-1}$. Let $\mathcal{I}$ be the collection of all subsets of I_d^{k-1} and clearly $|\mathcal{I}| = 2^{|I_d^{k-1}|} = 2^{\binom{d+k-2}{d-1}}$. We classify the vertices in V' by the types of the edges in which they are contained. Indeed, for $I \in \mathcal{I}$, let V_I be the collection of vertices $v \in V'$ such that the following two properties hold:

- for every $\mathbf{i} \in I$, there are at least μn^{k-1} edges e of H such that $v \in e$ and $\mathbf{i}_{\mathcal{P}_1}(e \setminus \{v\}) = \mathbf{i}$;
- for every $\mathbf{i} \notin I$, there are fewer than μn^{k-1} edges e of H such that $v \in e$ and $\mathbf{i}_{\mathcal{P}_1}(e \setminus \{v\}) = \mathbf{i}$.

Clearly this defines a partition of V'. Moreover, note that $V_\emptyset = \emptyset$ – this is because any vertex in $V_\emptyset$ has vertex degree at most $2\binom{d+k-2}{d-1}\mu n^{k-1} < \delta_1(H)$, violating the minimum degree assumption. In particular, this implies 4. Note that this partition can be built by reading the edges for each $v \in V'$, so in time $O(n^k)$. Next we collect the parts that are too small and put them into a trash set V_0 in a recursive manner.

We first sort V_I, $I \in \mathcal{I}$ such that $|V_I|$ is increasing. Next, starting from $V_0 = \emptyset$, we *recursively* check in time $O(|\mathcal{I}|n)$ if next V_I, $I \in \mathcal{I}$ in the sequence satisfies that

$$|V_I| < (k-1)|V_0| + b, \text{ where } b := k\binom{k + 2^{\binom{c+k-2}{c-1}} + c - 1}{k} + \binom{c+k-2}{c-1}C$$

and if yes, put all vertices of V_I to V_0 (note here that V_0 is dynamic). Because $|\mathcal{I}| = 2^{\binom{d+k-2}{d-1}}$, straightforward computation shows that after the process we have

$$|V_0| \leq \frac{k^{|\mathcal{I}|}-1}{k-1}b \leq k^{2^{\binom{c+k-2}{c-1}}}\left(k\binom{k + 2^{\binom{c+k-2}{c-1}} + c - 1}{k} + \binom{c+k-2}{c-1}C\right).$$

At last, in constant time we remove the empty clusters and relabel the parts V_I's to $V_1, \ldots, V_s$, and relabel the parts of $\mathcal{P}_1$ as $V_{s+1}, \ldots, V_r$. The resulting partition satisfies all desired properties in the lemma and the overall running time is $O(n^{2^{c-1}k+1})$.

7 An absorption lemma

The following result guarantees our collection $\mathcal{E}_{abs}$ of absorbing sets in the proof of Theorem 9. The absorption method is by now a standard way to turn an almost spanning structure to a spanning one. Here we use a variant called *lattice-based absorption method*, developed by Han [11]. We remark that the following lemma is very similar to that [11, Lemma 3.4], and the only difference is because of our refined definition of robust vectors $I_{\mathcal{P}}^{\mu}(H)$.

▶ **Lemma 14** (Absorption Lemma). *Suppose $k \geq 3$, $\delta > 0$ and let $t := 2^{\lfloor 1/\delta \rfloor}$. Suppose that*

$$1/n \ll 1/c' \ll \beta, \mu \ll 1/t, 1/k.$$

Let H be an n-vertex k-graph with a partition $\mathcal{P}$ of $V(H)$ satisfying Lemma 8 (1)-(5), where $n \geq n_0$ and k divides n. Let $n_1 := |\bigcup_{s+1\leq i\leq r} V_i|$ (where r, s are from the statement of Lemma 8). Then there is a family $\mathcal{E}_{abs}$ consisting of at most $c' \log n_1$ disjoint tk^2-sets such that for each $A \in \mathcal{E}_{abs}$, $H[A]$ contains a perfect matching and every k-set $S \subseteq V(H)$ with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$ has at least $\sqrt{\log n_1}$ absorbing tk^2-sets in $\mathcal{E}_{abs}$.

Proof. Roughly speaking, in the proof we first exhibit a large number of absorbing sets for each k-set S with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$, and then show that the desired family $\mathcal{E}_{abs}$ can be obtained by standard probabilistic arguments. Our first task is to prove the following claim.

▷ Claim 15. Any k-set S with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$ has at least $\mu^{t+1}\beta^{k+1}n_1^{tk^2}$ absorbing tk^2-sets which consist of vertices in $\bigcup_{s+1\leq i\leq r} V_i$ only.

Proof. We split the proof into two cases regarding to $I^{\mu}_{\mathcal{P},1}(H)$ and $I^{\mu}_{\mathcal{P},2}(H)$. Note that all reachable sets will be constructed with vertices in $\bigcup_{s+1\leq i\leq r} V_i$ only.

▶ **Case 1.** *Suppose $\mathbf{i} \in I^{\mu}_{\mathcal{P},1}(H)$.*

For a k-set $S = \{y_1, \ldots, y_k\}$ with $\mathbf{i}_{\mathcal{P}}(S) = \mathbf{i}$, we construct absorbing tk^2-sets for S as follows. We first fix an edge $W = \{x_1, \ldots, x_k\}$ in H such that $\mathbf{i}_{\mathcal{P}}(W) = \mathbf{i}$ and $W \cap S = \emptyset$. Note that we have at least $\mu n^k - kn_1^{k-1} > \frac{\mu}{2}n^k$ choices for such an edge. Without loss of generality, we may assume that for all $i \in [k]$, x_i, y_i are in the same part V_j of $\mathcal{P}$ for $j > s$. Recall that by Lemma 8 (5) V_j is (β, t)-closed in $H[\bigcup_{s+1\leq i\leq r} V_i]$. Since x_i is (β, t)-reachable to y_i, there are at least βn_1^{tk-1} $(tk-1)$-sets T_i such that both $H[T_i \cup \{x_i\}]$ and $H[T_i \cup \{y_i\}]$ have perfect matchings. We pick disjoint reachable $(tk-1)$-sets for each x_i, y_i, $i \in [k]$ greedily, while avoiding the existing vertices. Since the number of existing vertices is at most $tk^2 + k$, we have at least $\frac{\beta}{2}n_1^{tk-1}$ choices for such $(tk-1)$-sets in each step. Note that $W \cup T_1 \cup \cdots \cup T_k$ is an absorbing set for S. First, it contains a perfect matching because each $T_i \cup \{x_i\}$ for $i \in [k]$ spans t vertex-disjoint edges. Second, $H[W \cup T_1 \cup \cdots \cup T_k \cup S]$ also contains a perfect matching and each $T_i \cup \{y_i\}$ for $i \in [k]$ spans t vertex-disjoint edges. There were at least $\frac{\mu}{2}n_1^k$ choices for W and at least $\frac{\beta}{2}n_1^{tk-1}$ choices for each T_i. Thus we find at least

$$\frac{\mu}{2}n^k \times \frac{\beta^k}{2^k}n_1^{tk^2-k} \times \frac{1}{(tk^2)!} \geq \mu\beta^{k+1}n_1^{tk^2}$$

absorbing tk^2-sets for S.

▶ **Case 2.** *Suppose $\mathbf{i} \in I^{\mu}_{\mathcal{P},2}(H)$.*

Suppose $S = \{v_1, y_2, \ldots, y_k\}$ with $\mathbf{i}_{\mathcal{P}}(S) = \mathbf{i}$ and $v_1 \in V_i$ for some $i \in [s]$. We construct absorbing tk^2-sets for S as follows. We fix an edge with vertex set $W = \{v_1, x_2, \ldots, x_k\}$ for $x_2, \ldots, x_k \in \bigcup_{s+1\leq j\leq r} V_j \setminus \{y_2, \ldots, y_k\}$ such that $\mathbf{i}_{\mathcal{P}}(W) = \mathbf{i}_{\mathcal{P}}(S) = \mathbf{i}$ and $W \cap S = \{v_1\}$. Note that by Lemma 8 (4) we have at least $\mu n^{k-1} - (k-1)n_1^{k-2} > \frac{\mu}{2}n^{k-1}$ choices for W (and $x_2, \ldots, x_k$ are in $\bigcup_{s+1\leq i\leq r} V_i$, by the definition of $I^{\mu}_{\mathcal{P},2}(H)$). Without loss of generality, we may assume that for all $i \in \{2, \ldots, k\}$, x_i, y_i are in the same part V_j of $\mathcal{P}$, $j > s$. Since x_i is (β, t)-reachable to y_i, there are at least βn_1^{tk-1} $(tk-1)$-sets T_i in $V(H) \setminus V_0$ such that both $H[T_i \cup \{x_i\}]$ and $H[T_i \cup \{y_i\}]$ have perfect matchings. We pick disjoint reachable $(tk-1)$-sets in $V(H) \setminus V_0$ for each x_i, y_i, $i \in \{2, \ldots, k\}$ greedily, while avoiding the existing vertices. Since the number of existing vertices is at most $tk(k-1) + (k-1)$, we have at least

$\frac{\beta}{2}n_1^{tk-1}$ choices for such $(tk-1)$-sets in each step. At last, let us pick a matching M of size t in H that is vertex disjoint from the existing vertices (the purpose is to let the absorbing set contain exactly tk^2 vertices). For the number of choices for $V(M)$, we can sequentially choose disjoint edges satisfying any μ-robust edge vector $\mathbf{i} \in I^{\mu}_{\mathcal{P},1}(H)$ and infer that there are at least $\frac{1}{2}\mu^t n^{tk}$ choices.

Note that each choice of $(W \setminus \{v_1\}) \cup T_2 \cup \cdots \cup T_k \cup V(M)$ is an absorbing set for S. First, it contains a perfect matching because each $T_i \cup \{x_i\}$ for $i \in \{2, \ldots, k\}$ spans t vertex-disjoint edges and M is a matching. Second, $H[W \cup T_1 \cup \cdots \cup T_k \cup S]$ also contains a perfect matching as each $T_i \cup \{y_i\}$ for $i \in \{2, \ldots, k\}$ spans t vertex-disjoint edges, W is an edge and M is a matching. There were at least $\frac{\mu}{2}n_1^{k-1}$ choices for W and at least $\frac{\beta}{2}n_1^{tk-1}$ choices for each T_i and $\frac{1}{2}\mu^t n^{tk}$ choices for $V(M)$. Thus we find at least

$$\frac{\mu}{2}n^k \times \left(\frac{\beta}{2}n_1^{tk-1}\right)^{k-1} \times \frac{1}{2}\mu^t n^{tk} \times \frac{1}{(tk^2)!} \geq \mu^{t+1}\beta^k n_1^{tk^2}$$

absorbing tk^2-sets for S, with vertices from $\bigcup_{s+1\leq i\leq r} V_i$ only. ◁

Continuing the proof of Lemma 14, we pick a family $\mathcal{E}$ of tk^2-sets by including every tk^2-subset of $\bigcup_{s+1\leq i\leq r} V_i$ with probability $p = c' n_1^{-tk^2} \log n_1$ independently, uniformly at random. Then the expected number of elements in $\mathcal{E}$ is $p\binom{n_1}{tk^2} \leq \frac{c'}{tk^2}\log n_1$ and the expected number of intersecting pairs of tk^2-sets is at most

$$p^2\binom{n_1}{tk^2} \times tk^2 \times \binom{n_1}{tk^2-1} \leq \frac{c'^2(\log n_1)^2}{n_1} = o(1).$$

Then by Markov's inequality, with probability at least $1 - 1/(tk^2) - o(1)$, $\mathcal{E}$ contains at most $c' \log n_1$ sets and they are pairwise vertex disjoint.

For every k-set S with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$, let X_S be the number of absorbing sets for S in $\mathcal{E}$. Then by Claim 15,

$$\mathbb{E}(X_S) \geq p\mu^{t+1}\beta^{k+1}n_1^{tk^2} = \mu^{t+1}\beta^{k+1}c' \log n_1.$$

By Chernoff's bound,

$$\mathbb{P}\left(X_S \leq \frac{1}{2}\mathbb{E}(X_S)\right) \leq \exp\left\{-\frac{1}{8}\mathbb{E}(X_S)\right\} \leq \exp\left\{-\frac{\mu^{t+1}\beta^{k+1}c' \log n_1}{8}\right\} = o(n^{-k}),$$

since $1/c' \ll \beta, \mu \ll 1/m$. Thus, with probability $1 - o(1)$, for each k-set S with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$, there are at least

$$\frac{1}{2}\mathbb{E}(X_S) \geq \frac{\mu^{t+1}\beta^{k+1}c' \log n_1}{2} > \sqrt{\log n_1}$$

absorbing sets for S in $\mathcal{E}$. We obtain $\mathcal{E}_{abs}$ by deleting the elements of $\mathcal{E}$ that are not absorbing sets for any k-set S and thus $|\mathcal{E}_{abs}| \leq |\mathcal{E}| \leq c' \log n_1$. ◀

8 Proof of Theorem 9

Now we are ready to prove Theorem 9. Let H be an n-vertex k-graph, and let $\mathcal{P}$ be a partition given by Lemma 8 satisfying (1)-(5). We first prove the forward implication.

8.1 Proof of the forward implication of Theorem 9

If H contains a perfect matching M, then $\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M)) = \mathbf{0} \in L^{\mu}_{\mathcal{P}}(H)$. Let M' be the smallest submatching of M that covers V_0, so $|M'| \leq |V_0|$. We shall show that there exists a matching $M'' \subset (M \setminus M')$ such that $|M''| \leq q$ and $\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M' \cup M'')) \in L^{\mu}_{\mathcal{P}}(H)$, implying that $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ is (V_0, q)-soluble.

Indeed, suppose that $M'' \subset (M \setminus M')$ is a smallest matching such that $\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M' \cup M'')) \in L^{\mu}_{\mathcal{P}}(H)$ and $|M''| = m \geq q$. Let $M'' = \{e_1, \ldots, e_m\}$ and consider the $m+1$ partial sums

$$\sum_{i=1}^{j} \mathbf{i}_{\mathcal{P}}(e_i) + L^{\mu}_{\mathcal{P}}(H),$$

for $j = 0, 1, \ldots, m$. Since $|Q(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))| \leq q \leq m$, two of the sums must be in the same coset. That is, there exist $0 \leq j_1 < j_2 \leq m$ such that

$$\sum_{i=j_1+1}^{j_2} \mathbf{i}_{\mathcal{P}}(e_i) \in L^{\mu}_{\mathcal{P}}(H).$$

So the matching $M^* := M' \setminus \{e_{j_1+1}, \ldots, e_{j_2}\}$ satisfies that $\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M^* \cup M')) \in L^{\mu}_{\mathcal{P}}(H)$ and $|M^*| < |M''|$, a contradiction.

8.2 Proof of the backward implication of Theorem 9

We first introduce the following useful constant. Given a set I of k-vectors in $\mathbb{Z}^r$, and $m \in \mathbb{N}$, consider the set J of all m'-vectors that are in the lattice in $\mathbb{Z}^r$ generated by I with $0 \leq m' \leq m$. That is, for any $\mathbf{v} \in J$, there exist $a_{\mathbf{i}} \in \mathbb{Z}$, $\mathbf{i} \in I$ such that

$$\mathbf{v} = \sum_{\mathbf{i} \in I} a_{\mathbf{i}} \mathbf{i}$$

Then let $C^* := C^*(r, k, I, m)$ be the maximum of $|a_{\mathbf{i}}|$ over all such $\mathbf{v}$. Furthermore, let $C_{\max} := C_{\max}(k, m)$ be the maximum of $C^* = C^*(r, k, I, m)$ over all $r \leq r_0(k) := 2^{\binom{2k-1}{k-1}} + k$ and all families of k-vectors $I \subseteq \mathbb{Z}^r$.

Now we start the proof. Recall that $c^*_{k,\ell} \geq c^*_{k,k-1} = 1/k$. Then $\lfloor 1/c^*_{k,\ell} \rfloor \leq k$. Define constants

$$t := 2^k \quad \text{and} \quad C := C_{\max}(k, kq + k).$$

Define an additional constant $c' > 0$ so that

$$1/n_0 \ll 1/c' \ll \beta, \mu \ll \delta' \ll 1/k, 1/q, 1/C, 1/t.$$

Let $n \geq n_0$ be a multiple of k. Let H be as in the statement of the theorem and $\mathcal{P}$ be a partition of $V(H)$ satisfying Lemma 8 (1)-(5), where the C therein is as defined above. In particular, Property (5) and the choice of t imply that for $s+1 \leq i \leq r$, $|V_i| \geq \delta' n/2$ and V_i is (β, t)-closed in $H[\bigcup_{s+1 \leq i \leq r} V_i]$. Furthermore, assume that $(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ is (V_0, q)-soluble, that is, there is a matching M_1 of size at most $|V_0| + q$ such that M_1 covers V_0 and it is a $(|V_0| + q)$-solution, that is,

$$\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M_1)) \in L^{\mu}_{\mathcal{P}}(H).$$

Let $n_1 := |\bigcup_{s+1\le i\le r} V_i|$. We first apply Lemma 14 to H and get a family $\mathcal{E}_{abs}$ consisting of at most $c' \log n_1$ disjoint tk^2-sets such that every k-set S of vertices with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$ has at least $\sqrt{\log n_1}$ absorbing tk^2-sets in $\mathcal{E}_{abs}$.

Note that $V(M_1)$ may intersect $V(\mathcal{E}_{abs})$ in at most $(|V_0|+q)k$ absorbing sets of $\mathcal{E}_{abs}$. Let $\mathcal{E}_0$ be the subfamily of $\mathcal{E}_{abs}$ obtained from removing the tk^2-sets that intersect $V(M_1)$. Let M_0 be the perfect matching on $V(\mathcal{E}_0)$ that is the union of the perfect matchings on each member of $\mathcal{E}_0$. Note that every k-set S with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$ has at least $\sqrt{\log n_1} - (|V_0|+q)k$ absorbing sets in $\mathcal{E}_0$.

Next we want to "store" some disjoint edges for each k-vector in $I^{\mu}_{\mathcal{P}}(H)$ for later steps, and at the same time we also cover the rest vertices of $\bigcup_{1\le i\le s} V_i$ (recall that V_0 is covered by M_1). More precisely, set $C' := C^*(r,k,I^{\mu}_{\mathcal{P}}(H), kq+k) \le C$. Note that Lemma 8 (3) guarantees that for each $i \in [s]$, V_i has at least $\binom{2k-2}{k-1}C$ uncovered vertices. We construct a matching M_2 in $H \setminus V(M_0 \cup M_1)$ which consists of C' disjoint edges e with $\mathbf{i}_{\mathcal{P}}(e) = \mathbf{i}$ for every $\mathbf{i} \in I^{\mu}_{\mathcal{P}}(H)$ and also cover the rest vertices of $\bigcup_{1\le i\le s} V_i$ by μ-robust edges. So

$$|M_2| \le \binom{k+r-1}{k} C' + \left|\bigcup_{1\le i\le s} V_i\right|.$$

Note that the process is possible because H contains at least μn^k edges for each $\mathbf{i} \in I^{\mu}_{\mathcal{P},1}(H)$ and every vertex in $\bigcup_{1\le i\le s} V_i$ is in at least μn^{k-1} edges for $\mathbf{i} \in I^{\mu}_{\mathcal{P},2}(H)$ and

$$|V(M_0\cup M_1\cup M_2)| \le tk^2c' \log n_1 + (|V_0|+q)k + \left(\binom{k+r-1}{k} C' + \left|\bigcup_{1\le i\le s} V_i\right|\right) k < \mu n_1 < \mu n, \tag{2}$$

which allow us to choose desired edges in a greedy manner. Moreover, for every $i \in [s]$, the number of μ-robust index vectors $\mathbf{i}$ such that $\mathbf{i}|_i = 1$ is at most $\binom{2k-2}{k-1}$, and thus the process above needs at most $\binom{2k-2}{k-1}C$ uncovered vertices from V_i, which is okay by our construction[3].

Let $H' := H \setminus V(M_0 \cup M_1 \cup M_2)$ and $n' := |H'|$. So $n' \ge n - \mu n$ and by $\delta(k,\ell,k) \le c^*_{k,\ell}$ due to Theorem 11,

$$\delta_\ell(H') \ge \delta_\ell(H) - \mu n^{k-\ell} \ge (\delta(k,\ell,k) + \gamma/2)\binom{n'-\ell}{k-\ell}.$$

By the definition of $\delta(k,\ell,k)$, we have a matching M_3 in H covering all but at most k vertices. Let U be the set of vertices in H' uncovered by M_3. We are done if $U = \emptyset$. Otherwise because k divides n we have $|U| = k$.

We write $Q := Q(\mathcal{P}, L^{\mu}_{\mathcal{P}}(H))$ for brevity. Recall that $\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M_1)) \in L^{\mu}_{\mathcal{P}}(H)$. Note that by definition, the index vectors of all edges in M_2 are in $I^{\mu}_{\mathcal{P}}(H)$. So we have $\mathbf{i}_{\mathcal{P}}(V(H) \setminus V(M_1 \cup M_2)) \in L^{\mu}_{\mathcal{P}}(H)$, namely, $R_Q(V(H) \setminus V(M_1 \cup M_2)) = \mathbf{0} + L^{\mu}_{\mathcal{P}}(H)$. Thus,

$$\sum_{e\in M_0\cup M_3} R_Q(e) + R_Q(U) = \mathbf{0} + L^{\mu}_{\mathcal{P}}(H).$$

Suppose $R_Q(U) = \mathbf{v}_0 + L^{\mu}_{\mathcal{P}}(H)$ for some $\mathbf{v}_0 \in L^d_{\max}$; so

$$\sum_{e\in M_0\cup M_3} R_Q(e) = -\mathbf{v}_0 + L^{\mu}_{\mathcal{P}}(H).$$

[3] Remark. This is where we need Lemma 8 (3), the lower bound of $|V_i|$, $i \in [s]$.

We use the following claim proved in [13] (earlier versions appeared in [11, 18]). Its proof is via the coset arguments and is very similar to the one used in the proof of the forward implication.

▷ Claim 16 ([13, Claim 5.1]). There exist $e_1, \dots, e_p \in M_0 \cup M_3$ for some $p \le q-1$ such that

$$\sum_{i\in[p]} R_Q(e_i) = -\mathbf{v}_0 + L^{\mu}_{\mathcal{P}}(H). \tag{3}$$

That is, we have $\sum_{i\in[p]} \mathbf{i}_{\mathcal{P}}(e_i) + \mathbf{i}_{\mathcal{P}}(U) \in L^{\mu}_{\mathcal{P}}(H)$. Let $Y := \bigcup_{i\in[p]} e_i \cup U$ and thus $|Y| = pk + k \le qk + k$. We now complete the perfect matching by absorption. Since $\mathbf{i}_{\mathcal{P}}(Y) \in L^{\mu}_{\mathcal{P}}(H)$, we have the following equation

$$\mathbf{i}_{\mathcal{P}}(Y) = \sum_{\mathbf{v}\in I^{\mu}_{\mathcal{P}}(H)} a_{\mathbf{v}}\mathbf{v},$$

where $a_{\mathbf{v}} \in \mathbb{Z}$ for all $\mathbf{v} \in I^{\mu}_{\mathcal{P}}(H)$. Since $|Y| \le qk+k$, by the definition of C', we have $|a_{\mathbf{v}}| \le C'$ for all $\mathbf{v} \in I^{\mu}_{\mathcal{P}}(H)$. Noticing that $a_{\mathbf{v}}$ may be negative, we can assume $a_{\mathbf{v}} = b_{\mathbf{v}} - c_{\mathbf{v}}$ such that one of $b_{\mathbf{v}}, c_{\mathbf{v}}$ is $|a_{\mathbf{v}}|$ and the other is zero for all $\mathbf{v} \in I^{\mu}_{\mathcal{P}}(H)$. So we have

$$\sum_{\mathbf{v}\in I^{\mu}_{\mathcal{P}}(H)} c_{\mathbf{v}}\mathbf{v} + \mathbf{i}_{\mathcal{P}}(Y) = \sum_{\mathbf{v}\in I^{\mu}_{\mathcal{P}}(H)} b_{\mathbf{v}}\mathbf{v}.$$

This equation means that given a family $\mathcal{E} = \{W^{\mathbf{v}}_1, \dots, W^{\mathbf{v}}_{c_{\mathbf{v}}} : \mathbf{v} \in I^{\mu}_{\mathcal{P}}(H)\}$ of disjoint k-subsets of $V(H) \setminus Y$ such that $\mathbf{i}_{\mathcal{P}}(W^{\mathbf{v}}_i) = \mathbf{v}$ for all $i \in [c_{\mathbf{v}}]$, we can regard $V(\mathcal{E}) \cup Y$ as the union of disjoint k-sets $\{S^{\mathbf{v}}_1, \dots, S^{\mathbf{v}}_{b_{\mathbf{v}}} : \mathbf{v} \in I^{\mu}_{\mathcal{P}}(H)\}$ such that $\mathbf{i}_{\mathcal{P}}(S^{\mathbf{v}}_j) = \mathbf{v}$, $j \in [b_{\mathbf{v}}]$ for all $\mathbf{v} \in I^{\mu}_{\mathcal{P}}(H)$. Since $c_{\mathbf{v}} \le C'$ for all $\mathbf{v}$ and $V(M_2) \cap Y = \emptyset$, we can choose the family $\mathcal{E}$ as a subset of M_2. In summary, starting with the matching $M_0 \cup M_1 \cup M_2 \cup M_3$ leaving U uncovered, we delete the edges $e_1, \dots, e_p$ from $M_0 \cup M_3$ given by Claim 16 and then leave $Y = \bigcup_{i\in[p]} V(e_i) \cup U$ uncovered. Next we delete the family $\mathcal{E}$ of edges from M_2 and leave $V(\mathcal{E}) \cup Y$ uncovered. Finally, we regard $V(\mathcal{E}) \cup Y$ as the union of at most

$$|M_2| + qk + k \le \sqrt{\log n_1}/2$$

k-sets S each with $\mathbf{i}_{\mathcal{P}}(S) \in I^{\mu}_{\mathcal{P}}(H)$.

Note that by definition, Y may intersect at most $qk + k$ absorbing sets in $\mathcal{E}_0$, which cannot be used to absorb those sets we obtained above. Since each k-set S has at least $\sqrt{\log n_1} - (|V_0| + q)k > \sqrt{\log n_1}/2 + qk + k$ absorbing tk^2-sets in $\mathcal{E}_0$, we can greedily match each S with a distinct absorbing tk^2-set $E_S \in \mathcal{E}_0$ for S. Replacing the matching on $V(E_S)$ in M_0 by the perfect matching on $H[E_S \cup S]$ for each S gives a perfect matching in H.

References

1. Noga Alon, Peter Frankl, Hao Huang, Vojtech Rödl, Andrzej Ruciński, and Benny Sudakov. Large matchings in uniform hypergraphs and the conjecture of Erdős and Samuels. *J. Combin. Theory Ser. A*, 119(6):1200–1215, 2012. doi:10.1016/j.jcta.2012.02.004.
2. Arash Asadpour, Uriel Feige, and Amin Saber. Santa claus meets hypergraph matchings. *Approximation, randomization and combinatorial optimization, Lecture Notes in Comput. Sci., vol. 5171, Springer, Berlin*, pages 10–20, 2008.
3. Yulin Chang, Huifen Ge, Jie Han, and Guanghui Wang. Matching of given sizes in hypergraphs. *ArXiv eprint: 2106.16068*, 2021.
4. Laihao Ding, Jie Han, Shumin Sun, Guanghui Wang, and Wenling Zhou. F-factors in quasirandom hypergraphs. *J. London Math. Soc., to appear*, 2022.

5 Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

6 Peter Frankl and Andrey Kupavskii. The Erdős Matching Conjecture and concentration inequalities. *ArXiv e-prints*, 2018. arXiv:1806.08855.

7 Frederik Garbe and Richard Mycroft. Hamilton cycles in hypergraphs below the Dirac threshold. *J. Combin. Theory Ser. B*, 133:153–210, 2018. doi:10.1016/j.jctb.2018.04.010.

8 Stefan Glock, Daniela Kühn, Allan Lo, and Deryk Osthus. The existence of designs via iterative absorption: hypergraph F-designs for arbitrary F. *Mem. Amer. Math. Soc., accepted*, 2020.

9 Hiep Hàn, Yury Person, and Mathias Schacht. On perfect matchings in uniform hypergraphs with large minimum vertex degree. *SIAM J. Discrete Math*, 23:732–748, 2009.

10 Jie Han. Perfect matchings in hypergraphs and the Erdős matching conjecture. *SIAM J. Discrete Math.*, 30(3):1351–1357, 2016. doi:10.1137/16M1056079.

11 Jie Han. Decision problem for perfect matchings in dense k-uniform hypergraphs. *Trans. Amer. Math. Soc.*, 369(7):5197–5218, 2017.

12 Jie Han and Peter Keevash. Finding perfect matchings in dense hypergraphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2366–2377, 2020. doi:10.1137/1.9781611975994.145.

13 Jie Han and Andrew Treglown. The complexity of perfect matchings and packings in dense hypergraphs. *J. Combin. Theory Ser. B*, 141:72–104, 2020. doi:10.1016/j.jctb.2019.06.004.

14 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

15 Marek Karpiński, Andrzej Ruciński, and Edyta Szymańska. Computational complexity of the perfect matching problem in hypergraphs with subcritical density. *Internat. J. Found. Comput. Sci.*, 21(6):905–924, 2010. doi:10.1142/S0129054110007635.

16 Peter Keevash. The existence of designs. *preprint*, 2014.

17 Peter Keevash, Fiachra Knox, and Richard Mycroft. Polynomial-time perfect matchings in dense hypergraphs. *Proceedings of the 45th STOC (2013)*, 2013.

18 Peter Keevash, Fiachra Knox, and Richard Mycroft. Polynomial-time perfect matchings in dense hypergraphs. *Adv. Math.*, 269:265–334, 2015. doi:10.1016/j.aim.2014.10.009.

19 Daniela Kühn, Deryk Osthus, and Timothy Townsend. Fractional and integer matchings in uniform hypergraphs. *European J. Combin.*, 38:83–96, 2014. doi:10.1016/j.ejc.2013.11.006.

20 Allan Lo and Klas Markström. F-factors in hypergraphs via absorption. *Graphs Combin.*, 31(3):679–712, 2015. doi:10.1007/s00373-014-1410-8.

21 Vojtech Rödl, Andrzej Ruciński, and Endre Szemerédi. An approximate Dirac-type theorem for k-uniform hypergraphs. *Combinatorica*, 28(2):229–260, 2008.

22 Edyta Szymańska. The complexity of almost perfect matchings and other packing problems in uniform hypergraphs with high codegree. *European J. Combin.*, 34(3):632–646, 2013. doi:10.1016/j.ejc.2011.12.009.

23 Raphael Yuster. Combinatorial and computational aspects of graph packing and graph decomposition. *Computer Science Review*, 1(1):12–26, 2007. doi:10.1016/j.cosrev.2007.07.002.

Fully Functional Parameterized Suffix Trees in Compact Space

Arnab Ganguly
Dept. of Computer Science, University of Wisconsin, Whitewater, WI, USA

Rahul Shah
Dept. of Computer Science, Louisiana State University, Baton Rouge, LA, USA

Sharma V. Thankachan
Dept. of Computer Science, University of Central Florida, Orlando, FL, USA

Abstract

Two equal length strings are a parameterized match (p-match) iff there exists a one-to-one function that renames the symbols in one string to those in the other. The *Parameterized Suffix Tree* (PST) [Baker, STOC' 93] is a fundamental data structure that handles various string matching problems under this setting. The PST of a text $T[1,n]$ over an alphabet Σ of size σ takes $O(n \log n)$ bits of space. It can report any entry in (parameterized) (i) suffix array, (ii) inverse suffix array, and (iii) longest common prefix (LCP) array in $O(1)$ time. Given any pattern P as a query, a position i in T is an occurrence iff $T[i, i+|P|-1]$ and P are a p-match. The PST can count the number of occurrences of P in T in time $O(|P| \log \sigma)$ and then report each occurrence in time proportional to that of accessing a suffix array entry. An important question is, *can we obtain a compressed version of PST that takes space close to the text's size of $n \log \sigma$ bits and still support all three functionalities mentioned earlier*? In SODA' 17, Ganguly et al. answered this question partially by presenting an $O(n \log \sigma)$ bit index that can support (parameterized) suffix array and inverse suffix array operations in $O(\log n)$ time. However, the compression of the (parameterized) LCP array and the possibility of faster suffix array and inverse suffix array queries in compact space were left open. In this work, we obtain a compact representation of the (parameterized) LCP array. With this result, in conjunction with three new (parameterized) suffix array representations, we obtain the first set of PST representations in $o(n \log n)$ bits (when $\log \sigma = o(\log n)$) as follows. Here $\varepsilon > 0$ is an arbitrarily small constant.

- Space $O(n \log \sigma)$ bits and query time $O(\log_\sigma^\varepsilon n)$;
- Space $O(n \log \sigma \log\log_\sigma n)$ bits and query time $O(\log\log_\sigma n)$; and
- Space $O(n \log \sigma \log_\sigma^\varepsilon n)$ bits and query time $O(1)$.

The first trade-off is an improvement over Ganguly et al.'s result, whereas our third trade-off matches the optimal time performance of Baker's PST while squeezing the space by a factor roughly $\log_\sigma n$. We highlight that our trade-offs match the space-and-time bounds of the best-known compressed text indexes for exact pattern matching and further improvement is highly unlikely.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Data Structures, Suffix Trees, String Algorithms, Compression

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.65

Category Track A: Algorithms, Complexity and Games

Funding Supported under US NSF grants CCF-1527435, CCF-2112643 and CCF-2137057.

1 Introduction

Text Indexing is a classical problem in Computer Science with numerous applications. The objective is to pre-process a text $T[1,n]$ over an alphabet Σ of size σ to create a data structure, such that for any pattern P given as a query, we can count/report all the positions in T where P appear as a substring. The suffix trees and suffix arrays (along with Longest Common

Prefix array, LCP array in short) are the most widely-known text indexes [21, 37, 28]. They occupy $\Theta(n)$ words of space (equivalently, $\Theta(n \log n)$ bits) and can count the number of occurrences in time $\tilde{O}(|P|)$. The time for per occurrence reporting is a constant for both structures. Although the space is linear in the number of words, there is an $O(\log_\sigma n)$ factor blowup when we consider the actual text size, which is $n\lceil \log \sigma \rceil$ bits. This factor is not negligible when $\sigma \ll n$. For example, the space occupied by the suffix tree of the human genome, even with very efficient implementation, such as in [27], requires about 40 GB of space, whereas the genome occupies less than 1GB.

To address the above issue, Grossi and Vitter [20] and Ferragina and Manzini [7] introduced succinct/compressed space alternatives, respectively known as *Compressed Suffix Array* (CSA) and *FM Index*. They can answer counting queries in $\tilde{O}(|P|)$ time and reporting in $\tilde{O}(1)$ time per occurrence. In some sense, both structures exploit the so-called *rank-preserving property* of suffixes/leaves. Specifically, consider two leaves/suffixes in the sub-tree of a non-root node in the *classical* suffix tree. If one were to chop off the first character of the suffixes corresponding to these leaves, thus leading to two different suffixes, the relative ordering of the first two suffixes within the suffix tree would be the same as their chopped counterparts. This crucial property leads to an efficient implementation of *Last-to-Front (LF) mapping*, which is defined as follows: *given the leaf i corresponding to a suffix starting at position t in the text,* $\mathsf{LF}(i)$ *is the leaf corresponding to the suffix starting at position* $(t-1)$. The LF mapping (or its inverse Ψ function) plays a pivotal role in the working of FM-Index and CSA and their subsequent improvements. Later, Sadakane [35] showed that by storing $O(n)$ extra bits, we could also compute the LCP of any two suffixes in $\tilde{O}(1)$ time, leading to the first fully functional suffix tree representation in $O(n \log \sigma)$ bits – i.e., it can report suffix array, inverse suffix array and LCP values. See [30] for further reading.

For numerous variants of the text indexing problem [1, 3, 18, 23, 32, 36] (such as parameterized matching, order-preserving matching, two-dimensional matching, cartesian tree matching, etc.), although linear space indexes are known, designing succinct/compressed indexes has been challenging [4, 5, 10, 13, 15, 11, 17, 12, 14, 25, 26, 24, 33]. We focus on the parameterized matching problem [1] defined as follows: two equal-length strings X and Y are a parameterized match (p-match) if and only if there exists a one-to-one function $f : \Sigma \to \Sigma$ such that $Y[i] = f(X[i])$ for every $i \in [1, |Y|]$. For example, $xyxz$ and $yzyx$ are p-match, but $xyxz$ and $xywz$ are not p-match. The indexing version is to count/report all substrings of $T[1, n]$ that p-match with a query pattern P. An index of size $\Theta(n \log n)$ bits, namely *parameterized suffix tree* (PST), has been known due to Baker [1]. However, the problem of designing a space-efficient avatar of PST turns out to be challenging because the above described *rank-preserving property* is no longer valid here. To that end, Ganguly et al. [16] proposed the Parameterized Burrows-Wheeler Transform (pBWT) that can support (parameterized) LF mapping in $O(\log \sigma)$ time using space close to $n \log \sigma$ bits. This led to the first sub-linear space index $\big($when $\log \sigma = o(\log n)\big)$ that can support (parameterized) suffix array and inverse suffix array operations in $\tilde{O}(1)$ time. Although this index is a significant achievement, it does not support LCP queries. In this paper, we augment this missing functionality, leading to the first fully functional PST representation in compact space. Besides this, we present three new space-time trade-offs (for suffix array access and its inverse operation) that are clear improvements over the previous results.

1.1 Baker's Parameterized Suffix Tree

We will use the following terminologies: for a string S, $|S|$ is its length, $S[i], 1 \le i \le |S|$, is its ith character and $S[i, j] = S[i] \circ S[i + 1] \circ \cdots \circ S[j]$, where $\circ$ denotes *concatenation*. If $i > j$, $S[i, j]$ denotes an empty string. Also, S_i denotes the circular suffix starting at position i. Specifically, S_i is S if $i = 1$ and is $S[i, |S|] \circ S[1, i - 1]$ otherwise.

Baker [1] introduced the following encoding scheme for matching strings over Σ. Let \$ be a special character in Σ. A string S is encoded into a string $\mathsf{prev}(S)$ of length $|S|$ by replacing the first occurrence of every character (other than \$) in S by 0 and any other occurrence by the difference in text position from its previous occurrence. Specifically, for any $i \in [1, |S|]$, $\mathsf{prev}(S)[i] = S[i]$ if $S[i] = \$$; otherwise, $\mathsf{prev}(S)[i] = (i - j)$, where $j < i$ is the last occurrence of $S[i]$ before i. If j does not exist, then $\mathsf{prev}(S)[i] = 0$. For example, $\mathsf{prev}(xy\$x) = 00\$3$. Note that $\mathsf{prev}(S)$ is a string over $\Sigma' = \{\$, 0, 1, \ldots, |S| - 1\}$, and can be computed in time $O(|S| \log \sigma)$.

▶ **Convention 1.** *In Σ', the integer characters are lexicographically smaller than \$. An integer character i comes before another integer character j iff $i < j$.*

▶ **Fact 2** ([1])**.** *Two (equal length) strings S and S' are a p-match iff $\mathsf{prev}(S) = \mathsf{prev}(S')$. Also a string P and a prefix of S are a p-match iff $\mathsf{prev}(P)$ is a prefix of $\mathsf{prev}(S)$.*

The parameterized Suffix Tree (PST) of $T[1, n]$ is a compacted trie of all strings in $\mathcal{P} = \{\mathsf{prev}(T[k, n]) \mid 1 \leq k \leq n\}$. For convenience, we assume that $T[n] = \$$ and $T[i] \neq \$$ for all $i \neq n$. Each edge is labeled with a string over Σ'. We use $\mathsf{str}(u)$ to denote the concatenation of edge labels on the path from the root to node u and $\mathsf{strLen}(u) = |\mathsf{str}(u)|$. Clearly, PST consists of n leaves (one per each encoded suffix) and at most $n - 1$ internal nodes. We use ℓ_i to denote the ith leftmost leaf and $\mathsf{str}(\ell_i)$ to denote the ith lexicographically smallest string in $\mathcal{P}$. Also, $\mathsf{PSA}[1, n]$ is an associated array called the parameterized suffix array, where $\mathsf{PSA}[i] = j$ and $\mathsf{PSA}^{-1}[j] = i$ iff $\mathsf{prev}(T[j, n]) = \mathsf{str}(\ell_i)$. Let $\mathsf{plcp}(i, j)$ be $\mathsf{strLen}(u)$, where u is the lowest common ancestor (LCA) of ℓ_i and ℓ_j; equivalently the length of the LCP of $\mathsf{prev}(T_{\mathsf{PSA}[i]})$ and $\mathsf{prev}(T_{\mathsf{PSA}[j]})$. The parameterized LCP array $\mathsf{PLCP}[1, n)$ is defined as follows: $\mathsf{PLCP}[i] = \mathsf{plcp}(i, i + 1)$. See Figure 1 for an illustration. Since $\mathsf{plcp}(i, j)$ is the smallest element in $\mathsf{PLCP}[i, j - 1]$, by maintaining an $O(n)$-bit range minimum query data structure [8] over PLCP, we can compute $\mathsf{plcp}(i, j)$ for any i, j in $O(1)$ time.

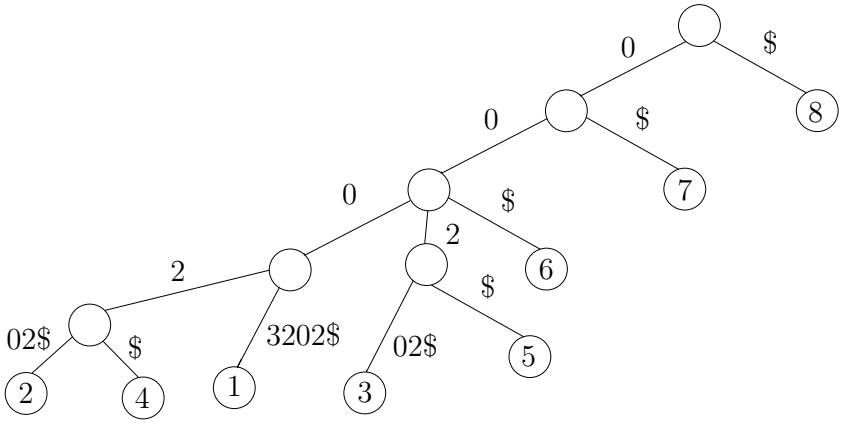
To answer a pattern matching query P (which is a string over $\Sigma - \{\$\}$), traverse the PST from the root and find the highest node u_P (if it exists) such that $\mathsf{str}(u_P)$ is prefixed by $\mathsf{prev}(P)$. This step takes $O(|P| \log \sigma)$ time. Then, find the range $[sp, ep]$ of leaves (called the suffix range of P) under u_P (this can be found in constant time by pre-processing the tree). Output $ep - sp + 1$ as the answer to counting and output $\{\mathsf{PSA}[i] \mid sp \leq i \leq ep\}$ as the answer to reporting. If u_P does not exist, we conclude that P does not have any p-match within T.

We refer to [6, 9, 29] for several other (linear space) data structures for parameterized pattern matching.

1.2 Compact Encoding of Parameterized Suffix Array

The parameterized LF mapping is defined as $\mathsf{PLF}(i) = \mathsf{PSA}^{-1}[\mathsf{PSA}[i] - 1]$. In [16], Ganguly et al. showed that one can implement PLF in $O(\log \sigma)$ time using an $n \log \sigma + o(n \log \sigma) + O(n)$ bit index. Their index constitutes the parameterized Burrows-Wheeler Transform (PWBT), which is an array of length n, such that $\mathsf{PBWT}[i]$ stores the number of distinct characters in (the prefix of) $T_{\mathsf{PSA}[i]}$ until the first occurrence $T[\mathsf{PSA}[i] - 1]$. See Figure 1 for an illustration.

By maintaining a Wavelet Tree [19] over PBWT, coupled with a succinct encoding [31] of the structure of the PST, they showed that PSA can be represented in $n \log \sigma + O\big(n + (n/\Delta) \log n\big)$ bits to support $\mathsf{PSA}[\cdot]/\mathsf{PSA}^{-1}[\cdot]$ queries in $t_{\mathsf{PSA}} = O(\Delta \cdot \log \sigma)$ time for any



i	T_i	prev(T_i)	prev$(T_{\mathsf{PSA}[i]})$	$T_{\mathsf{PSA}[i]}$	PSA$[i]$	f$_i$	PBWT$[i]$	W$[i]$	PLF(i)	$\Psi(i)$
1	xyzxzwz$	0003202$	000202$5	yzxzwz$x	2	8	3	4	3	4
2	yzxzwz$x	000202$5	0002$504	xzwz$xyz	4	5	2	3	4	5
3	zxzwz$xy	00202$50	0003202$	xyzxzwz$	1	3	$	3	8	1
4	xzwz$xyz	0002$504	00202$50	zxzwz$xy	3	2	4	2	1	2
5	zwz$xyzx	002$0043	002$0043	zwz$xyzx	5	2	3	2	2	6
6	wz$xyzxz	00$00432	00$00432	wz$xyzxz	6	8	2	4	5	7
7	z$xyzxzw	0$004320	0$004320	z$xyzxzw	7	4	4	3	6	8
8	$xyzxzwz	$0003202	$0003202	$xyzxzwz	8	∅	3	$	7	3

Figure 1 The text is $T[1,8] = xyzxzwz\$$, where $\Sigma = \{w, x, y, z, \$\}$.

$\Delta = O(\log_\sigma n)$ fixed in advance. For example, $O(n \log \sigma)$ bits of space and $O(\log n)$ query time by fixing $\Delta = \log_\sigma n$. This is the first succinct/compact space representation of PSA.[1] However, it does not support $\mathsf{plcp}(\cdot, \cdot)$ queries.

1.2.1 Challenges in Making PLF Computation Faster

Note that the product of space (in bits) and query time of Ganguly et al.'s PSA is always $\Theta(n \log n \log \sigma)$. A natural question is: *can we obtain better trade-offs*?

The current index is limited primarily because its main component for computing parameterized LF mapping needs to support queries of the following type: $\mathsf{RangeCount}_{\mathsf{PBWT}}(i, j, x, y) = |\{k \mid k \in [i, j], \mathsf{PBWT}[k] \in [x, y]\}|$. From the 4-sided range counting lower bound [34], any $O(n \log^{O(1)} \sigma)$-bit data structure needs $\Omega(1 + \log \sigma / \log \log n)$ time. This time becomes a bottleneck when it comes to some of the advanced suffix sampling techniques that are used for speeding up (classical) suffix array queries using additional space (as listed in Theorem 3); in fact, to adapt these techniques, LF mapping needs to be implemented in $O(1)$ time. Therefore, to prove Theorem 3, we need a new set of techniques.

1.2.2 Challenges in Compressing Parameterized LCP Array

Sadakane's LCP compression framework [35] for traditional text indexing relies on the following: *if two suffixes begin with the same character, their LCP after chopping the first character will be one less than their original LCP, and these two suffixes will retain*

[1] A succinct index for a data of size Z bits is a data structure having $Z + o(z)$ bits. On the other hand, a compact index needs $O(Z)$ bits.

their relative lexicographic rank after chopping. This allows one to compactly encode the LCP information. Unfortunately, this is not true for parameterized strings. For e.g., let $X = wxywabcdwx\$$ and $Y = abcdwx\$$ be two suffixes of T, then their respective prev encodings are $\mathsf{prev}(X) = 0003000058\$$ and $\mathsf{prev}(Y) = 000000\$$; hence, their p-LCP is 3. After chopping the first characters, the respective prev encodings are $000000058\$$ and $00000\$$, resulting in a p-LCP value of 5. Thus, chopping the first character can increase LCP; in fact, it can also decrease or even remain the same! Moreover, the order of the suffix may switch (as seen in this example), which adds to the difficulty. In short, the previous techniques are not adequate for compressing parameterized LCP array.

1.3 Our Results: Fully Functional PST in Compact Space

The suffix range $[sp, ep]$ of a pattern P can be computed in $O(|P| \log \sigma)$ time using Ganguly et al.'s index [16]; so we focus on speeding up suffix array queries and reporting LCP. We overcome the $O(\log \sigma)$ bottleneck of parameterized LF mapping by using its inverse, the Ψ-function, defined as $\Psi(i) = j$ iff $\mathsf{PLF}(j) = i$. This allows us to remove the dependence on 4-sided range-queries, instead of using simpler $\mathsf{partialRank}$ and select queries, which can be supported in $O(1)$ time using succinct space. With this, we implement Ψ-function in $O(1)$ time and thereby obtain three trade-offs, with space-time product near $n \log \sigma$. For our LCP framework, we essentially reduce a parameterized LCP query to a traditional LCP query; this allows us to leverage Sadakane's framework [35].

In summary, we have the following theorem.

▶ **Theorem 3.** *For the parameterized suffix tree of a text $T[1, n]$ over an alphabet of size σ, the following space-time trade-offs are possible in the word RAM model of computation with word-size $\Omega(\log n)$, where $\varepsilon > 0$ is an arbitrarily small constant.*

Index Size (in bits)	***Query Time (t_{PSA})***
$O(n \log \sigma)$	$O(\log_\sigma^\varepsilon n)$
$O(n \log \sigma \log \log_\sigma n)$	$O(\log \log_\sigma n)$
$O(n \log \sigma \log_\sigma^\varepsilon n)$	$O(1)$

All three basic queries (i.e., $\mathsf{PSA}[\cdot], \mathsf{PSA}^{-1}[\cdot]$ and $\mathsf{plcp}(\cdot, \cdot)$) are supported in $O(t_{\mathsf{PSA}})$ time.

Note that Baker's original definition also includes "*static characters*" for which the match has to be done in the traditional way. For the simplicity of exposition, we assume that all characters in Σ, except \$ are parameterized characters. We remark that our index can be extended to incorporate static characters without any sacrifice in time or space.

Outline. We start in Section 2 with a weaker version of Theorem 3 without the LCP claims. Specifically, we show that using an $O(n \log \sigma)$ bit index, we can support PSA and PSA^{-1} queries in $O(\log_\sigma n)$ time. Note that this is already a factor $(\log \sigma)$ faster than what is achievable using Ganguly et al.'s index [16]. Using more intricate techniques, we obtain the $\mathsf{PSA}[\cdot]/\mathsf{PSA}^{-1}[\cdot]$ trade-offs in Sections 3 and 4. Finally, the technique for encoding the LCP array is in Section 5.

2 Our Framework: A Compact Space Index

Let's start with a few definitions that we are going to use throughout this paper.

▶ **Definition 4.** *Define the following (see Figure 1 for an illustration):*

Notation	***Definition***
$\Psi(i)$	$\mathsf{PSA}^{-1}[1]$ *if* $\mathsf{PSA}[i] = n$, *else* $\mathsf{PSA}^{-1}[\mathsf{PSA}[i]+1]$
$\mathsf{PLF}(i)$	$\mathsf{PSA}^{-1}[n]$ *if* $\mathsf{PSA}[i] = 1$, *else* $\mathsf{PSA}^{-1}[\mathsf{PSA}[i]-1]$
f_i	$\varnothing$ if $i = n$, *else the first occurrence of* $T[\mathsf{PSA}[i]]$ *in* $T_{1+\mathsf{PSA}[i]}$
$\mathsf{W}[i]$	\$ if $i = n$, *else number of zeroes in* $\mathsf{prev}(T_{1+\mathsf{PSA}[i]})[1, \mathsf{f}_i]$
$\mathsf{PBWT}[i]$	$\mathsf{W}[\mathsf{PLF}(i)]$

Our goal is to prove the following theorem in this section.

▶ **Theorem 5.** *By using an $O(n \log \sigma)$-bit index, we can compute $\Psi(i)$ in $O(1)$ time.*

Before we prove this, we will see how we can use it to achieve an $O(n \log \sigma)$-bit index that supports PSA and PSA^{-1} queries in $O(\log_\sigma n)$ time. We explicitly store $\mathsf{PSA}[i]$ iff it equals n or is a multiple of $\Delta = \lceil \log_\sigma n \rceil$. Additionally, we store a bit-vector $B[1, n]$ as follows: set $B[i] = 1$ iff $\mathsf{PSA}[i]$ has been explicitly stored. For reporting, a $\mathsf{PSA}[j]$ can be retrieved in $O(1)$ time if $B[j] = 1$. Otherwise, we repeatedly apply Ψ starting from j until we reach an index $j' = \Psi\big(\ldots \Psi\big(\Psi\big(j\big)\big) \ldots\big)$ such that $B[j'] = 1$ (i.e., $\mathsf{PSA}[j']$ is explicitly stored). If the Ψ operation was applied k times, we get $\mathsf{PSA}[j] = \mathsf{PSA}[j'] - k$. The time complexity is $O(k)$. For PSA^{-1} queries, we store $\mathsf{PSA}^{-1}[i]$ if i equals n or if i is a multiple of Δ. To compute $\mathsf{PSA}^{-1}[j]$, we first find the largest number $j' \leq j$, such that j' is a multiple of Δ. Compute $j'' = \mathsf{PSA}^{-1}[j']$ from the sampled-PSA^{-1} in $O(1)$ time. Let $k = j - j' < \Delta$. Starting from j'' carry out k successive Ψ operations and report the final index as $\mathsf{PSA}^{-1}[j]$ in $O(k)$ time. Finally, $k < \Delta = \lceil \log_\sigma n \rceil$. The (extra) space needed is $(n/\Delta) \log n = O(n \log \sigma)$ bits. Other trade-offs may be obtained by tuning Δ, which is what we will do using a more sophisticated sampling technique along with a modified version of Theorem 5; details are in Section 4.

2.1 Succinct Data-Structure Toolkit

▶ **Fact 6** ([31]). *A tree having m nodes can be stored in $2m + o(m)$ bits, such that if each node is labeled by its pre-order rank, the following operations can be supported in $O(1)$ time:*

- $\mathsf{pre\text{-}order}(u)/\mathsf{post\text{-}order}(u)$ = *pre-order/post-order rank of node u.*
- $\mathsf{parent}(u)$ = *parent of node u.*
- $\mathsf{nodeDepth}(u)$ = *number of edges on the path from the root to u.*
- $\mathsf{lca}(u, v)$ = *lowest common ancestor (LCA) of two nodes u and v.*
- $\mathsf{lmostLeaf}(u)/\mathsf{rmostLeaf}(u)$ = *leftmost/rightmost leaf in the subtree rooted at u.*
- $\mathsf{levelAncestor}(u, D)$ = *ancestor of u such that* $\mathsf{nodeDepth}(u) = D$.

Also, we can find the pre-order rank of the ith leftmost leaf in $O(1)$ time.

▶ **Fact 7** ([2]). *Given an array $A[1, t]$ over $\Sigma = \{1, 2, \ldots, \sigma\}$, by storing an $O(t \log \sigma)$-bit structure, we can support the following operation in $O(1 + \log \frac{\log \sigma}{\log t})$ time:*

$\mathsf{rank}_A(i, c) =$ *number of occurrences of c in $A[1, i]$*

Additionally, the following operations can be supported in $O(1)$ time:

- *access $A[i]$*
- $\mathsf{partialRank}_A(i) = \mathsf{rank}(i, A[i])$, *i.e., the number of occurrences of $A[i]$ is the range $[1, i]$*
- $\mathsf{select}_A(i, c)$ = *the i^{th} occurrence of c in A*

2.2 Proof of Theorem 5

From their definitions, we observe: $\Psi(i) = j \iff \mathsf{PLF}(j) = i$, and $\mathsf{W}[i] = \mathsf{PBWT}[\Psi(i)]$. Additionally, we use $\chi(i)$ to denote the number of suffixes k such that $\Psi(k) \leq \Psi(i)$ and $\mathsf{W}[k] = \mathsf{W}[i]$. Based on these, it is easy to see that if $j = \Psi(i)$, then j is the $\chi(i)^{th}$ occurrence of $\mathsf{W}[i]$ in PBWT. Given $\chi(i)$, we can compute $\Psi(i)$ as:

$$\Psi(i) = \mathsf{select}_{\mathsf{PBWT}}\Big(\chi(i), \mathsf{W}[i]\Big)$$

Note that arrays $\mathsf{PBWT}[1, n]$ and $\mathsf{W}[1, n]$ take $O(\log \sigma)$ bits per entry. Therefore, we preprocess them into compact data structures that support access/select queries in $O(1)$ time (see Fact 7). Therefore, given $\chi(i)$, we can compute $\Psi(i)$ in constant time. To this end, we present the following lemma, which completes the proof of Theorem 5.

▶ **Lemma 8.** *By using an $O(n \log \sigma)$-bit data structure, we can compute $\chi(i)$ in $O(1)$ time.*

The rest of the section is dedicated to proving Lemma 8. For brevity, throughout we use "suffix i" to denote the suffix corresponding to leaf ℓ_i in the PST.

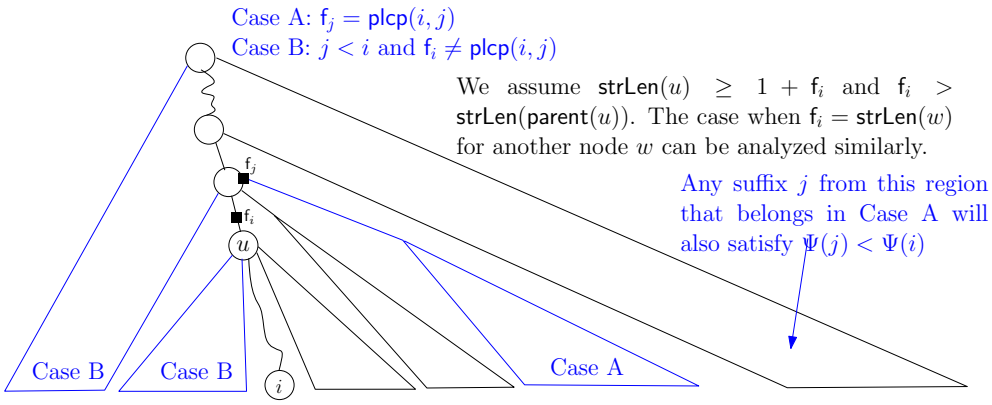
▶ **Lemma 9.** *Let $i < n$. Suppose u is the highest node on the path from the root to ℓ_i such that $\mathsf{f}_i \leq \mathsf{strLen}(u) - 1$. Then, for any leaf ℓ_j in the subtree of u, we have $\mathsf{f}_j = \mathsf{f}_i$*

Proof. Let $d = \mathsf{plcp}(i, j)$. Since $d \geq \mathsf{strLen}(u) \geq \mathsf{f}_i + 1$, we have $\mathsf{prev}\big(T_{\mathsf{PSA}[i]}[1, d]\big) = \mathsf{prev}\big(T_{\mathsf{PSA}[j]}[1, d]\big)$, i.e., the suffixes starting at $\mathsf{PSA}[i]$ and $\mathsf{PSA}[j]$ p-match until their first d characters. Clearly, the first occurrence of $T[\mathsf{PSA}[i]]$ in $T_{1+\mathsf{PSA}[i]}$ must be the same as the first occurrence of $T[\mathsf{PSA}[j]]$ in $T_{1+\mathsf{PSA}[j]}$, i.e., $\mathsf{f}_i = \mathsf{f}_j$. ◀

▶ **Lemma 10.** *If $\mathsf{W}[i] = \mathsf{W}[j]$, then $\Psi(j) < \Psi(i)$ iff*

- **Case A:** *either* $\mathsf{f}_j = \mathsf{plcp}(i, j)$
- **Case B:** *or,* $\mathsf{f}_j \neq \mathsf{plcp}(i, j)$, $j < i$, *and* $\mathsf{f}_i \neq \mathsf{plcp}(i, j)$

Proof. Recall Convention 1. Let $d = \mathsf{plcp}(i, j)$. If $\mathsf{f}_i = \varnothing$ or $\mathsf{f}_j = \varnothing$, then $\mathsf{W}[i] \neq \mathsf{W}[j]$. So, $\mathsf{f}_i, \mathsf{f}_j < n$. Also, $\mathsf{prev}(T_{\mathsf{PSA}[i]})[d+1] \neq \mathsf{prev}(T_{\mathsf{PSA}[j]})[d+1]$, by the definition of LCP. We now prove both cases (see Figure 2 for an illustration).



Blue suffixes are those suffixes j such that $\Psi(j) < \Psi(i)$

Figure 2 Illustration of Lemma 10.

- If $\mathsf{f}_j = d$, then $\mathsf{prev}(T_{\mathsf{PSA}[j]})[d+1] = d$ and $\mathsf{prev}(T_{\mathsf{PSA}[\Psi(j)]})[d] = 0$. From Lemma 9, we conclude $\mathsf{f}_i \geq d$. Moreover, $\mathsf{f}_i \neq d$ because $\mathsf{prev}(T_{\mathsf{PSA}[i]})[d+1] \neq d$ (by the definition of LCP). Therefore, $\mathsf{f}_i > d$. This implies $\mathsf{prev}(T_{\mathsf{PSA}[i]})[d+1] \neq 0$; otherwise, $\mathsf{W}[i] \neq \mathsf{W}[j]$, a contradiction. Consequently, either $\mathsf{prev}(T_{\mathsf{PSA}[\Psi(i)]})[d] > 0$ or $\mathsf{prev}(T_{\mathsf{PSA}[\Psi(i)]})[d] = \$$. Finally, note that $\mathsf{plcp}\big(\Psi(i), \Psi(j)\big) \geq d-1$. So after removing the first character of the two suffixes, their first $(d-1)$ characters will p-match. Hence, $\Psi(j) < \Psi(i)$ when $\mathsf{f}_j = \mathsf{plcp}(i,j)$.
- Now, assume $\mathsf{f}_j \neq d$. If $\mathsf{f}_j < d$, then $\mathsf{f}_i = \mathsf{f}_j$ (from Lemma 9) and $\mathsf{plcp}\big(\Psi(i), \Psi(j)\big) = d-1$. Then, $\Psi(j) < \Psi(i)$ iff $j < i$ because $\mathsf{prev}(T_{\mathsf{PSA}[\Psi(i)]})[d] = \mathsf{prev}(T_{\mathsf{PSA}[i]})[d+1]$ and $\mathsf{prev}(T_{\mathsf{PSA}[\Psi(j)]})[d] = \mathsf{prev}(T_{\mathsf{PSA}[j]})[d+1]$. If $\mathsf{f}_j > d$, then $\mathsf{prev}(T_{\mathsf{PSA}[j]})[d+1] = \mathsf{prev}(T_{\mathsf{PSA}[\Psi(j)]})[d]$. Also, $\mathsf{f}_i \geq d$ (from Lemma 9), implying either $\mathsf{prev}(T_{\mathsf{PSA}[\Psi(i)]})[d] = \mathsf{prev}(T_{\mathsf{PSA}[i]})[d+1]$ or $\mathsf{prev}(T_{\mathsf{PSA}[\Psi(i)]})[d] = 0$. The latter happens only when $\mathsf{f}_i = \mathsf{plcp}(i,j)$. Hence, we have $\Psi(j) < \Psi(i)$ when $j < i$ and $\mathsf{f}_i \neq \mathsf{plcp}(i,j)$.

This concludes the proof. ◀

To compute $\chi(i)$, we count the number n_A and n_B of Case A and B suffixes respectively; note that the cases are disjoint. Then, $\chi(i) = 1 + n_A + n_B$. We provide an overview first.

First, we locate the edge on which f_i lies, i.e., locate the edge $(\mathsf{parent}(u), u)$, such that $\mathsf{strLen}(\mathsf{parent}(u)) < 1 + \mathsf{f}_i \leq \mathsf{strLen}(u)$. This is facilitated by associating a bit with each node and set it to 1 iff $\mathsf{strLen}(\mathsf{parent}(\cdot)) < 1 + \mathsf{f}_j \leq \mathsf{strLen}(\cdot)$ for some suffix j in its sub-tree. Therefore, u is the lowest ancestor of ℓ_i that is associated with 1. By maintaining an $O(n)$ bit structure, we can answer this query in $O(1)$ time (see Lemma 11).

For counting n_A, we walk the path from root to $\mathsf{parent}(u)$, and for each node x on this path find out the number of suffixes j satisfying Case A: $\mathsf{f}_j = \mathsf{strLen}(x)$. Note that we afford to store f_j explicitly, instead store if f_j lies on an edge from node x to its child node y. Luckily, that's enough for us – for any suffix j, if $\mathsf{f}_j = \mathsf{strLen}(x)$ lies on the edge (x, y), then for all suffixes j' in the subtree of y, we have $f_{j'} = \mathsf{strLen}(x)$ (by Lemma 9). So, the count can be obtained via a simple unary encoding of suffixes of this kind. For counting Case B: $j < i$ and $\mathsf{f}_i \neq \mathsf{plcp}(i,j)$, walk from root to a node v; here, $v = u$ if $\mathsf{f}_i > \mathsf{strLen}(u)$, and $v = \mathsf{parent}(u)$ if $\mathsf{f}_i = \mathsf{strLen}(u)$. Initialize n_B to the number of leaves that lie to the left of this path. Then, add the number of leaves lying to the left of ℓ_i within the sub-tree of v to n_B. Note that for Cases A and B, we must consider only the suffixes j satisfying $\mathsf{W}[j] = \mathsf{W}[i]$; this is achieved by collecting suffixes based on their $\mathsf{W}[\cdot]$ values into different trees. Finally, we cannot afford to walk the path; therefore, we rely on the result in Lemma 12.

▶ **Lemma 11.** *Consider a compacted tree τ having L leaves, where each node is associated with a 0 or 1. By using an $O(L)$-bit data structure, given a query leaf ℓ, we can find the lowest ancestor v (if it exists) of ℓ associated with a 1 in $O(1)$ time.*

▶ **Lemma 12.** *Consider a compacted tree τ having L leaves, where each node w is associated with an integer $g(w) \geq 0$. For any node v, we have $\sum_{u \in \mathcal{S}_v} g(u) \leq L_v$, where $\mathcal{S}_v$ is the set of nodes in the subtree of v and L_v is the number of leaves in the subtree of v. By using an $O(L)$-bit data structure, given a query leaf ℓ, we can compute $G(\ell) = \sum_v g(v)$ in $O(1)$ time, where v is an ancestor of ℓ.*

The proofs of Lemmas 11 and 12 (deferred to Section 2.3) relies on mostly standard techniques from succinct data structures. Next we present the implementation details of $\chi(i)$ computation.

The Data Structure. To compute $\chi(i)$, we only need to consider those suffixes k, such that $\mathsf{W}[k] = \mathsf{W}[i]$. To this end, we create (at most) σ compacted tries $\mathsf{PST}_1, \mathsf{PST}_2, \ldots, \mathsf{PST}_\sigma$, where PST_α is the compacted trie of the strings in $\{\mathsf{prev}(T[\mathsf{PSA}[k], n]) \mid \mathsf{W}[k] = \alpha\}$. We do not store these trees explicitly; rather, we store their topology with succinct functionalities using Fact 6.

We pre-process each PST_α using Lemma 11 as follows: associate each node w in PST_α with 1 if $\mathsf{strLen}(\mathsf{parent}(w)) \leq \mathsf{f}_i < \mathsf{strLen}(w)$ for some leaf ℓ_i under w with $\mathsf{W}[i] = \alpha$. We also pre-process PST_α using Lemma 12 as follows: associate a node w with the number β_w, where β_w is the number of suffixes i in the subtree of w in PST_α such that $\mathsf{f}_i = \mathsf{strLen}(w)$. Note that $\sum_w \beta_w$ over all nodes w in all the trees is $O(n_\alpha)$, where n_α is the number of leaves in PST_α. The space needed for all such PST_α trees combined is $O(n)$ bits. Finally, we store a partial-rank data structure (Fact 7) on W. The total space needed is $O(n \log \sigma)$ bits.

Query Processing. Given i, in $O(1)$ time, we first jump to the corresponding leaf $\ell_{i'}$ in $\mathsf{PST}_{\mathsf{W}[i]}$ by using the $\mathsf{partialRank}_{\mathsf{W}}(i)$ query. Now locate the highest node u in $\mathsf{PST}_{\mathsf{W}[i]}$ such that $1 + \mathsf{f}_i \leq \mathsf{strLen}(u)$ in $O(1)$ time using Lemma 11. We consider the following two scenarios separately. To determine which case a suffix falls in, we store a bit-vector $F[1, n]$, such that $F[i] = 1$ iff the suffix i belongs to the first case. In each of the following cases, we can compute $\chi(i)$ in $O(1)$ time, which completes the proof of Lemma 8.

Case 1: $\mathsf{strLen}(\mathsf{parent}(u)) < \mathsf{f}_i < \mathsf{strLen}(u)$ for an ancestor node u of ℓ_i.
Let j be such that $\mathsf{W}[j] = \mathsf{W}[i]$. Applying Lemma 10, $\Psi(j) < \Psi(i)$ if either $j < i$ or $\mathsf{f}_j = \mathsf{plcp}(i, j)$. Thus, $\chi(i) = i' + \sum_v \beta_v$, where v is an ancestor of $\ell_{i'}$. The last term can be computed in $O(1)$ time using Lemma 12.

Case 2: $\mathsf{f}_i = \mathsf{strLen}(u)$ for an ancestor node u of ℓ_i.
Let j be such that $\mathsf{W}[j] = \mathsf{W}[i]$. Applying Lemma 10, $\Psi(j) < \Psi(i)$ if either (1) $j < i$ and $\mathsf{plcp}(i, j) \neq \mathsf{f}_i$, or (2) $\mathsf{f}_j = \mathsf{plcp}(i, j)$. Let w be the child of u on the path to $\ell_{i'}$. Using Fact 6, in $O(1)$ time, we compute $\mathsf{lmostLeaf}(u)$ and $\mathsf{lmostLeaf}(w)$, which are respectively the leftmost leaf in the subtree of u and the subtree of w. Thus,

$$\chi(i) = i' - (\mathsf{lmostLeaf}(w) - \mathsf{lmostLeaf}(u)) + \sum_v \beta_v$$

where v is an ancestor of $\ell_{i'}$. The last term can be computed in $O(1)$ time using Lemma 12. This completes the proof of Lemma 8.

2.3 Proofs of Lemma 11 and Lemma 12

We rely on standard techniques from succinct data structures. For both lemmas, we employ the following marking scheme. Starting from the leftmost leaf, every $C = c\lceil \log L \rceil$ leaves form a group, where c is a constant to be decided later. (The last group may have fewer than C leaves.) Mark the LCA of the first and last leaf of each group. The number of marked nodes is $O(L/C)$ [22].

We prove Lemma 11 first. At each marked node, we store the depth of its nearest ancestor (including itself) which is associated with a 1. Traverse the subtree τ_{u^*} of a marked node u^* in pre-order, and create a bit-string B_{u^*} as follows: when entering the subtree of a node w, append 1 if w is associated with a 1, followed by a 0 to B_{u^*}. Additionally, for every $i \in [1, L_{u^*}]$, store $A_{u^*}[i] =$ node depth of the nearest ancestor of $\ell_{u^*,i}$ associated with a 1 if the ancestor is in τ_{u^*}, else $A_{u^*}[i] = -1$. For each marked node u^*, maintain a pointer to the corresponding A_{u^*} and B_{u^*} pair. Pre-process τ_{u^*} with Fact 6. Lastly, we maintain a

bit-vector to detect in $O(1)$ time whether a leaf has an ancestor associated with 1, or not. The total space as before can be bounded by $O(L)$ bits. Given a query leaf ℓ_k, we check whether ℓ_k has an ancestor associated with 1 or not. Assume that it does. Then, we first locate the lowest marked node v^* as described earlier. Let d^* be the node depth stored at v^*. Let $k' = k - C\lfloor k/C \rfloor$. Check the k'th entry of the satellite array of v^*, and let it be d'. If $d' >= 0$, then the desired node is given by $\mathsf{levelAncestor}(\ell, d')$, else the desired node is given by $\mathsf{levelAncestor}(\ell, d^*)$.

Now, we prove Lemma 12. At each marked node u^*, store $G(u^*)$. Since the number of marked nodes is at most $\lceil L/C \rceil$, the space needed is $O(\frac{L}{C} \log L) = O(L)$ bits. Let τ_{u^*} be the subtree rooted at a marked node u^*. Note that τ_{u^*} has at most $2C$ nodes. Traverse tree τ_{u^*} in pre-order, and create a bit-string B_{u^*} as follows: when entering the subtree of a node w, append $g(w)$ in unaryto B_{u^*}. Additionally, for every $i \in [1, L_{u^*}]$, store $A_{u^*}[i] = G(\ell_{u^*,i})$, where L_{u^*} is the number of leaves in the subtree of u^*, and $\ell_{u^*,i}$ is the i^{th}-leftmost leaf in τ_{u^*}. The space needed to store the array A_{u^*} is $O(C \log C)$ bits. Note that $|B_{u^*}| \leq 2C$; hence, the number of possible such bit-strings is at most 2^{2C}. We store all possible combinations of A_{u^*} and B_{u^*}, which requires $O(2^{2C} C \log C)$ bits, which is $o(L)$ bits for $c = 1/4$. For each marked node u^*, maintain a pointer to the corresponding A_{u^*} and B_{u^*} pair, which requires $\frac{L}{C} \log(2^{2C}) = O(L)$ bits. Finally, pre-process τ_{u^*} with Fact 6. The total space needed is $O(L)$ bits. Given a query leaf ℓ_k, we first locate the lowest marked node $v^* = \mathsf{lca}(\ell_x, \ell_y)$ of ℓ_k, where $x = 1 + C\lfloor k/C \rfloor$, $y = \min\{L, C(1 + \lfloor k/C \rfloor)\}$. Let d^* be the value stored at v^*. Let $k' = k - C\lfloor k/C \rfloor$. Check the k'th entry of the satellite array of v^*, and let it be d'. Then, $G(\ell_k) = d^* + d'$ is computed in $O(1)$ time.

3 Generalized Ψ Function

We start with a definition that we are going to use throughout this section, as well as a couple of lemmas that will form the backbone of the indexes to achieve the three trade-offs.

▶ **Definition 13.** *Define* $\Psi^k(i) = \mathsf{PSA}^{-1}[\mathsf{PSA}[i] + k]$.

Our main arsenal to obtain the three trade-offs is the following version of Theorem 5, which enables the computation of "some" $\Psi^k(\cdot)$ in time faster than $O(k)$.

▶ **Lemma 14.** *For any predefined integer* Δ*, we can construct an* $O(n \log \sigma)$*-bit structure* $\mathsf{DS}(\Delta)$ *that computes* $\Psi^\Delta(i)$ *for any* i *with* $\mathsf{PSA}[i]$ *being a multiple of* Δ *in* $O(1)$ *time.*

We prove this lemma in this section. Let's start with the intuition. Note that in Lemma 14, if one is willing to relax the time to $O(\Delta)$, we can simply apply Theorem 5 Δ times. Here, we will chop off the first Δ characters of a suffix, where the characters are from Σ. To reduce the time to $O(1)$, the main idea is to consider a character from the alphabet Σ^Δ; clearly, chopping off one character from Σ^Δ is equivalent to chopping off Δ characters from Σ. Note that each character from Σ^Δ requires $\Delta \log \sigma$ bits for representation; however, since we sample $\approx n/\Delta$ suffixes, the total space will still be $O((n/\Delta) \cdot \Delta \log \sigma) = O(n \log \sigma)$ bits. The proof techniques are similar to that of Theorem 5.

▶ **Definition 15.** *A suffix is* Δ*-sampled if its starting position is a multiple of* Δ*. Let* S_Δ *be the collection of all* Δ*-sampled suffixes of* T*, i.e.,* $S_\Delta = \{T_\Delta, T_{2\Delta}, \dots\}$*. Let* $n_\Delta = |S_\Delta|$ *be the number of* Δ*-sampled suffixes. A* Δ*-sampled parameterized suffix array, denoted as* $\mathsf{PSA}_\Delta[1, n_\Delta]$*, stores the starting position of the suffixes in lexicographic order of their* prev*-encoding.*

▶ **Definition 16.** *Let $B_\Delta[1,n]$ be a bitmap, such that $B_\Delta[i] = 1$ iff $\mathsf{PSA}[i]$ is a multiple of Δ.*

▶ **Lemma 17.** *Given $i \in [1, n_\Delta]$, we can find a $j \in [1,n]$, such that $\mathsf{PSA}[j] = \mathsf{PSA}_\Delta[i]$ in $O(1)$ time by maintaining an $O(n)$-bit space structure.*

Proof. We maintain a bit-vector $B_\Delta[1,n]$ using Fact 7, where $B_\Delta[i] = 1$ iff $T_{\mathsf{PSA}[i]}$ is Δ-sampled. The total space needed is $O(n)$ bits. Observe that $T_{\mathsf{PSA}_\Delta[i]}$ is exactly the i^{th} Δ-sampled suffix in lexicographic order; thus, $j = \mathsf{select}_{B_\Delta}(i,1)$. ◀

▶ **Definition 18.** *Let $T_{\mathsf{PSA}[i]}$ be a Δ-sampled suffix. Define the following:*

Notation	***Definition ($\circ$ denotes concatenation)***
$\Psi^\Delta(i)$	$\mathsf{PSA}^{-1}[\mathsf{PSA}[i] + \Delta]$
$\mathsf{PLF}^\Delta(i)$	$\mathsf{PSA}^{-1}[\mathsf{PSA}[i] - \Delta]$
$\mathsf{W}^\Delta[i]$	$\mathsf{W}[i] \circ \mathsf{W}[\Psi(i)] \circ \cdots \circ \mathsf{W}[\Psi^{\Delta-1}(i)]$
$\mathsf{PBWT}^\Delta[i]$	$\mathsf{PBWT}[\mathsf{PLF}^{\Delta-1}(i)] \circ \mathsf{PBWT}[\mathsf{PLF}^{\Delta-2}(i)] \circ \cdots \circ \mathsf{PBWT}[i]$

▶ **Observation 19.** *Let $T_{\mathsf{PSA}[i]}$ be a Δ-sampled suffix. The following observations can be deduced from the above definitions:*

- $\Psi^\Delta(i) = j$ *iff* $\mathsf{PLF}^\Delta(j) = i$
- *If* $\Psi^\Delta(i) = j$, *then* $\mathsf{PBWT}^\Delta[i] = \mathsf{W}^\Delta[j]$

▶ **Definition 20.** *Let $T_{\mathsf{PSA}[i]}$ be a Δ-sampled suffix. Define*

$$\chi^\Delta(i) = |\{k, \text{ where } T_{\mathsf{PSA}[k]} \text{ is } \Delta\text{−sampled}, \Psi^\Delta(k) \leq \Psi^\Delta(i) \text{ and } \mathsf{W}^\Delta[k] = \mathsf{W}^\Delta[i]\}|$$

Reduction from function Ψ^Δ to χ^Δ. Using Observation 19, it is easy to see that if $j = \Psi^\Delta(i)$, then j is the $\chi^\Delta(i)^{th}$ occurrence of $\mathsf{W}^\Delta[i]$ in PBWT^Δ. Given $\chi^\Delta(i)$, we can compute $\Psi^\Delta(i)$ as: $\Psi^\Delta(i) = \mathsf{select}_{B_\Delta}\left(\mathsf{select}_{\mathsf{PBWT}^\Delta}\left(\chi^\Delta(i), \mathsf{W}^\Delta[i]\right), 1\right)$

Note that the arrays $\mathsf{PBWT}^\Delta[1, n_\Delta]$ and $\mathsf{W}^\Delta[1, n_\Delta]$ take $O(\Delta \log \sigma)$ bit per entry. We preprocess them into compact data structures that support access/select queries in $O(1)$ time (see Fact 7). The space needed is $O(n_\Delta \cdot \Delta \log \sigma) = O(n \log \sigma)$ bits. Thus, given $\chi^\Delta(i)$, we can compute $\Psi^\Delta(i)$ in constant time. To this end, we present the following lemma,which completes the proof of Lemma 14.

▶ **Lemma 21.** *Let $T_{\mathsf{PSA}[i]}$ be a Δ-sampled suffix. By using an $O(n \log \sigma)$-bit data structure, we can compute $\chi^\Delta(i)$ in $O(1)$ time.*

3.1 Proof of Lemma 21

For the ease of notation, let $i_d = \Psi^d(i)$, $j_d = \Psi^d(j)$, and $L_d = \mathsf{plcp}(i_d, j_d)$, where $d \in [1, \Delta]$. Let $i_0 = i$, $j_0 = j$, and $L_0 = \mathsf{plcp}(i,j)$. Our proof hinges on Lemma 22, which says, for any two suffixes $T_{\mathsf{PSA}[i]}$ and $T_{\mathsf{PSA}[j]}$ such that $i < j$ and $\mathsf{W}^\Delta[i] = \mathsf{W}^\Delta[j]$, the relative order between i and j can change at most once while applying the Ψ-operation Δ number of times.

▶ **Lemma 22.** *Consider two Δ-sampled suffixes $T_{\mathsf{PSA}[i]}$ and $T_{\mathsf{PSA}[j]}$ such that $i < j$, and $\mathsf{W}^\Delta[i] = \mathsf{W}^\Delta[j]$.*

- *If there exists a $\gamma \in [1, \Delta - 1]$ such that $i_\gamma > j_\gamma$, then $\forall \gamma' \in [\gamma + 1, \Delta]$, $i_{\gamma'} > j_{\gamma'}$.*
- *Consider the minimum $\gamma \in [1, \Delta]$ such that $i_\gamma > j_\gamma$, then $\mathsf{f}_{j_{\gamma-1}} = L_{\gamma-1}$.*

Proof. We prove the first part of the lemma; the second part follows directly. Consider the smallest $\gamma \in [1, \Delta - 1]$ such that $\Psi^{\gamma}(i) > \Psi^{\gamma}(j)$. Since $\Psi^{\gamma-1}(i) < \Psi^{\gamma-1}(j)$, we have $\mathsf{str}(\ell_{j_{\gamma-1}})[1 + L_{\gamma-1}] > \mathsf{str}(\ell_{i_{\gamma-1}})[1 + L_{\gamma-1}]$. Then, $\mathsf{str}(\ell_{j_\gamma})[1 + L_\gamma] = 0 < \mathsf{str}(\ell_{i_\gamma})[1 + L_\gamma]$; otherwise, applying Lemma 9, it is easy to show that $\mathsf{W}^{\Delta}[i] \neq \mathsf{W}^{\Delta}[j]$. For the purpose of contradiction, consider the smallest $\gamma' > \gamma$ such that $\Psi^{\gamma'}(i) < \Psi^{\gamma'}(j)$. Note that $L_{\gamma'-1} = L_\gamma - (\gamma' - \gamma - 1)$. Hence, $\mathsf{str}(\ell_{j_{\gamma'-1}})[1 + L_{\gamma'-1}] = 0 < \mathsf{str}(\ell_{i_{\gamma'-1}})[1 + L_{\gamma'-1}]$. Since $\Psi^{\gamma'}(i) < \Psi^{\gamma'}(j)$, applying Lemma 9, $\mathsf{str}(\ell_{i_{\gamma'}})[1 + L_{\gamma'}] = 0$ and $\mathsf{f}_{i_{\gamma'}} = L_{\gamma'}$, which contradicts $\mathsf{W}^{\Delta}[i] = \mathsf{W}^{\Delta}[j]$. This completes the proof. ◀

Using Lemmas 9, 10, and 22, we get the following.

▶ **Lemma 23.** *Consider two Δ-sampled suffixes $T_{\mathsf{PSA}[i]}$ and $T_{\mathsf{PSA}[j]}$ such that $i < j$, and $\mathsf{W}^{\Delta}[i] = \mathsf{W}^{\Delta}[j]$. If there exists a $\gamma \in [1, \Delta]$ such that $i_\gamma > j_\gamma$, then*

- *$i_\Delta > j_\Delta$, and*
- *for any k such that $T_{\mathsf{PSA}[k]}$ is Δ-sampled, $\mathsf{W}^{\Delta}[i] = \mathsf{W}^{\Delta}[k]$, and $\mathsf{plcp}(k, j) > \mathsf{plcp}(i, j)$, we have $i_\Delta > k_\Delta$.*

To compute $\chi^{\Delta}(i)$, note that we only need to consider those suffixes k, such that $\mathsf{W}^{\Delta}[k] = \mathsf{W}^{\Delta}[i]$. To this end, we create (at most) σ^{Δ} compact tries $\mathsf{PST}^{\Delta}_1, \mathsf{PST}^{\Delta}_2, \ldots, \mathsf{PST}^{\Delta}_{\sigma^\Delta}$, where PST_α is the compacted trie of the strings in

$$\{\mathsf{prev}(T[\mathsf{PSA}[k], n]) \mid \mathsf{W}^{\Delta}[k] = \alpha \text{ and } T_{\mathsf{PSA}[k]} \text{ is } \Delta \text{ sampled}\}$$

We do not store these trees explicitly; rather, we maintain the data-structure of Fact 6 for each tree topology. Note that given a leaf k in PST, where $T_{\mathsf{PSA}[k]}$ is Δ sampled, we can jump to its corresponding leaf k' in $\mathsf{PST}_{\mathsf{W}^\Delta}$ in $O(1)$ time using an $O(n \log \sigma)$ structure (the bit-array B_Δ, and Fact 7 over W^{Δ}).

Consider a tree PST^{Δ}_x. Let the number of suffixes lying in this tree be m_x. For any leaf j' in PST^{Δ}_x, let $\mathsf{map}(j')$ be the equivalent leaf in PST. Consider a node u in PST^{Δ}_x. For each $\gamma \in [1, \Delta]$ we write two numbers $G_\gamma(u)$ and $H_\gamma(u)$ defined as:

- $G_\gamma(u)$ = the number of leaves j' in the subtree of u such that $\mathsf{f}_{j_\gamma} = \mathsf{strLen}(u)$, where $j = \mathsf{map}(j')$
- If $\mathsf{f}_{j_\gamma} \neq \mathsf{strLen}(\mathsf{parent}(u))$, where j' is a leaf in the subtree of u and $j = \mathsf{map}(j')$, then $H_\gamma(u) = 0$. Else, $H_\gamma(u)$ = the number of leaves k' in the subtree of $\mathsf{parent}(u)$ such that $\mathsf{f}_{k_\gamma} \neq \mathsf{strLen}(\mathsf{parent}(u))$ and $\mathsf{pre\text{-}order}(\ell_{k'}) < \mathsf{pre\text{-}order}(u)$, where $k = \mathsf{map}(k')$

Note that $\sum_u G_\gamma(u) \leq m_x$ and $\sum_u H_\gamma(u) \leq m_x$ (using Lemmas 22 and 23). Hence, $\sum_u G_\gamma(u)$ and $\sum_u H_\gamma(u)$ over $\gamma \in [1, \Delta]$ can be stored in $O(m_x \Delta)$ bits using unary encoding. To compute $\chi^{\Delta}(i)$, we first jump to the corresponding leaf i' in $\mathsf{PST}^{\Delta}_{\mathsf{W}^\Delta[i]}$ in $O(1)$ time. Let $\mathcal{U}$ be the set of ancestors of $\ell_{i'}$ Now, $\chi(i) = i' + \sum_{\gamma=1}^{\Delta} \sum_{u \in \mathcal{U}} G_u(\ell_{i'}) - \sum_{\gamma=1}^{\Delta} \sum_{u \in \mathcal{U}} H_u(\ell_{i'})$, which can be computed in $O(1)$ time using (slightly adapted versions of) Lemmas 11 and 12. Since $\sum_{x=1}^{\sigma^\Delta} m_x = n_\Delta$, the total space is $O(n \log \sigma)$ bits; recall that m_x is the number of suffixes in PST^{Δ}_x. This concludes the proof.

4 Achieving the Three Trade-offs of PSA

We prove the trade-offs using the result in Lemma 14 as a black box. Additionally, we will use the sampled PSA and sampled PSA^{-1} in Lemma 24 for all the three cases. Let $\lambda = 2^{\lceil \log \log_\sigma n \rceil}$, the next highest power of 2 greater than or equal to $\log_\sigma n$. The strategy for computing $\mathsf{PSA}[i]$ is the same as before (refer to Section 2), i.e., find the smallest $k < \lambda$, such that $\mathsf{PSA}[i] = \mathsf{PSA}[j] - k$, where $j = \Psi^k(i)$ and $B_\lambda[j] = 1$, but in fewer number of steps.

▶ **Lemma 24** (Sampled PSA and PSA^{-1}). *A sampled-*PSA *is a structure that supports the following query: for any i, it reports* $\mathsf{PSA}[i]$ *if* $\mathsf{PSA}[i]$ *is a multiple of λ, and ∞ otherwise. Similarly, a sampled-*PSA^{-1} *is a structure that computes* $\mathsf{PSA}^{-1}[i]$ *for any i which is a multiple of λ. We can maintain them in $O(n \log \sigma)$ bits and answer queries in $O(1)$ time.*

Proof. Note that the sampled-PSA^{-1} is an array of size $O(n/\lambda)$ in which each entry can be recorded using $\lceil \log n \rceil$ bits and accessed in $O(1)$ time. For the sampled-PSA, we maintain the bitmap $B_\lambda[1, n]$ in Definition 16 by choosing $\Delta = \lambda$. Additionally, we associate $\mathsf{PSA}[i]$ with those i's where $B_\lambda[i] = 1$. The space required is $(n + (n/\lambda) \log n) = O(n \log \sigma)$ bits, and the query can be easily handled in $O(1)$ time. ◀

4.1 Achieving $t_{\mathsf{PSA}} = O(\log^{\varepsilon}_{\sigma} n)$ using $O(n \log \sigma)$ bits

Let Δ_t be $(\log^{\varepsilon}_{\sigma} n)^t$ rounded to the next highest power of 2. We maintain $\mathsf{DS}(\Delta_t)$ and $B_{\Delta_t}[1, n]$ for $t = 0, 1, 2, 3, \ldots, 1/\varepsilon$. (Recall Lemma 14 and Definition 16 for definitions of this data structures.) The space is $1/\varepsilon \times O(n \log \sigma)$ bits, as desired.

To compute $\mathsf{PSA}[i]$, we initialize $k = 0, j = i, t = 0$ and follow the steps below.

1. If $B_\lambda[j] = 1$, access $\mathsf{PSA}[j]$ in $O(1)$ time from the sampled-PSA and report $\mathsf{PSA}[j] - k$.
2. Else if $B_{\Delta_{t+1}}[j] = 1$, update $t \leftarrow t + 1$ and go to Step 1.
3. Else we compute $j' = \Psi^{\Delta_t}(j)$ using $\mathsf{DS}(\Delta_t)$ in $O(1)$ time, update $j \leftarrow j'$, $k \leftarrow k + \Delta_t$, and then we repeat from Step 2.

Then, the number of times we perform (constant time) $\Psi^{\Delta_t}(\cdot)$ operations on $\mathsf{DS}(\Delta_t)$ is at most $\Delta_{t+1}/\Delta_t = \log^{\varepsilon}_{\sigma} n$. Therefore, the overall time complexity is $O(\frac{1}{\varepsilon} \log^{\varepsilon}_{\sigma} n)$.

Note that the algorithm for $\mathsf{PSA}[i]$ computes several j's, starting with $j = i$, such that the j computed after t^{th} "step 1" guarantees that $\mathsf{PSA}[j] \geq \mathsf{PSA}[i]$ is the smallest number divisible by Δ_t. Therefore, the correctness follows from that fact that $\mathsf{PSA}[i]$ is $\mathsf{PSA}[j] - k$.

The computation of $\mathsf{PSA}^{-1}[\cdot]$ is analogous, but in the reverse order, as desired. Specifically, we perform queries on $\mathsf{DS}(\Delta_t)$'s, in descending order of t.

4.2 Achieving $t_{\mathsf{PSA}} = O(\log \log_{\sigma} n)$ using $O(n \log \sigma \log \log_{\sigma} n)$ bits

We maintain $\mathsf{DS}(\Delta_t)$ structure of Lemma 14 and $B_{\Delta_t}[1, n]$, where $\Delta_t = 2^t$, for $t = 0, 1, 2, 3, \ldots, \log \lambda$, where $\lambda = 2^{\lceil \log \log_{\sigma} n \rceil}$ as defined in Lemma 24. The space is $O(n \log \sigma) \times \log \lambda$ bits, as desired.

To compute $\mathsf{PSA}[i]$, we initialize $k = 0, j = i, t = 0$ and follow the steps below.

1. If $B_\lambda[j] = 1$, access $\mathsf{PSA}[j]$ in $O(1)$ time from the sampled-PSA and report $\mathsf{PSA}[j] - k$.
2. Else if $B_{\Delta_{t+1}}[j] = 1$, update $t \leftarrow t + 1$ and go to Step 1.
3. Else compute $j' = \Psi^{\Delta_t}(j)$ using $\mathsf{DS}(\Delta_t)$ in $O(1)$ time, update $j \leftarrow j'$ and $k \leftarrow k + \Delta_t$. Then update $t \leftarrow t + 1$ and go to Step 1.

We perform at most $\log \lambda$ constant-time operations on $\mathsf{DS}(\cdot)$, hence $t_{\mathsf{PSA}} = O(\log \log_{\sigma} n)$. The computation of $\mathsf{PSA}^{-1}[\cdot]$ (and correctness proof) is analogous as in the previous section.

4.3 Achieving $t_{\mathsf{PSA}} = O(1)$ using $O(n \log \sigma \log^{\varepsilon}_{\sigma} n)$ bits

Here we use Lemma 25, which is a slight modification of Lemma 14. We remark that the proof is rather straightforward given the proof of Lemma 14; so, we omit it.

▶ **Lemma 25.** *For any predefined integers Δ and $\delta < \Delta$ (both are powers of 2), we can construct an $O(n \log \sigma)$-bit structure that computes $\Psi^{\delta}(i)$ for any i with $(\mathsf{PSA}[i] + \delta)$ being a multiple of δ in $O(1)$ time. We call this data structure $\mathsf{DS}(\Delta, \delta)$.*

We define an array $E^{\delta}_{\Delta}[1, n]$ such that $E^{\delta}_{\Delta}[i]$ is

- $-\infty$ if $\mathsf{PSA}[i]$ is not a multiple of Δ
- an integer $f \in [0, \Delta/\delta)$, such that $(\mathsf{PSA}[i] + f \cdot \delta)$ is a multiple of Δ.

Let Δ_t be $(\log^{\varepsilon}_{\sigma} n)^t$ rounded to the next highest power of 2. We store $\mathsf{DS}(\Delta_{t+1}, f \cdot \Delta_t)$ for all $t \in [0, 1/\varepsilon]$ and $f \in [0, \Delta_{t+1}/\Delta_t)$. Additionally, we store $E^{\Delta_t}_{\Delta_{t+1}}[1, n]$ for all $t \in [0, 1/\varepsilon]$. Therefore, the total space is $n/\varepsilon \times O\big(\log \sigma \log^{\varepsilon}_{\sigma} n + \log(\log^{\varepsilon}_{\sigma} n)\big)$ bits.
To compute $\mathsf{PSA}[i]$, we initialize $k = 0, j = i, t = 0$ and follow the steps below.

1. If $B_{\lambda}[j] = 1$, access $\mathsf{PSA}[j]$ in $O(1)$ time from the sampled-PSA and report $\mathsf{PSA}[j] - k$.
2. Else if $B_{\Delta_{t+1}}[j] = 1$, update $t \leftarrow t + 1$ and go to Step 1.
3. Else find f from $E^{\Delta_t}_{\Delta_{t+1}}[1, n]$, such that $(\mathsf{PSA}[i] + f \cdot \Delta_t)$ is a multiple of Δ_{t+1}. Compute $j' = \Psi^{f \cdot \Delta_t}(j)$ using $\mathsf{DS}(\Delta_{t+1}, f \cdot \Delta_t)$ in $O(1)$ time, update $j \leftarrow j'$ and $k \leftarrow k + f \cdot \Delta_t$. Then go to Step 2.

We issue at most one (constant time) query on $\mathsf{DS}(\Delta_t, \cdot)$ per t. Therefore, $t_{\mathsf{PSA}} = O(1/\varepsilon)$. The computation of $\mathsf{PSA}^{-1}[\cdot]$ (and correctness proof) is analogous to the discussion in the previous two sections.

5 Encoding Parameterized Longest Common Prefix (pLCP) Array

Recall that $\mathsf{PLCP}[i] = \mathsf{plcp}(i, i+1)$ for $1 \le i < n$. We introduce a new encoding scheme, which converts a string S to a string $\mathsf{encode}(S)$ over an alphabet $\Sigma'' = \{0, 1, \ldots, \sigma\}$ as follows. We replace each character $S[i]$ with 0 if i is the first occurrence of $S[i]$, else replace it with the number of distinct characters in $S[j, i]$, where $j < i$ is the rightmost occurrence of $S[i]$ before i. For example, $\mathsf{encode}(xyxxzyx) = 0021033$. For any two strings S and S', $\mathsf{encode}(S) = \mathsf{encode}(S')$ iff $\mathsf{prev}(S) = \mathsf{prev}(S')$; the proof is straightforward using mathematical induction.

Let $T' = \mathsf{encode}(T)$. Let $\mathsf{SA}_{T'}[1, n]$ be the suffix array of T', i.e., $\mathsf{SA}_{T'}[i] = j$ and $\mathsf{SA}^{-1}_{T'}[j] = i$ iff the i^{th} lexicographically smallest suffix of T' starts at position j. Also, let $\mathsf{lcp}_{T'}(i, j)$ be the length of the longest common prefix of the suffixes of T' starting at $\mathsf{SA}_{T'}[i]$ and $\mathsf{SA}_{T'}[j]$. The following is immediate from known results on encoding suffix trees.

▶ **Fact 26** ([20, 35]). *We can answer $\mathsf{SA}_{T'}[\cdot]$ and $\mathsf{SA}^{-1}_{T'}[\cdot]$ queries as follows:*

- *in $\mathsf{t}_{\mathsf{SA}} = O(\log^{\varepsilon}_{\sigma} n)$ time using an $O(n \log \sigma)$-bit index*
- *in $\mathsf{t}_{\mathsf{SA}} = O(\log \log_{\sigma} n)$ time using an $O(n \log \sigma \log \log_{\sigma} n)$-bit index*
- *in $\mathsf{t}_{\mathsf{SA}} = O(1)$ time using an $O(n \log \sigma \log^{\varepsilon}_{\sigma} n)$-bit index*

Moreover, we can answer $\mathsf{lcp}_{T'}(\cdot, \cdot)$ queries in $O(t_{\mathsf{t}_{\mathsf{SA}}})$ time using $O(n)$ extra bits.

We have the following crucial lemma.

▶ **Lemma 27.** *Let x_i be the smallest number such that the number of distinct characters in $T_{\mathsf{PSA}[i]}[1, \mathsf{PLCP}[i]]$ and $T_{\mathsf{PSA}[i]}[1, x_i]$ are the same. Then,*

$$\mathsf{PLCP}[i] = x_i + \mathsf{lcp}_{T'}\Big(\mathsf{SA}^{-1}_{T'}\big[\mathsf{PSA}[i] + x_i\big], \mathsf{SA}^{-1}_{T'}\big[\mathsf{PSA}[i+1] + x_i\big]\Big)$$

Proof. Since $\mathsf{PLCP}[i] \geq x_i$, $y_i = \mathsf{PLCP}[i] - x_i$ is the length of the longest common prefix of the strings obtained by deleting the first x_i characters of $\mathsf{prev}(T_{\mathsf{PSA}[i]})$ and $\mathsf{prev}(T_{\mathsf{PSA}[i+1]})$ respectively. Equivalently, y_i is the longest common prefix of the suffixes of $\mathsf{encode}(T)$ starting at positions $\mathsf{PSA}[i] + x_i$ and $\mathsf{PSA}[i+1] + x_i$ respectively. The proof follows from the definition of x_i. ◀

▶ **Theorem 28.** *Suppose* $\mathsf{PSA}[\cdot]$ *and* $\mathsf{SA}^{-1}_{T'}[\cdot]$ *values are accessible in times* $\mathsf{t_{PSA}}$ *and* $\mathsf{t_{SA}}$ *respectively, we can compute* $\mathsf{PLCP}[i] = x_i + y_i$ *for any* i *in time* $O(\mathsf{t_{SA}} + \mathsf{t_{PSA}})$ *using an* $O(n \log \sigma)$*-bit structure. We can also support* $\mathsf{plcp}(\cdot, \cdot)$ *queries in the same time.*

Proof. We first describe the structure for computing x_i. If $\sigma > \log n$, we store x_i explicitly in $\log n$ bits if $x_i > \sigma \log n$ and in $O(\log(\sigma \log n))$ bits otherwise. All x_i's that are larger than $\sigma \log n$ can be stored in $O(n)$ bits as they are no more than $n/\log n$. The space needed for the rest is $n \log(\sigma \log n) = O(n \log \sigma)$ bits. If $\sigma \leq \log n$, maintain an array C, where $C[i] = T_{\mathsf{PSA}[i]}[x_i]$ and a rank-select data structure (Fact 7) over T. The space is $O(n \log \sigma)$ bits. Since x_i is the first occurrence of $C[i]$ in $T[\mathsf{PSA}[i], n]$, we compute $x_i = \mathsf{select}_T\big(\mathsf{rank}_T\big(\mathsf{PSA}[i] - 1, C[i]\big) + 1, C[i]\big) - \mathsf{PSA}[i] + 1$ in time $t_{\mathsf{PSA}} + O(\log(\log \sigma / \log \log n)) = O(t_{\mathsf{PSA}})$.

We now focus on computing y_i. Find $j = \mathsf{SA}^{-1}_{T'}[\mathsf{PSA}[i] + x_i]$ and $k = \mathsf{SA}^{-1}_{T'}[\mathsf{PSA}[i+1] + x_i]$ in time $O(\mathsf{t_{SA}} + \mathsf{t_{PSA}})$. From Lemma 27, we have $y_i = \mathsf{lcp}_{T'}(j, k)$. We handle $\mathsf{lcp}_{T'}(\cdot, \cdot)$ queries in time $O(t_{\mathsf{SA}})$ using $O(n)$ extra bits [35].

Finally, to answer $\mathsf{plcp}(\cdot, \cdot)$ queries, we maintain a Range Minimum Query (RMQ) structure [8] of size $2n + o(n)$ over the PLCP array with $O(1)$ query time. Then, given any i and $j > i$, compute $k = \arg\min\{\mathsf{PLCP}[k] \mid k \in [i, j)\}$ and report $\mathsf{plcp}(i, j) = \mathsf{PLCP}[k]$. ◀

Theorem 3 follows from Theorem 28, Fact 26, and the trade-offs in Section 4.

References

1. Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 71–80, 1993. doi:10.1145/167088.167115.
2. Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, 2015. doi:10.1145/2629339.
3. Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Order-preserving incomplete suffix trees and order-preserving indexes. In *String Processing and Information Retrieval - 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, pages 84–95, 2013. doi:10.1007/978-3-319-02432-5_13.
4. Gianni Decaroli, Travis Gagie, and Giovanni Manzini. A compact index for order-preserving pattern matching. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 72–81. IEEE, 2017. doi:10.1109/DCC.2017.35.
5. Gianni Decaroli, Travis Gagie, and Giovanni Manzini. A compact index for order-preserving pattern matching. *Softw. Pract. Exp.*, 49(6):1041–1051, 2019. doi:10.1002/spe.2694.
6. Diptarama, Takashi Katsura, Yuhei Otomo, Kazuyuki Narisawa, and Ayumi Shinohara. Position heaps for parameterized strings. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.8.
7. Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

8 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comput.*, 40(2):465–492, 2011. doi:10.1137/090779759.

9 Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. The parameterized suffix tray. In Tiziana Calamoneri and Federico Corò, editors, *Algorithms and Complexity - 12th International Conference, CIAC 2021, Virtual Event, May 10-12, 2021, Proceedings*, volume 12701 of *Lecture Notes in Computer Science*, pages 258–270. Springer, 2021. doi:10.1007/978-3-030-75242-2_18.

10 Travis Gagie, Giovanni Manzini, and Rossano Venturini. An encoding for order-preserving matching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.38.

11 Arnab Ganguly, Wing-Kai Hon, Yu-An Huang, Solon P. Pissis, Rahul Shah, and Sharma V. Thankachan. Parameterized text indexing with one wildcard. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2019, Snowbird, UT, USA, March 26-29, 2019*, pages 152–161. IEEE, 2019. doi:10.1109/DCC.2019.00023.

12 Arnab Ganguly, Wing-Kai Hon, Kunihiko Sadakane, Rahul Shah, Sharma V. Thankachan, and Yilin Yang. Space-efficient dictionaries for parameterized and order-preserving pattern matching. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPIcs*, pages 2:1–2:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.2.

13 Arnab Ganguly, Wing-Kai Hon, Kunihiko Sadakane, Rahul Shah, Sharma V. Thankachan, and Yilin Yang. A framework for designing space-efficient dictionaries for parameterized and order-preserving matching. *Theor. Comput. Sci.*, 854:52–62, 2021. doi:10.1016/j.tcs.2020.11.036.

14 Arnab Ganguly, Wing-Kai Hon, and Rahul Shah. A framework for dynamic parameterized dictionary matching. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 10:1–10:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.SWAT.2016.10.

15 Arnab Ganguly, Dhrumil Patel, Rahul Shah, and Sharma V. Thankachan. LF successor: Compact space indexing for order-isomorphic pattern matching. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 71:1–71:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.71.

16 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pbwt: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 397–407, 2017. doi:10.1137/1.9781611974782.25.

17 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Structural pattern matching - succinctly. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*, volume 92 of *LIPIcs*, pages 35:1–35:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ISAAC.2017.35.

18 Raffaele Giancarlo. The suffix of a square matrix, with applications. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 402–411, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313842.

19 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 841–850, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

20 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

21 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

22 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *J. ACM*, 61(2):9:1–9:36, 2014. doi:10.1145/2590774.

23 Dong Kyue Kim, Yoo Ah Kim, and Kunsoo Park. Generalizations of suffix arrays to multi-dimensional matrices. *Theor. Comput. Sci.*, 302(1-3):223–238, 2003. doi:10.1016/S0304-3975(02)00828-9.

24 Sung-Hwan Kim and Hwan-Gue Cho. Indexing isodirectional pointer sequences. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 35:1–35:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.35.

25 Sung-Hwan Kim and Hwan-Gue Cho. A compact index for cartesian tree matching. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021, July 5-7, 2021, Wroclaw, Poland*, volume 191 of *LIPIcs*, pages 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CPM.2021.18.

26 Sung-Hwan Kim and Hwan-Gue Cho. Simpler fm-index for parameterized string matching. *Inf. Process. Lett.*, 165:106026, 2021. doi:10.1016/j.ipl.2020.106026.

27 Stefan Kurtz. Reducing the space requirement of suffix trees. *Softw., Pract. Exper.*, 29(13):1149–1171, 1999. doi:10.1002/(SICI)1097-024X(199911)29:13<1149::AID-SPE274>3.0.CO;2-O.

28 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22:935–948, 1993. doi:10.1137/0222058.

29 Katsuhito Nakashima, Noriki Fujisato, Diptarama Hendrian, Yuto Nakashima, Ryo Yoshinaka, Shunsuke Inenaga, Hideo Bannai, Ayumi Shinohara, and Masayuki Takeda. Dawgs for parameterized matching: Online construction and related indexing structures. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPIcs*, pages 26:1–26:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.26.

30 Gonzalo Navarro. *Compact data structures: A practical approach.* Cambridge University Press, 2016.

31 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. doi:10.1145/2601073.

32 Sung Gwan Park, Amihood Amir, Gad M. Landau, and Kunsoo Park. Cartesian tree matching and indexing. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.16.

33 Dhrumil Patel and Rahul Shah. Inverse suffix array queries for 2-dimensional pattern matching in near-compact space. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 60:1–60:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.60.

34 Mihai Patrascu. Lower bounds for 2-dimensional range counting. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 40–46. ACM, 2007. doi:10.1145/1250790.1250797.

35 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/s00224-006-1198-x.

36 Tetsuo Shibuya. Generalization of a suffix tree for RNA structural pattern matching. In *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, pages 393–406, 2000. doi:10.1007/3-540-44985-X_34.

37 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

The Fine-Grained Complexity of Graph Homomorphism Parameterized by Clique-Width

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Thekla Hamm
Algorithms and Complexity Group, TU Wien, Austria

Viktoriia Korchemna
Algorithms and Complexity Group, TU Wien, Austria

Karolina Okrasa
Faculty of Matematics and Information Science, Warsaw University of Technology, Poland
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Kirill Simonov
Algorithms and Complexity Group, TU Wien, Austria

Abstract

The generic homomorphism problem, which asks whether an input graph G admits a homomorphism into a fixed target graph H, has been widely studied in the literature. In this article, we provide a fine-grained complexity classification of the running time of the homomorphism problem with respect to the clique-width of G (denoted cw) for virtually all choices of H under the Strong Exponential Time Hypothesis. In particular, we identify a property of H called the signature number $s(H)$ and show that for each H, the homomorphism problem can be solved in time $\mathcal{O}^*(s(H)^{\mathrm{cw}})$. Crucially, we then show that this algorithm can be used to obtain essentially tight upper bounds. Specifically, we provide a reduction that yields matching lower bounds for each H that is either a projective core or a graph admitting a factorization with additional properties – allowing us to cover all possible target graphs under long-standing conjectures.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases homomorphism, clique-width, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.66

Category Track A: Algorithms, Complexity and Games

Funding *Robert Ganian*: Robert Ganian acknowledges support by the Austrian Science Fund (FWF, projects Y1329 and P31336).
Thekla Hamm: Thekla Hamm acknowledges support by the Austrian Science Fund (FWF, projects P31336 and Y1329).
Viktoriia Korchemna: Viktoriia Korchemna acknowledges support by the Austrian Science Fund (FWF, project Y1329).
Karolina Okrasa: Karolina Okrasa acknowledges support by the European Research Council, grant agreement No 714704. Parts of this work were performed while visiting TU Wien, Vienna, Austria.
Kirill Simonov: Kirill Simonov acknowledges support by the Austrian Science Fund (FWF, project P31336).

1 Introduction

A *homomorphism* from a graph G to a graph H is an edge-preserving mapping from the vertices of G to the vertices of H. Homomorphisms are fundamental constructs which have been studied from a wide variety of perspectives [2, 3, 13]. Our focus here will be on the

class of problems which ask whether an input n-vertex graph G admits a homomorphism to a fixed target graph H. This "meta-problem" – which we simply call $\textsc{Hom}(H)$ – captures, among others, the classical c-Coloring problems when H is set to the complete graph on c vertices. Famously, Hell and Nešetřil [15] proved that $\textsc{Hom}(H)$ is polynomial-time solvable if H is bipartite or has a loop, and NP-complete otherwise.

While the aforementioned result provides a basic classification of the complexity of $\textsc{Hom}(H)$, it does not say much in terms of how quickly one can actually solve these problems. Indeed, the usual assumption that $\mathsf{P} \neq \mathsf{NP}$ is not sufficient to obtain tight bounds for the running times of algorithms. While upper bounds can be straightforwardly obtained by designing a suitable algorithm, the corresponding lower bounds usually rely on the Exponential Time Hypothesis (ETH) or the Strong Exponential Time Hypothesis (SETH), which allows for even tighter bounds [18, 19, 23]. It is not difficult to design a brute-force algorithm for the homomorphism problem that runs in time $\mathcal{O}^*(|V(H)|^n)$ for every choice of H, and thanks to the breakthrough result of Cygan et al. we now know that this running time is essentially tight under the Exponential Time Hypothesis (ETH) [6] as long as one considers only the dependency on n and $|V(H)|$.

Still, it is often possible to circumvent this lower bound and obtain significantly better runtime guarantees. One approach to do so is to consider restrictions on the class of targets: if H is a complete graph then $\textsc{Hom}(H)$ can be solved in time $\mathcal{O}^*(2^n)$, and there are also several algorithms that achieve running times of the form $\mathcal{O}^*(\alpha(H)^n)$ where $\alpha(H)$ is some structural parameter of H [12, 29, 30]. The other is to exploit the properties of the input graph G, which are commonly captured by a suitably defined structural parameter. The most commonly used graph parameter in this respect is *treewidth* [28], which informally measures how "tree-like" a graph is.

When considering treewidth, it is once again not difficult to obtain an algorithm that runs in time $\mathcal{O}^*(|V(H)|^{\mathrm{tw}})$, where tw is the treewidth of G; as before, it was much more difficult to show that this is essentially optimal. The first SETH-based tight lower bound in this setting was actually shown for special cases of the related problem of $\textsc{LHom}(H)$, where each vertex in the graph G comes with a list of admissible targets for the homomorphism [9]; this was later lifted to a full classification [24]. A nearly-complete SETH-based lower bound result for $\textsc{Hom}(H)$ itself was only obtained recently by Okrasa and Paweł Rzążewski [25]; in particular, the result covers all targets which are so-called *projective cores*. It is known that almost all graphs are projective cores [16, 25, 31], and it is worth noting that the authors showed that their result can be lifted to all targets under long-standing conjectures on the properties of projective cores [21, 22].

While treewidth is the most prominent structural graph parameter, it is not the most general[1] one that can be used to efficiently solve $\textsc{Hom}(H)$. Indeed, standard dynamic programming techniques can be used to obtain a $\mathcal{O}^*((2^{|V(H)|})^{\mathrm{cw}})$ time algorithm for the problem, where cw stands for *clique-width* [4]: a well-studied graph parameter that is bounded not only on all graph classes of bounded treewidth, but also on well-structured dense classes such as complete graphs. But is this basic algorithm generally optimal (mirroring the situation for treewidth [25]), or can one obtain better runtime dependencies on clique-width?

Contribution. Our aim is to obtain a detailed understanding of the fine-grained complexity of $\textsc{Hom}(H)$ in terms of the clique-width of G and the fixed target H. As a starting point for our investigation, we note that Lampis used the SETH to obtain tight bounds for c-Coloring

[1] There is a hierarchy of graph parameters (see, e.g., [1, Figure 1]), where parameter $\mathbb{A}$ is more general than parameter $\mathbb{B}$ if there are graph classes of bounded $\mathbb{A}$ and unbounded $\mathbb{B}$ but the opposite is not true.

with respect to clique-width [20]. Interestingly, already for this special case, the upper and lower bounds differ from those of the aforementioned simple dynamic programming algorithm: if H is a complete graph, then $\textsc{Hom}(H)$ can be solved in time $\mathcal{O}^*((2^{|V(H)|}-2)^{\mathrm{cw}})$ [20] and this is tight under the SETH. However, as noted by Piecyk and Rzążewski [27], it was not at all obvious how these bounds can be lifted to general choices of H.

In order to achieve our goals we need to improve upon the basic dynamic programming idea to identify a "hopefully correct" base of the exponent for every choice of H. Towards our first result, we identify a structural property of H called the *signature number* (denoted $s(H)$) which, intuitively, captures the number of non-trivial neighborhood classes of vertex subsets in H (the *signature set*). We then obtain a non-trivial dynamic programming algorithm that solves $\textsc{Hom}(H)$ in time where the base of the exponent is precisely the signature number. We note that $s(H)$ is $2^{|V(H)|}-2$ for complete graphs H, and so this result also provides a succinct and broader explanation for the running time of Lampis' algorithm [20].

▶ **Theorem 1.** *Let H be a fixed graph. $\textsc{Hom}(H)$ can be solved in time $\mathcal{O}^*(s(H)^{\mathrm{cw}(G)})$ for each input graph G, assuming an optimal clique-width expression of G is provided as part of the input.*

With this upper bound, we proceed to the main technical contribution of this paper: establishing a corresponding lower bound under the SETH. The main difficulty here is that we need a reduction that is delicate on one hand, since it needs to preserve the clique-width, but is on the other hand also flexible enough to work for many different choices of H; moreover, the reduction has to rely on the signature numbers of these graphs in some way.

To provide an intuitive description of the reduction, let us focus for now on the case where H is a projective core. On a high level, the main building block is an *S-gadget* which, given an arbitrary set S of pairs of vertices in H and two vertices p and q of the input graph G, ensures that every homomorphism f satisfies $(f(p), f(q)) \in S$. After providing a generic construction for such S-gadgets which is clique-width preserving and works for every valid choice of H, we use these to obtain *implication gadgets* and *or gadgets* which restrict how a solution homomorphism can behave on a selected set of vertices in G. The formalization of these gadgets is the main technical hurdle towards the desired result; once that is done, we can lift the idea used in the earlier reduction of Lampis [20] that established clique-width lower bounds for c-Coloring by reducing from Constraint Satisfaction (CSP) to $\textsc{Hom}(H)$. One crucial distinction in our reduction is that we use elements of the signature set (as opposed to color sets) to represent domain values in the CSP instance.

To lift these considerations to cases where H is not a projective core, we unfortunately need to add an extra layer of complexity. Similarly as in the previous treewidth-based lower bound for $\textsc{Hom}(H)$ [25], one can base this step on conjectures of Larose and Tardif [21,22] that classify all remaining targets as certain graph products with special properties (notably, all of the factors must be "truly projective"). The approach used for treewidth [25] was then to essentially repeat all steps of the proof for projective cores, with the added difficulty that one uses the properties of products instead of dealing directly with projective cores.

While this approach could be used here as well, instead we unify the two cases (H being a projective core, and H being a product) by defining the notion of W-projectivity for some factor W of H. In particular, if H is a projective core then it itself is H-projective, while if H is a product with truly projective factor H_i then it is H_i-projective. As our main result, we obtain an SETH-based lower bound which essentially shows that for each W-projective graph H, $s(W)$ is the optimal base of the clique-width exponent for solving $\textsc{Hom}(H)$:

▶ **Theorem 2.** *Let H be a fixed non-trivial core with prime factorization $H_1 \times \ldots \times H_m$. Assume that H is H_i-projective for some $i \in [m]$. Then there is no algorithm solving $\textsc{Hom}(H)$ in time $\mathcal{O}^*((s(H_i) - \varepsilon)^{\mathrm{cw}(G)})$ for any $\varepsilon > 0$, unless the* SETH *fails.*

By also deliberately considering prime factorizations in the algorithm which we provide for Theorem 1, we can obtain an upper bound on the complexity of $\textsc{Hom}(H)$ that matches the lower bound from Theorem 2. For a discussion explicitly relating these complexity bounds in the context of the aforementioned conjectures of Larose and Tardif, we refer to Section 6.

2 Preliminaries

We use standard terminology for graph theory [7]. Let $[i]$ denote the set $\{1, \ldots, i\}$. For a mapping $f : A \to B$ and $A' \subseteq A$, let $f|_{A'}$ denote the restriction of f to A'. We will use the $\mathcal{O}^*(\cdot)$ notation to suppress factors polynomial in the input size.

Homomorphisms and Cores

For two graphs G and H, a *homomorphism* from G to H is a mapping $h : V(G) \to V(H)$, such that for every $uv \in E(G)$ we have $h(u)h(v) \in E(H)$. If there exists a homomorphism from G to H, we denote this fact by $G \to H$, and if h is a homomorphism from G to H, we denote that by $h : G \to H$. If there is no homomorphism from G to H, we write $G \not\to H$. If $G \to H$ and $H \to G$, we say that G and H are *homomorphically equivalent*. In particular, since the composition of homomorphisms is a homomorphism, if G and H are homomorphically equivalent, then for every graph F we have that $F \to G$ if and only if $F \to H$. It is straightforward to verify that homomorphic equivalence is an equivalence relation on the class of all graphs. On the other hand, if $G \not\to H$ and $H \not\to G$ for some graphs G, H, we say that G and H are *incomparable*.

We say that a graph H is a *core* if every homomorphism $h : H \to H$ is an automorphism. Equivalently, H is a core if for every proper induced subgraph H' of H it holds that $H \not\to H'$. We say that a core H' is a *core of* H if H' is an induced subgraph of H and $H \to H'$. Clearly, each core graph is a core of itself. Each graph has a unique (up to isomorphism) core, and the core of H can be equivalently defined as the smallest (with respect to the number of vertices) graph that is homomorphically equivalent with H [16].

A graph H is *ramified* if $N(u) \not\subseteq N(v)$ for every two distinct vertices u, v of H. Observe that each core is ramified; otherwise one could define $f : H \to H$ that is an identity on all vertices of H but u and set $f(u) = v$. This would be a homomorphism to a proper subgraph of H, contradicting the fact that H is a core.

We say that a graph H is *trivial* if its core has at most two vertices.

▶ **Observation 3** ([15])**.** *A graph H is trivial if and only if it is either bipartite or contains a vertex with a loop.*

Proof. It is straightforward to observe that there exist three trivial cores: K_1, K_2, and K_1^*, where by K_1^* we denote the graph that consists of one vertex with a loop.

If H contains a vertex with a loop, then K_1^* is the core of H. If H is bipartite, then the core of H is either K_1 (if H has no edges) or K_2 (since mapping the vertices of one bipartition class to one vertex of K_2, and another bipartition graph to the other, is a homomorphism).

For the other direction, assume that H is a non-bipartite loopless graph. Since it is loopless, K_1^* cannot be its core. Clearly, H has at least one edge, and therefore $H \not\to K_2$ Moreover, H contains an odd cycle C_{2k+1} as a subgraph, hence, $C_{2k+1} \to H$. If now $H \to K_2$, composition of these homomorphism gives that $C_{2k+1} \to K_2$, which is equivalent to stating that C_{2k+1} is 2-colorable, a contradiction. ◀

Observe that trivial cores H correspond precisely to the polynomial cases of the $\textsc{Hom}(H)$ problem. Since our aim is to focus on the NP-hard cases of the problem, from here onward we will assume that the target graph is non-trivial.

Signature Sets

For a vertex v of a graph H, let $N_H(v)$ denote the set of neighbors of v in H. If the graph is clear from the context, we will omit the subscript H and write $N(v)$.

For a non-empty set $T \subseteq V(H)$ we say that $S(T)$ is the *signature set of* T if $S(T) = \bigcap_{t \in T} N(t)$. We say that a non-empty set $S \subseteq V(H)$ is a *signature set*, if there exists T such that $S = S(T)$. We denote by $\mathcal{S}(H)$ the set of all signature sets of H, and we note that $\{V(H), \emptyset\} \cap \mathcal{S}(H) = \emptyset$.

▶ **Observation 4.** *If T is a proper non-empty subset of $V(H)$, and $a \in T$, $b \in S(T)$, then $ab \in E(H)$. Moreover, for non-empty subsets $A, B \subseteq V(H)$, $S(A \cup B) = S(A) \cap S(B)$.*

We note that the operation of taking a signature set is reversible on $\mathcal{S}(H)$:

▶ **Observation 5.** *For every $A \in \mathcal{S}(H)$, $S(S(A)) = A$.*

Proof. By the definition of signature set, $A \times S(A) \subseteq E(H)$, so $A \subseteq S(S(A))$. For the converse direction observe that as $A \in \mathcal{S}(H)$, there exists a non-empty subset T of $V(H)$ such that $A = S(T)$. Pick any $x \in S(S(A))$, then $E(H) \supseteq \{x\} \times S(A) = \{x\} \times S(S(T)) \supseteq \{x\} \times T$. Hence by definition $x \in S(T) = A$. ◀

Let the *signature number* of H, denoted $s(H)$, be defined as $|\mathcal{S}(H)|$. As mentioned in the introduction, the signature number will play a crucial role in our upper and lower bounds.

Observe that, if H is a target and hence non-trivial, for every nonempty $T \subseteq V(H)$ we have that $S(T) \cap T = \emptyset$. From that it is easy to see that $V(H)$ never belongs to $\mathcal{S}(H)$. Since, by definition, $\emptyset \notin \mathcal{S}(H)$, we get the following bounds for $s(H)$.

▶ **Observation 6.** *Let H be a graph with no loops. Then $s(H) \leqslant 2^{|V(H)|} - 2$.*

Notice that since $2^{|V(H)|} - 2$ is the number of all proper non-empty subsets of $V(H)$, the equality in Observation 6 holds if and only if H is a clique.

If $S \in \mathcal{S}(H)$, we call T such that $S(T) = S$ a *witness* of S. Clearly, we can have distinct T_1, T_2 such that $S(T_1) = S(T_2)$, however, notice that in such a case there exists $T = T_1 \cup T_2$ such that $S(T) = S(T_1) = S(T_2)$. Hence, there exists a unique maximal (with respect to inclusion) witness of S, and we denote it by $M(S)$. In fact, it is not difficult to see that $M(S) = \{v \in V(H) \mid S \subseteq N_H(v)\}$; for $S(M(S)) = S$ to hold, it is clearly necessary that $S \subseteq N_H(v)$ for all $v \in M(S)$. On the other hand, as $M(S)$ is maximal all v for which this is true are contained in $M(S)$.

In this way signature sets and their witnesses are in one-to-one correspondence. While not necessary to obtain our algorithmic and lower bounds for $\textsc{Hom}(H)$ parameterized by clique-width, this offers an alternative perspective on the role of signatures in our results.

In fact, the signature number could equivalently be defined as the "maximal witness number" and signature sets could be replaced by maximal witnesses in all our proofs: Let $\mathcal{M}(H) = \{M(S) : S \in \mathcal{S}(H)\}$.

▶ **Observation 7.** $\mathcal{S}(H) = \mathcal{M}(H)$.

Proof. Let T be a fixed non-empty subset of $V(H)$ such that $S(T) \neq \emptyset$. Observe that the definition and the maximality of $M(S(T))$ implies that $S(S(T)) = \bigcap_{t \in S(T)} N(t) = M(S(T))$ Since T and $S(T)$ are non-empty, we get that $\mathcal{S}(H) \supset \mathcal{M}(H)$. For the other direction observe that $M(M(S(T))) = S(T)$. Assume the contrary, then there exists a vertex $v \notin S(T)$ such that $N(v) \supset M(S(T))$. However, $T \subset M(S(T)) \subset N(v)$ meaning that $v \in S(T)$, a contradiction. Thus, $\mathcal{S}(H) = \mathcal{M}(H)$. ◀

We note that if H is a core graph, we can also bound the minimum cardinality of $\mathcal{S}(H)$.

▶ **Observation 8.** *Let H be a core graph, $H \neq K_1$. Then $s(H) \geqslant |V(H)|$.*

Proof. Observe that if H is a core distinct from K_1, then it does not contain isolated vertices. Therefore, for each $v \in V(H)$ we have $N(v) \in \mathcal{S}(H)$. On the other hand, since H is a core, it is ramified. In particular, for every distinct $v, w \in V(H)$ we have $N(v) \neq N(w)$. Hence different vertices give rise to different signature sets. ◀

Clique-Width and Clique-Width Expressions

For a positive integer k, we let a *k-graph* be a graph whose vertices are labeled by $[k]$. For convenience, we consider a graph to be a k-graph with all vertices labeled by 1. We call the k-graph consisting of exactly one vertex v (say, labeled by i) an initial k-graph and denote it by $i(v)$.

The *clique-width* of a graph G is the smallest integer k such that G can be constructed from initial k-graphs by means of iterative application of the following three operations:

1. Disjoint union (denoted by $\oplus$);
2. Relabeling: changing all labels i to j (denoted by $\rho_{i \to j}$);
3. Edge insertion: adding an edge from each vertex labeled by i to each vertex labeled by j ($i \neq j$; denoted by $\eta_{i,j}$).

A construction of a k-graph G using the above operations can be represented by an algebraic term composed of $\oplus$, $p_{i \to j}$ and $\eta_{i,j}$ (where $i \neq j$ and $i, j \in [k]$). Such a term is called a k-expression defining G, and we often view it as a tree with each node labeled with the appropriate operation. Conversely, we call the k-graph that arises from a k-expression its *evaluation*. The *clique-width* of G is the smallest integer k such that G can be defined by a k-expression which we then also call a *clique-width expression* of G.

Many graph classes are known to have constant clique-width; examples include all graph classes of constant treewidth and co-graphs [5]. Moreover a fixed-parameter algorithm is known to compute a k-expression of the input where k is bounded in $f(\mathrm{cw})$ [26].

3 Algorithm

As our first contribution, we obtain an algorithm that will play a crucial role for upper-bounding the fine-grained complexity of $\textsc{Hom}(H)$.

▶ **Theorem 1.** *Let H be a fixed graph. $\textsc{Hom}(H)$ can be solved in time $\mathcal{O}^*(s(H)^{\mathrm{cw}(G)})$ for each input graph G, assuming an optimal clique-width expression of G is provided as part of the input.*

Proof. Assume, w.l.o.g., that G is connected and $|V(G)| > 1$. We will describe a dynamic program that proceeds in a leaf-to-root fashion along the provided k-expression σ of G. For a subexpression $\tau \subseteq \sigma$, we denote the evaluation of τ by G_τ, and by $V_\tau^i \subseteq V(G_\tau)$ the vertex

set that has label i in G_τ. We say that i is a *live label* in τ if there is an edge of G which is incident to V_τ^i and does not appear in G_τ. Denote the set of live labels in τ by L_τ. Since G is connected, $L_\tau \neq \emptyset$ for any proper subexpression τ of σ.

For each subexpression τ of σ, we will compute a set P_τ consisting of functions $p\colon L_\tau \to \mathcal{S}(H)$ where $p \in P_\tau$ if and only if there exists a homomorphism h_p from G_τ to H such that $p(i) \subseteq S(h_p(V_\tau^i))$, $i \in L_\tau$. We will say that $p \in P_\tau$ *describes* the homomorphism h_p in τ or, equivalently, that h_p *witnesses* p in τ. Intuitively, we will use $p(i)$ to preemptively store the images of the neighbors of V_τ^i in the final graph G – that is why we store not only the exact signature, but all signatures that occur as subsets. We remark that storing the "current" images of the neighbors of V_τ^i in G_τ would be sufficient to obtain a conceptually simpler fixed-parameter algorithm parameterized by clique-width, but in that case it is not obvious how one can avoid a quadratic dependency on clique-width in the exponent.

Observe that for any homomorhism $h\colon G_\tau \to H$, images of vertices with live labels should be connected in H with images of their future neighbors. In particular, for any $i \in L_\tau$, $S(h(V_\tau^i)) \neq \emptyset$ and hence $S(h(V_\tau^i)) \in \mathcal{S}(H)$. Therefore h is described in τ by some $p \in P_\tau$. By definition, $L_\sigma = \emptyset$ and hence G is homomorphic to H if and only if P_σ contains the empty mapping, i.e., if $P_\sigma = \{\emptyset\}$ (as opposed to $P_\sigma = \emptyset$). It remains to show how to correctly compute each P_τ. To do so, we distinguish based on the outermost operation of τ:

(a) $\tau = i(v)$ for some $i \in [\mathrm{cw}(G)]$. In this case $L_\tau = \{i\}$, and P_τ contains all functions $p : \{i\} \mapsto \mathcal{S}(H)$ such that $p(i) \subseteq N_H(u)$ for some $u \in V(H)$.

(b) $\tau = \rho_{i\to j}(\tau')$ and $P_{\tau'}$ has already been computed. If $i \notin L_{\tau'}$, we can correctly set $L_\tau = L_{\tau'}$ and $P_\tau = P_{\tau'}$. If $i \in L_{\tau'}$ and $j \notin L_{\tau'}$, then $L_\tau = (L_{\tau'} \setminus \{i\}) \cup \{j\}$ and

$$P_\tau = \left\{ p\colon L_\tau \to \mathcal{S}(H) \;\middle|\; \exists p' \in P_{\tau'} : \quad p(\ell) = \begin{cases} p'(\ell) & \text{if } \ell \neq j \\ p'(i) & \text{if } \ell = j \end{cases} \right\}.$$

Finally, if $\{i, j\} \subseteq L_{\tau'}$, then $L_\tau = L_{\tau'} \setminus \{i\}$ and $P_\tau = \{p'|_{L_\tau} \mid p' \in P_{\tau'} \wedge p'(i) = p'(j)\}$.

For correctness in the last case, let h be a homomorphism from G_τ to H and $S_\ell \in \mathcal{S}(H)$ be such that $S_\ell \subseteq S(h(V_\tau^\ell))$, $\ell \in L_\tau$. Observe that $V_\tau^\ell = V_{\tau'}^\ell$ for $\ell \in L_\tau \setminus \{j\}$ and $V_\tau^j = V_{\tau'}^j \cup V_{\tau'}^i$. In particular, $S_j \subseteq S(h(V_{\tau'}^i))$ and $S_j \subseteq S(h(V_{\tau'}^j))$. By definition of $P_{\tau'}$, there exists $p' \in P_{\tau'}$ such that $p'(\ell) = S_\ell$ for $\ell \in L_\tau \setminus \{j\}$ and $p'(i) = p'(j) = S_j$. The function $p \in P_\tau$, defined by $p = p'|_{L_\tau}$, satisfies $p'(\ell) = S_\ell$, $\ell \in L_\tau$.

On the other hand, fix some $p \in P_\tau$. Let $p' \in P_{\tau'}$ be a function such that p arises from p' in the construction of P_τ. Consider a witness h of p' in τ'. For every $\ell \in L_\tau \setminus \{j\}$ we have $V_\tau^\ell = V_{\tau'}^\ell$ and so $p(\ell) = p'(\ell) \subseteq S(h(V_\tau^\ell))$. Moreover, $p(j) = p'(j) \subseteq S(h(V_{\tau'}^j))$ and $p(j) = p'(i) \subseteq S(h(V_{\tau'}^i))$. By Observation 4, we have $p(j) \subseteq S(h(V_{\tau'}^i)) \cap S(h(V_{\tau'}^j)) = S(h(V_{\tau'}^i \cup V_{\tau'}^j)) = S(h(V_\tau^j))$. Hence p witnesses h in τ.

(c) $\tau = \tau^{(1)} \oplus \tau^{(2)}$ where $P_{\tau^{(1)}}$ and $P_{\tau^{(2)}}$ have already been computed. In this case $L_\tau = L_{\tau^{(1)}} \cup L_{\tau^{(2)}}$. We define

$$P_\tau = \{p = p_1 \cup p_2 \mid p_1 \in P_{\tau^{(1)}} \wedge p_2 \in P_{\tau^{(2)}} \wedge (\forall \ell \in L_{\tau^{(1)}} \cap L_{\tau^{(2)}} : p_1(\ell) = p_2(\ell))\}$$

Intuitively, we construct a homomorphism on the disjoint union of two subgraphs by "gluing together" the homomorphisms on the subgraphs. If the subgraphs share any live labels, after this step they will all be treated equally. For this reason we require the images of the neighbors of such labels to be the same in both subgraphs.

For correctness, let h be a homomorphism from G_τ to H and $S_\ell \in \mathcal{S}(H)$ be such that $S_\ell \subseteq S(h(V_\tau^\ell))$, $\ell \in L_\tau$. Observe that for every $\ell \in L_{\tau^{(1)}} \cap L_{\tau^{(2)}}$, $V_\tau^\ell \supseteq V_{\tau^{(i)}}^\ell$, so $S_\ell \subseteq S(h(V_{\tau^{(i)}}^\ell))$, $i = 1, 2$. By definition, there exists $p_i \in P_{\tau^{(i)}}$ such that $p_i(\ell) = S_\ell$ for $\ell \in L_{\tau^{(i)}}$, $i = 1, 2$. For $p = p_1 \cup p_2$ we have $p(\ell) = S_\ell$, $\ell \in L_\tau$.

For the converse, fix some $p \in P_\tau$. Let $p_1 \in P_{\tau^{(1)}}$, $p_2 \in P_{\tau^{(2)}}$ be functions such that $p = p_1 \cup p_2$. Let h_i be a witness of p_i in $\tau^{(i)}$, $i = 1, 2$. We define $h = h_1 \cup h_2$. Since G_τ doesn't contain edges between $V(G_{\tau^{(1)}})$ and $V(G_{\tau^{(2)}})$, h is a homomorphism from G_τ to H. For all $\ell \in L_{\tau^{(1)}} \setminus L_{\tau^{(2)}}$, we have $p(\ell) = p_1(\ell) \subseteq S(h_1(V_\tau^\ell)) = S(h(V_\tau^\ell))$, similarly for $\ell \in L_{\tau^{(1)}} \setminus L_{\tau^{(2)}}$. For $\ell \in L_{\tau^{(1)}} \cap L_{\tau^{(2)}}$, we have $p(\ell) = p_i(\ell) \subseteq S(h_i(V_{\tau^{(i)}}^\ell))$, $i = 1, 2$, so $p(\ell) \subseteq S(h_1(V_{\tau^{(1)}}^\ell)) \cap S(h_2(V_{\tau^{(2)}}^\ell)) = S(h(V_\tau^\ell))$. Hence h is a witness of p in τ.

(d) $\tau = \eta_{i,j}(\tau')$ and $P_{\tau'}$ has already been computed. In this case $L_\tau = L_{\tau'} \setminus I$ where $I \subseteq \{i, j\}$ is the set of live labels in τ' that are no longer live labels in τ. We set $P_\tau = \{p : L_\tau \to \mathcal{S}(H) \mid \exists p' \in P_{\tau'} \colon \big(p'(i) \supseteq S(p'(j)) \wedge p|_{L_\tau \setminus \{i,j\}} = p'|_{L_\tau \setminus \{i,j\}} \wedge p(i) \subseteq p'(i) \wedge p(j) \subseteq p'(j)\big)\}$.

Intuitively, we can add the edges between two live labels if and only if there are edges between their images in H. Our restriction on p' is an expression of this condition in terms of images of neighbors and their signatures. Indeed, for correctness, let h be a homomorphism from G_τ to H and $S_\ell \in \mathcal{S}(H)$ be such that $S_\ell \subseteq S(h(V_\tau^\ell))$, $\ell \in L_\tau$. There exists $p' \in P_{\tau'}$ such that $p'(\ell) = S_\ell$ for all $\ell \in L_\tau \setminus \{i, j\}$, $p'(i) = S(h(V_\tau^i))$ and $p'(j) = S(h(V_\tau^j))$. As h is a homomorphism, we have $h(V_{\tau'}^i) \times h(V_{\tau'}^j) \subseteq E(H)$, which means that $S(h(V_{\tau'}^j)) \supseteq h(V_{\tau'}^i)$, i.e. $p'(j) \supseteq h(V_{\tau'}^i)$. Then $S(p'(j)) \subseteq S(h(V_{\tau'}^i)) = p'(i)$ and hence p' gives rise to $p \in P_\tau$ such that $p(\ell) = S_\ell$, $\ell \in L_\tau$.

On the other hand, let $p \in P_\tau$ arise from $p' \in P_{\tau'}$. Consider a witness $h \colon G_{\tau'} \to H$ of p' in τ'. To see that h preserves edges between $V_{\tau'}^i$ and $V_{\tau'}^j$, recall that $p'(i) \supseteq S(p'(j))$, so $S(h(V_{\tau'}^i)) \supseteq p'(i) \supseteq S(p'(j)) \supseteq S(S(h(V_{\tau'}^j))) \supseteq h(V_{\tau'}^j)$. Hence $h(V_{\tau'}^i) \times h(V_{\tau'}^j) \subseteq E(H)$ and h is a homomorphism from G_τ to H. By construction, for every $\ell \in L_\tau$ it holds that $p(\ell) \subseteq p'(\ell) \subseteq S(h(V_\tau^\ell))$, so h witnesses p in τ.

It is easy to verify that $|P_\tau| \leqslant s(H)^{\mathrm{cw}(G)}$ for each subexpression τ of σ. This means that in each step, the computation requires time $\mathcal{O}(\mathrm{cw}(G) s(H)^2 s(H)^{\mathrm{cw}((G))})$. Overall this yields a complexity of $\mathcal{O}(|V(G)| \mathrm{cw}(G) s(H)^2 s(H)^{\mathrm{cw}((G))}) \subseteq \mathcal{O}^*(s(H)^{\mathrm{cw}(G)})$. ◀

4 On Products and Projectivity

While Theorem 1 will serve as the upper bound that will match our target SETH-based lower bounds for $\textsc{Hom}(H)$ for the "most difficult" choices of H, in many cases one can in fact supersede the algorithm's runtime by exploiting well-known properties of target graphs.

As a simple example showcasing this, consider the *wheel* graph W_6 (see Figure 1). Since W_6 is 3-colorable, it holds that $W_6 \to K_3$, and since K_3 is a core and an induced subgraph of W_6, it is the core of W_6. We recall that if H is a core of H', then for every graph G it holds that $G \to H$ if and only if $G \to H'$. Hence, having an instance G of $\textsc{Hom}(W_6)$, we can compute a core of W_6 (since we assume that the target graph is fixed, this can be done in constant time), and use Theorem 1 for $H = K_3$ to decide whether $G \to W_6$ in total running time $\mathcal{O}^*(s(K_3)^{\mathrm{cw}(G)})$. As $s(K_3) < s(W_6)$ (as showcased in Figure 1), this yields a better running time bound than the direct use of Theorem 1. While this example shows that the signature number can decrease by taking an induced subgraph, we remark that it can never increase.

▶ **Observation 9.** *Let H and H' be graphs such that H is an induced subgraph of H'. Then $s(H) \leqslant s(H')$.*

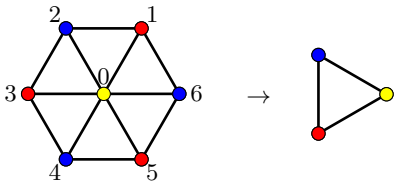


Figure 1 The graphs W_6 (left) and K_3 (right). Colors on the vertices of W_6 indicate the homomorphism $h : W_6 \to K_3$. We note that $\{\{0\}\} \cup \{\{0, i\} \mid i \in [6]\} \subseteq \mathcal{S}(W_6)$. Since K_3 is a clique, we have $s(K_3) = 6 < 7 \leqslant s(W_6)$.

Proof. Given a connected graph Q without loops, one may consider an equivalence relation $\sim_Q$ on the set of proper nonempty subsets of $V(Q)$ defined as follows: $V_1 \sim_Q V_2$ if and only if V_1 and V_2 have the same signature sets in Q. Observe that $s(Q)$ is equal to the number of equivalence classes of $\sim_Q$. Hence to prove the claim, it suffices to show that whenever two sets belong to different equivalence classes of $\sim_H$, they also belong to different equivalence classes of $\sim_{H'}$. For this, consider any two proper non-empty subsets V_1 and V_2 of $V(H)$ such that $V_1 \nsim_H V_2$. Without loss of generality, we assume that there exists $v \in (\bigcap_{t \in V_1} N_H(t)) \setminus (\bigcap_{t \in V_2} N_H(t))$. Then $vt \in E(H) \subseteq E(H')$ for every $t \in V_1$, i.e., v belongs to the signature set of V_1 in H'. On the other hand, $vt_0 \notin E(H)$ for some $t_0 \in V_2$. As H is induced subgraph of H', it means that $vt_0 \notin E(H')$, so v doesn't belong to the signature set of V_2 in H' and hence $V_1 \nsim_{H'} V_2$. ◀

At this point, we may ask whether the procedure of simply computing the unique core H of the fixed target H' and then applying Theorem 1 for H could yield a tight upper bound for $\textsc{Hom}(H')$. Unfortunately, the situation is more complicated than that, and we need to introduce a few important notions in order to capture the problem's fine-grained complexity.

Let the *direct product* $H_1 \times H_2$ of graphs H_1, H_2 be the graph defined as follows:

$$V(H_1 \times H_2) = V(H_1) \times V(H_2),$$
$$E(H_1 \times H_2) = \{(x_1, x_2)(y_1, y_2) : x_i y_i \in E(H_i) \text{ for every } i \in \{1, 2\}\}.$$

We call H_1 and H_2 the *factors* of $H_1 \times H_2$. Clearly, the operation $\times$ is commutative, and since it is also associative, we can naturally extend the definition of direct product to more than two factors, i.e., $H_1 \times H_2 \times \ldots \times H_m = H_1 \times (H_2 \times \ldots \times H_m)$. Note that for every graph H that contains at least one edge, it holds that $H \times K_1^* = H$.

In the remaining part of the paper we will often consider vertices that are tuples. If such a vertex is an argument of some function and in cases where this does not lead to confusion, we omit one pair of brackets; similarly, we omit internal brackets in nested tuples where this does not lead to confusion. Moreover, for any graph H and for an integer ℓ, we denote by H^ℓ the graph $\overbrace{H \times \ldots \times H}^{\ell}$. As an example, instead of writing $((x_1, x_2), y_1) \in ((H_1 \times H_1) \times H_2)$, we write $(x_1, x_2, y_1) \in (H_1^2 \times H_2)$.

If $H = H_1 \times \ldots \times H_m$ for some graphs $H_1, \ldots, H_m$, we say that $H_1 \times \ldots \times H_m$ is a *factorization* of H. A graph H on at least two vertices is *prime* if the fact that $H = H_1 \times H_2$ for some graphs H_1, H_2 implies that $H_1 = K_1^*$ or $H_2 = K_1^*$. If H has a factorization $H_1 \times \ldots \times H_m$ such that for every $i \in [m]$ the graph H_i is prime, we call $H_1 \times \ldots \times H_m$ a *prime factorization* of H.

▶ **Theorem 10** ([8, 14]). *Any connected non-bipartite graph with more than one vertex has a unique prime factorization (into factors with possible loops).*

Consider a graph $H_1 \times \ldots \times H_m$, and let $i \in [m]$. A mapping $\pi_i : V(H_1 \times \ldots \times H_m) \to V(H_i)$ such that $\pi_i(x_1, \ldots, x_\ell) = x_i$ is called the (i-th) *projection* of $H_1 \times \ldots \times H_m$. Clearly, such a mapping is always a homomorphism.

▶ **Observation 11.** *Let $G, H_1, \ldots, H_m$ be graphs. Then $G \to H_1 \times \ldots \times H_m$ if and only if for every $i \in [m]$ we have $G \to H_i$.*

Proof. Let $f : G \to H_1 \times \ldots \times H_m$. Then for every $i \in [m]$ (by $[m]$ we denote the set of integers $\{1, \ldots, m\}$) we have a homomorphism $f_i : G \to H_i$. Conversely, if for every $i \in [m]$ we have $g_i : G \to H_i$, then we can define $g : G \to H_1 \times \ldots \times H_m$ as $g(x) = (g_1(x), \ldots, g_m(x))$. ◀

Crucially, in some cases Observation 11 allows us to improve the bounds given by Theorem 1 even if H is a core, simply by considering all possible factorizations of H.

▶ **Corollary 12.** *Let H be a graph with factorization $H_1 \times \ldots \times H_m$, and let G be an instance graph of Hom(H). Assuming that the clique-width expression σ of G of width $\mathrm{cw}(G)$ is given, the Hom(H) problem can be solved in time $\max_{i \in [m]} \mathcal{O}^*\big(s(H_i)^{\mathrm{cw}(G)}\big)$.*

Proof. Observe that if G is an instance of Hom(H), by Theorem 1 for every $i \in [m]$ we can decide whether $G \to H_i$ in time $\mathcal{O}^*\big(s(H_i)^{\mathrm{cw}(G)}\big)$. Then, if G is a yes-instance of Hom(H_i) for every $i \in [m]$, we return that G is a yes-instance of Hom(H). Otherwise, we return that G is a no-instance of Hom(H). The correctness of this procedure follows from Observation 11. ◀

On the other hand, the notion of signature sets we introduced in the previous section behaves multiplicatively with respect to taking direct product of graphs.

▶ **Observation 13.** *Let $H = H_1 \times H_2$. Then $\mathcal{S}(H) = \mathcal{S}(H_1) \times \mathcal{S}(H_2)$.*

Proof. We prove that $\mathcal{S}(H)$ is of form $\{S(T_1) \times S(T_2) : T_i \subseteq V(H_i), S(T_i) \neq \emptyset \text{ for } i = 1, 2\}$.

Let T_1 and T_2 be some subsets of, respectively, $V(H_1)$ and $V(H_2)$. Clearly,

$$S(T_1) \times S(T_2) = [\bigcap_{t \in T_1} N(t)] \times [\bigcap_{t' \in T_2} N(t')] = \bigcap_{(t,t') \in T_1 \times T_2} N(t, t') = S(T_1 \times T_2). \tag{1}$$

Therefore, if $S(T_1)$ and $S(T_2)$ are non-empty, we get that $S(T_1) \times S(T_2) \in \mathcal{S}(H)$.

To see that $\mathcal{S}(H) \subseteq \mathcal{S}(H_1) \times \mathcal{S}(H_2)$, we show that for every $T \subseteq V(H)$ set $S(T)$ is of the form $S(T_1) \times S(T_2)$ for some T_1, T_2. Define T_1 and T_2 to be minimal sets such that $T \subseteq T_1 \times T_2$. Hence, by (1), $S(T_1) \times S(T_2) = S(T_1 \times T_2) \subseteq S(T)$. On the other hand, for every $(s, s') \in S(T)$ we have $s \in \bigcap_{t \in T_1} N(t)$ and $s' \in \bigcap_{t' \in T_2} N(t')$, so the equality follows. ◀

It follows from Observation 13 that if H is a graph with factorization $H_1 \times \ldots \times H_m$, then $s(H) = s(H_1) \cdot \ldots \cdot s(H_m)$. Therefore if there exist at least two factors H_i, H_j such that $s(H_i), s(H_j) > 1$, Corollary 12 yields a better running time than Theorem 1.

In order to analyze the possible matching lower bounds for our algorithms, in the remaining part of the section, we focus only on connected non-trivial cores H that are provided with their unique prime factorization $H_1 \times \ldots \times H_m$; if H is prime, we technically consider this factorization to be $H \times K_1^*$ (noting that this is not a prime factorization, and that K_1^* is the only non-simple graph in this article). We note that the factors of a core must satisfy some necessary conditions.

▶ **Observation 14** ([25])**.** *Let H be a connected, non-trivial core with factorization $H = H_1 \times \ldots \times H_m$ such that $H_i \neq K_1^*$ for all $i \in [m]$. Then for every $i \in [m]$ the graph H_i is a connected non-trivial core, incomparable with H_j for $j \in [m] - \{i\}$.*

Observation 14 in particular implies that if H is a connected non-trivial graph with factorization $H_1 \times \ldots \times H_m$, then at least one of the factors H_i must be non-trivial, and that K_1 and K_2 never appear as factors of a connected non-trivial graph.

In the remaining part of this section we introduce a few more important definitions, in particular, the well-established notion of *projectivity* for non-trivial graphs H.

We say that a homomorphism $f : H^\ell \to H$, for some $\ell \geqslant 2$, is *idempotent* if for each $x \in V(H)$ it holds that $f(x, \ldots, x) = x$. Graph H is *projective* if for every $\ell \geqslant 2$, every idempotent homomorphism $f : H^\ell \to H$ is a projection. We note that every projective graph on at least three vertices must be connected, ramified, non-bipartite and prime [22].

Here, we introduce a generalization of the projectivity property for non-trivial cores, which turns out to be the central component required to establish the lower bound for our problem. As a first step towards this, we lift the notion of idempotency as follows. Let H be a connected, non-trivial prime core, and let W be either a connected core on at least three vertices incomparable with H, or the graph K_1^*. Let $f : A \to H$ be a homomorphism where $A = H^\ell \times W$; observe that ℓ is uniquely determined by either W being incomparable with the prime core H, or W being K_1^*. We say that f is *H-idempotent* if for each $x \in V(H), y \in V(W)$ it holds that $f(x, \ldots, x, y) = x$.

Now, let us consider a non-trivial core H which admits a prime factorization $H_1 \times \ldots \times H_m$ and let $i \in [m]$. We say that H is *H_i-projective* if H_i is non-trivial and every H_i-idempotent homomorphism $f : H_1 \times \ldots \times H_{i-1} \times H_i^\ell \times H_{i+1} \times \ldots \times H_m \to H_i$ is a projection. In other words, for every homomorphism $f : H_1 \times \ldots H_{i-1} \times H_i^\ell \times H_{i+1} \times \ldots \times H_m \to H_i$ such that for every $x \in V(H_i), y_j \in V(H_j)$ for $j \in [m] - \{i\}$ it holds that $f(y_1, \ldots, y_{i-1}, x, \ldots, x, y_{i+1}, \ldots, y_m) = x$, we must have that there exists $q \in \{i, \ldots, i + \ell - 1\}$ such that $f \equiv \pi_q$. Recall that if H is a non-trivial projective core, then it must be prime, so $H \times K_1^*$ is its only possible factorization. It is straightforward to verify that in a such case H is H-projective.

Since the direct product of graphs is commutative, if $H = H_1 \times \ldots \times H_m$ is H_i-projective for some $i \in [m]$, to simplify the notation we will often assume w.l.o.g. that $i = 1$.

5 Hardness

In this section, we focus on establishing the desired lower bounds, stated below.

▶ **Theorem 2.** *Let H be a fixed non-trivial core with prime factorization $H_1 \times \ldots \times H_m$. Assume that H is H_i-projective for some $i \in [m]$. Then there is no algorithm solving $\textsc{Hom}(H)$ in time $\mathcal{O}^*((s(H_i) - \varepsilon)^{\mathrm{cw}(G)})$ for any $\varepsilon > 0$, unless the* SETH *fails.*

We divide our proof into two main steps. First, we show that in our setting, instead of considering the $\textsc{Hom}(H)$ problem, we may focus on the Homomorphism Extension problem, denoted $\textsc{HomExt}(H)$. For a fixed H, $\textsc{HomExt}(H)$ takes as an instance a pair (G', h'), where $h' : V' \to V(H)$ is a mapping from some $V' \subseteq V(G')$. We ask whether there exists *an extension* of h' to G', i.e., a homomorphism $h : G' \to H$ such that $h|_{V'} \equiv h'$.

The $\textsc{HomExt}(H)$ is clearly a generalization of the $\textsc{Hom}(H)$ problem. However, as the first step of our proof, we show that if H is a fixed non-trivial core, each instance (G', h') of $\textsc{HomExt}(H)$ can be transformed in polynomial time into an instance G of $\textsc{Hom}(H)$, such that $\mathrm{cw}(G')$ and $\mathrm{cw}(G)$ differ only by a constant.

▶ **Theorem 15.** *Let H be a fixed non-trivial core. Given an instance (G', h') of $\textsc{HomExt}(H)$, we can construct an equivalent instance G of $\textsc{Hom}(H)$ such that $\mathrm{cw}(G) \leqslant \mathrm{cw}(G') + |V(H)|$.*

Proof. Let $V' \subseteq V(G')$ be the domain of h'. We construct G by taking a copy $\hat{G}'$ of G' and a copy $\hat{H}$ of H. Then, for every $v \in V'$ we add all the edges with one endpoint in v and another one in $N_{\hat{H}}(h'(v))$.

Observe that if there exists an extension $h : G' \to H$ of h', then h can be also extended to G, by setting $h(v) = v$ for every $v \in V(\hat{H})$. Indeed, let $uv \in E(G)$. If $u, v \in V(\hat{G}')$, then $h(u)h(v) \in E(H)$, by definition of the extension of h' to G'. If $u, v \in V(\hat{H})$, then $h(u)h(v) = uv \in E(H)$. Finally, assume that $u \in V(\hat{G}'), v \in V(\hat{H})$. Note that, by definition of G, this can happen only if $u \in V'$ and v is adjacent to $h'(u)$. Hence, $h(u)h(v) = h'(u)v \in E(H)$.

For the reverse direction, assume that there exists a homomorphism $f : G \to H$. We show that there exists an extension $h : G' \to H$ of h'. Let $\sigma : H \to H$ be a restriction of f to H. Since H is a core, σ is an automorphism of H. We claim that $g = \sigma^{-1} \circ f : G \to H$, restricted to G', is an extension of h'. Clearly, g is a composition of homomorphisms, so also a homomorphism. Therefore, it remains to show that for every $v \in V'$ we have $h'(v) = g(v)$. Since H is a core, and $N_H(h'(v)) \subseteq N_G(v)$, we have that $f(v) = f(h'(v))$. It follows that $g(v) = \sigma^{-1} \circ f(v) = \sigma^{-1} \circ f(h'(v)) = g(h'(v))$. However, recall that for every $u \in V(H)$ we have that $g(u) = u$, so in particular, $g(h'(v)) = h'(v)$.

To see that $\mathrm{cw}(G) \leqslant \mathrm{cw}(G') + |V(H)|$, observe that we added exactly $|V(H)|$ vertices to G'. This means we can modify a clique-width expression σ for G' to obtain a clique-width expression of G as follows. Each added vertex is introduced with a designated label that is distinct from all labels used in σ. Then each subexpression of σ that introduces a vertex of G' can be replaced by an expression that introduces the vertex and inserts all required edges to the added vertices. Finally, one can insert the missing edges between added vertices. ◀

As the second step, we prove the following theorem.

▶ **Theorem 16.** *Let H be a fixed non-trivial core with prime factorization $H_1 \times \ldots \times H_m$. Assume that H is H_i-projective for some $i \in [m]$. Then there is no algorithm solving* HomExt*(H) in time $\mathcal{O}^*((s(H_i) - \varepsilon)^{\mathrm{cw}(G')})$ for any $\varepsilon > 0$, unless the SETH fails.*

Before we proceed to the proof of Theorem 16, we show that it implies Theorem 2.

Theorem 16 → Theorem 2: Let H be a non-trivial core with a prime factorization $H_1 \times \ldots \times H_m$. W.l.o.g. assume that H is H_1-projective. Suppose that Theorem 2 does not hold, i.e., there exists an algorithm A that solves every instance G of Hom(H) in time $\mathcal{O}^*((s(H_1) - \varepsilon)^{\mathrm{cw}(G)})$.

Let (G', h') be an instance of HomExt(H). We use Theorem 15 to transform (G', h') into an equivalent instance G of Hom(H), such that $\mathrm{cw}(G) \leqslant \mathrm{cw}(G') + |V(H)|$. Then, we use A to decide whether $G \to H$ in time

$$\mathcal{O}^*\left((s(H_1) - \varepsilon)^{\mathrm{cw}(G)} = \mathcal{O}^*\left((s(H_1) - \varepsilon)^{\mathrm{cw}(G')} \cdot (s(H_1) - \varepsilon)^{|V(H)|}\right)\right.\!.$$

Since H is a fixed graph, $(s(H_1) - \varepsilon)^{|V(H)|}$ is a constant, and therefore $\mathcal{O}^*\left((s(H_1) - \varepsilon)^{\mathrm{cw}(G)}\right) = \mathcal{O}^*\left((s(H_1) - \varepsilon)^{\mathrm{cw}(G')}\right)$. Since $G \to H$ if and only if (G', h') is a yes-instance of HomExt(H), we get a contradiction with Theorem 16.

We will prove Theorem 16 for $i = 1$, which covers other cases by commutativity of direct products. We begin by constructing certain gadgets that will be used later. Let H be a fixed core with factorization $H_1 \times \ldots \times H_m$. We define $W = H_2 \times \ldots \times H_m$ if $m \geqslant 2$, and $W = K_1^*$ otherwise. Clearly, $H_1 \times W$ is a (not necessarily prime) factorization of H. Moreover, if for some graph G we have a homomorphism $f : G \to H_1 \times \ldots, \times H_m$, for $i \in [m]$ we denote by f_i the homomorphism $\pi_i \circ f : G \to H_i$.

Let S be a set of pairs of vertices of H_1, and let $w, w' \in V(W)$. We say that a tuple (F, h', p, q), such that F is a graph, $h' : V' \to H$ is a mapping with domain $V' \subseteq V(F)$, and $p, q \in V(F)$, is an (S, w, w')-*gadget* if

(S1) for every extension $h : F \to H$ of h', it holds that $(h_1(p), h_1(q)) \in S$,

(S2) for every pair $(s_1, s_2) \in S$ there exists an extension $h : F \to H$ of h' such that $h(p) = (s_1, w)$ and $h(q) = (s_2, w')$.

▶ **Lemma 17.** *Let H be a non-trivial connected core with factorization $H_1 \times W$, let $S \subseteq V(H_1)^2$, and let $w, w' \in V(W)$. Assume that H is H_1-projective. Then there exists an (S, w, w')-gadget.*

Proof. Let $S = \{(s_1^1, s_2^1), \ldots, (s_1^\ell, s_2^\ell)\}$. Define

$$F = H_1^\ell \times W, \quad \text{and} \quad p = (s_1^1, \ldots, s_1^\ell, w), \quad \text{and } q = (s_2^1, \ldots, s_2^\ell, w').$$

Let $V' = \{(x, x, \ldots, x, y) \mid x \in V(H_1), y \in V(W)\}$, and let $h'(x, \ldots, x, y) = (x, y)$. We claim that (F, h', p, q) is an (S, w, w')-gadget.

The condition 1 follows from the fact that H is H_1-projective. Indeed, observe that if $h : F \to H_1 \times W$ is an extension of h', then h_1 must be H_1-idempotent, and hence a projection on one of the ℓ first coordinates. Therefore, we must have $(h_1(p), h_1(q)) \in S$.

For (S2), take any $(s_1^i, s_2^i) \in S$ and let $h : F \to H_1 \times W$, $h(x) = (\pi_i(x), \pi_{\ell+1}(x))$. Clearly, h is an extension of h', and it is easy to verify that $h(p) = (s_1^i, w)$ and $h(q) = (s_2^i, w')$. ◀

We say that $S \subseteq V(H_1)^2$ is *proper*, if for every coordinate there exist two elements is S that differ on that coordinate, i.e., S is not of the form $\{s\} \times U$ nor $U \times \{s\}$ for some $s \in V(H_1)$ and $U \subseteq V(H_1)$. Note that if S is proper and (F, h', p, q) is an (S, w, w')-gadget, then neither p nor q belong to the domain of h'.

For fixed vertices $a, b \in V(H_1)$, let $S_{a,b} = \{(a', b') : a' \neq a, b' \in V(H_1)\} \cup \{(a, b)\}$. We call the $(S_{a,b}, w, w')$-gadget (F, h', p, q) an $((a, b), w, w')$-*implication-gadget*. Intuitively, an $((a, b), w, w')$-implication-gadget works as the implication $a \Rightarrow b$, since in every homomorphism $h : F \to H$ that extends h', if $h_1(p) = a$, then $h_1(q) = b$.

Let $a, b, c \in V(H_1)$, $w \in V(W)$, and let t be an integer. A triple (F, h', R) such that F is a graph, $h' : V' \to H_1 \times W$ is a partial mapping from some $V' \subseteq V(F)$, and R is a subset of $V(F)$ of cardinality t is an *t-or-gadget with domain* $((a, b, c), w)$ if

(O1) for every homomorphism $h : F \to H$ that is an extension of h', and for every $u \in R$ we have that $h_1(u) \in \{a, b, c\}$ and there exists $v \in R$ such that $h_1(v) = a$,

(O2) for every $v \in R$ there exists a homomorphism $h : F \to H$ that is an extension of h', such that $h(v) = (a, w)$ and for every $u \in R - \{v\}$ it holds that $h(u) \in \{(b, w), (c, w)\}$.

▶ **Lemma 18.** *Let H be a non-trivial core with factorization $H_1 \times W$. Assume that H is H_1-projective. Then for every distinct $a, b, c \in V(H_1)$, every $w \in V(W)$ and every t, there exists a t-or-gadget (F, h', R) with domain $((a, b, c), w)$.*

Proof. We consider separately the cases $t = 1$ and $t = 2$. Observe that in case $t = 1$ our gadget needs to be a graph that has a vertex $r \in R$ that is always mapped to (a, w). Hence, we set $F = K_1$, $R = V(F)$, and $h'(v) = (a, w)$ for $v \in V(F)$.

If $t = 2$, let $S = \{(a, b), (b, a), (a, a)\}$, we introduce an independent set $R = \{r_1, r_2\}$ and (S, w, w)-gadget (F, h', r_1, r_2). To see that (F, h', R) satisfies 1, consider any extension $f : F \to H$ of h'. As (F, h', r_1, r_2) is (S, w, w)-gadget, we have $(f_1(r_1), f_1(r_2)) \in \{(a, b), (a, a), (b, a)\}$. For 2, recall that by the property 2 of S-gadget there exist extensions $f^{(1)}$ and $f^{(2)}$ of h' such that $(f_1^{(1)}(r_1), f_1^{(1)}(r_2)) = (a, b)$ and $(f_1^{(2)}(r_1), f_1^{(2)}(r_2)) = (b, a)$.

Assume then that $t \geqslant 2$, and let

$$\begin{aligned} S &= \{a,b,c\}^2 - \{(b,c),(c,b)\}, \\ S_{\text{left}} &= \{(a,a),(a,b),(a,c),(c,a),(c,c)\}, \\ S_{\text{right}} &= \{(a,a),(a,b),(b,a),(b,b),(c,a)\} \end{aligned}$$

be subsets of $V(H_1)^2$. We introduce an independent set $R = \{r_1, \ldots, r_t\}$ of t vertices and one copy of (S_{left}, w, w)-gadget (F_1, h'_1, p_1, q_1) from r_1 to r_2. Then, for $j \in \{2, \ldots, t-2\}$, we introduce an (S, w, w)-gadget (F_j, h'_j, p_j, q_j) from r_j to r_{j+1} (we note that if $t \leqslant 3$, we do not introduce these). Last, we introduce one copy of (S_{right}, w, w)-gadget $(F_{t-1}, h'_{t-1}, p_{t-1}, q_{t-1})$ from r_{t-1} to r_t. We note that sets S, S_{left}, and S_{right} are proper, so the domains of the partial mappings h'_j, $j \in \{1, \ldots, t-1\}$, are pairwise disjoint. In particular, the union $h' = \bigcup_{j=1}^{t} h'_j$ is a well-defined mapping. We define F to be the union of all the graphs from the introduced gadgets and claim that (F, h', R) is a t-or-gadget. We first show that 1 holds. Assume that there exists an extension $f : F \to H$ of h', and $j' \in [t]$ such that $f_1(r_{j'}) \notin \{a,b,c\}$. This implies that there exists $j \in \{j'-1, j'\}$ such that $(f_1(r_j), f_1(r_{j+1})) \notin S'$ for any $S' \in \{S_{\text{left}}, S, S_{\text{right}}\}$. This is a contradiction with (F, h'_j, p_j, q_j) being an (S', w, w)-gadget, as it violates 1.

Now assume that there exists an extension $f : F \to H$ of h' such that for every $j \in [t]$ we have that $f_1(r_j) \in \{b,c\}$. The definition of S_{left} and S_{right}, respectively, implies that $f_1(r_1) = c$ and $f_1(r_t) = b$. Hence, there exists $j \in [t-1]$ such that $f_1(r_j) = c$ and $f_1(r_{j+1}) = b$. However, observe that the pair (c,b) does not belong to set S', for $S' \in \{S_{\text{left}}, S, S_{\text{right}}\}$, and since we introduced an S'-gadget from r_j to r_{j+1}, this leads to a contradiction.

To see that 2 holds as well, fix some $r_j \in R$ and define

$$f'(r_\ell) = \begin{cases} (a,w), & \text{if } \ell = j, \\ (c,w), & \text{if } \ell < j, \\ (b,w), & \text{if } \ell > j, \end{cases}$$

If $j = 1$, then since $(a,b) \in S_{\text{left}}$, $(b,b) \in S$ and $(b,b) \in S_{\text{right}}$, the property 2 asserts that we can construct a homomorphism $f : F \to H$ that extends h' and f'. The same holds also if $j = t$, (since $(c,c) \in S_{\text{left}}$, $(c,c) \in S$ and $(c,a) \in S_{\text{right}}$), and if $1 < j < t$ (since $(c,c) \in S_{\text{left}}$, $(c,c),(c,a),(a,b),(b,b) \in S$ and $(b,b) \in S_{\text{right}}$). ◀

Finally, all that remains is to prove Theorem 16. Our reduction generalizes the construction used by Lampis [20] to reduce an SETH lower-bounded constraint satisfaction problem to k-Coloring. Intuitively speaking, in that construction possible variable assignments are encoded by mapping specified vertices to arbitrary non-trivial subsets of the colors. The straightforward generalization of this approach to our setting would be to map to non-trivial subsets of $V(H)$. However, the structure of H allows only certain configurations of subsets as images for the specified vertices in a solution for $\text{Hom}(H)$ – which is precisely where the signature sets come into play.

Let $q, B \geqslant 2$ be integers. We will reduce from the q-CSP-B problem that is defined as follows. An instance of q-CSP-B consists of a set X of variables and a set C of q-constraints. A q-constraint $c \in C$ is a q-tuple of elements from X and a set $P(c)$ of q-tuples of elements from $[B]$ (i.e., $P(c) \subseteq [B]^q$). The q-CSP-B problem asks whether there exists an assignment $\gamma : X \to [B]$, such that each constraint is satisfied, i.e., if $c = ((x_1, \ldots, x_q), P(c)) \in C$, then $(\gamma(x_1), \ldots, \gamma(x_q)) \in P(c)$. Note that we can assume that q-constraints in our q-CSP-B instance may have less than q vertices, as it is always possible to add at most $q-1$ dummy variables to X and add them to constraints that are of smaller size.

We will use the following theorem.

▶ **Theorem 19** ([20]). *For any $B \geqslant 2, \varepsilon > 0$ we have the following: assuming the SETH, there exists q such that n-variable q-CSP-B cannot be solved in time $\mathcal{O}^*((B-\varepsilon)^n)$.*

We have all the tools to perform the final reduction.

Proof of Theorem 16. Recall that it is sufficient to prove the theorem when $H = H_1 \times W$ is non-trivial H_1-projective core ($W = K_1^*$ if $H = H_1$). Fix $\varepsilon > 0$ and set $B = s(H_1)$. As H is H_1-projective, H_1 is non-trivial and hence contains at least three distinct vertices a, b and c. In particular, $B \geqslant 3$ by Observation 8. Since $H = H_1 \times W$ is a non-trivial core, W must have at least one edge ww' (it may happen that $w = w'$). From now on a, b, c, w and w' are fixed. Let q be the smallest number such that q-CSP-B on n variables cannot be solved in time $\mathcal{O}^*((B-\varepsilon)^n)$ assuming the SETH, given by Theorem 19.

Let φ be an instance of q-CSP-B, where $X = \{x_1, \ldots, x_n\}$ is the set of variables and $C = \{c_0, \ldots, c_{m-1}\}$ is the set of constraints. For every $j \in \{0, \ldots, m-1\}$ denote by X_j the set of variables that appear in the constraint c_j. Let $P(c_j) = \{f_1^j, \ldots, f_{p_j}^j\}$ be the set of assignments from X_j to $[B]$ that satisfy the constraint c_j. Let $L = m(n|H_1| + 1)$, and let $\lambda : [B] \to \mathcal{S}(H_1)$ be some fixed bijection.

We construct the instance G_φ of HomExt(H). For each $j \in \{0, \ldots L-1\}$, let $j' = j \mod m$. Let $R_j = \{r_1^j, \ldots, r_{p_{j'}}^j\}$, where each vertex r_k^j corresponds to the assignment $f_k^{j'}$. We introduce the $p_{j'}$-or-gadget (F_j, h'_j, R_j) with domain $((a,b,c), w)$.

For each $x_i \in X_{j'}$, and for each $f_k^{j'} \in P(c_{j'})$ we do the following:

1. Let $y = f_k^{j'}(x_i) \in [B]$. Construct an independent set $V_i^{j,k}$ of $|\lambda(y)|$ vertices and an independent set $U_i^{j,k}$ of $|S(\lambda(y))|$ vertices.
2. For each $d \in \lambda(y)$ select a distinct vertex $z \in V_i^{j,k}$ and add an $((a,d), w, w')$-implication-gadget from r_k^j to z. For each $d \in S(\lambda(y))$ select a distinct vertex $z \in U_i^{j,k}$ and add an $((a,d), w, w')$-implication-gadget from r_k^j to z.
3. Connect all vertices of $U_i^{j,k}$ with all vertices of previously constructed sets $V_i^{\ell,k'}$ for $\ell < j$ and $k' \in [p_\ell]$.

The partial mapping h' is the union of all the partial mappings that are introduced by all the gadgets. This finishes the construction of the instance (G_φ, h') of HomExt(H).

▷ Claim 20. If φ is a yes-instance of q-CSP-B, then there exists a homomorphism $h : G_\varphi \to H$ that extends h'.

Proof of Claim. If φ is a yes-instance of q-CSP-B, then there exists an assignment $\gamma : X \to [B]$ satisfying each constraint. We define $h : G_\varphi \to H$ as follows.

Fix $j \in \{0, \ldots, L-1\}$, and consider the or-gadget (F_j, h'_j, R_j). Recall that the set $P(c_{j'})$ consists of all assignments of variables in $X_{j'}$ that satisfy the constraint $c_{j'}$. Therefore, there exists an assignment $f_k^{j'} \in P(c_{j'})$ such that $\gamma|_{X_{j'}} \equiv f_k^{j'}$. Consider the vertex r_k^j that corresponds to that assignment. By the property 2 of the or-gadget, we know that there exists a H-coloring of F_j that extends h', such that (i) $h_1(r_k^j) = a$ and (ii) for every $r_{k'}^j \in R_j, k' \neq k$ we have that $h_1(r_{k'}^j) \in \{b, c\}$.

Let $x_i \in X_{j'}$ and let $y = f_k^{j'}(x_i) \in [B]$. Since for each $d \in \lambda(y)$ there exists a vertex $z \in V_i^{j,k}$ such that there is an $((a,d), w, w')$-implication-gadget from r_k^j to z, the condition (i) implies that $h_1(V_i^{j,k}) = \lambda(y)$. We color the vertices of $V_i^{j,k}$ in a way that $h(V_i^{j,k}) = \lambda(y) \times \{w\}$.

Also, since for each $d \in S(\lambda(y))$ there exists a vertex $z \in U_i^{j,k}$ such that there is an (a,d)-implication-gadget from r_k^j to z, the condition (i) implies that $h_1(U_i^{j,k}) = S(\lambda(y))$. We color the vertices of $U_i^{j,k}$ in a way that $h(U_i^{j,k}) = S(\lambda(y)) \times \{w'\}$.

Because of (ii), the implication gadgets from $r_{k'}$ to the vertices of $V_i^{j,k'} \cup U_i^{j,k'}$ do not put any constraints on the coloring of the sets $V_i^{j,k'}$ and $U_i^{j,k'}$. Therefore, for each $v \in V_i^{j,k'}$ we set $h(v)$ to be any vertex from $\lambda(y)$. Similarly, for each $u \in U_i^{j,k'}$ we set $h(u)$ to be any vertex from $S(\lambda(y))$. Since $(b,x),(c,x) \in S_{a,b}$ for any $x \in V(H_1)$, property 2 applied to the implication gadgets asserts that since $h(r_{k'}) \in \{b,c\}$, we can always extend this mapping to a homomorphism of the whole gadget to H.

It remains to argue that the edges between the sets $V_i^{j_1,k_1}$ and $U_i^{j_2,k_2}$ are mapped to edges of H, for any $j_1 < j_2$ and k_1, k_2. However, observe that since γ is an extension of some $f_{k_1}^{j_1'} \in S_{j_1'}$ and $f_{k_2}^{j_2'} \in S_{j_2'}$, we must have $f_{k_1}^{j_1'}(x_i) = f_{k_2}^{j_2'}(x_i) = y$. Hence, h maps every $v \in V_i^{j_1,k_1}$ to some element of $\lambda(y) \times \{w\}$, and every $u \in U_i^{j_2,k_2}$ to some element of $S(\lambda(y)) \times \{w'\}$. By Observation 4, and since $ww' \in E(W)$, we get that $h(v)h(u) \in E(H)$. That concludes the proof of the claim. ◁

▷ Claim 21. If there exists a homomorphism $h : G_\varphi \to H$ that extends h', then φ is a yes-instance of q-CSP-B.

Proof of Claim. We will define the assignment $\gamma : X \to [B]$ that makes every constraint from C satisfied.

Fix $j \in \{0,\ldots,L-1\}$, and consider the $p_{j'}$-or-gadget (F_j, h'_j, R_j). By the property 1 of the or-gadget, there exists $k_j \in [p_{j'}]$ such that $h_1(r_{k_j}^j) = a$. Implication gadgets whose p-vertices were identified with $r_{k_j}^j$ assert that $h_1(V_i^{j,k_j}) \in \mathcal{S}(H)$ and $h_1(U_i^{j,k_j})$ is the signature of $h_1(V_i^{j,k_j})$. Then, by Observation 5, $h_1(V_i^{j,k_j})$ is a signature of $h_1(U_i^{j,k_j})$. Denote $h_1(U_i^{j,k_j})$ by T_i^j, then $h_1(V_i^{j,k_j}) = S(T_i^j)$ and $S(S(T_i^j)) = T_i^j$ Let $y_i^j = f_{k_j}^{j'}(x_i)$ be the *candidate assignment* for $x_i \in X_{j'}$ at index j, recall that $y_i^j = \lambda^{-1}(S(T_i^j))$.

Let $i \in [n]$ be fixed and let $j_1, j_2 \in [L], j_1 < j_2$ be such that $x_i \in X_{j_1'} \cap X_{j_2'}$. Observe that in such case $T_i^{j_1} \supseteq T_i^{j_2}$. Indeed, denote $k_1 = k_{j_1}$, $k_2 = k_{j_2}$, then we have (1) $h_1(V_i^{j_1,k_1}) = S(T_i^{j_1})$ and $h_1(U_i^{j_1,k_1}) = T_i^{j_1}$, and (2) $h_1(V_i^{j_2,k_2}) = S(T_i^{j_2})$ and $h_1(U_i^{j_2,k_2}) = T_i^{j_2}$. Recall that each vertex from $U_i^{j_2,k_2}$ is adjacent to each vertex from $V_i^{j_1,k_1}$. Since h_1 is a homomorphism, the same holds for their images: each vertex from $T_i^{j_2}$ is adjacent to each vertex from $S(T_i^{j_1})$. Then $S(T_i^{j_2}) \supseteq S(T_i^{j_1})$, so $T_i^{j_1} = S(S(T_i^{j_1})) \supseteq S(S(T_i^{j_2})) \supseteq T_i^{j_2}$.

We say that the index $j_1 \in \{0,\ldots,L-1\}$ is *problematic* for i if there is $j_2 > j_1$ such that $x_i \in X_{j_1'} \cap X_{j_2'}$ and $T_i^{j_1} \neq T_i^{j_2}$. Since for each variable we have at most $|H_i|$ problematic indices, there are at most $|H_i| \cdot n$ problematic indices for all variables. Since $L = m(|H_i| \cdot n + 1)$, by pigeonhole principle we get that there exists a set $J \subseteq \{0,\ldots,L-1\}$ of m consecutive indices such that none of them is problematic for any i. For every $i \in [n]$, we fix some $j \in J$ such that $x_i \in X_{j'}$ and set $\gamma(x_i) = y_i^j$ (observe that the choice of j does not matter).

We claim that γ is an assignment that satisfies every constraint from φ. Indeed, for any $j' \in [m]$ there exists $j \in J$ such that $j' = j \mod m$. For every $i \in X_{j'}$, we have $\gamma(x_i) = y_i^j = f_{k_j}^{j'}(x_i)$, so γ satisfies the constraint $c_{j'}$. ◁

Finally, it remains to adapt the arguments of Lampis [20] to establish the desired linear clique-width bound.

▷ Claim 22. G_φ can be constructed in time polynomial in $|\varphi|$, and we have $\mathrm{cw}(G_\varphi) \leqslant n + f(\varepsilon,\nu)$ for some function f, where $\nu = |V(H)|$.

Proof of Claim. Observe that any (S,w,w')-gadget constructed as in Lemma 17 for $i = 1$ has at most $|V(H_i)|^{|S|-1} \cdot |V(H)| \leqslant \nu^{|S|}$ vertices. In particular, we can ensure that every implication gadget in G_φ has at most $\nu^{\mathcal{O}(\nu^2)}$ vertices. Moreover, we will assume that all the

or-gadgets of G_φ are constructed as in Lemma 18 and the subgadgets for S, S_{left} and S_{right} contain at most ν^7 vertices. Then for every $j \in \{0, \ldots, L-1\}$, $p_{j'}$-or-gadget (F_j, h'_j, R_j) has at most $(p_{j'} - 1) \cdot \nu^7$ vertices.

For fixed H and $\varepsilon > 0$ we have that $B = s(H_i) \leqslant 2^{|V(H_i)|} - 2$ and q is a constant that only depends on B, ε (that is, on $|V(H_i)|$, ε). Each constraint of the $q - \text{CSP} - B$ instance has at most B^q satisfying assignments. In particular, the number of vertices in each or-gadget is upper-bounded by $B^q \cdot \nu^7$. Therefore, it is not hard to see that the whole construction can be performed in polynomial time, if H is fixed and ε is a constant. For clique-width we use the following labels:

1. n *main* labels, representing the variables of φ.
2. A single *done* label. Its informal meaning is that a vertex that receives this label will not be connected to anything else not yet introduced in the graph.
3. $B^q \cdot \nu^7$ *constraint work* labels.
4. $qB^q \cdot \nu^{\mathcal{O}(\nu^2)}$ *variable-constraint incidence work* labels.

To give a clique-width expression we will describe how to build the graph, following essentially the steps given in the description of the construction by maintaining the following invariant: before starting iteration j, all vertices of the set $W_i^j = \bigcup_{j'<j} \bigcup_{k \in [p_{j'}]} V_i^{j',k}$ have label i, and all other vertices have the *done* label. This invariant is vacuously satisfied before the first iteration, since the graph is empty. Suppose that for some $j \in \{0, \ldots, L-1\}$ the invariant is true. We use the $B^q \cdot \nu^7$ constraint work labels to introduce the vertices of the $p_{j'}$-or-gadget (F_j, h'_j, R_j), giving each vertex a distinct label. We use join operations to construct the internal edges of the or-gadget. Then, for each variable x_i that appears in the current constraint we do the following: we use $B^q \cdot \nu^{\mathcal{O}(\nu^2)}$ of the variable-constraint incidence work labels to introduce for all $k \in [p_{j'}]$ the vertices of $V_i^{j,k}$ and $U_i^{j,k}$ as well as the implication gadgets connecting these to r_k^j. Again we use a distinct label for each vertex, but the number of vertices (including internal vertices of the implication gadgets) is $B^q \cdot \nu^{\mathcal{O}(\nu^2)}$, so we have sufficiently many labels to use distinct labels for each of the q variables of the constraint. We use join operations to add the edges inside all implication gadgets. Then we use join operations to connect $U_i^{j,k}$ to all vertices W_i^j. This is possible, since the invariant states that all the vertices of W_i^j have the same label i. We then rename all the vertices of $U_i^{j,k}$ for all k to the done label, and do the same also for internal vertices of all implication gadgets. We proceed to the next variable of the same constraint and handle it using its own $B^q \cdot \nu^{\mathcal{O}(\nu^2)}$ labels. Once we have handled all variables of the current constraint, we rename all vertices of each $V_i^{j,k}$ to label i for all k. We then rename all vertices of the $p_{j'}$-or-gadget (F_j, h'_j, R_j) gadget to the done label and increase j by 1. It is not hard to see that we have maintained the invariant and constructed all edges induced by the vertices introduced in steps up to j, so repeating this process constructs the graph. ◁

Together the claims imply Theorem 16 in the following way: For an arbitrary instance of q-CSP-B, our construction produces an instance of HomExt(H), and the instances are equivalent by Claim 20 and Claim 21. If one could solve HomExt(H) in $\mathcal{O}^*((s(H_i)-\varepsilon)^{\text{cw}(G)})$ for some $\varepsilon > 0$, one could use our construction to solve q-CSP-B, and by our choice of B and Claim 22 this procedure would have complexity $\mathcal{O}^*((B-\varepsilon)^{n+c})$ for some constant c. By our choice of q according to Theorem 19, this contradicts the SETH. ◀

6 Summary and Concluding Remarks

Extensions and Corollaries. We observe that Corollary 12 can be combined with Theorem 2 to obtain the following statement, which summarizes our results.

▶ **Theorem 23.** *Let H' be a fixed graph with the non-trivial connected core H. Let $H_1 \times \ldots \times H_m$ be the prime factorization of H. Let $i \in [m]$ be such that $s(H_i) = \max_{j\in[m]} s(H_j)$. Let G be an instance of Hom(H').*

1. *Assuming a clique-width expression σ of G of width $\mathrm{cw}(G)$ is given, the Hom(H') problem can be solved in time $\max_{i\in[m]} \mathcal{O}^*(s(H_i)^{\mathrm{cw}(G)})$.*
2. *Assuming that H is H_i-projective, there is no algorithm to solve Hom(H') in time $\max_{i\in[m]} \mathcal{O}^*((s(H_i)-\varepsilon)^{\mathrm{cw}(G)})$ for any $\varepsilon > 0$, unless the SETH fails.*

We note that the restriction to connected targets can be avoided by known properties of homomorphisms to disconnected graphs [25]; on the algorithmic side, one branches over all connected components of H, while for the lower bound one considers the component with maximum signature number.

It is clear that obtaining a full complexity classification with respect to clique-width may require weakening the assumption in the second statement of Theorem 23. We recall that an analogous situation occurs in the work of Okrasa and Rzążewski [25]; as mentioned in the introduction, the authors obtain the SETH-conditioned tight complexity bound for the Hom(H) problem parameterized by treewidth for all targets H, assuming two conjectures of Larose and Tardif [21,22]. The notion of H_i-projectivity allows us to restate these conjectures as one, which is not only sufficient in our setting but is also weaker in the sense of it being implied by the former two conjectures, but not necessarily equivalent to them.

▶ **Conjecture 1.** *Let H be a non-trivial core with prime factorization $H_1 \times \ldots \times H_m$ and let $i \in [m]$. Then H is H_i-projective.*

Using Conjecture 1, we can restate our main result as follows.

▶ **Theorem 24.** *Let H' be a fixed graph with the non-trivial connected core H. Let $H_1 \times \ldots \times H_m$ be the prime factorization of H. Let G be an instance of Hom(H').*

1. *Assuming the clique-width expression σ of G of width $\mathrm{cw}(G)$ is given, the Hom(H') problem can be solved in time $\max_{i\in[m]} \mathcal{O}^*(s(H_i)^{\mathrm{cw}(G)})$.*
2. *Assuming that Conjecture 1 holds, there is no algorithm to solve Hom(H') in time $\max_{i\in[m]} \mathcal{O}^*((s(H_i)-\varepsilon)^{\mathrm{cw}(G)})$ for any $\varepsilon > 0$, unless the SETH fails.*

We also observe that since each non-trivial projective core H is H-projective, in this case we already obtain a tight complexity bound.

▶ **Corollary 25.** *Let H' be a fixed graph with the non-trivial connected projective core H. Let G be an instance of Hom(H').*

1. *Assuming the clique-width expression σ of G of width $\mathrm{cw}(G)$ is given, the Hom(H') problem can be solved in time $\mathcal{O}^*(s(H)^{\mathrm{cw}(G)})$.*
2. *There is no algorithm to solve Hom(H') in time $\mathcal{O}^*((s(H)-\varepsilon)^{\mathrm{cw}(G)})$ for any $\varepsilon > 0$, unless the SETH fails.*

Generalizations and Other Research Directions. We remark that our hardness reduction is via HomExt(H), and in fact our algorithm can also easily be adapted to this setting (by removing all records that do not adhere to the partial mapping from the input graph to H) without an increase in complexity. However, since the dichotomy between P and NP-complete cases of HomExt(H) is more complicated (see [11], studied as the graph-retract problem) there exist target graphs H that are not covered by Theorem 24. On a similar note, let us also point out that setting up the SETH-conditioned tight complexity bounds for clique-width for a more general list problem LHom(H) [10,27] is widely open.

Another direction that is very closely related to our results is to determine similarly tight complexity bounds for the *rank-width* (rw) of the input graph: rank-width [17, 26] is a graph parameter that is known to be asymptotically equivalent to clique-width and is in fact used as an approximation of clique-width that can be computed in fixed-parameter tractable time. Our results together with the known relationship between clique-width and rank-width imply an upper bound of $\mathcal{O}^*(s(H)^{2^{\mathrm{rw}}+1})$ and a SETH lower bound of $(s(H)-\varepsilon)^{\mathrm{rw}}$ on the complexity of $\textsc{Hom}(H)$ for projective H parameterized by the rank-width of the input.

References

1 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

2 Jan Böker. Graph similarity and homomorphism densities. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 32:1–32:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

3 Andrei A. Bulatov and Amineh Dadsetan. Counting homomorphisms in plain exponential time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 21:1–21:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

4 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

5 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

6 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socała. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017. doi:10.1145/3051094.

7 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

8 Willibald Dörfler. Primfaktorzerlegung und Automorphismen des Kardinalproduktes von Graphen. *Glasnik Matematički*, 9:15–27, 1974.

9 László Egri, Dániel Marx, and Paweł Rzążewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 27:1–27:15, 2018.

10 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *J. Graph Theory*, 42(1):61–80, 2003. doi:10.1002/jgt.10073.

11 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

12 Fedor V. Fomin, Pinar Heggernes, and Dieter Kratsch. Exact algorithms for graph homomorphisms. *Theory Comput. Syst.*, 41(2):381–393, 2007. doi:10.1007/s00224-007-2007-x.

13 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007.

14 Richard H. Hammack, Wilfried Imrich, and Sandi Klavžar. *Handbook of product graphs*. CRC press, 2011.

15 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

16 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discret. Math.*, 109(1-3):117–126, 1992.

17 Sang il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

20 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

21 Benoît Larose. Families of strongly projective graphs. *Discuss. Math. Graph Theory*, 22(2):271–292, 2002. doi:10.7151/dmgt.1175.

22 Benoît Larose and Claude Tardif. Strongly rigid graphs and projectivity. *Multiple-Valued Logic*, 7:339–361, 2001.

23 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011.

24 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

25 Karolina Okrasa and Paweł Rzążewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021.

26 Sang-il Oum. Approximating rank-width and clique-width quickly. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science*, pages 49–58, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

27 Marta Piecyk and Paweł Rzążewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

28 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

29 Paweł Rzążewski. Exact algorithm for graph homomorphism and locally injective graph homomorphism. *Inf. Process. Lett.*, 114(7):387–391, 2014.

30 Magnus Wahlström. New plain-exponential time classes for graph homomorphism. *Theory Comput. Syst.*, 49(2):273–282, 2011. doi:10.1007/s00224-010-9261-z.

31 Tomasz Łuczak and Jaroslav Nešetřil. Note on projective graphs. *J. Graph Theory*, 47(2):81–86, 2004.

Sublinear Dynamic Interval Scheduling (On One or Multiple Machines)

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Karol Pokorski
Institute of Computer Science, University of Wrocław, Poland

Abstract

We revisit the complexity of the classical Interval Scheduling in the dynamic setting. In this problem, the goal is to maintain a set of intervals under insertions and deletions and report the size of the maximum size subset of pairwise disjoint intervals after each update. Nontrivial approximation algorithms are known for this problem, for both the unweighted and weighted versions [Henzinger, Neumann, Wiese, SoCG 2020]. Surprisingly, it was not known if the general exact version admits an exact solution working in sublinear time, that is, without recomputing the answer after each update.

Our first contribution is a structure for Dynamic Interval Scheduling with amortized $\tilde{\mathcal{O}}(n^{1/3})$ update time. Then, building on the ideas used for the case of one machine, we design a sublinear solution for any constant number of machines: we describe a structure for Dynamic Interval Scheduling on $m \geq 2$ machines with amortized $\tilde{\mathcal{O}}(n^{1-1/m})$ update time.

We complement the above results by considering Dynamic Weighted Interval Scheduling on one machine, that is maintaining (the weight of) the maximum weight subset of pairwise disjoint intervals. We show an almost linear lower bound (conditioned on the hardness of Minimum Weight k-Clique) for the update/query time of any structure for this problem. Hence, in the weighted case one should indeed seek approximate solutions.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases interval scheduling, dynamic problems, data structures, greedy algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.67

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2203.14310

1 Introduction

The Interval Scheduling (IS) problem is often used as one of the very first examples of problems that can be solved with a greedy approach. In this problem, we have a set of jobs, the i-th job represented by an interval (s_i, f_i). Given n such intervals, we want to find a maximum size subset of pairwise disjoint intervals. In this context, disjoint intervals are usually called compatible. This admits a natural interpretation as a scheduling problem, where each request corresponds to a job that cannot be interrupted and requires exclusive access to a machine. Then, the goal is to schedule as many jobs as possible using a single machine. The folklore greedy algorithm solves this problem in $\mathcal{O}(n)$ time, assuming that the intervals are sorted by the values of f_i [19]. While it may appear to be just a puzzle, interval scheduling admits multiple applications in areas such as logistics, telecommunication, manufacturing, or personnel scheduling. For more applications and a detailed summary of different variants of interval scheduling, we refer to [21].

In many real-world applications, there is a need for maintaining the input under certain updates (for example, insertions and deletions of items), so that we can report the optimal solution (or its cost) after each operation. The goal is to avoid the possibly very expensive


recalculation of the answer (which surely takes at least linear time in the size of the input) by maintaining some kind of additional structure. The first step in this line of research is to design a structure with sublinear update/query time. Then, the next goal is to bring down the time complexities to polylogarithmic (in the size of the input). Examples of problems in which this has been successfully accomplished include dynamic graph connectivity [16, 17, 23], dynamic longest increasing subsequence [13, 20], dynamic suffix array [2, 18], dynamic graph clustering [10], and many others. For some dynamic problems no such solutions are known, and we have tools for proving (conditional) polynomial hardness for dynamic algorithms [14].

This suggests the following Dynamic Interval Scheduling (DIS) problem, in which we want to maintain a set S of intervals subject to insert and delete operations. After each update, we should report the size of the maximum size subset of pairwise compatible intervals. Note that reporting the subset itself might be not feasible, as it might contain $\Omega(n)$ intervals. Similarly, neither is explicitly maintaining this subset, as an update might trigger even $\Omega(n)$ changes in the unique optimal subset. Thus, the challenge is to maintain an implicit representation of the current solution that avoids recomputing the answer after each update, that is, supports each update in sublinear time. Besides being a natural extension of a very classical problem, we see this question as possibly relevant in practical application in which we need to cope with a dynamically changing set of jobs.

1.1 Previous work

Surprisingly, to the best of our knowledge, the complexity of general exact DIS was not considered in the literature. However, Gavruskin et al. [12] considered its restricted version, in which there is an extra constraint on the set S. Namely, it should be *monotonic* at all times: for any two intervals $(s_i, f_i), (s_j, f_j) \in S$ we should have $s_i < s_j$ and $f_i < f_j$ or vice versa. Under such assumption, there is a structure with $\mathcal{O}(\log^2 n)$ amortized time per update and $\mathcal{O}(\log n)$ amortized time per query. Alternatively, the update time can be decreased to $\mathcal{O}(\log n)$ if the query only returns if a given interval belongs to the optimal solution.

For the general version of DIS, Henzinger, Neumann and Wiese [15] designed an efficient approximation algorithm that maintains an $(1 + \epsilon)$-approximate solution in polylogarithmic time. The dependency on ϵ has been very recently improved from exponential to polynomial by Compton, Mitrović and Rubinfeld [7]. In fact, both solutions work for the weighted version of the problem, called Dynamic Weighted Interval Scheduling (DWIS). In this problem, each interval has its associated weight, and the goal is to maintain a subset of pairwise compatible intervals with the largest total weight. Note that the static version of this problem, called Weighted Interval Scheduling (WIS), can be solved by a straightforward dynamic programming algorithm [19] (but the greedy strategy no longer works now that we have weights). This brings the challenge of determining if the unweighted (and weighted) version of the problem admits an efficient exact solution.

A natural generalization of interval scheduling is to consider multiple machines. In such a problem, there is a shared set of jobs to process, each job can be either discarded or scheduled on one of the available m machines. Jobs scheduled on each machine must be pairwise compatible. The goal is to maximize the number (or the total weight) of scheduled intervals. IS on multiple machines (IS+) can be solved by extending the greedy algorithm considering intervals by the earliest end time. For each considered interval, if no machine is free at the respective time, the interval is discarded. If there are some free machines, the interval is assigned to the available machine that was busy at the latest. A direct implementation of this approach incurs a factor of m in the running time, but this can be avoided [6, 11]. The weighted version of the problem (WIS+) can be formulated and solved as a min-cost flow

problem [3, 5]. For the dynamic version, Compton, Mitrović and Rubinfeld [7] extend their methods for maintaining an approximate answer to multiple machines, however, their bounds are mostly relevant for the unweighted case. A related (but not directly connected) question is to maintain the smallest number of machines necessary to schedule all jobs in the current set [12].

1.2 Our contribution

In this paper, we consider dynamic interval scheduling on one and multiple machines. We show that the unweighted version of the problem admits a sublinear dynamic solution, and furthermore, we make non-trivial progress on decreasing the exponent in the time complexity of the solution.

The starting point is a simple structure for the general DIS problem with $\mathcal{O}(\sqrt{n}\log n)$ amortized update/query time. This is then improved to $\tilde{\mathcal{O}}(n^{1/3})$ amortized update/query time. For multiple machines, we begin with $m = 2$, and show how to solve the corresponding problem, denoted DIS2, in $\tilde{\mathcal{O}}(\sqrt{n})$ amortized time per update. Next, we use this solution to solve the general DIS+ problem in $\tilde{\mathcal{O}}(n^{1-1/m})$ amortized time per update. While designing a solution working in $\tilde{\mathcal{O}}(n^{1-1/(m+1)})$ time is not very difficult, our improved time bounds require some structural insight that might be of independent interest.

▶ **Theorem 1.** *There is a date structure for Dynamic Interval Scheduling on $m \geq 1$ machines that supports any update in $\tilde{\mathcal{O}}(\max(n^{1/3}, n^{1-1/m}))$ amortized time.*

We complement the above result by a (conditional) lower bound for the weighted version of the problem, even with $m = 1$. We show that, for every $\epsilon > 0$, under the Minimum Weight $(2\ell+1)$-Clique Hypothesis, it is not possible to maintain a structure that solves DWIS in $\mathcal{O}(n^{1-\epsilon})$ time per operation. This shows an interesting difference between the static and dynamic complexities of the unweighted and weighted versions: despite both IS and WIS admitting simple efficient algorithms, DIS admits a sublinear solution while DWIS (probably) does not.

1.3 Techniques and ideas

A natural approach to DIS is to efficiently simulate the execution of the greedy algorithm.

▶ **Definition 2.** *For an interval $I_i = (s_i, f_i)$, the* leftmost compatible *interval $LC(I_i)$ is the interval $(s_{i'}, f_{i'}) \in S$ with the smallest $f_{i'}$ such that $s_{i'} \geq f_i$ or $\perp$ if there is no such interval.*

Note that if the greedy algorithm includes I_i in the solution then it also includes $\mathrm{LC}(I_i)$. Thus, it is easy to prove that if I_i is the interval with the smallest f_i in S, then the (optimal) solution generated by the greedy algorithm is $\{I_i, \mathrm{LC}(I_i), \mathrm{LC}^2(I_i), \dots\}$.

One can consider a forest in which each interval is represented as a node and an interval I_i has parent $\mathrm{LC}(I_i)$. By creating an artificial root and connecting all forest roots' to it, we make this representation a tree. We call it *the greedy tree* (of S). The answer to the DIS query is the length of the longest path from any node to the root in the tree. We know this is actually the path from the earliest ending interval thanks to the greedy algorithm.

A standard approach used in dynamic problems is splitting the current input into several smaller parts and recomputing some information only in the part containing the updated item. Then, the answer is obtained by using the information precomputed for every part. An attempt to use such an approach for DIS could be as follows. We partition S into parts, either by the start or the end times, and in every part we precompute the result of running the

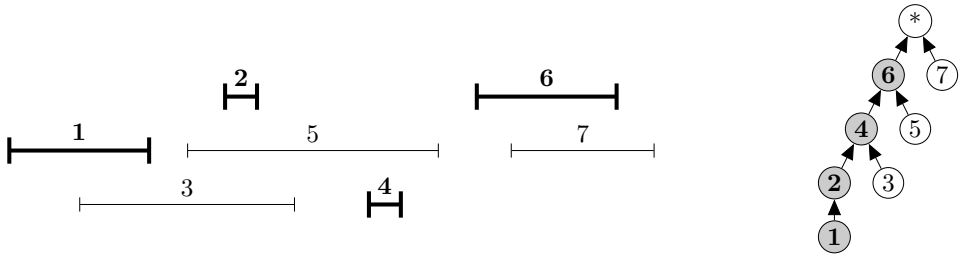


Figure 1 An input instance for DIS with the optimal solution generated by the greedy algorithm marked using bold lines and the corresponding greedy tree.

greedy algorithm from every possible state. The goal is to accelerate running the algorithm by being able to jump over the parts. For $m = 1$, we can simply maintain the greedy tree, as it allows us to simulate running the greedy algorithm not only from the interval with the smallest end time but in fact from an arbitrary interval I_i. We call this *resuming the greedy algorithm from* I_i. This allows us to jump over the whole part efficiently, and by appropriately balancing the size of each part we obtain a data structure with $\tilde{\mathcal{O}}(n^{1/2})$ time per update. This is described in detail in Appendix A. A similar approach works for $m > 1$, except that instead of the greedy tree we need to preprocess the answer for every m-tuple of intervals, resulting in $\tilde{\mathcal{O}}(n^{1-1/(m+1)})$ time per update.

We improve on this basic idea for both $m = 1$ and $m > 1$. For $m = 1$, we design a way to solve the decremental variant of DIS in only (amortized) polylogarithmic time per update, and couple this with maintaining a buffer of the most recent insertions. For $m = 2$, the greedy tree is no longer sufficient to capture all possible states of the greedy algorithm. However, by a careful inspection, we prove that for a part consisting of n intervals, instead of precomputing the answers for all $\Theta(n^2)$ possible states, it is enough to consider only $\mathcal{O}(n)$ carefully selected states. For $m > 2$, we further extend this insight by identifying only $\mathcal{O}(n^{1-1/m})$ states, called *compressible*. Interestingly, using these states to simulate the greedy algorithm starting from an arbitrary state requires a separate $\mathcal{O}(n^2)$ precomputation, hence we need to consider the case $m = 2$ separately.

2 Interval scheduling on one machine

For our structures, it is sufficient that s_i and f_i characterizing intervals are pairwise comparable but to simplify the presentation, we assume that $s_i, f_i \in \mathbb{R}_+$. One can also use an order maintenance structure [4, 9] to achieve worst-case constant time comparisons between endpoints even if we only assume that when inserting an interval (s_i, f_i) we know just the endpoints of existing intervals in S that are the nearest predecessors of s_i and f_i. We make endpoints of all intervals pairwise distinct with the standard perturbation. We assume that each insert operation returns a handle to the interval which can later be used to delete.

Our structures work in epochs. At the beginning of each epoch, we set N to be the number of intervals in S. When the number of intervals is outside range $[\frac{N}{2}, 2N]$, the new epoch begins. At the beginning of an epoch, we construct an additional data structure $\mathcal{D}$ of all intervals in S by a sequence of inserts in any order. These reconstructions have no impact on the amortized update time complexity as n actual operations are turned into $\mathcal{O}(n)$ insertions and deletions. We maintain $\mathcal{D}$ during the epoch.

We maintain a global successor structure storing all intervals sorted by their end time that enables efficient computation of $\mathrm{LC}(\cdot)$. There are k separators that split the universe of coordinates into parts of similar size. Intervals are assigned into parts $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_k$ by their

start time. Some intervals are *internal* (if they fully fit in the part) and other are *external* (otherwise). Both $\tilde{\mathcal{O}}(n^{1/2})$ and $\tilde{\mathcal{O}}(n^{1/3})$ structures are able to efficiently find the internal result for an interval I_i in a part, that is how many intervals the greedy algorithm can choose from I_i until reaching the exit (the last selected) interval of the part, so DIS query is solved by iterating over these parts and applying the exit of one part as an input to the next one.

We recommend reading Appendix A, where we introduce the above idea by showing a simpler but slower algorithm. Here we extend this approach and present a data structure showing the following.

▶ **Theorem 3.** *There is a data structure for DIS that supports any sequence of n insert/delete/query on intervals in $\tilde{\mathcal{O}}(n^{1/3})$ amortized time per each operation.*

The separators are chosen such that each part has size at most $2N^{2/3}$ and for any two consecutive parts $\mathcal{P}_j$ and $\mathcal{P}_{j+1}$ at least one has size at least $\frac{1}{2}N^{2/3}$. Thus, there are always $\mathcal{O}(n^{1/3})$ parts. More details on how to maintain this partition are provided in Appendix A.

Since our goal is to achieve $\tilde{\mathcal{O}}(n^{1/3})$ update time and parts are larger, we cannot afford to recompute the whole part from scratch for every update in it (as we did in Appendix A). Instead, we keep internal intervals of a part in two structures: a decremental structure and a buffer. External intervals are only kept in the global balanced binary search tree containing all the intervals. We first sketch the idea and describe the details in the following subsections.

The decremental structure of each part contains $\mathcal{O}(n^{2/3})$ intervals, has no information about buffer intervals, can be built in $\tilde{\mathcal{O}}(n^{2/3})$ time and allows deletions in $\mathcal{O}(\text{polylog}\, n)$ time. The buffer $\mathcal{B}_j \subseteq \mathcal{P}_j$ contains only at most $N^{1/3}$ last inserted internal intervals in $\mathcal{P}_j$. Each operation in a part leads to the recomputation of information associated with the buffer in $\tilde{\mathcal{O}}(n^{1/3})$ time. When $\mathcal{B}_j$ overflows, we rebuild the decremental structure from scratch using all internal intervals from the part and clear the buffer. Such recomputation happens every $\Omega(n^{1/3})$ updates inside a part. This way the update time of our solution can still be within the claimed bound.

As the optimal solution may use intervals both from the decremental collection and the buffer interchangeably, we need to combine information stored for these sets. For buffer intervals, we can afford to precompute the whole internal result and the exit of the part being fully aware of the content of the decremental collection. However, we also need to "notify" intervals of the decremental collection about potential better solutions that can be obtained by switching to buffer intervals. For this we store an additional structure of total size of $\tilde{\mathcal{O}}(n^{1/3})$, recomputed every update in a part, specifying for which intervals of the decremental collection there exists an "interesting" buffer interval.

2.1 Active and inactive intervals

▶ **Definition 4.** *An interval $I_i = (s_i, f_i)$ in a collection C of intervals is* active *if there is no other $(s_{i'}, f_{i'}) \in C$ such that $s_i \leq s_{i'} \leq f_{i'} \leq f_i$. Otherwise I_i is* inactive.

▶ **Lemma 5.** *For any set S of intervals and an interval $I_i \in S$, the greedy algorithm for IS resumed from I_i chooses (after I_i) only active intervals from S.*

Proof. Assume there are two intervals $I_1 = (s_1, f_1)$ and $I_2 = (s_2, f_2)$ such that $s_2 < s_1 < f_1 < f_2$. I_1 is considered earlier by the greedy algorithm. If it is scheduled, I_2 can no longer be scheduled as I_1 and I_2 are overlapping. If it is not, I_2 also can not be scheduled as the set of compatible intervals with I_2 is the subset of the compatible intervals with I_1. ◀

A collection of only active intervals is monotonic by definition. This provides a natural linear order on the active intervals in the collection: $(s_i, f_i) \prec (s_{i'}, f_{i'}) \Leftrightarrow s_i < s_{i'} \Leftrightarrow f_i < f_{i'}$. This allows us to focus on describing how to maintain the subset of active intervals inside a collection and only look for the solution of (D)IS in this subset.

The decremental structure $\mathcal{D}_j$ in each part only allows rebuilding and deletions. We maintain set of active intervals $\mathcal{A}_j \subseteq \mathcal{D}_j$ in the decremental collection. When an interval from $\mathcal{A}_j$ is deleted, the set should report new active intervals. We stress that the decremental structure is not aware of any buffer intervals of $\mathcal{P}_j$ and in order to determine if a particular interval is active in the decremental collection we do not take into account any buffer intervals.

▶ **Lemma 6.** *There is a structure that allows maintaining the subset of active intervals in a delete-only or insert-only collection of size n in $\mathcal{O}(\log n)$ amortized time per insertion/deletion and can be built in $\mathcal{O}(n \log n)$ time.*

Proof. Each interval (s_i, f_i) is translated into a point $(s_i, -f_i)$ in a plane. We say that point (x, y) *dominates* point (x', y') if $x > x' \wedge y > y'$. Point (x', y') is then *dominated by* (x, y). We say that a point is *dominated* if there is a point that dominates it. The interval is active in the collection if and only if the point representing it is not dominated. The set of non-dominated points forms a linear order: the larger x-coordinate implies the smaller y-coordinate. We store the front of non-dominated points in a predecessor/successor structure. Additionally, we maintain a range search tree indexed by x storing in each node the points of the appropriate range of x-coordinates and what is the point with the maximum y among them.

We start by describing the insert-only structure. When a point (x, y) is inserted, we search for its predecessor (x_ℓ, y_ℓ) and its successor (x_r, y_r) in the front of non-dominated points. This way we can either find if (x, y) is dominated by (x_r, y_r) or if it dominates (x_ℓ, y_ℓ). We then update the front and the range search tree appropriately.

To build the delete-only structure, we insert points one by one in any order as described above. When a point (x, y) is deleted, we search for its predecessor (x_ℓ, y_ℓ) and its successor (x_r, y_r) in the front and find what are the points in the range (x_ℓ, x_r) that become non-dominated, that is what are new maximums of nodes in the range search tree after removal of (x, y) from appropriate nodes. These new non-dominated points are added to the front and each interval from the decremental structure is activated only at most once. Thus, the time charged to each interval in the collection is bounded by $\mathcal{O}(\log n)$. ◀

2.2 Decremental structure

For $I_i \in \mathcal{P}_j$, we define LC-DECR(I_i) to be the next greedy choice in $\mathcal{A}_j$ after I_i.

▶ **Proposition 7.** *The set of greedy predecessors of I_i ($\{I_{i'} : \text{LC-DECR}(I_{i'}) = I_i\}$) forms a continuous range of active intervals in $\mathcal{A}_j$.*

Proof. For any active intervals I_1, I_2, we have $I_1 \prec I_2 \Rightarrow \text{LC-DECR}(I_1) \preceq \text{LC-DECR}(I_2)$, so if there are three active intervals $I_1 \prec I_3 \prec I_2$ such that LC-DECR$(I_1) =$ LC-DECR(I_2) then also LC-DECR$(I_3) =$ LC-DECR(I_1). ◀

Intervals of $\mathcal{A}_j$ form a forest where a node representing an interval I_i is the parent of $I_{i'}$'s node when LC-DECR$(I_{i'}) = I_i$. As in the previous section, we add an auxiliary interval to make this representation a tree, we denote it $\mathcal{T}_j$ and call it *a greedy tree of the part $\mathcal{P}_j$*. Greedy predecessors of I_i are the children of node I_i in the greedy tree. We stress that the greedy tree is built only for the intervals of the decremental collection.

We internally represent the greedy tree as an augmented top tree $\mathfrak{T}_j$ [1]. This allows maintaining underlying fully dynamic forest (updates are insertions/deletions of edges and changes to node/edge weights). Because deletions and activations of intervals in the decremental structure may change values of LC-DECR$(\cdot)$ for many nodes, we slightly alter the structure as described in Section 2.3. This is also one of the reasons why one cannot apply techniques described in [12] to solve even the decremental variant of DIS despite being able to efficiently maintain the (monotonic) set of active intervals.

When $I_i \in \mathcal{A}_j$ is to be deleted, its children $C = c_1 \prec c_2 \prec \cdots \prec c_\ell$ have to connect to other nodes of the greedy tree. Let $I_{k+1} = \text{LC-DECR}(I_i)$ before deletion and $I_1 \prec I_2 \prec \cdots \prec I_k$ are the intervals activated after removing I_i. Note that $I_k \prec I_{k+1}$. Nodes other than the elements of C do not change their parents.

We first observe that $I_1, I_2, \ldots, I_{k+1}$ are the only possible parents for nodes in C, remind the fact that $I_{i'} \prec I_{i''} \Rightarrow \text{LC}(I_{i'}) \preceq \text{LC}(I_{i''})$ and use Proposition 7 to see that some (possibly empty) prefix of children sequence $(c_1, c_2, \ldots, c_{r_1})$ has to be connected to I_1, then the next range $(c_{r_1+1}, c_{r_1+2}, \ldots, c_{r_2})$ has to be connected to I_2 and so on until finally some suffix of children sequence $(c_{r_k+1}, c_{r_k+2}, \ldots, c_\ell)$ has to be connected to I_{k+1}. We use binary search on the children sequence to find indices $r_1, r_2, \ldots, r_k$ in this order. We update the parents of the nodes in the found ranges in the greedy tree as described in Section 2.3 and it takes $\mathcal{O}(\text{polylog}\, n)$ per each **activated** interval.

Using the appropriate query to the top tree, we can resume the execution of the greedy algorithm restricted to $\mathcal{A}_j$ from any $I_i \in \mathcal{A}_j$ in $\mathcal{O}(\text{polylog}\, n)$ time.

2.3 Top tree

The underlying information maintained in $\mathfrak{T}_j$ is chosen to compute the following:

- weighted level ancestors,
- nearest marked ancestors,
- the total path weight from a node to the root (the sum of weights).

The discussion on how to maintain information that allows efficient computation of the above in $\mathfrak{T}_j$ can be found in [1].

$\mathfrak{T}_j$ represents an underlying modified greedy tree $\overline{\mathcal{T}}_j$, namely, we binarize the tree by reorganizing the children of each non-leaf node and adding auxiliary nodes as presented in Figure 2. A node in such a modified greedy tree that represents an actual interval has a weight 1, all other auxiliary nodes have a weight 0. The weight of the path between nodes is the sum of the weights of the nodes on the path (including the endpoints). This way, the weight of a path from a node representing an interval I_i to the root of the modified greedy tree represents the number of intervals chosen by the greedy algorithm from I_i.

$\mathcal{A}_j$ is always monotonic so we use $\prec$ order on children. This way, we can update values of LC-DECR$(\cdot)$ for a range of children of a node in $\mathcal{O}(\text{polylog}\, n)$ time by the appropriate splits and joins in $\mathfrak{T}_j$. Apart from auxiliary nodes, pre-order traversals of $\mathcal{T}_j$ and $\overline{\mathcal{T}}_j$ are equal.

$\overline{\mathcal{T}}_j$ and $\mathfrak{T}_j$ are only internal representations of $\mathcal{T}_j$ that enable efficient implementation of the necessary operations. Any updates of $\mathcal{T}_j$ are naturally translated into updates of $\overline{\mathcal{T}}_j$ and $\mathfrak{T}_j$ or were described above. We proceed with describing the further details on $\mathcal{T}_j$.

▶ **Definition 8.** *For an interval $I_i \in \mathcal{A}_j$, we define its* depth *as the depth in $\mathcal{T}_j$. The set of intervals of the same depth d is called* a layer d *in $\mathcal{T}_j$ (or $\mathcal{P}_j$).*

Note that if we traverse the greedy tree in BFS order (visiting children left-to-right) we obtain exactly $\prec$ order. Thus, when comparing two intervals on the same layer we can just see which one is earlier in the pre-order traversal of $\mathcal{T}_j$. This way we can treat layers as sorted collections of intervals (actually, subranges of $\prec$).

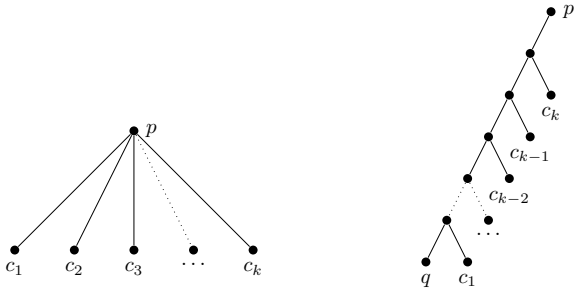


Figure 2 A part of a greedy tree is shown on the left and the modified greedy tree represented by $\mathfrak{T}_j$ is shown on the right. We assume $c_1 \prec c_2 \prec c_3 \prec \ldots \prec c_k$. One can retrieve i-th child of a node by querying for the level ancestor from node q (ignoring the weights of nodes).

▶ Remark 9. We already have all the ingredients for the algorithm to solve the delete-only DIS variant in $\mathcal{O}(\text{polylog}\, n)$ time. In this case, we do not partition intervals nor use a buffer. Instead, we only use the top tree representing the greedy tree of all the active intervals in the whole decremental collection of intervals.

Similarly, we remind that the structure for maintaining the subset of active intervals can be also maintained for the insert-only variant of DIS (Lemma 6). Now we also observe that we can maintain the greedy tree when the intervals are only inserted. A new interval I_i may only improve $\text{LC}(\cdot)$ for some continuous range of intervals and we can binary search the endpoints of this range. To account for the cost of reconnecting these nodes, which may have many different parents, we observe that for any insertion, there is only at most one interval that loses a child in the greedy tree and is not deactivated. We charge the time of reconnection of the range of its children to the insertion of I_i. We charge the time needed to reconnect other nodes to the insertion of their (deactivated, thus actually deleted) parent. This establishes the time complexity of the insert-only variant of DIS to be $\mathcal{O}(\text{polylog}\, n)$.

2.4 Buffer

▶ **Definition 10.** *For intervals $I_1 \in \mathcal{A}_j$ and $I_2 \in \mathcal{B}_j$, we say that* I_1 directly wants to switch *to I_2 if and only if all the following conditions hold:*

- *I_1 ends earlier than I_2,*
- *I_1 and I_2 are compatible,*
- *$I_2 \prec$ LC-DECR(I_1).*

The aim of the above definition is to capture that sometimes the value of $\text{LC}(\cdot)$ may be different from LC-DECR$(\cdot)$. Note that if I_1 directly wants to switch to I_2 it does not necessarily imply that $\text{LC}(I_1) = I_2$. It just means that I_2 is (in sense of $\prec$) a better next greedy choice for I_1 than it appears from the computation in the decremental collection. Note that it also means that the greedy algorithm resumed from any node in the subtree of I_1 in the greedy tree will not choose LC-DECR(I_1). Thus we define the following.

▶ **Definition 11.** *For intervals $I_1 \in \mathcal{A}_j$, $I_2 \in \mathcal{B}_j$ we say that* I_1 wants to switch *to I_2 if and only if there exists an integer $k \geq 0$ such that LC-DECR$^k(I_1)$ directly wants to switch to I_2.*

▶ **Proposition 12.** *For an interval $I_i \in \mathcal{B}_j$, there exist an integer d such that the set of intervals in $\mathcal{A}_j$ that directly want to switch to I_i is either:*

- *a continuous range of a layer d,*
- *a suffix of layer d and a prefix of layer $d+1$.*

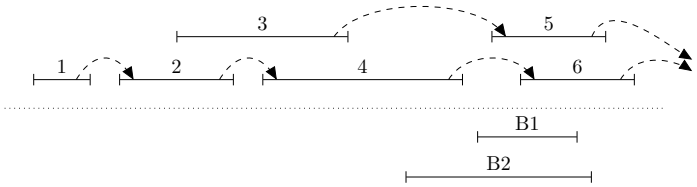


Figure 3 Part of an instance of DIS. Intervals in the decremental collection are shown above the dotted line and buffer intervals are below. Dashed arrows connect intervals with their respective LC-DECR(·). Here intervals 1, 2, 3 and 4 want to switch to B1 (3 and 4 directly) and interval 3 wants to (directly) switch to B2.

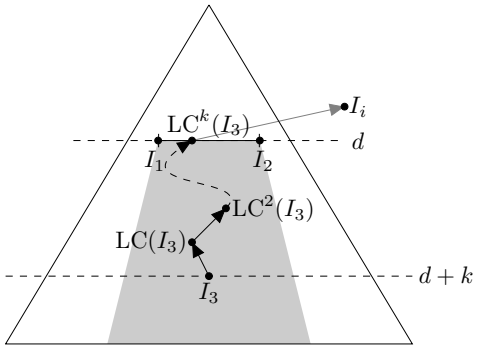


Figure 4 Indirect possibility of switching to $I_i \in \mathcal{B}_j$ for $I_3 \in \mathcal{A}_j$. Nodes of d-th layer in range from I_1 to I_2 directly want to switch to I_i. In the gray area are the nodes that want to switch to I_i.

Proof. Let $I_1 \prec I_3 \prec I_2$ and assume that I_1 and I_2 want to switch to I_i. Then, also I_3 wants to switch to I_i: I_i ends earlier than LC-DECR(I_3) because I_1 wants to switch and I_3 can switch to I_i because I_2 can. This shows that the nodes that want to switch to I_i form a continuous range in $\prec$. Active intervals that directly want to switch to any particular I_i are pairwise overlapping. Indeed, consider two such intervals $I_1 \prec I_2$. If they are compatible then $\mathrm{LC}(I_1) \preceq I_2$, so $\mathrm{LC}(I_1)$ ends earlier than any buffer interval compatible with I_2 to the right of I_2, a contradiction. This also proves that a node and its parent in the greedy tree cannot both directly want to switch to the same buffer interval thus completing the proof. ◀

Proposition 12 shows that the actual size of the information needed to notify the intervals from $\mathcal{A}_j$ that want to directly switch to a particular buffer interval is short. For each buffer interval, it is enough to remember endpoints of at most two ranges.

We also want to efficiently maintain information also indirect switching. Intervals that want to switch to I_i are the nodes in subtree of any node in ranges from Proposition 12. For range from I_1 to I_2 on layer d that wants to directly switch to I_i, any node I_3 on layer $d' = d + k \geq d$ satisfying $I_1 \preceq \text{LC-DECR}^k(I_3) \preceq I_2$ wants to switch to I_i, see Figure 4. We use a 2D range search tree indexed by depth and intervals of $\mathcal{A}_j$ in $\prec$ order. The structure allows us to store a collection of three-sided rectangles, so that given query point we can check if it is contained in at least one of the rectangles. To mark nodes as in Figure 4 we add $[d, +\infty) \times [I_1, I_2]$ to the tree.

An interval may want to switch to multiple intervals but the actual switching point for any $I_i \in \mathcal{A}_j$ is the earliest in $\prec$ (the deepest in $\mathcal{T}_j$) interval that wants to directly switch to a buffer interval on the path from I_i to the root in $\mathcal{T}_j$. We can deduce the actual earliest

switching to buffer interval from any $I_i \in \mathcal{A}_j$ on layer d in $\mathcal{O}(\text{polylog}\, n)$ time by using a binary search on depth $d' \leq d$, each time querying the 2D range search tree if a point (d', I_i) is covered by at least one rectangle. The result for the prefix of the path until reaching the buffer can be obtained from the top tree $\mathfrak{T}_j$. We recreate the whole range search tree after an update in the part.

For any $I_i \in \mathcal{B}_j$ we store the total length of the path to the root of $\mathcal{T}_j$ (this is the internal result for I_i in $\mathcal{P}_j$) and the latest actual interval of $\mathcal{P}_j$ just before reaching the root (this is the exit for I_i in $\mathcal{P}_j$). This information is recomputed for all buffer intervals in $\mathcal{P}_j$ using dynamic programming by iterating the buffer intervals by decreasing end times as follows. For $I_i \in \mathcal{B}_j$, we compute $\text{LC}(I_i)$ and if it is a buffer interval, we use its exit result and its internal result plus 1 as the information for I_i (and, by the order of the computation, we already know these). If $\text{LC}(I_i) \in \mathcal{A}_j$, we query the decremental collection for the next buffer interval after I_i selected by the greedy algorithm as described above and combine its result with the prefix of the traversed path from $\text{LC}(I_i)$ in the decremental collection. This is computed in $\tilde{\mathcal{O}}(n^{1/3})$ time.

3 Interval scheduling on multiple machines

We stress that we assume that there are constant number of machines thus we are going to ignore $\mathcal{O}(\text{poly}\, m)$ factors in time complexities. The difference between naive application of standard techniques and our algorithms is negligible when m is large.

As the main idea of our algorithm is to efficiently simulate the folklore greedy algorithm for IS+ (described in [6, 11]), we now remind it. The intervals are considered separately by the earliest end time. For each considered interval, if there is no available machine at the time, the job is rejected. Otherwise, it is accepted and assigned the available machine that was busy at the latest time. The proof of correctness is a standard exchange argument.

The state of a partial execution (up to some time t) of the greedy algorithm can be fully described by the sequence of length m, where i-th entry describes which interval was last scheduled on i-th machine before or at time t. Some of the entry intervals to $\mathcal{P}_j$ may not belong to $\mathcal{P}_{j-1}$ if some machine had not accepted any intervals in $\mathcal{P}_{j-1}$. At the same time, we want to preprocess information only for tuples of intervals from $\mathcal{P}_j$, thus we need the following additional notation.

▶ **Definition 13.** *The* greedy state G_t *(at time t) is the (multi)set of m input intervals. Each element $I_i = (s_i, f_i) \in G_t$ means that at time t there is a machine that was busy up to time f_i. We use elements $\overline{I_i}$ to indicate that there is a machine which was busy up to time s_i.*

$\overline{\cdot}$ indicates that the particular machine is blocked for all intervals that start too early. Thus, despite each interval can only be selected once, we may want to mark that some machines are busy up to the same time. For this reason, we decided to use multisets for greedy states. $\overline{(s_i, f_i)}$ can be simulated by an artificial interval $(-\infty, s_i)$.

The greedy algorithm only considers values of t that are end times of intervals $I_i = (s_i, f_i)$ in the input. We slightly abuse the notation and use G_k to denote the greedy state at time f_k and assume the intervals are ordered according to the order of the IS+ algorithm i.e. $f_1 < f_2 < \ldots < f_n$. To not consider cases with $|G_k| < m$ we add m pairwise overlapping intervals all ending earlier than the beginning of any actual input interval.

If $G_{k-1} \neq G_k$, exactly one element of G_{k-1} needs to be updated to obtain G_k. It is the one that is ending the latest among the elements of G_{k-1} compatible with I_k. One can see the same from a slightly different perspective. Let us assume that i is the index for which

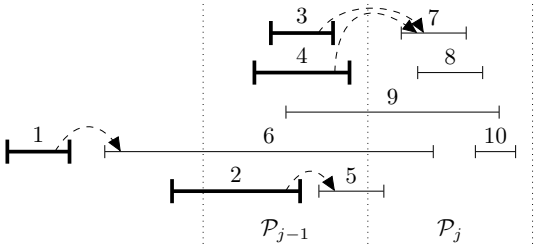


Figure 5 Translation of an exit greedy state $G = \{1, 2, 3, 4\}$ from part $\mathcal{P}_{j-1}$. Each interval of G is rounded to the earliest starting interval in $\mathcal{P}_j$ that is later than the end of the interval (denoted by dashed directed edge). Thus, we can assume that the entry greedy state in $\mathcal{P}_j$ is $\{\overline{5}, \overline{6}, \overline{7}, \overline{7}\}$.

$\mathrm{LC}(I_i)$ is the earliest ending interval among G_k. Then $G_k = G_{k+1} = \ldots = G_{k'-1} \neq G_{k'}$ and $G_{k'} = G_k \setminus \{I_i\} \cup \{\mathrm{LC}(I_i)\}$. We call $G_{k'}$ the *next greedy state after* G_k and denote it $\textsc{Next}(G_k)$. Because $\mathrm{LC}(\cdot)$ can be computed in $\mathcal{O}(\mathrm{polylog}\, n)$ time using the appropriate structure as described in Appendix A, we iterate through all candidates for I_i in the greedy state and thus have the following.

▶ **Corollary 14.** *$\textsc{Next}(G_k)$ can be computed in $\tilde{\mathcal{O}}(m)$ time for any G_k.*

We use insights from Section 2 and Appendix A and split the intervals into $\mathcal{O}(n^{1-1/m})$ parts of size at most $\mathcal{O}(n^{1/m})$. We restrict $\mathrm{LC}(\cdot)$ to only consider intervals in the same part as the argument of the operation (it can return $\perp$). We build an additional structure for internal intervals in each part and rebuild it every update in the part. As in the case of interval scheduling on one machine, our goal is to be able to efficiently handle (in $\mathcal{O}(\mathrm{polylog}\, n)$ time) a query for the internal result (the number of accepted intervals) and the exit greedy state from the part for a given entry greedy state G_k in the part – we call this the *part query from the greedy state G_k*.

Notice that during the execution of the greedy algorithm up to $\mathcal{P}_{j-1}$, it may happen that some machine will not accept any new interval in $\mathcal{P}_{j-1}$, so the exit greedy state coming from $\mathcal{P}_{j-1}$ may contain intervals also from earlier parts. Let us now describe how to translate such an exit greedy state coming from $\mathcal{P}_{j-1}$ into an entry greedy state of $\mathcal{P}_j$, so we can later only consider the content of one part. We observe that the decisions of the greedy algorithm only depend on the relative order of endpoints of the considered intervals. If a machine was busy up to time t and there are no intervals starting before time $t' > t$, we can safely assume that the machine is busy up to time t' without changing the execution of the greedy algorithm. Thus, we round up the end of each interval in the greedy state to the earliest start of some interval in $\mathcal{P}_j$. See Figure 5. We stress that the result of rounding is not necessarily part of the solution generated by our algorithm. It just indicates times up to which the machines are busy. After computing the exit greedy state for $\mathcal{P}_j$, we inspect if there are machines that have not accepted any intervals from $\mathcal{P}_j$ and revert the rounding for these.

We stick to Definition 4, but we cannot make direct use of Lemma 5 because in the case of multiple machines it may happen that inactive intervals are part of the optimal solution. As these intervals may not form a monotonic collection, we redefine $\prec$ order as follows: $(s_1, f_1) \prec (s_2, f_2) \Leftrightarrow f_1 < f_2$. We still maintain the greedy tree $\mathcal{T}_j$ and the top tree $\mathfrak{T}_j$[1] as described for one machine. We identify intervals with the nodes representing them in $\mathcal{T}_j$.

[1] We could also use simpler structures as we only need a subset of operations provided by the top tree and we can afford to rebuild the structure from scratch every update.

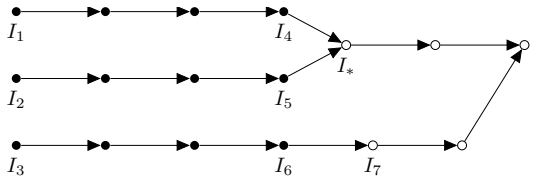


Figure 6 Lemma 16 for $G = \{I_1, I_2, I_3\}$. Here we assume $I_6 \prec I_* \prec I_7$. Elements of $\mathrm{N}(\cdot, \mathrm{I}_*)$ are filled dots. We have $\textsc{Next}^9(G) = \{I_4, I_5, I_6\}$ and $I_* \in \textsc{Next}^{10}(G)$.

▶ **Lemma 15.** *Let $I_i = (s_i, f_i)$ be an inactive interval and let $I_{i'} = (s_{i'}, f_{i'})$ be the latest (in $\prec$) interval contained inside I_i. If $I_i \in G_i$ then also $I_{i'} \in G_i$.*

Proof. Any interval compatible with I_i is also compatible with $I_{i'}$ and $I_{i'}$ ends earlier than I_i. This means that if I_i is accepted then also $I_{i'}$ is (at time $f_{i'}$). From time $f_{i'}$ up to time f_i the machine that accepted $I_{i'}$ cannot accept other interval: it would have to start after $f_{i'}$ and end before f_i thus violating our assumption that $I_{i'}$ is the latest interval contained inside I_i. This implies that $I_{i'} \in G_i$. ◀

▶ **Lemma 16.** *Let $G = \{I_1, I_2, \ldots, I_m\}$ be a greedy state for which all elements are active intervals. Let $I_* = (s_*, f_*)$ be the earliest (in $\prec$) interval being a common ancestor of any pair of elements of G. Let $\mathrm{N}(I_i, I_*)$ be a prefix of $\{I_i, LC(I_i), LC^2(I_i), \ldots\}$ intervals preceding I_* and let $\mathrm{S}(I_i, I_*)$ be the latest of $\mathrm{N}(I_i, I_*)$.*

Then $\mathrm{N}(I_1, I_*) \cup \mathrm{N}(I_2, I_*) \cup \ldots \cup \mathrm{N}(I_m, I_*)$ are the only elements scheduled by the greedy algorithm for IS+ resumed from G before reaching time f_*. Additionally, just before time f_* the greedy state of the algorithm is $\{\mathrm{S}(I_i, I_*) : i \in \{1, 2, \ldots, m\}\}$.

Proof. First, we make a technical note that thanks to the artificial root added to form the greedy tree, the interval I_* always exists.

The candidates for values of $\textsc{Next}(G_t)$ are G_ts with exactly one of the intervals replaced by its $\mathrm{LC}(\cdot)$ assigned to the same machine. Thus, by using this reasoning inductively for $\textsc{Next}^k(G)$ for increasing k, we observe that when moving forward along the path from any $I_i \in G$ to the root in the greedy tree, at least until reaching some interval $\succeq I_*$, all the traversed intervals will be scheduled on the same machine as I_i. Additionally, for different $I_{i'}, I_{i''} \in G$, the paths from $I_{i'}$ and $I_{i''}$ in the greedy tree do not share any nodes that are $\prec I_*$ (by definition). This way, all and the only elements that are included in some greedy state after G before considering I_* are the elements of $\mathrm{N}(I_i, I_*)$ and also just before considering I_* all the latest elements of $\mathrm{N}(\cdot, I_*)$ are in the greedy state. See Figure 6. ◀

If the elements of greedy state G are all active, we can naively compute I_* as in Lemma 16 by checking LCAs of all pairs of intervals in G in the greedy tree and then proceeding to the last interval before I_* independently from each node to obtain the last greedy state before reaching I_* as in Figure 6. Thus we have the following.

▶ **Corollary 17.** *Let $G = \{I_1, I_2, \ldots, I_m\}$ be a greedy state with only active intervals and let I_* be defined as in Lemma 16. It is possible to compute both the smallest k for which $I_* \in \textsc{Next}^k(G)$ and the value of $\textsc{Next}^k(G)$ itself in $\tilde{\mathcal{O}}(m^2)$ time.*

3.1 An $\tilde{\mathcal{O}}(n^{1/2})$-time algorithm for two machines

In this section, we focus on describing an efficient algorithm for dynamic interval scheduling on two machines and prove the following.

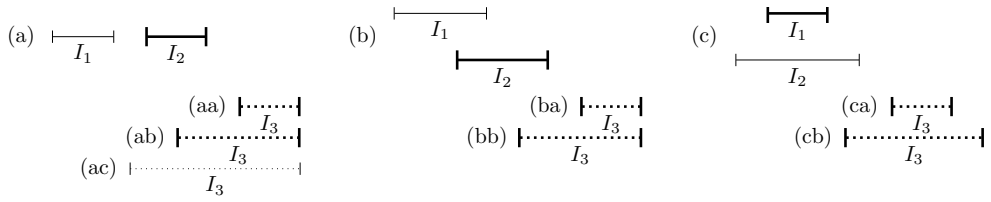


Figure 7 Three possible forms of a greedy state and cases as in Lemma 19. Active intervals are marked with bold lines and potential cases for I_3 are marked with dotted lines. Note that I_1 in forms (a) and (b) may be either active or inactive.

▶ **Theorem 18.** *There is a data structure for DIS2 that supports any sequence of n insert/delete/query on intervals in $\tilde{\mathcal{O}}(n^{1/2})$ amortized time per each operation.*

▶ **Lemma 19.** *There are only three possible forms of a greedy state for two machines.*
(a) $\{I_1, I_2\}$ *where* $I_1 \prec I_2$, I_1 *and* I_2 *are compatible and* I_2 *is active,*
(b) $\{I_1, I_2\}$ *where* I_1 *ends earlier than* I_2, I_1 *and* I_2 *are overlapping and* I_2 *is active,*
(c) $\{I_1, I_2\}$ *where an active interval* I_1 *is fully contained inside (an inactive)* I_2.

Proof. First we assume, without losing generality, that the greedy algorithm considered at least two intervals and resumes from the greedy state G_2 that is of the form (a) and $I_1 = (s_1, f_1)$, $I_2 = (s_2, f_2)$. One can easily prepend any instance of DIS+ with few intervals to achieve this.

Let assume that the next accepted interval by the greedy algorithm is $I_3 = (s_3, f_3)$ and the next greedy state after G_2 is G_3. There are three cases (see Figure 7):
(aa) $f_2 < s_3$ – then I_3 is an active interval and $G_3 = \{I_1, I_3\}$ is of the form (a),
(ab) $s_2 < s_3 < f_2$ – then I_3 is an active interval and $G_3 = \{I_2, I_3\}$ is of the form (b),
(ac) $s_3 < s_2$ – then I_3 is an inactive interval and $G_3 = \{I_2, I_3\}$ is of the form (c).
We now proceed to similar analysis of what are the forms of next greedy states that can be reached from states of the form (b) and (c).

If G_2 is of the form (b) then I_3 is either compatible with I_2 (case (ba)) and G_3 is of the form (a), or it overlaps with I_2 (case (bb)) and G_3 is of the form (b). Note that I_3 can not overlap with I_1 as then I_3 would be rejected.

Similarly, if G_2 is of the form (c) then I_3 is either compatible with I_2 (case (ca)) and G_3 is of the form (a), or it overlaps with I_2 (case (cb)) and G_3 is of the form (b).

No other forms than (a), (b) or (c) are reachable from (a) and this concludes the proof. ◀

We now describe our algorithm for DIS2. For each part it maintains the following:
- $B[I_i]$ for all active intervals I_i – the result of part query from the greedy state $\{I_i, I_{i'}\}$ of the form (b) where $I_{i'}$ is direct successor (in $\prec$ order),
- $C[I_i]$ for all inactive intervals I_i – the result of part query from the greedy state $\{I_i, I_{i'}\}$ of the form (c) where $I_{i'}$ is the latest (in $\prec$ order) active interval fully inside I_i.

When a part is updated, $B[\cdot]$ and $C[\cdot]$ structures are rebuilt from scratch. Computation of $B[I_i]$ or $C[I_i]$ is nothing else than answering a part query for the appropriate greedy state. We ask these queries in decreasing order of the sum of indices (in $\prec$ order) of the two intervals of the greedy state. This way, during the recomputation of $B[\cdot]$ and $C[\cdot]$ structures, whenever the algorithm is going to use some other result of $B[\cdot]$ or $C[\cdot]$ it is already computed as the queried sum of indices will be larger.

Additionally, for each active interval I_i we precompute the earliest (in $\prec$) interval $I_{i'}$ not on the path from I_i to the root of $\mathcal{T}_j$. We do this using dynamic programming, inspecting all the intervals in decreasing order of $\prec$ and it takes $\tilde{\mathcal{O}}(n^{1/2})$ time. Similarly, we precompute the number of intervals on the path from I_i to the latest interval ending earlier than $I_{i'}$.

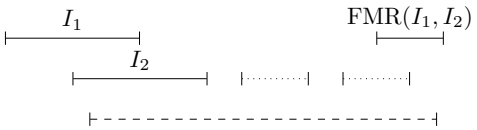


Figure 8 Both dotted intervals are accepted by the machine that accepted I_2 and the dashed interval overlaps with I_1 so is rejected. The left dotted interval in the example is $I_{i'''}$, the solution of the subproblem from the computation of FMR(I_1, I_2).

We now describe how to answer the part query from a greedy state G_i following the proof of Lemma 19 and considering all forms of G_i.

If G_i is of the form (a) we focus on finding the greedy state $G_{i'} = \textsc{Next}^k(G_i)$ for which k is the smallest such that $I_1 \notin \textsc{Next}^k(G_i)$. If I_1 is replaced in $G_{i'}$ by an active interval $I_{i'}$, it has to be the earliest (in $\prec$) interval overlapping with an interval $I_{i''}$ on the path from I_2 to the root in $\mathcal{T}_j$ (it can also be I_2 itself). We know which one and what is the contribution to the internal result as we precomputed it. Moreover, we observe that $G_{i'} = \{I_{i'}, I_{i''}\}$ and its part result is stored in $B[I_{i''}]$ so we just read the result from there. If I_1 is replaced in $G_{i'}$ by an inactive interval $I_{i'}$ it has to be the earliest (in $\prec$) interval compatible with I_1. Then $G_{i'} = \{I_{i'}, I_{i''}\}$ where $I_{i''}$ is the latest (in $\prec$ order) active interval fully inside $I_{i'}$. Thus, we read the part result for $G_{i'}$ from $C[I_{i'}]$.

If G_i is of the form (b), then $\textsc{Next}(G_i)$ is either of the form (a) for which we proceed as described above or of the form (b) but with both greedy state intervals active (case (bb) of the proof of Lemma 19), for which we use Corollary 17 to reach the greedy state of the form (a) and later proceed as described above.

If G_i is of the form (c), then $\textsc{Next}(G_i)$ is either of the form (a) or (b) and we proceed as described above.

3.2 An $\tilde{\mathcal{O}}(n^{1-1/m})$-time algorithm for $m \geq 3$ machines

Surprisingly, before we start describing the final algorithm for $m \geq 3$ machines, we need an additional building block for the two machine case.

▶ **Definition 20.** *For a collection of intervals S, for $I_1 \prec I_2$ from S, we define the* first machine replacement *$FMR(I_1, I_2)$ to be the interval in S which replaces I_1 in the greedy state when resuming the greedy execution from the greedy state $\{I_1, I_2\}$ on two machines. In other words, $FMR(I_1, I_2)$ is the earliest ending accepted interval after I_2 that will be scheduled on the same machine as I_1 by the greedy algorithm for IS+.*

Within the desired time bounds, for $m \geq 3$, we can afford recomputing FMR$(\cdot, \cdot)$ in parts from scratch for every pair of intervals in the updated part, as long as this recomputation takes $\tilde{\mathcal{O}}(|\mathcal{P}_j|^2)$ time. We could not do the same for $m = 2$.

▶ **Lemma 21.** *The values of $FMR(\cdot, \cdot)$ for all pairs of intervals in a collection of n intervals, can be computed in $\tilde{\mathcal{O}}(n^2)$ time.*

Proof. Assuming that intervals in part are ordered by $\prec$ and given names $I_1, I_2, \ldots$ in line with this order, we compute FMR$(I_{i'}, I_{i''})$ in decreasing order of the sum of $i' + i''$ indices. To compute FMR$(I_{i'}, I_{i''})$, for $I_{i'} = (s_{i'}, f_{i'})$ and $I_{i''} = (s_{i''}, f_{i''})$ we first find the earliest ending interval $I_{i'''}$ that ends later than $I_{i''}$ and is compatible with $I_{i'}$. To solve this subproblem we take a geometric view: each interval (s_i, f_i) is converted into a point (s_i, f_i) in 2D plane, the goal is to find the point with smallest y-coordinate above and to the right of $(s_{i'}, f_{i''})$. This

is solved by a 2D range search tree indexed by (x, y)-coordinates storing the appropriate result. Thus, the subproblem is solved. We proceed with the computation of $\mathrm{FMR}(I_{i'}, I_{i''})$. We have two cases: either $I_{i'''}$ is overlapping with $I_{i''}$ and then $\mathrm{FMR}(I_{i'}, I_{i''}) = I_{i'''}$ or $I_{i'''}$ is compatible with $I_{i''}$ and then $\mathrm{FMR}(I_{i'}, I_{i''}) = \mathrm{FMR}(I_{i'}, I_{i'''})$ which is already known by the order of the computation. We can also compute the number of intervals chosen by the greedy algorithm when resumed from state $\{I_{i'}, I_{i''}\}$ until reaching $\mathrm{FMR}(I_{i'}, I_{i''})$ (just 1 or the number chosen from $I_{i'}, I_{i'''}$ plus 1 depending on the above cases). ◀

As it turns out, the values of $\mathrm{FMR}(\cdot, \cdot)$ play important role in the algorithm for $m \geq 3$. We want to preprocess tuples of possible entry greedy states for a part to be able to efficiently answer part queries. The problem is that we have $\mathcal{O}(n^{1/m})$ intervals in each part, but we aim at $\tilde{\mathcal{O}}(n^{1-1/m})$ time complexity. Thus, we cannot precompute part queries for all possible greedy states. Instead, we carefully select specific *compressible* greedy states for which part query results are actually stored and design an algorithm that can push the simulation forward to the next compressible state or the exit state from the part.

▶ **Definition 22.** *Let $G_t = \{I_1, I_2, \ldots, I_m\}$ be a greedy state. We assume $I_1 \preceq I_2 \preceq \ldots \preceq I_m$. We say that G_t is* compressible *if at least one of the following conditions hold:*
(a) *I_m is inactive,*
(b) *I_m is active and exists active interval I_p such that $LC(I_p) = I_m$,*
(c) *$I_m = FMR(I_1, I_{m-1})$.*

▶ **Lemma 23.** *In a part of n intervals for DIS+ on m machines, there are only $\mathcal{O}(n^{m-1})$ compressible greedy states.*

Proof. We consider all the forms of the compressible greedy state as in Definition 22.
(a) from Lemma 15 we know that the greedy state also contains the latest interval fully inside I_m and thus we can forget this interval, so there are $\mathcal{O}(n^{m-1})$ such states,
(b) there is an edge (I_p, I_m) in the greedy tree of $\mathcal{P}_j$, we can store $(m-2)$-tuple of other intervals and the identifier of the appropriate edge, so there are $\mathcal{O}(n^{m-1})$ such states,
(c) we forget I_m as it is equal to $\mathrm{FMR}(I_1, I_{m-1})$, so there are $\mathcal{O}(n^{m-1})$ such states. ◀

Note that we can decompress the representations from Lemma 23 in $\mathcal{O}(m)$ time to obtain a full greedy state of size m. Also, by taking into account the sizes of the parts, we obtain that there are only $\mathcal{O}(n^{1-1/m})$ compressible greedy states for n intervals in S.

For an update in $\mathcal{P}_j$, we recompute part query results for all compressible greedy states in $\mathcal{P}_j$. As in Section 3.1, we do this using dynamic programming, in decreasing order of the sum of indices of the uncompressed state. The problem of computing the results for the states stored in the dynamic programming table is once again translated into the general query that has m-tuple as an input and has to push the simulation forward either to the next part or at least to a compressible greedy state from which we read the already preprocessed result and combine it with the traversed prefix of the path. We proceed with describing how to solve this general query.

We distinguish three forms of the greedy state $G = \{I_1, I_2, \ldots, I_m\}$ for $I_1 \preceq I_2 \preceq \ldots \preceq I_m$:
(*) I_m is inactive,
(**) there is $1 < p \leq m$ such that all intervals $I_p, I_{p+1}, \ldots, I_m$ are active,
(***) all $I_1, I_2, \ldots, I_m$ are active.

For case (*), we compute $G' = \textsc{Next}(G)$. Either the latest accepted interval in G' is inactive and then G' is compressible of type (a) or it is active, thus G' is of the form (**) or (***) and we proceed with it as described below.

For case (***), we use Corollary 17 to find the earliest greedy state $G' = \textsc{Next}^k(G)$ for which $I_* \in G'$. We observe that such G' is compressible of type (b), as both I_* and at least one of its children are elements of G'.

We now consider case (**). We compute $G' = \textsc{Next}(G)$ and consider the following subcases depending on the latest accepted interval I_+ in G':

(1) I_+ is inactive,
(2) I_+ is active, overlapping with I_m and $I_1 \in G'$,
(3) I_+ is active, overlapping with I_m and $I_1 \notin G'$,
(4) I_+ is active and not overlapping with I_m.

In case (1), we see that G' is compressible of type (a). In case (2), we observe that $I_+ = \mathrm{FMR}(I_1, I_m)$, so G' is compressible of type (c). In case (3), we observe that G' remains of type (**), but with smaller p. We proceed with computing $\textsc{Next}(G')$ until we reach any other case, which happens after at most $\mathcal{O}(m)$ iterations. In case (4), we observe that $G' = G \setminus \{I_m\} \cup \{I_+\}$ and I_+ is compatible with every other interval from G'. We read $\mathrm{FMR}(I_1, I_+)$ and push the simulation forward until reaching the first greedy state G'' with accepted interval I_{++} that will be scheduled on a different machine than I_m. Notice that if I_{++} is active then it is compatible with I_1 so $I_{++} = \mathrm{FMR}(I_1, I_m)$. As $m \geq 3$, I_{++} will replace $I_{m-1} \neq I_1$ in the greedy state thus G'' is compressible of type (c) and if I_{++} is inactive then G'' is compressible of type (a).

4 Lower bound for Dynamic Weighted Interval Scheduling

The Minimum Weight k-Clique problem is to find, in an edge-weighted graph, a clique of exactly k nodes having the minimum total weight of edges.

The following hypothesis about Minimum Weight k-Clique problem was formulated.

▶ **Conjecture 24** (Min Weight $(2\ell+1)$-Clique Hypothesis [22]). *There is a constant $c > 1$ such that, on a Word-RAM with $\mathcal{O}(\log n)$-bit words, finding a k-Clique of minimum total edge weight in an n-node graph with non-negative integer edge weights in $[1, n^{ck}]$ requires $n^{k-o(1)}$ time.*

The Minimum Weight $(2\ell+1)$-Cycle problem is to find, in an edge-weighted graph, a cycle consisting exactly $2\ell+1$ edges having the minimum total weight.

▶ **Theorem 25** ([22]). *If there is an integer $\ell \geq 1$ and a constant $\epsilon > 0$ such that Minimum Weight $(2\ell+1)$-Cycle in a directed weighted n-node $m = \Theta(n^{1+1/\ell})$-edge graph can be solved in $O(mn^{1-\epsilon} + n^2)$ time, then the Min Weight $(2\ell+1)$-Clique Hypothesis is false.*

Based on the above, we formulate the following.

▶ **Theorem 26.** *Unless the Min Weight $(2\ell+1)$-Clique Hypothesis is false, for all $\epsilon > 0$ there is no algorithm for DWIS problem with $O(n^{1-\epsilon})$ update and query time.*

Proof. As in [22], we use the fact that Minimum Weight $(2\ell+1)$-Cycle is still hard if restricted only to k-circle layered graphs, that is k-partite graphs in which, for each $i \in [k]$, all edges from nodes in i-th part end in $(i \bmod k+1)$-th part.

We reduce Minimum Weight $(2\ell+1)$-Cycle in a weighted $(2\ell+1)$-circle layered graph to DWIS. The input instance has n nodes and $m = \Theta(n^{1+1/\ell})$ edges of integer weights in range $[n^{c\ell} = W]$ for large enough c. We enumerate parts from 1 to $2\ell+1$ and we enumerate nodes independently in each parts starting from 0. For all $p \in [2\ell]$, for all edges from u-th node in p-th part to v-th node in $(p+1)$-th part, we insert an interval $[(p-1)n + u,\ pn + v)$ of weight $(f_i - s_i)(2\ell+1)(W+1) + (W - w)$ where w is the edge weight.

The optimal cycle has to go through some node s in the first part. We guess this node by inserting an interval $[-1, s)$ of weight $(f_i - s_i)(2\ell+1)(W+1)$ and, for all edges from u-th node in $(2\ell+1)$-th part to s of weight w, we insert an interval $[2\ell \cdot n + u,\ (2\ell+1) \cdot n)$ of weight $(f_i - s_i)(2\ell+1)(W+1) + (W-w)$. To start with another choice of s, we delete the corresponding intervals before inserting the new ones.

The selection of edge weights in our instance guarantees that the optimal solution maximizes the total length of chosen intervals and then minimizes the weight resulting from weights of edges in the graph, as each unit of length increases the value of the solution by $(2\ell+1)(W+1)$ while the additional gain from edge weights is, in total, at most $(2\ell+1)W$.

The only possibility to obtain the value of at least $(2\ell+1)(W+1)n$ is to choose the intervals spanning the whole interval $[-1, (2\ell+1)n)$ in the created instance. Such selection ensures that an interval representing node s in the first part is selected, as well as all intervals representing the edges on the cycle, including the last edge going to the first part represented by the interval with $f_i = (2\ell+1)(W+1)$. Because in this scenario there is no gap nor overlap in coordinates of the selected intervals, any two consecutive edges share a common node, so they form a $(2\ell+1)$-cycle. Thus, there is 1-1 correspondence between $(2\ell+1)$-cycles going through node s in the first part and solutions of weight at least $(2\ell+1)(W+1)n$. Nodes of the optimal $(2\ell+1)$-cycle can be deduced by inspecting endpoints of the selected intervals.

To solve Minimum Weight $(2\ell+1)$-Cycle by the above reduction we invoked $\mathcal{O}(m)$ insertions and deletions to DWIS structure. By choosing the input instance to have $\ell = \frac{1}{\epsilon}$ and $n = c\ell$ for large enough c, and assuming (ad absurdum) that these $\mathcal{O}(m) = \mathcal{O}(n^{1+\epsilon})$ operations took $\mathcal{O}(m \cdot m^{1-\epsilon})$ time, we obtained $\mathcal{O}(n^{2+\epsilon-\epsilon^2})$-time algorithm for the Minimum Weight $(2\ell+1)$-Cycle problem, thus violating Theorem 25. ◀

References

1. Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, October 2005. doi:10.1145/1103963.1103966.
2. Amihood Amir and Itai Boneh. Dynamic suffix array with sub-linear update time and poly-logarithmic lookup time. *CoRR*, abs/2112.12678, 2021.
3. Esther M. Arkin and Ellen B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1–8, 1987. doi:10.1016/0166-218X(87)90037-0.
4. Michael A Bender, Richard Cole, Erik D Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *European Symposium on Algorithms*, pages 152–164. Springer, 2002.
5. Khalid I. Bouzina and Hamilton Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3):379–393, December 1996. doi:10.1007/BF00121680.
6. Martin C. Carlisle and Errol L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(3):225–235, 1995. doi:10.1016/0166-218X(95)80003-M.
7. Spencer Compton, Slobodan Mitrović, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling, 2020. arXiv:2012.15002.
8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
9. P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 365–372, New York, NY, USA, 1987. Association for Computing Machinery. doi:10.1145/28395.28434.
10. Christof Doll, Tanja Hartmann, and Dorothea Wagner. Fully-dynamic hierarchical graph clustering using cut trees. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 338–349, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
11. Ulrich Faigle and Willem M. Nawijn. Note on scheduling intervals on-line. *Discrete Applied Mathematics*, 58(1):13–17, 1995. doi:10.1016/0166-218X(95)00112-5.

12 Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theoretical Computer Science*, 562:227–242, 2015. doi:10.1016/j.tcs.2014.09.046.

13 Pawel Gawrychowski and Wojciech Janczewski. Fully dynamic approximation of LIS in polylogarithmic time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 654–667. ACM, 2021. doi:10.1145/3406325.3451137.

14 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015.

15 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic Approximate Maximum Independent Set of Intervals, Hypercubes and Hyperrectangles. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2020.51.

16 Monika R. Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, July 1999. doi:10.1145/320211.320215.

17 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001. doi:10.1145/502090.502095.

18 Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. *CoRR*, abs/2201.01285, 2022. arXiv:2201.01285.

19 Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison Wesley, January 2006.

20 Tomasz Kociumaka and Saeed Seddighin. Improved dynamic algorithms for longest increasing subsequence. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 640–653, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451026.

21 Antoon W.J. Kolen, Jan Karel Lenstra, Christos H. Papadimitriou, and Frits C.R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007. doi:10.1002/nav.20231.

22 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. *Tight Hardness for Shortest Cycles and Paths in Sparse Graphs*, pages 1236–1252. Society for Industrial and Applied Mathematics, 2018. doi:10.1137/1.9781611975031.80.

23 Mihai Patrascu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. doi:10.1137/S0097539705447256.

24 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

A An $\tilde{\mathcal{O}}(n^{1/2})$-time algorithm for one machine

Here we show a simple structure that already needs a subset of the ideas used in more complicated and faster structures described in the paper. We show the data structure for DIS showing the following.

▶ **Theorem 27.** *There is a data structure for DIS that supports any sequence of n insert/delete operations in $\mathcal{O}(\sqrt{n}\log n)$ amortized time per update.*

We choose the set of *separators* $x_1 < x_2 < \cdots < x_k$. Separators split the universe of coordinates into $k+1$ *parts*: $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_k$. Assuming that $x_{k+1} = +\infty$, each part $\mathcal{P}_j$ contains intervals (s_i, f_i) with $x_j \le s_i < x_{j+1}$. Thus, at any time parts represent partition of intervals. We define the *size* of a part as the number of intervals in it. Intervals store references to parts in which are contained.

Separators (and pointers to the appropriate parts) are stored in a predecessor/successor data structure (we use balanced binary search trees [8] [2]) and are chosen to satisfy the following invariant: each part has size at most $2\sqrt{N}$ and for every two consecutive parts $\mathcal{P}_j$ and $\mathcal{P}_{j+1}$ at least one has size at least $\frac{1}{2}\sqrt{N}$. Thus, there are $\mathcal{O}(\sqrt{n})$ parts at any time and local rebuild of parts of size $\mathcal{O}(\sqrt{n})$ happens after $\Omega(\sqrt{n})$ operations affecting the part. As these rebuilds are simply appropriate separate insertions, the amortized update time complexity does not change.

Intervals in part $\mathcal{P}_j$ satisfying $f_i < x_{j+1}$ are called *internal* and all the others are called *external*. Internal intervals are stored in predecessor/successor data structures: sorted by s_i and, separately, sorted by f_i. Additionally, we have the same structures defined globally, for all the intervals in S. This allows to compute $\mathrm{LC}(I_i)$ in $\mathcal{O}(\log n)$ time.

For each internal interval I_i in part $\mathcal{P}_j$ we store its leftmost compatible internal interval in the same part, denoted by $\text{LC-INT}(I_i)$ (either $\mathrm{LC}(I_i)$ or $\perp$ in case $\mathrm{LC}(I_i) \notin \mathcal{P}_j$ or is external). Additionally, we store the information to resume the greedy execution from an internal interval I_i to the latest interval in the same part. This includes: $\text{RES-INT}(I_i)$ – the largest $r \geq 0$ such that $\text{LC-INT}^r(I_i) \neq \perp$ and $\text{EXIT-INT}(I_i) = \mathrm{LC}^{\text{RES-INT}(I_i)-1}(I_i)$.

When the content of $\mathcal{P}_j$ is updated, all the above values for intervals of $\mathcal{P}_j$ are recomputed naively from scratch: we start with computing $\text{LC-INT}(\cdot)$ in decreasing order of f_i. We set $\text{LC-INT}((s_i, f_i))$ to be the interval I' with the smallest $f_{i'}$ among intervals with $s_{i'} \geq f_i$ or $\perp$ if there is no such interval. We update which interval is I' whenever the computation of $\text{LC-INT}(\cdot)$ proceeds to smaller values of f_i by querying the appropriate part structure (containing only internal intervals) sorted by s_i. Overall, this naive recomputation of all information for all internal intervals in the part takes $\mathcal{O}(\sqrt{n}\log n)$ time.

With the above, we can resume the greedy algorithm from any I_i in any $\mathcal{P}_j$ until reaching the earliest interval in the solution outside $\mathcal{P}_j$ in $\mathcal{O}(\log n)$ time. For an external interval I_i it is enough to proceed to $\mathrm{LC}(I_i)$ to exit $\mathcal{P}_j$. If I_i is internal, we increase the total result by the number of selected intervals in the part (the internal result for I_i) and proceed to $I_{i'} = \mathrm{LC}(\text{EXIT-INT}(I_i))$. $I_{i'}$ may already be in some further part or it may be an external interval in $\mathcal{P}_j$ and then we proceed to $\mathrm{LC}(I_{i'}) \notin \mathcal{P}_j$.

To answer a DIS query, we simulate the execution of the greedy algorithm starting from the earliest ending interval and traversing the parts as described above. The query as described takes $\mathcal{O}(\sqrt{n}\log n)$ time.

To insert an interval (s_i, f_i) to S, we first locate the appropriate part $\mathcal{P}_j$ in the separators structure, insert the interval into $\mathcal{P}_j$, recompute the additional information associated with $\mathcal{P}_j$ and update the global structures. All these takes $\mathcal{O}(\sqrt{n}\log n)$ time. During insertion, it may happen that $\mathcal{P}_j$ becomes too large. In this case, if the size reached s, we naively find (using an appropriate predecessor/successor structure) $\lceil \frac{s}{2} \rceil$-th value x in the set of s_is of all intervals in $\mathcal{P}_j$ and add x as the new separator. This splits $\mathcal{P}_j$ into two new parts, which we recompute from scratch. This, again, works in $\mathcal{O}(\sqrt{n}\log n)$ time.

Deletion of an interval of $\mathcal{P}_j$ is similar and in the case of underflow of pair $\mathcal{P}_j$ and $\mathcal{P}_{j+1}$ or $\mathcal{P}_{j-1}$ and $\mathcal{P}_j$, we merge the parts by removing the separator between them and recompute the new part.

[2] If for all intervals s_i, f_i are small integers bounded by U, we could use y-fast tries [24]. This way we could achieve $\mathcal{O}(\sqrt{n}\log\log U)$ amortized time per operation.

Galloping in Fast-Growth Natural Merge Sorts

Elahe Ghasemi
Sharif University of Technology, Teheran, Iran
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Vincent Jugé
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Ghazal Khalighinejad
Duke University, Durham, NC, USA
Sharif University of Technology, Teheran, Iran

Abstract

We study the impact of sub-array merging routines on merge-based sorting algorithms. More precisely, we focus on the *galloping* sub-routine that TimSort uses to merge monotonic (non-decreasing) sub-arrays, hereafter called *runs*, and on the impact on the number of element comparisons performed if one uses this sub-routine instead of a naive merging routine.

The efficiency of TimSort and of similar sorting algorithms has often been explained by using the notion of *runs* and the associated *run-length entropy*. Here, we focus on the related notion of *dual runs*, which was introduced in the 1990s, and the associated *dual run-length entropy*. We prove, for this complexity measure, results that are similar to those already known when considering standard run-induced measures: in particular, TimSort requires only $\mathcal{O}(n + n\log(\sigma))$ element comparisons to sort arrays of length n with σ distinct values.

In order to do so, we introduce new notions of *fast-* and *middle-growth* for natural merge sorts (i.e., algorithms based on merging runs). By using these notions, we prove that several merge sorting algorithms, provided that they use TimSort's galloping sub-routine for merging runs, are as efficient as TimSort at sorting arrays with low run-induced or dual-run-induced complexities.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases Sorting algorithms, Merge sorting algorithms, Analysis of algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.68

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2012.03996

1 Introduction

In 2002, Tim Peters, a software engineer, created a new sorting algorithm, which was called TimSort [20] and was built on ideas from McIlroy [17]. This algorithm immediately demonstrated its efficiency for sorting actual data, and was adopted as the standard sorting algorithm in core libraries of widespread programming languages such as Python and Java. Hence, the prominence of such a custom-made algorithm over previously preferred *optimal* algorithms contributed to the regain of interest in the study of sorting algorithms.

$$S = (\underbrace{12, 7, 6, 5}_{\text{first run}}, \underbrace{5, 7, 14, 36}_{\text{second run}}, \underbrace{3, 3, 5, 21, 21}_{\text{third run}}, \underbrace{20, 8, 5, 1}_{\text{fourth run}})$$

Figure 1 A sequence and its *run decomposition* computed by a greedy algorithm: for each run, the first two elements determine if the run is non-decreasing or decreasing, then the run continues with the maximum number of consecutive elements that preserve its monotonicity.

Among the best-identified reasons behind the success of TimSort are the fact that this algorithm is well adapted to the architecture of computers (e.g., for dealing with cache issues) and to realistic distributions of data. In particular, the very conception of TimSort makes it particularly well-suited to sorting data whose *run decompositions* [3, 9] (see Figure 1) are simple. Such decompositions were already used in 1973 by Knuth's NaturalMergeSort [14, Section 5.2.4], which adapted the traditional MergeSort algorithm as follows: NaturalMergeSort is based on splitting arrays into monotonic subsequences, also called *runs*, and on merging these runs together. Thus, all algorithms sharing this feature of NaturalMergeSort are also called *natural* merge sorts.

In addition to being a natural merge sort, TimSort includes many optimisations, which were carefully engineered, through extensive testing, to offer the best complexity performances. As a result, the general structure of TimSort can be split into three main components: (i) a variant of an insertion sort, which is used to deal with *small* runs, e.g., runs of length less than 32, (ii) a simple policy for choosing which *large* runs to merge, (iii) a sub-routine for merging these runs, based on a so-called *galloping* strategy. The second component has been subject to an intense scrutiny these last few years, thereby giving birth to a great variety of TimSort-like algorithms, such as α-StackSort [2], α-MergeSort [7], ShiversSort [22] (which *predated* TimSort), adaptive ShiversSort [13], PeekSort and PowerSort [19]. On the contrary, the first and third components, which seem more complicated and whose effect might be harder to quantify, have often been used as black boxes when studying TimSort or designing variants thereof.

In what follows, we focus on the third component and prove that it is very efficient: whereas TimSort requires $\mathcal{O}(n + n\log(\rho))$ comparisons to sort arrays of length n that can be decomposed as a concatenation of ρ non-decreasing arrays, this component makes TimSort require only $\mathcal{O}(n + n\log(\sigma))$ comparisons to sort arrays of length n with σ distinct values.

Context and related work

The success of TimSort has nurtured the interest in the quest for sorting algorithms that would be both excellent all-around and adapted to arrays with few runs. However, its *ad hoc* conception made its complexity analysis harder than what one might have hoped, and it is only in 2015, a decade after TimSort had been largely deployed, that Auger et al. [2] proved that TimSort required $\mathcal{O}(n\log(n))$ comparisons for sorting arrays of length n.

This is optimal in the model of sorting by comparisons, if the input array can be an arbitrary array of length n. However, taking into account the run decompositions of the input array allows using finer-grained complexity classes, as follows. First, one may consider only arrays whose run decomposition consists of ρ monotonic runs. On such arrays, the best worst-case time complexity one may hope for is $\mathcal{O}(n + n\log(\rho))$ [16]. Second, we may consider even more restricted classes of input arrays, and focus only on those arrays that consist of ρ runs of lengths $r_1, \ldots, r_\rho$. In that case, every comparison-based sorting algorithm requires at least $n\mathcal{H} + \mathcal{O}(n)$ element comparisons on average, where $\mathcal{H}$ is defined as $\mathcal{H} = H(r_1/n, \ldots, r_\rho/n)$ and $H(x_1, \ldots, x_\rho) = -\sum_{i=1}^{\rho} x_i \log_2(x_i)$ is the general entropy function [3, 13, 17]. The number $\mathcal{H}$ is called the *run-length entropy* of the array.

Since the early 2000s, several natural merge sorts were proposed, all of which were meant to offer easy-to-prove complexity guarantees: ShiversSort, which runs in time $\mathcal{O}(n\log(n))$; α-StackSort, which, like NaturalMergeSort, runs in time $\mathcal{O}(n + n\log(\rho))$; α-MergeSort, which, like TimSort, runs in time $\mathcal{O}(n + n\mathcal{H})$; adaptive ShiversSort, PeekSort and PowerSort, which run in time $n\mathcal{H} + \mathcal{O}(n)$.

Except TimSort, these algorithms are, in fact, described only as policies for deciding which runs to merge, the actual sub-routine used for merging runs being left implicit: since choosing a naive merging sub-routine does not harm the worst-case time complexities considered above, all authors identified the cost of merging two runs of lengths m and n with the sum $m + n$, and the complexity of the algorithm with the sum of the costs of the merges performed.

One notable exception is that of Munro and Wild [19]. They compared the running times of TimSort and of TimSort's variant obtained by using a naive merging routine instead of TimSort's galloping sub-routine. However, and although they mentioned the challenge of finding distributions on arrays that might benefit from galloping, they did not address this challenge, and focused only on arrays with a low entropy $\mathcal{H}$. As a result, they unsurprisingly observed that the galloping sub-routine looked *slower* than the naive one.

Galloping turns out to be very efficient when sorting arrays with few distinct values, a class of arrays that had also been intensively studied. As soon as 1976, Munro and Spira [18] proposed a complexity measure $\mathcal{H}^*$ related to the run-length entropy, with the property that $\mathcal{H}^* \leqslant \log_2(\sigma)$ for arrays with σ values. They also proposed an algorithm for sorting arrays of length n with σ values by using $\mathcal{O}(n + n\mathcal{H}^*)$ comparisons. McIlroy [17] then extended their work to arrays representing a permutation π, identifying $\mathcal{H}^*$ with the run-length entropy of π^{-1} and proposing a variant of Munro and Spira's algorithm that would use $\mathcal{O}(n + n\mathcal{H}^*)$ comparisons in this generalised setting. Similarly, Barbay et al. [4] invented the algorithm QuickSynergySort, which aimed at minimising the number of comparisons, achieving a $\mathcal{O}(n + n\mathcal{H}^*)$ upper bound and further refining the parameters it used, by taking into account the interleaving between runs and dual runs. Yet, all of these algorithms require $\omega(n + n\mathcal{H})$ element moves in the worst case.

Furthermore, as a side effect of being rather complicated and lacking a proper analysis, except that of [19] that hinted at its inefficiency, the galloping sub-routine has been omitted in various mainstream implementations of natural merge sorts, in which it was replaced by its naive variant. This is the case, for instance, in library TimSort implementations of the programming languages Swift [8] and Rust [21]. On the contrary, TimSort's implementation in other languages, such as Java [6], Octave [24] or the V8 JavaScript engine [25], and PowerSort's implementation in Python [23] include the galloping sub-routine.

Contributions

We study the time complexity of various natural merge sort algorithms in a context where arrays are not just parametrised by their lengths. More precisely, we focus on a decomposition of input arrays that is dual to the decomposition of arrays into monotonic runs, and that was proposed by McIlroy [17].

Consider an array A that we want to sort in a *stable* manner, i.e., in which two elements can always considered to be distinct, if only because their positions in A are distinct. Without loss of generality, we identify the values $A[1], A[2], \ldots, A[n]$ with the integers from 1 to n, thereby making A a permutation of the set $\{1, 2, \ldots, n\}$. A common measure of presortedness consists in subdividing A into distinct monotonic *runs*, i.e., partitioning the set $\{1, 2, \ldots, n\}$ into intervals $R_1, R_2, \ldots, R_\rho$ on which the function $x \mapsto A[x]$ is monotonic.

Here, we adopt a dual approach, which consists in partitioning the set $\{1, 2, \ldots, n\}$ into the *increasing* runs $S_1, S_2, \ldots, S_\sigma$ of the inverse permutation A^{-1}. These intervals S_i are already known under the name of *shuffled up-sequences* [3, 15] or *riffle shuffles* [17]. In order to underline their connection with runs, we say that these intervals are the *dual runs* of A, and we denote their lengths by s_i. The process of transforming an array into a permutation and then extracting its dual runs is illustrated in Figure 2.

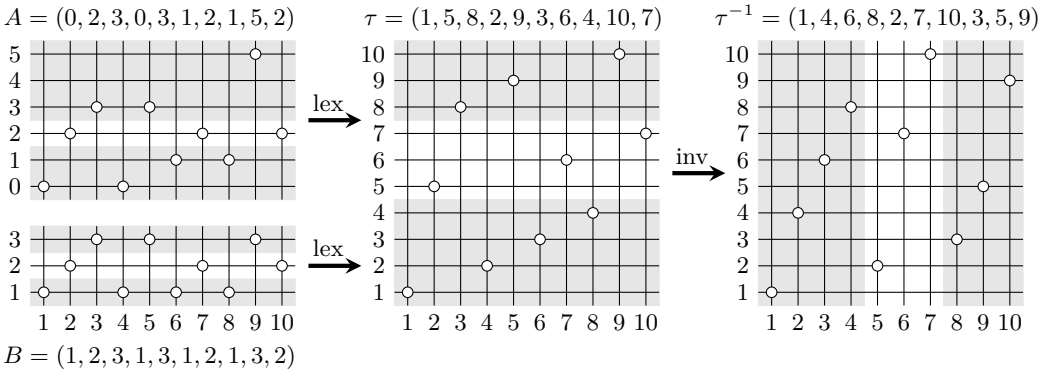


Figure 2 The arrays A and B are lexicographically equivalent to the permutation τ. Their dual runs, represented with gray and white horizontal stripes, have respective lengths 4, 3 and 3. The mappings $A \mapsto \tau$ and $B \mapsto \tau$ identify them with the dual runs of τ, i.e., with the runs of the permutation τ^{-1}. Note that A has only 3 dual runs, although it takes 5 distinct values.

When A is *not* a permutation of $\{1, 2, \ldots, n\}$, the dual runs of A are simply the maximal intervals S_i such that A is non-decreasing on the set of positions $\{j : A[j] \in S_i\} \subseteq \{1, 2, \ldots, n\}$. The length of a dual run is then defined as the cardinality of that set of positions. Thus, two lexicographically equivalent arrays have dual runs of the same lengths: this is the case, for instance, of the arrays A, B and τ in Figure 2.

In particular, we may see τ and B as canonical representatives of the array A: these are the unique permutation of $\{1, 2, \ldots, n\}$ and the unique array with values in $\{1, 2, \ldots, \sigma\}$ that are lexicographically equivalent with A. More generally, an array that contains σ distinct values cannot have more than σ dual runs.

Note that, in general, there is no non-trivial connection between the runs of a permutation and its dual runs. For instance, a permutation with a given number of runs may have arbitrarily many (or few) dual runs, and conversely.

In this article, we prove that, by using TimSort's galloping sub-routine, several natural merge sorts require $\mathcal{O}(n + n\mathcal{H}^*)$ comparisons, or even $n\mathcal{H}^* + \mathcal{O}(n)$ comparisons, where $\mathcal{H}^* = H(s_1/n, \ldots, s_\sigma/n) \leqslant \log_2(\sigma)$ is called the *dual run-length entropy* of the array, s_i is the length of the dual run S_i, and H is the general entropy function already mentioned above.

This legitimates using TimSort's arguably complicated galloping sub-routine rather than its naive alternative, in particular when sorting arrays that are constrained to have relatively few distinct values.

This also subsumes results that have been known since the 1970s. For instance, adapting the optimal constructions for alphabetic Huffman codes by Hu and Tucker [12] or Garsia and Wachs [10] to *merge trees* (described in Section 3) already provided sorting algorithms working in time $n\mathcal{H} + \mathcal{O}(n)$.

Our new results rely on notions that we call *fast-* and *middle-growth* properties, and which are found in natural merge sorts like α-MergeSort, α-StackSort, adaptive ShiversSort, ShiversSort, PeekSort, PowerSort or TimSort. More precisely, we prove that merge sorts require $\mathcal{O}(n + n\mathcal{H})$ comparisons *and* element moves when they possess the fast-growth property, thereby encompassing complexity results that were proved separately for each of these algorithms [1, 7, 13, 19], and $\mathcal{O}(n + n\mathcal{H}^*)$ comparisons when they possess the fast- or middle-growth property, which is a completely new result.

Finally, we prove finer complexity bounds on the number of comparisons used by adaptive ShiversSort, ShiversSort, NaturalMergeSort, PeekSort and PowerSort, which require only $n\mathcal{H}^* + \mathcal{O}(n + n\log(\mathcal{H}^* + 1))$ comparisons, nearly matching the $n\mathcal{H}+\mathcal{O}(n)$ (or $n\log_2(n)+\mathcal{O}(n)$ and $n\log_2(\rho) + \mathcal{O}(n)$, in the cases of ShiversSort and NaturalMergeSort) complexity upper bound they already enjoy in terms of comparisons and element moves. These results are summarised in Table 1.

Table 1 Element moves and comparisons needed by various algorithms using appropriate galloping sub-routines to sort arrays of length n with ρ runs, run-length entropy $\mathcal{H}$ and dual run-length entropy $\mathcal{H}^*$.

Algorithm	Element moves	Element comparisons
ShiversSort	$n\log_2(n) + \mathcal{O}(n)$	$n\mathcal{H}^* + \mathcal{O}(n + n\log(\mathcal{H}^* + 1))$
α-StackSort	$\mathcal{O}(n + n\log_2(\rho))$	$\mathcal{O}(n + n\min\{\log_2(\rho), \mathcal{H}^*\})$
NaturalMergeSort	$n\log_2(\rho) + \mathcal{O}(n)$	$n\min\{\log_2(\rho), \mathcal{H}^*\} + \mathcal{O}(n + n\log(\mathcal{H}^* + 1))$
α-MergeSort	$\mathcal{O}(n + n\mathcal{H})$	$\mathcal{O}(n + n\min\{\mathcal{H}, \mathcal{H}^*\})$
TimSort	$3n\mathcal{H}/2 + \mathcal{O}(n)$	$\mathcal{O}(n + n\min\{\mathcal{H}, \mathcal{H}^*\})$
adaptive ShiversSort	$n\mathcal{H} + \mathcal{O}(n)$	$n\min\{\mathcal{H}, \mathcal{H}^*\} + \mathcal{O}(n + n\log(\mathcal{H}^* + 1))$
PeekSort	$n\mathcal{H} + \mathcal{O}(n)$	$n\min\{\mathcal{H}, \mathcal{H}^*\} + \mathcal{O}(n + n\log(\mathcal{H}^* + 1))$
PowerSort	$n\mathcal{H} + \mathcal{O}(n)$	$n\min\{\mathcal{H}, \mathcal{H}^*\} + \mathcal{O}(n + n\log(\mathcal{H}^* + 1))$

2 The galloping sub-routine for merging runs

Here, we describe the galloping sub-routine that the algorithm TimSort uses to merge adjacent non-decreasing runs. This sub-routine is a blend between a naive merging algorithm, which requires $a + b - 1$ comparisons to merge runs A and B of lengths a and b, and a dichotomy-based algorithm, which requires $\mathcal{O}(\log(a + b))$ comparisons in the best case, and $\mathcal{O}(a + b)$ comparisons in the worst case. It depends on a parameter $\mathbf{t}$, and works as follows.

When merging runs A and B into one large run C, we first need to find the least integers k and ℓ such that $B[0] < A[k] \leqslant B[\ell]$: the $k + \ell$ first elements of C are $A[0], A[1], \ldots, A[k-1]$, $B[0], B[1], \ldots, B[\ell - 1]$, and the remaining elements of C are obtained by merging the sub-array of A that spans positions k to a and the sub-array of B that spans positions ℓ to b. Computing k and ℓ efficiently is therefore a crucial step towards reducing the number of comparisons required by the merging sub-routine (and, thus, by the sorting algorithm).

This computation is a special case of the following problem: if one wishes to find a secret integer $m \geqslant 1$ by choosing integers $x \geqslant 1$ and testing whether $x \geqslant m$, what is, as a function of m, the least number of tests that one must perform? Bentley and Yao [5] answer this question by providing simple strategies, which they number $\mathsf{B}_0, \mathsf{B}_1, \ldots$:

B_0: choose $x = 1$, then $x = 2$, and so on, until one chooses $x = m$, thereby finding m in m queries;

B_1: first use B_0 to find $\lceil\log_2(m)\rceil + 1$ in $\lceil\log_2(m)\rceil + 1$ queries, i.e., choose $x = 2^k$ until $x \geqslant m$, then compute the bits of m (from the most significant bit of m to the least significant one) in $\lceil\log_2(m)\rceil - 1$ additional queries; Bentley and Yao call this strategy a *galloping* (or *exponential search*) technique;

B_{k+1}: like B_1, except that one finds $\lceil\log_2(m)\rceil + 1$ by using B_k instead of B_0.

Strategy B_0 uses m queries, B_1 uses $2\lceil\log_2(m)\rceil$ queries (except for $m = 1$, where it uses one query), and each strategy B_k with $k \geqslant 2$ uses $\log_2(m) + o(\log(m))$ queries. Thus, if m is known to be arbitrarily large, one should favour some strategy B_k (with $k \geqslant 1$) over the

naive strategy B_0. However, when merging runs taken from a permutation chosen uniformly at random over the $n!$ permutations of $\{1, 2, \ldots, n\}$, the integer m is frequently small, which makes B_0 suddenly more attractive. In particular, the overhead of using B_1 instead of B_0 is a prohibitive $+20\%$ or $+33\%$ when $m = 5$ or $m = 3$, as illustrated in the black cells of Table 2.

Table 2 Comparison requests needed by strategies B_0 and B_1 to find a secret integer $m \geqslant 1$.

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
B_0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
B_1	1	2	**4**	4	**6**	6	6	6	8	8	8	8	8	8	8	8	10

McIlroy [17] addresses this issue by choosing a parameter $\mathbf{t}$ and using a blend between the strategies B_0 and B_1, which consists in two successive steps C_1 and C_2:

C_1: one first follows B_0 for up to $\mathbf{t}$ steps, thereby choosing $x = 1$, $x = 2$, …, $x = \mathbf{t}$ (if $m \leqslant \mathbf{t} - 1$, one stops after choosing $x = m$);

C_2: if $m \geqslant \mathbf{t} + 1$, one switches to B_1 (or, more precisely, to a version of B_1 translated by $\mathbf{t}$, since the precondition $m \geqslant 1$ is now $m \geqslant \mathbf{t} + 1$).

Once such a parameter $\mathbf{t}$ is fixed, McIlroy's mixed strategy allows retrieving m in $\mathsf{cost}_{\mathbf{t}}(m)$ queries, where $\mathsf{cost}_{\mathbf{t}}(m) = m$ if $m \leqslant \mathbf{t} + 2$, and $\mathsf{cost}_{\mathbf{t}}(m) = \mathbf{t} + 2\lceil \log_2(m - \mathbf{t}) \rceil$ if $m \geqslant \mathbf{t} + 3$. In practice, however, we will replace this cost function by the following simpler upper bound.

▶ **Lemma 1.** *For all $\mathbf{t} \geqslant 0$ and $m \geqslant 1$, we have $\mathsf{cost}_{\mathbf{t}}(m) \leqslant \mathsf{cost}^*_{\mathbf{t}}(m)$, where*

$$\mathsf{cost}^*_{\mathbf{t}}(m) = \min\{(1 + 1/(\mathbf{t} + 3))m, \mathbf{t} + 2 + 2\log_2(m + 1)\}.$$

Proof. Since the desired inequality is immediate when $m \leqslant \mathbf{t} + 2$, we assume that $m \geqslant \mathbf{t} + 3$. In that case, we already have $\mathsf{cost}_{\mathbf{t}}(m) \leqslant \mathbf{t} + 2(\log_2(m - \mathbf{t}) + 1) \leqslant \mathbf{t} + 2 + 2\log_2(m + 1)$, and we prove now that $\mathsf{cost}_{\mathbf{t}}(m) \leqslant m + 1$. Indeed, let $u = m - \mathbf{t}$ and let $f \colon x \mapsto x - 1 - 2\log_2(x)$. The function f is positive and increasing on the interval $[7, +\infty)$. Thus, it suffices to check by hand that $(m+1) - \mathsf{cost}_{\mathbf{t}}(m) = 0, 1, 0, 1$ when $u = 3, 4, 5, 6$, and that $(m+1) - \mathsf{cost}_{\mathbf{t}}(m) \geqslant f(u) > 0$ when $u \geqslant 7$. It follows, as expected, that $\mathsf{cost}_{\mathbf{t}}(m) \leqslant m + 1 \leqslant (1 + 1/(\mathbf{t} + 3))m$. ◀

The above discussion immediately provides us with a cost model for the number of comparisons performed when merging two runs.

▶ **Proposition 2.** *Let A and B be two non-decreasing runs of lengths a and b, with values in $\{1, 2, \ldots, \sigma\}$. For each integer $i \leqslant \sigma$, let $a_{\to i}$ (respectively, $b_{\to i}$) be the number of elements in A (respectively, in B) with value i. Using a merging sub-routine based on McIlroy's mixed strategy for a fixed parameter $\mathbf{t}$, we need at most*

$$1 + \sum_{i=1}^{\sigma} \mathsf{cost}^*_{\mathbf{t}}(a_{\to i}) + \mathsf{cost}^*_{\mathbf{t}}(b_{\to i})$$

element comparisons to merge the runs A and B.

Proof. First, assume that $a_{\to i} = 0$ for some $i \geqslant 2$. Replacing every value $j \geqslant i + 1$ with the value $j - 1$ in both arrays A and B does not change the behaviour of the sub-routine and decreases the value of σ. Moreover, the function $\mathsf{cost}^*_{\mathbf{t}}$ is sub-additive, i.e., we have $\mathsf{cost}^*_{\mathbf{t}}(m) + \mathsf{cost}^*_{\mathbf{t}}(m') \geqslant \mathsf{cost}^*_{\mathbf{t}}(m + m')$ for all $m \geqslant 0$ and $m' \geqslant 0$. Hence, without loss of generality, we assume that $a_{\to i} \geqslant 1$ for all $i \geqslant 2$. Similarly, we assume without loss of generality that $b_{\to i} \geqslant 1$ for all $i \leqslant \sigma - 1$.

Under these assumptions, the array C that results from merging A and B consists of $a_{\to 1}$ elements from A, then $b_{\to 1}$ elements from B, $a_{\to 2}$ elements from A, $b_{\to 2}$ elements from B, $\dots$, $a_{\to \sigma}$ elements from A and $b_{\to \sigma}$ elements from B. Thus, the galloping sub-routine consists in discovering successively the integers $a_{\to 1}, b_{\to 1}, a_{\to 2}, b_{\to 2}, \dots, a_{\to \sigma}$, each time using McIlroy's strategy based on the two steps C_1 and C_2; checking whether $a_{\to 1} = 0$ requires one more comparison than prescribed by McIlroy's strategy, and the integer $b_{\to \sigma}$ does not need to be discovered once the entire run A has been scanned be the merging sub-routine. ◀

We simply call $\mathbf{t}$-*galloping sub-routine* the merging sub-routine based on McIlroy's mixed strategy for a fixed parameter $\mathbf{t}$; when the value of $\mathbf{t}$ is irrelevant, we simply omit mentioning it. Then, the quantity

$$1 + \sum_{i=1}^{\sigma} \mathsf{cost}^*_{\mathbf{t}}(a_i) + \mathsf{cost}^*_{\mathbf{t}}(b_i)$$

is called the ($\mathbf{t}$-)*galloping cost* of merging A and B. By construction, this cost never exceeds $1 + 1/(\mathbf{t}+3)$ times the *naive cost* of merging A and B, which is simply defined as $a + b$. Below, we study the impact of using the galloping sub-routine instead of the naive one, which amounts to replacing naive merge costs by their galloping variants.

Note that using this new galloping cost measure is relevant only if the cost of element comparisons is significantly larger than the cost of element (or pointer) moves. For example, even if we were lucky enough to observe that each element in B is smaller than each element in A, we would perform only $\mathcal{O}(\log(a+b))$ element comparisons, but as many as $\Theta(a+b)$ element moves.

Updating the parameter $\mathbf{t}$

We assumed above that the parameter $\mathbf{t}$ did not vary while the runs A and B were being merged with each other. This is not how $\mathbf{t}$ behaves in TimSort's implementation of the galloping sub-routine. Instead, the parameter $\mathbf{t}$ is initially set to a constant ($\mathbf{t} = 7$ in Java), and may change during the algorithm as follows. In step C_2, after using the strategy B_1, and depending on the value of m that we found, one may realise that using B_0 might have been less expensive than using B_1. In that case, the value of $\mathbf{t}$ increases by 1, and otherwise (i.e., if using B_1 was indeed a smart move), it decreases by 1 (with a minimum of 0).

When sorting a random permutation, changing the value of $\mathbf{t}$ in that way decreases the average overhead of sometimes using B_1 instead of B_0 to a constant. More generally, even in the worst case, this overhead is linear in n.

▶ **Proposition 3.** *Let $\mathcal{A}$ be a stable natural merge sort algorithm, and let A be an array of length n. Let c_1 be the number of comparisons that $\mathcal{A}$ requires to sort A when it uses the naive sub-routine, and let c_2 be the number of comparisons that $\mathcal{A}$ requires to sort A when it uses the galloping sub-routine with TimSort's update policy for the parameter $\mathbf{t}$. We have $\mathsf{c}_2 \leqslant \mathsf{c}_1 + \mathcal{O}(n)$.*

Proof. Below, we group the comparisons that $\mathcal{A}$ performs while sorting A into *steps*, which we will consider as individual units. Steps are formed as follows. Let R and R' be consecutive runs that $\mathcal{A}$ is about to merge, and let us subdivide their concatenation $R \cdot R'$ into σ dual runs $S_1, S_2, \dots, S_\sigma$ (note that these are the dual runs of $R \cdot R'$ and not the dual runs of A, i.e., some elements of R may belong to a given dual run S_i of $R \cdot R'$ while belonging to distinct dual runs or A).

Each step consists of those comparisons used to discover the elements of R (resp., R') that belong to a given dual run S_i. Thus, the comparisons used to merge R and R' are partitioned into $2\sigma - 1$ steps: in the first step, we discover those elements of R that belong to S_1; in the second step, those elements of R' that belong to S_1; then, those elements of R that belong to S_2 ; ...; and, finally, those elements of R that belong to S_σ.

Let $s_1, s_2, \ldots, s_\ell$ be the steps into which the comparisons performed by $\mathcal{A}$ are grouped. By construction, each step s_i consists in finding an integer $m_i \geqslant 1$ (or, possibly, $m_i = 0$ if s_i is the first step of a merge between two consecutive runs). If $\mathcal{A}$ uses TimSort's update policy, the step s_i consists in using McIlroy's strategy for a given parameter $\mathbf{t}(s_i)$ that depends on s_i. We also denote by $\mathbf{t}(s_{\ell+1})$ the parameter value obtained after $\mathcal{A}$ has finished sorting the array A.

Strategy B_0 requires m_i comparisons to find that integer m_i, and McIlroy's strategy requires m_i comparisons if $m_i \leqslant \mathbf{t}(s_i)$, up to $m_i + 1$ comparisons if $\mathbf{t}(s_i) + 1 \leqslant m_i \leqslant \mathbf{t}(s_i) + 6$, and up to $m_i - 1$ comparisons if $m_i \geqslant \mathbf{t}(s_i) + 7$. Since $\mathbf{t}(s_{i+1}) = \mathbf{t}(s_i)$ in the first case, $\mathbf{t}(s_{i+1}) = \mathbf{t}(s_i) + 1$ in the second case and $\mathbf{t}(s_{i+1}) = \max\{0, \mathbf{t}(s_i) - 1\}$ in the third case, McIlroy's strategy never uses more than $m_i + \mathbf{t}(s_{i+1}) - \mathbf{t}(s_i)$ comparisons. Consequently, the overhead of using TimSort's update policy instead of a naive merging sub-routine is at most $\mathbf{t}(s_{\ell+1}) - \mathbf{t}(s_1)$.

Moreover, let $\mu_i = m_1 + m_2 + \ldots + m_i$ for all $i \leqslant \ell$. We show by induction on τ that, whenever $\mathbf{t}(s_i) = \mathbf{t}(s_1) + \tau$, we have $2\mu_i \geqslant \tau^2$. Indeed, if $\tau \geqslant 1$ and if s_i is the first step for which $\mathbf{t}(s_i) = \mathbf{t}(s_1) + \tau$, we have $\mathbf{t}(s_{i-1}) = \mathbf{t}(s_1) + \tau - 1$. It follows that $2\mu_{i-1} \geqslant (\tau - 1)^2$ and $m_i \geqslant \mathbf{t}(s_1) + \tau \geqslant \tau$, which proves that $2\mu_i \geqslant (\tau - 1)^2 + 2\tau = \tau^2 + 1$. Thus, we conclude that $\mathbf{t}(s_\ell + 1) - \mathbf{t}(s_1) \leqslant \sqrt{2\mu_\ell}$.

Finally, μ_ℓ is equal to the number of element comparisons that $\mathcal{A}$ would perform if it used the naive merging strategy, i.e., $\mu_\ell = \mathsf{c}_1$. No merge sort requires more than $\mathcal{O}(n^2)$ comparisons, and therefore $\mu_\ell = \mathcal{O}(n^2)$, which is why the overhead of using TimSort's update policy is at most $\mathbf{t}(s_{\ell+1}) - \mathbf{t}(s_1) \leqslant \sqrt{2\mu_\ell} = \mathcal{O}(n)$. ◀

Deciding whether our results remain valid when $\mathbf{t}$ is updated like in TimSort remains an open question. However, in Section 5.3, we propose and study the following alternative update policy: when merging runs of lengths a and b, we set $\mathbf{t} = \lceil \log_2(a + b) \rceil$.

3 A fast-growth property and its consequences

In this section, we focus on two novel properties of stable natural merge sorts, which we call *fast-growth* and *middle-growth*. These properties capture all TimSort-like natural merge sorts invented in the last decade, and explain why these sorting algorithms require only $\mathcal{O}(n + n\mathcal{H})$ element moves and $\mathcal{O}(n + n \min\{\mathcal{H}, \mathcal{H}^*\})$ element comparisons. We will prove in subsequent sections that many algorithms have these properties.

When applying a stable natural merge sort on an array A, the elements of A are clustered into monotonic sub-arrays called *runs*, and the algorithm consists in repeatedly merging consecutive runs into one larger run until the array itself contains only one run. Consequently, each element may undergo several successive merge operations. *Merge trees* [3, 13, 19] are a convenient way to represent the succession of runs that ever occur while A is being sorted.

▶ **Definition 4.** *The* merge tree *induced by a stable natural merge sort algorithm on an array A is the binary rooted tree $\mathcal{T}$ defined as follows. The nodes of $\mathcal{T}$ are all the runs that were present in the initial array A or that resulted from merging two runs. The runs of the initial array are the leaves of $\mathcal{T}$, and when two consecutive runs R_1 and R_2 are merged with each other into a new run $\overline{R}$, the run R_1 spanning positions immediately to the left of those of R_2, they form the left and the right children of the node $\overline{R}$, respectively.*

Such trees ease the task of referring to several runs that might not have occurred simultaneously. In particular, we will often refer to the i^{th} *ancestor* or a run R, which is just R itself if $i = 0$, or the parent, in the tree $\mathcal{T}$, of the $(i-1)^{\text{th}}$ ancestor of R if $i \geqslant 1$. That ancestor will be denoted by $R^{(i)}$.

Before further manipulating these runs, let us first present some notation about runs and their lengths, which we will frequently use. We will commonly denote runs with capital letters, possibly with some index or adornment, and we will then denote the length of such a run with the same small-case letter and the same index or adornment. For instance, runs named R, R_i, Q' and $\overline{S}$ will have respective lengths r, r_i, q' and $\overline{s}$.

▶ **Definition 5.** *We say that a stable natural merge sort algorithm $\mathcal{A}$ has the* fast-growth property *if it satisfies the following statement:*

> *There exist an integer $\ell \geqslant 1$ and a real number $\theta > 1$ such that, for every merge tree $\mathcal{T}$ induced by $\mathcal{A}$ and every run at depth ℓ or more in $\mathcal{T}$, we have $r^{(\ell)} \geqslant \theta r$.*

We also say that $\mathcal{A}$ has the middle-growth property *if it satisfies the following statement:*

> *There exists a real number $\kappa > 1$ such that, for every merge tree $\mathcal{T}$ induced by $\mathcal{A}$, every integer $h \geqslant 0$ and every run R of height h in $\mathcal{T}$, we have $r \geqslant \kappa^h$.*

Since every node of height $h \geqslant 1$ in a merge tree is a run of length at least 2, each algorithm with the fast-growth property also has the middle-growth property: indeed, it suffices to choose $\kappa = \min\{2, \theta\}^{1/\ell}$. As a result, the former property is stronger than the latter one, and indeed it has stronger consequences.

▶ **Theorem 6.** *Let $\mathcal{A}$ be a stable natural merge sort algorithm with the fast-growth property. If $\mathcal{A}$ uses either the galloping or the naive sub-routine for merging runs, it requires $\mathcal{O}(n + n\mathcal{H})$ element comparisons and moves to sort arrays of length n and run-length entropy $\mathcal{H}$.*

Proof. Let $\ell \geqslant 1$ and $\theta > 1$ be the integer and the real number mentioned in the definition of the statement "$\mathcal{A}$ has the fast-growth property". Let A be an array of length n with ρ runs of lengths $r_1, r_2, \ldots, r_\rho$, let $\mathcal{T}$ be the merge tree induced by $\mathcal{A}$ on A, and let d_i be the depth of the run R_i in the tree $\mathcal{T}$.

The algorithm $\mathcal{A}$ uses $\mathcal{O}(n)$ element comparisons and element moves to delimit the runs it will then merge and to make them non-decreasing. Then, both the galloping and the naive merging sub-routine require $\mathcal{O}(a + b)$ element comparisons and moves to merge two runs A and B of lengths a and b. Therefore, it suffices to prove that $\sum_{R \in \mathcal{T}} r = \mathcal{O}(n + n\mathcal{H})$.

Consider some leaf R_i of the tree $\mathcal{T}$, and let $k = \lfloor d_i/\ell \rfloor$. The $(k\ell)^{\text{th}}$ ancestor of R_i is a node R of size $r \geqslant \theta^k r_i$, and thus $n \geqslant r \geqslant \theta^k r_i$. Hence, $d_i + 1 \leqslant \ell(k+1) \leqslant \ell\,(\log_\theta(n/r_i) + 1)$, and we conclude that

$$\sum_{R \in \mathcal{T}} r = \sum_{i=1}^{\rho} (d_i + 1) r_i \leqslant \ell \sum_{i=1}^{\rho} (r_i \log_\theta(n/r_i) + r_i) = \ell(n\mathcal{H}/\log_2(\theta) + n) = \mathcal{O}(n + n\mathcal{H}). \quad \blacktriangleleft$$

A similar, weaker result also holds for algorithms with the middle-growth property.

▶ **Theorem 7.** *Let $\mathcal{A}$ be a stable natural merge sort algorithm with the middle-growth property. If $\mathcal{A}$ uses either the galloping or the naive sub-routine for merging runs, it requires $\mathcal{O}(n \log(n))$ element comparisons and moves to sort arrays of length n.*

Proof. Let us borrow the notations used when proving Theorem 6, and let $\kappa > 1$ be the real number mentioned in the definition of the statement "$\mathcal{A}$ has the middle-growth property". Like in the proof of Theorem 6, it suffices to show that $\sum_{R \in \mathcal{T}} r = \mathcal{O}(n \log(n))$.

The $d_i{}^{\text{th}}$ ancestor of a run R_i is the root of $\mathcal{T}$, and thus $n \geqslant \kappa^{d_i}$. Hence, $d_i \leqslant \log_\kappa(n)$, and we conclude that

$$\sum_{R\in\mathcal{T}} r = \sum_{i=1}^{\rho}(d_i+1)r_i \leqslant \sum_{i=1}^{\rho}(\log_\kappa(n)+1)r_i = (\log_\kappa(n)+1)n = \mathcal{O}(n\log(n)). \qquad \blacktriangleleft$$

Theorems 6 and 7 provide us with a simple framework for recovering well-known results on the complexity of many algorithms. By contrast, Theorem 8 consists in new complexity guarantees on the number of element comparisons performed by algorithms with the middle-growth property, provided that they use the galloping sub-routine.

▶ **Theorem 8.** *Let $\mathcal{A}$ be a stable natural merge sort algorithm with the middle-growth property. If $\mathcal{A}$ uses the galloping sub-routine for merging runs, it requires $\mathcal{O}(n + n\mathcal{H}^*)$ element comparisons to sort arrays of length n and dual run-length entropy $\mathcal{H}^*$.*

Proof. All comparisons performed by the galloping sub-routine are of the form $A[i] \leqslant^? A[j]$, where i and j are positions such that $i < j$. Thus, the behaviour of $\mathcal{A}$, i.e., the element comparisons and element moves it performs, is invariant under lexicographic equivalence, as illustrated in Figure 2. Consequently, starting from an array A of length n with σ dual runs $S_1, S_2, \ldots, S_\sigma$, we create a new array B of length n with σ distinct values, setting $B[j] \stackrel{\text{def}}{=\!=} i$ whenever $A[j]$ belongs to the dual run S_i, and we may now assume that A coincides with B. This assumption allows us to directly use Proposition 2, whose presentation would have been more complicated if we had referred to dual runs of an underlying array instead of referring directly to distinct values.

Now, let $\kappa > 1$ be the real number mentioned in the definition of the statement "$\mathcal{A}$ has the middle-growth property". Let A be an array of length n and whose values are integers from 1 to σ, let $s_1, s_2, \ldots, s_\sigma$ be the lengths of its dual runs, and let $\mathcal{T}$ be the merge tree induced by $\mathcal{A}$ on A.

The algorithm $\mathcal{A}$ uses $\mathcal{O}(n)$ element comparisons to delimit the runs it will then merge and to make them non-decreasing. We prove now that merging these runs requires only $\mathcal{O}(n + n\mathcal{H}^*)$ comparisons. For every run R in $\mathcal{T}$ and every integer $i \leqslant \sigma$, let $r_{\to i}$ be the number of elements of R with value i. In the galloping cost model, merging two runs R and R' requires at most

$$1 + \sum_{i=1}^{\sigma} \mathsf{cost}^*_{\mathbf{t}}(r_{\to i}) + \mathsf{cost}^*_{\mathbf{t}}(r'_{\to i})$$

element comparisons. Since less than n such merge operations are performed, and since $n = \sum_{i=1}^{\sigma} s_i$ and $n\mathcal{H}^* = \sum_{i=1}^{\sigma} s_i \log(n/s_i)$, it remains to show that

$$\sum_{R\in\mathcal{T}} \mathsf{cost}^*_{\mathbf{t}}(r_{\to i}) = \mathcal{O}(s_i + s_i\log(n/s_i))$$

for all $i \leqslant \sigma$. Then, since $\mathsf{cost}^*_{\mathbf{t}}(m) \leqslant (\mathbf{t}+1)\mathsf{cost}^*_0(m)$ for all parameter values $\mathbf{t} \geqslant 0$ and all $m \geqslant 0$, we assume without loss of generality that $\mathbf{t} = 0$.

Now, consider some integer $h \geqslant 0$, let $\mathcal{R}_h$ be the set of runs at height h in $\mathcal{T}$, and let $\mathsf{C}_0(h) = \sum_{R\in\mathcal{R}_h} \mathsf{cost}^*_0(r_{\to i})$. Since no run in $\mathcal{R}_h$ descends from another one, we already have

$$\mathsf{C}_0(h) \leqslant 2\sum_{R\in\mathcal{R}_h} r_{\to i} \leqslant 2s_i \text{ and } \sum_{R\in\mathcal{R}_h} r \leqslant n.$$

Moreover, by definition of κ, each run $R \in \mathcal{R}_h$ is of length $r \geqslant \kappa^h$, and thus $|\mathcal{R}_h| \leqslant n/\kappa^h$.

Then, consider the constant $\lambda = \lceil \log_\kappa(n/s_i) \rceil$ and the functions $f\colon x \mapsto \mathbf{t}+2+2\log_2(x+1)$ and $g\colon x \mapsto x\, f(s_i/x)$. Both f and g are positive and concave on the interval $(0, +\infty)$, thereby also being increasing. It follows that, for all $h \geqslant 0$,

$$\begin{aligned}\mathsf{C}_0(\lambda+h) &\leqslant \sum_{R\in\mathcal{R}_{\lambda+h}} f(r_{\to i}) \leqslant |\mathcal{R}_{\lambda+h}|\, f\big(\textstyle\sum_{R\in\mathcal{R}_{\lambda+h}} r_{\to_i}/|\mathcal{R}_{\lambda+h}|\big) \leqslant g\big(|\mathcal{R}_{\lambda+h}|\big)\\ &\leqslant g\big(n/\kappa^{\lambda+h}\big) \leqslant g\big(s_i\kappa^{-h}\big) = \big(2+2\log_2(\kappa^h+1)\big)s_i\kappa^{-h}\\ &\leqslant \big(2+2\log_2(2\kappa^h)\big)s_i\kappa^{-h} = \big(4+2h\log_2(\kappa)\big)s_i\kappa^{-h}.\end{aligned}$$

The inequalities on the first line respectively hold by definition of cost_1^*, because f is concave, and because f is increasing and $\sum_{R\in\mathcal{R}_h} r_{\to_i} \leqslant s_i$; the inequalities on the second line hold because g is increasing and $|\mathcal{R}_h| \leqslant n/\kappa^h$.

We conclude that

$$\begin{aligned}\sum_{R\in\mathcal{T}} \mathsf{cost}_0^*(r_{\to i}) = \sum_{h\geqslant 0} \mathsf{C}_0(h) &= \sum_{h=0}^{\lambda-1} \mathsf{C}_0(h) + \sum_{h\geqslant 0} \mathsf{C}_0(\lambda+h)\\ &\leqslant 2\lambda s_i + 4s_i \sum_{h\geqslant 0} \kappa^{-h} + 2\log_2(\kappa) s_i \sum_{h\geqslant 0} h\kappa^{-h}\\ &\leqslant \mathcal{O}\big(s_i(1+\log(n/s_i))\big) + \mathcal{O}\big(s_i\big) + \mathcal{O}\big(s_i\big) = \mathcal{O}\big(s_i + s_i\log(n/s_i)\big).\end{aligned}$$

◀

4 PowerSort has the fast-growth property

In this section, we prove that PowerSort and many TimSort-like algorithms enjoy the fast- or middle-growth properties. To that aim, we first define the run merge policy of PowerSort, by introducing the notion of *power* of a run endpoint or of a run, and then characterising the merge trees that PowerSort induces.

▶ **Definition 9.** *Let A be an array of length n, whose run decomposition consists of runs $R_1, R_2, \ldots, R_\rho$, ordered from left to right. For all integers $i \leqslant \rho$, let $e_i = r_1 + \ldots + r_i$. We also abusively set $e_{-1} = -\infty$ and $e_{\rho+1} = n$.*

When $0 \leqslant i \leqslant \rho$, we denote by $\mathbf{I}(i)$ the half-open interval $(e_{i-1}+e_i, e_i+e_{i+1}]$. The power *of e_i, which we denote by p_i, is then defined as the least integer p such that $\mathbf{I}(i)$ contains an element of the set $\{kn/2^{p-1} : k \in \mathbb{Z}\}$. Thus, we (abusively) have $p_0 = -\infty$ and $p_\rho = 0$.*

Finally, let $R_{i\ldots j}$ be a run obtained by merging consecutive runs $R_i, R_{i+1}, \ldots, R_j$. The power *of the run R is defined as $\max\{p_{i-1}, p_j\}$.*

The notion of power quickly comes with nice properties, two of which we mention now.

▶ **Lemma 10.** *For each non-empty sub-interval I of the set $\{0, \ldots, \rho\}$, there exists a unique integer $i \in I$ such that $p_i \leqslant p_j$ for all $j \in I$.*

Proof. Assume that the integer i is not unique. Since e_0 is the only endpoint with power $-\infty$, we know that $0 \notin I$. Then, let a and b be elements of I such that $a < b$ and $p_a = p_b \leqslant p_j$ for all $j \in I$, and let $p = p_a = p_b$. By definition of p_a and p_b, there exist odd integers k and ℓ such that $kn/2^{p-1} \in \mathbf{I}(a)$ and $\ell n/2^{p-1} \in \mathbf{I}(b)$. Since $\ell \geqslant k+1$, the fraction $(k+1)n/2^{p-1}$ belongs to some interval $\mathbf{I}(j)$ such that $a \leqslant j \leqslant b$. But since $k+1$ is even, we know that $p_j < p$, which is absurd. Thus, our initial assumption is invalid, which completes the proof. ◀

▶ **Lemma 11.** *Let $R_1, \ldots, R_\rho$ be the run decomposition of an array A. There is exactly one tree $\mathcal{T}$ that is induced on A and in which every inner node has a smaller power than its children. Furthermore, for every run $R_{i\ldots j}$ in $\mathcal{T}$, we have $\max\{p_{i-1}, p_j\} < \min\{p_i, p_{i+1}, \ldots, p_{j-1}\}$.*

Proof. Given a merge tree $\mathcal{T}$, let us prove that the following statements are equivalent:
S_1: each inner node of $\mathcal{T}$ has a smaller power than its children;
S_2: each run $R_{i\dots j}$ that belongs to $\mathcal{T}$ has a power that is smaller than all of $p_i, \dots, p_{j-1}$;
S_3: if a run $R_{i\dots j}$ is an inner node of $\mathcal{T}$, its children are the two runs $R_{i\dots k}$ and $R_{k+1\dots j}$ such that $p_k = \min\{p_i, \dots, p_{j-1}\}$.

First, if S_1 holds, we prove S_3 by induction on the height h of the run $R_{i\dots j}$. Indeed, if the restriction of S_3 to runs of height less than h holds, let $R_{i\dots k}$ and $R_{k+1\dots j}$ be the children of a run $R_{i\dots j}$ of height h. If $i < k$, the run $R_{i\dots k}$ has two children $R_{i\dots \ell}$ and $R_{\ell+1\dots k}$ such that $p_\ell = \min\{p_i, \dots, p_{k-1}\}$, and the powers of these runs, i.e., $\max\{p_{i-1}, p_\ell\}$ and $\max\{p_\ell, p_k\}$, are greater than the power of $R_{i\dots k}$, i.e., $\max\{p_{i-1}, p_k\}$, which proves that $p_\ell > p_k$. It follows that $p_k = \min\{p_i, \dots, p_k\}$, and one proves similarly that $p_k = \min\{p_k, \dots, p_{j-1}\}$, thereby showing that S_3 also holds for runs of height h.

Then, if S_3 holds, we prove S_2 by induction on the depth d of the run $R_{i\dots j}$. Indeed, if the restriction of S_2 to runs of depth less than d holds, let $R_{i\dots k}$ and $R_{k+1\dots j}$ be the children of a run $R_{i\dots j}$ of depth d. Lemma 10 and S_3 prove that p_k is the unique smallest element of $\{p_i, \dots, p_{j-1}\}$, and the induction hypothesis proves that $\max\{p_{i-1}, p_j\} < p_k$. It follows that both powers $\max\{p_{i-1}, p_k\}$ and $\max\{p_k, p_j\}$ are smaller than all of $p_i, \dots, p_{k-1}, p_{k+1}, \dots, p_{j-1}$, thereby showing that S_2 also holds for runs of depth d.

Finally, if S_2 holds, let $R_{i\dots j}$ be an inner node of $\mathcal{T}$, with children $R_{i\dots k}$ and $R_{k+1\dots j}$. Property S_2 ensures that $\max\{p_{i-1}, p_j\} < p_k$, and thus that $\max\{p_{i-1}, p_j\}$ is smaller than both $\max\{p_{i-1}, p_k\}$ and $\max\{p_k, p_j\}$, i.e., that $R_{i\dots j}$ has a smaller power that its children, thereby proving S_1.

In particular, once the array A and its run decomposition $R_1, \dots, R_\rho$ are fixed, S_3 provides us with a deterministic top-down construction of the unique merge tree $\mathcal{T}$ induced on A and that satisfies S_1: the root of $\mathcal{T}$ must be the run $R_{1\dots\rho}$ and, provided that some run $R_{i\dots j}$ belongs to $\mathcal{T}$, where $i < j$, Lemma 10 proves that the integer k mentioned in S_3 is unique, which means that S_3 unambiguously describes the children of $R_{i\dots j}$ in the tree $\mathcal{T}$.

This proves the first claim of Lemma 11, and the second claim of Lemma 11 follows from the equivalence between the statements S_1 and S_2. ◀

This leads to the following characterisation of the algorithm PowerSort, which is proved in [19, Lemma 4] and which we consider as an alternative definition of PowerSort.

▶ **Definition 12.** *In every merge tree that PowerSort induces, inner nodes have a smaller power than their children.*

▶ **Lemma 13.** *Let $\mathcal{T}$ be a merge tree induced by PowerSort, let R be a run of $\mathcal{T}$ with power p, and let $R^{(2)}$ be its grandparent. We have $2^{p-2}r < n < 2^p r^{(2)}$.*

Proof. Let $R_{i\dots j}$ be the run R. Without loss of generality, we assume that $p = p_j$, the case $p = p_{i-1}$ being entirely symmetric. Lemma 11 states that all of $p_i, \dots, p_{j-1}$ are larger than p. Thus, the union of intervals $\mathbf{I}(i) \cup \dots \cup \mathbf{I}(j) = (e_{i-1} + e_i, e_j + e_{j+1}]$ does not contain any element of the set $\mathcal{S} = \{kn/2^{p-2} : k \in \mathbb{Z}\}$. The bounds $e_{i-1} + e_i$ and $e_j + e_{j+1}$ are therefore contained between two consecutive elements of $\mathcal{S}$, i.e., there exists an integer ℓ such that

$$\ell n/2^{p-2} \leqslant e_{i-1} + e_i \leqslant e_j + e_{j+1} < (\ell+1)n/2^{p-2},$$

and we conclude that

$$r = e_j - e_{i-1} \leqslant (e_j + e_{j+1}) - (e_{i-1} + e_i) < n/2^{p-2}.$$

We prove now that $n \leqslant 2^p r^{(2)}$. To that end, we assume that both R and its parent are left children, the other possible cases being symmetric. There exist integers u and v such that the parent of R is $R_{i\ldots u}$, and its grandparent is $R_{i\ldots v}$. Hence, $\max\{p_{i-1}, p_u\} < \max\{p_{i-1}, p_j\} = p$, which shows that $p_u < p_j = p$. Thus, both intervals $\mathbf{I}(j)$ and $\mathbf{I}(u)$, which are subintervals of $(2e_{i-1}, 2e_v]$, contain elements of the set $\mathcal{S}' = \{kn/2^{p-1} : k \in \mathbb{Z}\}$. This means that there exist two integers k and ℓ such that $2e_{i-1} < kn/2^{p-1} < \ell n/2^{p-1} \leqslant 2e_v$, from which we conclude that

$$r^{(2)} = e_v - e_{i-1} > (\ell - k)n/2^p \geqslant n/2^p. \qquad \blacktriangleleft$$

▶ **Theorem 14.** *The algorithm PowerSort has the fast-growth property.*

Proof. Let $\mathcal{T}$ be a merge tree induced by PowerSort. Then, let R be a run in $\mathcal{T}$, and let p and $p^{(3)}$ be the respective powers of the runs R and $R^{(3)}$. Definition 12 ensures that $p \geqslant p^{(3)} + 3$, and therefore Lemma 13 proves that

$$2^{p^{(3)}+1} r \leqslant 2^{p-2} r < n < 2^{p^{(3)}} r^{(5)}.$$

This means that $r^{(5)} \geqslant 2r$, and therefore that PowerSort has the fast-growth property. ◀

For the sake of conciseness, we only list which algorithms were found to have the fast- or middle-growth property. Proofs that they do can be found in the complete version of this article [11].

▶ **Theorem 15.** *The algorithms TimSort, α-MergeSort, PeekSort and adaptive ShiversSort have the fast-growth property.*

An immediate consequence of Theorems 6 and 8 is that these algorithms sort arrays of length n and run-length entropy $\mathcal{H}$ in time $\mathcal{O}(n + n\mathcal{H})$ – which was already well-known – and that, if used with the galloping merging sub-routine, they only need $\mathcal{O}(n + n\mathcal{H}^*)$ comparisons to sort arrays of length n and dual run-length entropy $\mathcal{H}^*$ – which is a new result.

▶ **Theorem 16.** *The algorithms NaturalMergeSort, ShiversSort and α-StackSort have the middle-growth property.*

Theorem 8 proves that, if these three algorithms are used with the galloping merging sub-routine, they only need $\mathcal{O}(n + n\mathcal{H}^*)$ comparisons to sort arrays of length n and dual run-length entropy $\mathcal{H}^*$. By contrast, observe that they can be implemented by using a stack, following TimSort's own implementation, but where only the two top runs of the stack could be merged. It is proved in [13] that such algorithms may require $\omega(n + n\mathcal{H})$ comparisons to sort arrays of length n and run-length entropy $\mathcal{H}$. Hence, Theorem 6 shows that these three algorithms do *not* have the fast-growth property.

5 Refined complexity bounds for PowerSort

One weakness of Theorem 8 is that it cannot help us to distinguish the complexity upper bounds of those algorithms that have the middle-growth property, although the constants hidden in the $\mathcal{O}$ symbol could be dramatically different. In this section, we study these constants, thereby focusing on upper bounds of the type $\mathsf{c}n\mathcal{H}^* + \mathcal{O}(n)$ or $\mathsf{c}n(1+o(1))\mathcal{H}^* + \mathcal{O}(n)$.

Since sorting arrays of length n, in general, requires at least $\log_2(n!) = n\log_2(n) + \mathcal{O}(n)$ comparisons, and since $\mathcal{H}^* \leqslant \log_2(n)$ for all arrays, we already know that $\mathsf{c} \geqslant 1$ for any such constant c. Below, we focus on finding matching upper bounds in two regimes: first using a fixed parameter $\mathbf{t}$, thereby obtaining a constant $\mathsf{c} > 1$, and then letting $\mathbf{t}$ depend on the lengths of those runs that are being merged, in which case we reach the constant $\mathsf{c} = 1$.

5.1 A tight middle-growth property

Below, we aim at computing the least constant that might lie hidden in the $\mathcal{O}(n + n\mathcal{H}^*)$ upper bound of Theorem 8. If we were to simply extract that constant from the proof we gave, this constant would depend directly on the real number κ mentioned in the definition of the statement "$\mathcal{A}$ has the middle-growth property". To that aim, we make that statement more precise.

▶ **Definition 17.** *We say that a stable natural merge sort algorithm $\mathcal{A}$ has the* tight *middle-growth property if it satisfies the following statement:*

> *There exists an integer $\theta \geqslant 0$ such that, for every merge tree $\mathcal{T}$ induced by $\mathcal{A}$, every integer $h \geqslant 0$ and every run R of height h in $\mathcal{T}$, we have $r \geqslant 2^{h-\theta}$.*

Since every node of height $h \geqslant 1$ in a merge tree is a run of length at least 2, each algorithm with the tight middle-growth property also has the middle-growth property: indeed, it suffices to choose $\kappa = 2^{1/(\theta+1)}$. The tight middle-growth property is incomparable with the fast-growth property since, for instance, adaptive ShiversSort and PeekSort fail to have the tight middle-growth property. In practice, we might also introduce a related notion of tight fast-growth property, which would be useful when evaluating the constant hidden in the $\mathcal{O}(n + n\mathcal{H})$ upper bound on the complexity of sorting algorithms.

▶ **Theorem 18.** *The algorithm PowerSort has the tight middle-growth property.*

Proof. Let $\mathcal{T}$ be a merge tree induced by PowerSort and let R be a run in $\mathcal{T}$ at depth at least h. We will prove that $r^{(h)} \geqslant 2^{h-4}$.

If $h \leqslant 4$, the desired inequality is immediate. Then, if $h \geqslant 5$, let n be the length of the array on which $\mathcal{T}$ is induced. Let also p and $p^{(h-2)}$ be the respective powers of the runs R and $R^{(h-2)}$. Definition 12 and Lemma 13 prove that $2^{p^{(h-2)}+h-4} \leqslant 2^{p-2} \leqslant 2^{p-2}r < n < 2^{p^{(h-2)}}r^{(h)}$. ◀

5.2 Using a fixed parameter $\mathbf{t}$

Following the structure of Section 3, we prove now that each algorithm with the tight middle-growth property, such as PowerSort, enjoys excellent upper bounds on the number of element comparisons it requires.

▶ **Theorem 19.** *Let $\mathcal{A}$ be a stable natural merge sort algorithm with the tight middle-growth property. For each parameter $\mathbf{t} \geqslant 0$, if $\mathcal{A}$ uses the $\mathbf{t}$-galloping sub-routine for merging runs, it requires at most $(1 + 1/(\mathbf{t}+3))n\mathcal{H}^* + \log_2(\mathbf{t}+1)n + \mathcal{O}(n)$ element comparisons to sort arrays of length n and dual run-length entropy $\mathcal{H}^*$.*

Proof. Let us follow a variant of the proof of Theorem 8. Let θ be the integer mentioned in the definition of the statement "$\mathcal{A}$ has the tight middle-growth property", let $\mathcal{T}$ be the merge tree induced by $\mathcal{A}$ on an array A of length n, and let $s_1, s_2, \ldots, s_\sigma$ be the lengths of the dual runs of A. Like in the proof of Theorem 8, we just need to prove that

$$\sum_{R \in \mathcal{T}} \mathsf{cost}^*_{\mathbf{t}}(r_{\to i}) \leqslant (1 + 1/(\mathbf{t}+3))s_i \log_2(n/s_i) + s_i \log_2(\mathbf{t}+1) + \mathcal{O}(s_i)$$

for all $i \leqslant \sigma$.

Then, let $\mathcal{R}_h$ be the set of runs at height h in $\mathcal{T}$. By construction, no run in $\mathcal{R}_h$ descends from another one, which proves that

$$\sum_{R \in \mathcal{R}_h} r_{\to i} \leqslant s_i \text{ and that } \sum_{R \in \mathcal{R}_h} r \leqslant n.$$

Since each run $R \in \mathcal{R}_h$ is of length $r \geqslant 2^{h-\theta}$, it follows that $|\mathcal{R}_h| \leqslant n/2^{h-\theta}$.

Then, consider the function

$$\mathsf{C}_{\mathbf{t}}(h) = \sum_{R \in \mathcal{R}_h} \mathsf{cost}^*_{\mathbf{t}}(r_{\to i}).$$

We noted above that

$$\mathsf{C}_{\mathbf{t}}(h) \leqslant (1 + 1/(\mathbf{t}+3)) \sum_{R \in \mathcal{R}_h} r_{\to i} \leqslant (1 + 1/(\mathbf{t}+3))s_i$$

for all $h \geqslant 0$.

Let also $f\colon x \mapsto \mathbf{t} + 2 + 2\log_2(x+1)$, $g\colon x \mapsto x\, f(s_i/x)$ and $\mu = \lceil \log_2((\mathbf{t}+1)n/s_i) \rceil$. Both functions f and g are positive, concave and increasing on $(0, +\infty)$, which shows that

$$\begin{aligned}
\mathsf{C}_{\mathbf{t}}(\mu+\theta+h) &\leqslant \sum_{R \in \mathcal{R}_{\mu+\theta+h}} f(r_{\to i}) \leqslant |\mathcal{R}_{\mu+\theta+h}|\, f\big(\textstyle\sum_{R \in \mathcal{R}_{\mu+\theta+h}} r_{\to i} / |\mathcal{R}_{\mu+\theta+h}|\big) \leqslant g\big(|\mathcal{R}_{\mu+\theta+h}|\big) \\
&\leqslant g\big(n/2^{\mu+h}\big) \leqslant g\big(2^{-h} s_i/(\mathbf{t}+1)\big) \\
&\leqslant \big(\mathbf{t} + 2 + 2\log_2\big(2^h(\mathbf{t}+1)+1\big)\big) 2^{-h} s_i/(\mathbf{t}+1) \\
&\leqslant \big(\mathbf{t} + 2 + 2(h(\mathbf{t}+1)+1)\big) 2^{-h} s_i/(\mathbf{t}+1) \leqslant (4+2h) 2^{-h} s_i.
\end{aligned}$$

We conclude that

$$\begin{aligned}
\sum_{R \in \mathcal{T}} \mathsf{cost}^*_{\mathbf{t}}(r_{\to i}) = \sum_{h \geqslant 0} \mathsf{C}_{\mathbf{t}}(h) &= \sum_{h=0}^{\mu+\theta-1} \mathsf{C}_{\mathbf{t}}(h) + \sum_{h \geqslant 0} \mathsf{C}_{\mathbf{t}}(\mu+\theta+h) \\
&\leqslant (1 + 1/(\mathbf{t}+3))(\mu+\theta)s_i + 4s_i \sum_{h \geqslant 0} 2^{-h} + 2s_i \sum_{h \geqslant 0} h 2^{-h} \\
&\leqslant (1 + 1/(\mathbf{t}+3))(\log_2(n/s_i) + \log_2(\mathbf{t}+1))s_i + \mathcal{O}(s_i) \\
&\leqslant (1 + 1/(\mathbf{t}+3))\log_2(n/s_i)s_i + \log_2(\mathbf{t}+1)s_i + \mathcal{O}(s_i).
\end{aligned}$$

◀

5.3 Using a parameter $\mathbf{t}$ with logarithmic growth

The upper bound provided by Theorem 19 is minimal when $\mathbf{t} = \Theta(\mathcal{H}^*)$, in which case it simply becomes $n\mathcal{H}^* + \log_2(\mathcal{H}^*+1)n + \mathcal{O}(n)$. However, computing $\mathcal{H}^*$ before starting the actual sorting process is not reasonable. Instead, we update the parameter $\mathbf{t}$ as follows, which will provide us with a slightly larger upper bound.

▶ **Definition 20.** *We call* logarithmic *galloping sub-routine the merging sub-routine that, when merging adjacent runs of lengths a and b, performs the same comparisons and element moves as the $\mathbf{t}$-galloping sub-routine for $\mathbf{t} = \lceil \log_2(a+b) \rceil$.*

▶ **Theorem 21.** *Let $\mathcal{A}$ be a stable natural merge sort algorithm with the tight middle-growth property. If $\mathcal{A}$ uses the logarithmic galloping sub-routine for merging runs, it requires at most $n\mathcal{H}^* + 2\log_2(\mathcal{H}^*+1)n + \mathcal{O}(n)$ element comparisons to sort arrays of length n and dual run-length entropy $\mathcal{H}^*$.*

Proof. Let us refine and adapt the proofs of Theorems 8 and 19. Let θ be the integer mentioned in the definition of the statement "$\mathcal{A}$ has the tight middle-growth property", let $\mathcal{T}$ be the merge tree induced by $\mathcal{A}$ on an array A of length n, and let $s_1, s_2, \ldots, s_\sigma$ be the lengths of the dual runs of A.

Using a parameter $\mathbf{t} = \lceil \log_2(r) \rceil$ to merge runs R' and R'' into one run R requires at most

$$1 + \sum_{i=1}^{\sigma} \mathsf{cost}^*_{\lceil \log_2(r) \rceil}(r'_{\to i}) + \mathsf{cost}^*_{\lceil \log_2(r) \rceil}(r''_{\to i})$$

element comparisons. Given that

$$\begin{aligned}\mathsf{cost}^*_{\lceil \log_2(r) \rceil}(r'_{\to i}) &\leqslant \min\{(1 + 1/\log_2(r))r'_{\to i}, \log_2(r) + 3 + 2\log_2(r'_{\to i} + 1)\} \\ &\leqslant \min\{(1 + 1/\log_2(r))r'_{\to i}, 3\log_2(r+1) + 3\},\end{aligned}$$

and that $r'_{\to i} + r''_{\to i} = r_{\to i}$, this makes a total of at most

$$1 + \sum_{i=1}^{\sigma} \mathsf{cost}^*_{\log}(r, r_{\to i})$$

element comparisons, where $\mathsf{cost}^*_{\log}(r, m) = \min\{(1 + 1/\log_2(r))m, 6\log_2(r+1) + 6\}$.

Then, let $\mathcal{T}^*$ denote the tree obtained after removing the leaves of $\mathcal{T}$. We focus on proving that

$$\sum_{R \in \mathcal{T}^*} \mathsf{cost}^*_{\log}(r, r_{\to i}) \leqslant s_i \log_2(n/s_i) + 2s_i \log_2(\log_2(2n/s_i)) + \mathcal{O}(s_i)$$

for all $i \leqslant \sigma$. Indeed, finding the run decomposition $R_1, R_2, \ldots, R_\rho$ of A requires $n-1$ comparisons, and $\rho - 1 \leqslant n - 1$ merges are then performed, which will make a total of up to

$$\begin{aligned}2n + \sum_{R \in \mathcal{T}^*} \sum_{i=1}^{\sigma} \mathsf{cost}^*_{\log}(r, r_{\to i}) &\leqslant 2n + \sum_{i=1}^{\sigma} s_i \log_2(n/s_i) + 2s_i \log_2(\log_2(n/s_i) + 1) + \mathcal{O}(s_i) \\ &\leqslant n\mathcal{H}^* + 2\log_2(\mathcal{H}^* + 1)n + \mathcal{O}(n)\end{aligned}$$

comparisons, the latter inequality being due to the concavity of the function $x \mapsto \log_2(x+1)$.

Then, let $\mathcal{R}_h$ be the set of runs at height h in $\mathcal{T}$, and let

$$\mathsf{C}_{\log}(h) = \sum_{R \in \mathcal{R}_h} \mathsf{cost}^*_{\log}(r, r_{\to i}).$$

No run in $\mathcal{R}_h$ descends from another one, and each run $R \in \mathcal{R}_h$ has length $r \geqslant 2^{\max\{1, h-\theta\}}$, which proves that $|\mathcal{R}_h| \leqslant n/2^{h-\theta}$ and that

$$\mathsf{C}_{\log}(h) \leqslant \sum_{R \in \mathcal{R}_h} (1+1/\log_2(r))r_{\to i} \leqslant \sum_{R \in \mathcal{R}_h} (1+1/\max\{1, h-\theta\})r_{\to i} \leqslant (1+1/\max\{1, h-\theta\})s_i.$$

Finally, let $z = n/s_i \geqslant 1$, and consider the constant $\nu = \lceil \log_2(z\log_2(2z)) \rceil + \theta$ and the functions $f\colon x \mapsto 1 + \log_2(x+1)$ and $g\colon x \mapsto x\, f(n/x)$. Both functions f and g are concave, positive and increasing on $(0, +\infty)$, which proves that

$$\begin{aligned}\mathsf{C}_{\log}(\nu + h)/6 \leqslant \sum_{R \in \mathcal{R}_{\nu+h}} f(r) &\leqslant |\mathcal{R}_{\nu+h}|\, f\big(\textstyle\sum_{R \in \mathcal{R}_{\nu+h}} r/|\mathcal{R}_{\nu+h}|\big) \leqslant g(|\mathcal{R}_{\nu+h}|) \leqslant g(n/2^{\nu-\theta+h}) \\ &\leqslant g\big(2^{-h}s_i/\log_2(2z)\big) = 2^{-h}s_i f(2^h z\log_2(2z))/\log_2(2z) \\ &\leqslant 2^{-h}s_i\big(1 + \log_2(2^h(2z)^2)\big)/\log_2(2z) = 2^{-h}(h+1)s_i/\log_2(2z) + 2^{1-h}s_i \\ &\leqslant (h+3)2^{-h}s_i,\end{aligned}$$

where the inequality between the second and third line simply comes from the fact that

$$1 + 2^h z \log_2(2z) \leqslant 1 + 2^{h+1} z^2 \leqslant 2^h (1 + 2z^2) \leqslant 2^h \times (2z)^2$$

whenever $h \geqslant 0$ and $z \geqslant 1$.

It follows that

$$\sum_{h=1}^{\theta} \mathsf{C}_{\log}(h) + \sum_{h \geqslant 0} \mathsf{C}_{\log}(\nu + h) \leqslant 2\theta s_i + 6 \sum_{h \geqslant 0} (h+3) 2^{-h} s_i = \mathcal{O}(s_i),$$

whereas

$$\sum_{h=1}^{\nu-\theta-1} \mathsf{C}_{\log}(\theta + h) \leqslant \sum_{h=1}^{\nu-1} (1 + 1/h) s_i \leqslant ((\nu - 1) + 1 + \ln(\nu - 1)) s_i = (\nu + \ln(\nu)) s_i.$$

Thus, we conclude that

$$\begin{aligned} \sum_{R \in \mathcal{T}^*} \mathsf{cost}^*_{\log}(r, r_{\to i}) = \sum_{h \geqslant 1} \mathsf{C}_{\log}(h) &\leqslant s_i(\nu + \ln(\nu)) + \mathcal{O}(s_i) \\ &\leqslant s_i \log_2(z) + s_i \log_2(\log_2(2z)) + s_i \log_2(\log_2(2z^2)) + \mathcal{O}(s_i) \\ &\leqslant s_i \log_2(z) + 2 s_i \log_2(\log_2(2z)) + \mathcal{O}(s_i). \end{aligned}$$

◀

It is of course possible to marginally improve our update policy in order to approach the $n\mathcal{H}^* + \log_2(\mathcal{H}^* + 1)n + \mathcal{O}(n)$ upper bound. For instance, choosing $\mathbf{t} = \tau\lceil \log_2(a+b) \rceil$ for a given constant $\tau \geqslant 1$ provides us with an $n\mathcal{H}^* + (1 + 1/\tau) \log_2(\mathcal{H}^* + 1)n + \log_2(\tau) + \mathcal{O}(n)$ upper bound, and choosing $\mathbf{t} = \lceil \log_2(a+b) \rceil \times \lceil \log_2(\log_2(a+b)) \rceil$ further improves that upper bound. However, such improvements soon become negligible in comparison with the overhead of having to compute the value of $\mathbf{t}$.

Finally, and like in Section 4, we list a few algorithms that enjoy similar complexity upper bounds. Proofs that they do can be found in the complete version of this article [11].

▶ **Theorem 22.** *The algorithms* NaturalMergeSort *and* ShiversSort *have the tight middle-growth property.*

▶ **Theorem 23.** *Theorems 19 and 21 remain valid if we consider the algorithms* PeekSort *and* adaptive ShiversSort *instead of an algorithm with the tight middle-growth property.*

By contrast, we conjecture that no choice policy for the parameters $\mathbf{t}$ would provide us with $(1 + o(1))n\mathcal{H}^* + \mathcal{O}(n)$ upper bounds on the number of element comparisons performed by TimSort, α-StackSort or α-MergeSort. However, finding precise characterisations of the best constants c that could be achieved for these algorithms is a wide open question.

6 Conclusion: An idealistic galloping cost model

In the above sections, we observed the impact of using a galloping sub-routine for a fixed or a variable parameter $\mathbf{t}$. Although choosing a constant value of $\mathbf{t}$ (e.g., $\mathbf{t} = 7$, as advocated in [17]) already leads to very good results, letting $\mathbf{t}$ vary, for instance by using the logarithmic variant of the sub-routine, provides us with even better complexity guarantees, with an often negligible overhead of $\mathcal{O}(n \log(\mathcal{H}^* + 1) + n)$ element comparisons: up to a small error, this provides us with the following *idealistic* cost model for run merges, allowing us to simultaneously identify the parameter $\mathbf{t}$ with $+\infty$ and with a constant.

▶ **Definition 24.** *Let A and B be two non-decreasing runs with $a_{\to i}$ (respectively, $b_{\to i}$) elements of value i for all $i \in \{1, 2, \ldots, \sigma\}$. The* idealistic galloping cost *of merging A and B is defined as the quantity*

$$\sum_{i=1}^{\sigma} \mathsf{cost}^*_{\mathsf{ideal}}(a_{\to i}) + \mathsf{cost}^*_{\mathsf{ideal}}(b_{\to i}),$$

where $\mathsf{cost}^*_{\mathsf{ideal}}(m) = \min\{m, \log_2(m+1) + \mathcal{O}(1)\}$.

Indeed, abusively identifying the real number of element comparisons performed while runs A and B with this idealistic galloping cost, we may prove that every stable natural merge sort with the tight middle-growth property (but also related algorithms such as PeekSort and adaptive ShiversSort) requires at most $n\mathcal{H}^* + \mathcal{O}(n)$ element comparisons to sort arrays of length n and dual run-length entropy $\mathcal{H}^*$.

We think that this idealistic cost model is both simple and precise enough to allow studying the complexity of natural merge sorts in general, provided that they use the galloping sub-routine. Thus, it would be interesting to use that cost model in order to study, for instance, the least constant c for which various algorithms such as TimSort or α-MergeSort require up to $\mathsf{c}n(1 + o(1))\mathcal{H}^* + \mathcal{O}(n)$ element comparisons.

We also hope that this simpler framework will foster the interest for the galloping merging sub-routine of TimSort, and possibly lead to amending Swift and Rust implementations of TimSort to include that sub-routine, which we believe is too efficient in relevant cases to be omitted.

References

1. Nicolas Auger, Vincent Jugé, Cyril Nicaud, and Carine Pivoteau. On the worst-case complexity of timsort. In *26th Annual European Symposium on Algorithms (ESA)*, pages 4:1–13, 2018. Extended version available at: `arXiv:1805.08612`.
2. Nicolas Auger, Cyril Nicaud, and Carine Pivoteau. Merge strategies: from merge sort to timsort. Research report hal-01212839, 2015.
3. Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theoretical Computer Science*, 513:109–123, 2013.
4. Jérémy Barbay, Carlos Ochoa, and Srinivasa Rao Satti. Synergistic solutions on multisets. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 31:2–14, 2017.
5. Jon Bentley and Andrew Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976.
6. Josh Bloch. Timsort implementation in java 13, retrieved 01/01/2022. URL: `https://github.com/openjdk/jdk/blob/3afeb2cb4861f95fd20c3c04f04be93b435527c0/src/java.base/share/classes/java/util/ComparableTimSort.java`.
7. Sam Buss and Alexander Knop. Strategies for stable merge sorting. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1290, 2019.
8. Ben Cohen. Timsort implementation in java 13, retrieved 01/01/2022. URL: `https://github.com/apple/swift/blob/4c1d46bc0980d84bf3178bc42295522c242fec86/ stdlib/public/core/Sort.swift`.
9. Vladmir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.
10. Adriano Garsia and Michelle Wachs. A new algorithm for minimal binary search trees. *SIAM Journal on Computing*, 6(4):622–642, 1977.
11. Elahe Ghasemi, Vincent Jugé, and Ghazal Khalighinejad. Galloping in natural merge sorts. In *49th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 61:1–61:19, 2022. Extended version available at: `arrXiv:2012.03996`.

12 Te Hu and Alan Tucker. Optimal computer search trees and variable-length alphabetical codes. *SIAM Journal on Applied Mathematics*, 21(4):514–532, 1971.

13 Vincent Jugé. Adaptive shivers sort: an alternative sorting algorithm. In *31th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1639–1654, 2020.

14 Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publish. Co., 1998.

15 Christos Levcopoulos and Ola Petersson. Sorting shuffled monotone sequences. *Information and Computation*, 112(1):37–50, 1994.

16 Heikki Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Trans. Computers*, 34(4):318–325, 1985.

17 Peter McIlroy. Optimistic sorting and information theoretic complexity. In *4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 467–474, 1993.

18 Ian Munro and Philip Spira. Sorting and searching in multisets. *SIAM journal on Computing*, 5(1):1–8, 1976.

19 Ian Munro and Sebastian Wild. Nearly-optimal mergesorts: Fast, practical sorting methods that optimally adapt to existing runs. In *26th Annual European Symposium on Algorithms (ESA 2018)*, pages 63:1–63:15, 2018.

20 Tim Peters. Timsort description, retrieved 01/09/2021. URL: `https://github.com/python/cpython/blob/24e5ad4689de9adc8e4a7d8c08fe400dcea668e6/Objects/listsort.txt`.

21 Clément Renault et al. Timsort implementation in rust, retrieved 01/01/2022. URL: `https://github.com/rust-lang/rust/blob/a5a91c8e0732753de7c028182cbb02901fe1b608/library/alloc/src/slice.rs`.

22 Olin Shivers. A simple and efficient natural merge sort. Technical report, Georgia Institute of Technology, 2002.

23 Guido van Rossum et al. Powersort implementation in cpython, retrieved 01/01/2022. URL: `https://github.com/python/cpython/blob/c8749b578324ad4089c8d014d9136bc42b065343/Objects/listobject.c`.

24 Jeff Weaton and Markus Mützel. Timsort implementation in octave, retrieved 01/01/2022. URL: `https://github.com/gnu-octave/octave/blob/7e6da6d504ad962a8d33622b6955b0210ff62365/liboctave/util/oct-sort.cc`.

25 Simon Zünd et al. Timsort implementation in v8, retrieved 01/01/2022. URL: `https://github.com/v8/v8/blob/25f0e32915930df1d53722b91177b1dee5202499/third_party/v8/builtins/array-sort.tq`.

Tolerant Bipartiteness Testing in Dense Graphs

Arijit Ghosh
Indian Statistical Institute, Kolkata, India

Gopinath Mishra
University of Warwick, Coventry, UK

Rahul Raychaudhury
Duke University, Durham, NC, USA

Sayantan Sen
Indian Statistical Institute, Kolkata, India

Abstract

Bipartite testing has been a central problem in the area of property testing since its inception in the seminal work of Goldreich, Goldwasser, and Ron. Though the non-tolerant version of bipartite testing has been extensively studied in the literature, the tolerant variant is not well understood. In this paper, we consider the following version of tolerant bipartite testing problem: Given two parameters $\varepsilon, \delta \in (0,1)$, with $\delta > \varepsilon$, and access to the adjacency matrix of a graph G, we have to decide whether G can be made bipartite by editing at most εn^2 entries of the adjacency matrix of G, or we have to edit at least δn^2 entries of the adjacency matrix to make G bipartite. In this paper, we prove that for $\delta = (2 + \Omega(1))\varepsilon$, tolerant bipartite testing can be decided by performing $\widetilde{\mathcal{O}}\left(1/\varepsilon^3\right)$ many adjacency queries and in $2^{\widetilde{\mathcal{O}}(1/\varepsilon)}$ time complexity. This improves upon the state-of-the-art query and time complexities of this problem of $\widetilde{\mathcal{O}}\left(1/\varepsilon^6\right)$ and $2^{\widetilde{\mathcal{O}}(1/\varepsilon^2)}$, respectively, due to Alon, Fernandez de la Vega, Kannan and Karpinski, where $\widetilde{\mathcal{O}}(\cdot)$ hides a factor polynomial in $\log(1/\varepsilon)$.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Tolerant Testing, Bipartite Testing, Query Complexity, Graph Property Testing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.69

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2204.12397

Funding Gopinath Mishra's research is supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP) and by EPSRC award EP/V01305X/1.

Acknowledgements The authors would like to thank Yufei Zhao, Dingding Dong and Nitya Mani for pointing out a mistake in an earlier version of this paper, as well as the reviewers of ICALP for various suggestions that improved the presentation of the paper.

1 Introduction

The field of property testing refers to the model where the main goal is to design efficient algorithms that reads only "small" part of the input. Over the past few years, the field has had very rapid growth, and several interesting techniques and results have emerged. See, for example, the new property testing book by Goldreich [11] for an introduction to property testing.

The field of graph property testing was first introduced in the seminal work of Goldreich, Goldwasser, and Ron [12]. In that work, the authors studied various interesting and important problems in dense graphs and testing bipartiteness was one of them. In non-tolerant variant

of bipartiteness testing, we are given a dense graph G and a proximity parameter $\varepsilon \in (0,1)$ as the input, and the goal is to decide if G is bipartite, or do we need to modify at least εn^2 many entries of the adjacency matrix of G to make it bipartite, using as few queries to the adjacency matrix of G as possible.

Due to the fundamental nature of the problem, *bipartite testing* has been extensively studied over the past two decades [12]. Though there are several works on non-tolerant testing of various graph properties across all models in graph property testing [12, 13, 7], there are very few works related to their tolerant counterparts (See, for example, the property testing book by Goldreich [11] for an extensive list of various results). To the best of our knowledge, this is the first time tolerant bipartite testing has been *explicitly* studied in the literature.

Now we formally define the notion of *bipartite distance* and state our main result. Then we discuss our result vis-a-vis the related works.

▶ **Definition 1.1** (Bipartite distance). *A* bipartition *of a graph G is a function $f : V(G) \to \{L, R\}$, where $V(G)$ denotes the vertex set of G* [1]*. The* bipartite *distance of G with respect to the bipartition f is denoted and defined as*

$$d_{bip}(G, f) := \left[\sum_{v \in V : f(v) = L} |N(v) \cap f^{-1}(L)| + \sum_{v \in V : f(v) = R} |N(v) \cap f^{-1}(R)| \right].$$

Here $N(v)$ denotes the neighborhood of v in G. Informally, $d_{bip}(G, f)$ measures the distance of the graph G from being bipartite, with respect to the bipartition f. The *bipartite distance* of G is defined as the minimum bipartite distance of G over all possible bipartitions f of G, that is,

$$d_{bip}(G) := \min_{f} d_{bip}(G, f).$$

Now we are ready to formally state our result.

▶ **Theorem 1.2** (Main result). *Given query access to the adjacency matrix of a dense graph G with n vertices and a proximity parameter $\varepsilon \in (0,1)$, there exists an algorithm that, with probability at least $\frac{9}{10}$, decides whether $d_{bip}(G) \leq \varepsilon n^2$ or $d_{bip}(G) \geq (2+\Omega(1))\varepsilon n^2$, by sampling $\mathcal{O}\left(\frac{1}{\varepsilon^3} \log \frac{1}{\varepsilon}\right)$ many vertices in $2^{\mathcal{O}\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)}$ time, and performs $\mathcal{O}\left(\frac{1}{\varepsilon^3} \log^2 \frac{1}{\varepsilon}\right)$ many queries.*

1.1 Our result in the context of literature

Recall that non-tolerant bipartite testing refers to the problem where we are given query access to the adjacency matrix of an unknown graph G and a proximity parameter $\varepsilon \in (0,1)$, and the objective is to decide whether $d_{bip}(G) = 0$ or $d_{bip}(G) \geq \varepsilon n^2$. The problem of non-tolerant bipartite testing in the dense graph model was first studied in the seminal work of Goldreich, Goldwasser, and Ron [12], and they showed that it admits an algorithm with query complexity $\widetilde{\mathcal{O}}\left(1/\varepsilon^3\right)$. Later, Alon and Krivelevich [4] improved the query complexity of the problem to $\widetilde{\mathcal{O}}\left(1/\varepsilon^2\right)$. They further studied the problem of testing c-colorability of dense graph. Note that bipartite testing is a special case of testing c-colorability, when $c = 2$. They proved that c-colorability can be tested by performing $\widetilde{\mathcal{O}}\left(1/\varepsilon^4\right)$ many queries, for $c \geq 3$. This bound was later improved to $\widetilde{\mathcal{O}}\left(1/\varepsilon^2\right)$ by Sohler [18]. On the other hand, for

[1] L and R denote left and right respectively.

non-tolerant bipartite testing, Bogdanov and Trevisan [6] proved that $\Omega(1/\varepsilon^2)$ and $\Omega(1/\varepsilon^{3/2})$ many adjacency queries are required by any non-adaptive and adaptive testers, respectively. Later, Gonen and Ron [14] further explored the power of adaptive queries for bipartiteness testing. Bogdanov and Li [5] showed that bipartiteness can be tested with one-sided error in $\mathcal{O}(1/\varepsilon^c)$ queries, for some constant $c < 2$, assuming a conjecture [2].

Though the non-tolerant variant of bipartite testing is well understood, the query complexity of the tolerant version (even for restricted cases like we consider in Theorem 1.2) has not yet been addressed in the literature. From the result of Alon, Vega, Kannan and Karpinski [1],for estimating MaxCut [3] for any given ε $(0 < \varepsilon < 1)$, it implies that that the bipartite distance of a (dense) graph G can be estimated up to an additive error of εn^2, by performing $\widetilde{\mathcal{O}}\left(1/\varepsilon^6\right)$ many queries (see Appendix A for details, and in particular, see Corollary A.2). Even for the tolerant version that we consider in Theorem 1.2, their algorithm does not give any bound better than $\widetilde{\mathcal{O}}\left(1/\varepsilon^6\right)$. Note that Alon, Vega, Kannan and Karpinski [1] improved the result of Goldreich, Goldwasser, and Ron [12], who were the first to prove that MaxCut can be estimated with an additive error of εn^2 by performing $\widetilde{\mathcal{O}}(1/\varepsilon^7)$ many adjacency queries and with time complexity $2^{\widetilde{\mathcal{O}}(1/\varepsilon^3)}$. Though we improve the bound for tolerant bipartite testing (for the restricted case as stated in Theorem 1.2) substantially from the work of Alon et al. [1], we would like to note that this is the first work that studies tolerant bipartite testing explicitly.

1.2 Other related works

Apart from the dense graph model, this problem has also been studied in other models of property testing. Goldreich and Ron [13] studied the problem of bipartiteness testing for bounded degree graphs, where they gave an algorithm of $\widetilde{\mathcal{O}}(\sqrt{n})$ queries, where n denotes the number of vertices of the graph. Later, Kaufman, Krivelevich and Ron [15] studied the problem in the general graph model and gave an algorithm with query complexity $\widetilde{\mathcal{O}}\left(\min\left\{\sqrt{n}, n^2/m\right\}\right)$, where m denotes the number of edges of the graph. Few years back, Czumaj, Monemizadeh, Onak, and Sohler [7] studied the problem for planar graphs (more generally, for any minor-free graphs), where they employed a random walk based technique, and proved that constant number of queries are enough for bipartiteness testing. Apart from bipartite testing, there have been extensive works related to property testing in the dense graph model and its connection to the regularity lemma [3, 2, 9].

1.3 Organization

In Section 2, we present an overview of our algorithm along with a brief description of its analysis. In Section 3, we formally describe our algorithm, followed by its correctness analysis in Section 4. Finally, we conclude in Section 5. The proofs that are omitted in the main text are either presented in the appendix or in the full version of the paper [10].

1.4 Notations

All graphs considered here are undirected, unweighted, and have no self-loops or parallel edges. For a graph $G = (V(G), E(G))$, $V(G)$ and $E(G)$ denote the vertex and edge sets of G respectively. For a vertex $v \in V(G)$, $N_G(v)$ denotes the neighborhood of a vertex v in

[2] The conjecture is that if the graph G is ε-far from bipartite, then an induced subgraph on $\widetilde{\mathcal{O}}(1/\varepsilon)$ many random vertices would be $\widetilde{\Omega}(\varepsilon)$-far from being bipartite with probability at least 1/2.

[3] MaxCut of a graph G denotes the size of the largest *cut* in G.

G, and we will write it as $N(v)$ when the graph G is clear from the context. Since we are only considering undirected graphs, we write an edge as $\{u, v\} \in E(G)$. For a set of pairs of vertices Z, we will denote the set of vertices present in at least one pair in Z by $V(Z)$. For a function $f : V(G) \to \{L, R\}$, $f^{-1}(L)$ and $f^{-1}(R)$ represent the set of vertices that are mapped to L and R by f respectively. We denote by $\binom{V(G)}{2}$ the set of unordered pairs of the vertices of G. Finally, $a = (1 \pm \varepsilon)b$ represents $(1 - \varepsilon)b \leq a \leq (1 + \varepsilon)b$.

2 Overview of the proof of Theorem 1.2

In this section, we give an overview of our algorithm. The detailed description of the algorithm is presented in Section 3, while its analysis is presented in Section 4. We will prove the following theorem, which is our main technical result.

▶ **Theorem 2.1.** *There exists an algorithm* Tol-Bip-Dist(G, ε) *that given adjacency query access to a dense graph G with n vertices and a parameter $\varepsilon \in (0, 1)$, decides with probability at least $9/10$, whether $d_{bip}(G) \leq \varepsilon n^2$ or $d_{bip}(G) \geq (2 + k)\varepsilon n^2$, by sampling $\mathcal{O}(\frac{1}{k^5\varepsilon^2} \log \frac{1}{k\varepsilon})$ many vertices in $2^{\mathcal{O}(\frac{1}{k^3\varepsilon} \log \frac{1}{k\varepsilon})}$ time, using $\mathcal{O}(\frac{1}{k^8\varepsilon^3} \log^2 \frac{1}{k\varepsilon})$ many queries to the adjacency matrix of G.*

Note that Theorem 2.1 implies Theorem 1.2, by taking $k = \Omega(1)$.

2.1 Brief description of the algorithm

Assume C_1, C_2, C_3 are three suitably chosen large absolute constants. At the beginning of our algorithm, we generate t many subsets of vertices $X_1, \ldots, X_t$, each with $\lceil \frac{C_2}{k^3\varepsilon} \log \frac{1}{k\varepsilon} \rceil$ many vertices chosen randomly, where $t = \lceil \log \frac{C_1}{k\varepsilon} \rceil$. Let $\mathcal{C} = X_1 \cup \ldots \cup X_t$. Apart from the X_i's, we also randomly select a set of pairs of vertices Z, with $|Z| = \lceil \frac{C_3}{k^5\varepsilon^2} \log \frac{1}{k\varepsilon} \rceil$. We find the neighbors of each vertex of Z in $\mathcal{C}$. Then for each vertex pair in Z, we check whether it is an edge in the graph or not. Loosely speaking, the set of edges between $\mathcal{C}$ and $V(Z)$ [4] will help us generate partial bipartitions, restricted to $X_i \cup V(Z)$'s, for each $i \in [t]$, and the edges among the pairs of vertices of Z will help us in estimating the bipartite distance of some *specific kind* of bipartitions of G. Here we would like to note that no further query will be performed by the algorithm. The set of edges with one vertex in $\mathcal{C}$ and the other in $V(Z)$, and the set of edges among the vertex pairs in Z, when treated in a *specific* manner, will give us the desired result. Observe that the number of adjacency queries performed by our algorithm is $\mathcal{O}(\frac{1}{k^8\varepsilon^3} \log^2 \frac{1}{k\varepsilon})$.

For each $i \in [t]$, we do the following. We consider all possible bipartitions $\mathcal{F}_i$ of X_i. For each bipartition f_{ij} (of X_i) in $\mathcal{F}_i$, we extend f_{ij} to a bipartition of $X_i \cup V(Z)$, say f'_{ij}, such that both f_{ij} and f'_{ij} are identical with respect to X_i. Moreover, we assign $f'_{ij}(z)$ (to either L or R), for each $z \in V(Z) \setminus X_i$, based on the neighbors of z in X_i. To design a rule of assigning $f'_{ij}(z)$, for each $z \in V(Z) \setminus X_i$ for our purpose, we define the notions of *heavy* and *balanced* vertices, with respect to a bipartition (see Definitions 4.1 and 4.2). Heavy and balanced vertices are defined in such a manner that when the bipartite distance of G is at most εn^2 (that is, G is ε-close), we can infer the following interesting connections. Let f be a bipartition of $V(G)$ such that $d_{bip}(G, f) \leq \varepsilon n^2$. We will prove that the total number of edges, with no endpoints in X_i and whose at least one endpoint is a balanced vertex with respect to f, is bounded (see Claim 4.12). Moreover, if we generate a bipartition f' such that

[4] Recall that $V(Z)$ denotes the set of vertices present in at least one pair in Z.

f and f' differ for *large* number of heavy vertices, then the bipartite distance with respect to f' cannot be small. To guarantee the correctness of our algorithm, we will prove that a heavy vertex v with respect to f, can be detected and $f(v)$ can be determined, with probability at least $1 - o(k\varepsilon)$. Note that the testing of being a heavy vertex will be performed only for the vertices in $V(Z)$. We will see shortly how this will help us to guarantee the completeness of our algorithm.

Finally, our algorithm computes ζ_{ij}, that is, the fraction of vertex pairs in Z that are monochromatic [5] edges with respect to f'_{ij}. If we find at least one i and j such that $\zeta_{ij} \leq \left(2 + \frac{k}{20}\right)\varepsilon$, the algorithm decides that $d_{bip}(G) \leq \varepsilon n^2$. Otherwise, it will report that $d_{bip}(G) \geq (2+k)\varepsilon n^2$.

2.2 Completeness

Let us assume that the bipartite distance of G is at most εn^2, and let f be a bipartition of $V(G)$ that is optimal. Let us now focus on a particular $i \in [t]$, that is, an X_i. Since we are considering all possible bipartitions $\mathcal{F}_i$ of X_i, there exists a $f_{ij} \in \mathcal{F}_i$, such that f_{ij} and f are identical with respect to X_i. To complete our argument, we introduce (in Definition 4.3) the notion of SPECIAL bipartition $\textsc{Spl}_i^f : V(G) \to \{L, R\}$, with respect to f by f_{ij} such that $f(v)$, $f_{ij}(v)$ and $\textsc{Spl}_i^f(v)$ are identical for each $v \in X_i$, and at least $1 - o(k\varepsilon)$ fraction of heavy vertices, with respect to f, are mapped identically both by f and $\textsc{Spl}_i^f$. We shall prove that the bipartite distance of G with respect to $\textsc{Spl}_i^f$ is at most $\left(2 + \frac{k}{50}\right)\varepsilon n^2$ (see Lemma 4.6). Now let us think of generating a bipartition f''_{ij} of $V(G)$ such that, for each $v \in V(G) \setminus X_i$, if we determine $f''_{ij}(v)$ by the same rule used by our algorithm to determine $f_{ij}(z)$, for each $z \in V(Z) \setminus X_i$. Note that our algorithm does not find f''_{ij} explicitly, it is used only for the analysis purpose. The number of heavy vertices, with respect to the bipartition f, that have different mappings by f and f''_{ij}, is at most $o(k\varepsilon n)$ with constant probability. So, with a constant probability, f''_{ij} is a SPECIAL bipartition with respect to f by f_{ij}. Note that, if we take $|Z| = \mathcal{O}(\frac{1}{k^5\varepsilon^2} \log \frac{1}{k\varepsilon})$ many random vertex pairs and determine the fraction χ_{ij}^f of pairs that form monochromatic edges with respect to the SPECIAL bipartition f''_{ij}, we can show that $\chi_{ij}^f \leq (2 + \frac{k}{20})\varepsilon$, with probability at least $1 - 2^{-\Omega(\frac{1}{k^3\varepsilon} \log \frac{1}{k\varepsilon})} \geq \frac{9}{10}$. However, we are not finding either f''_{ij} or χ_{ij}^f explicitly. We just find ζ_{ij}, that is, the fraction of vertex pairs in Z that are monochromatic edges with respect to f'_{ij}. But the above argument still holds, since Z is chosen randomly and there exists a f''_{ij}, such that $f''_{ij}(z) = f'_{ij}(z)$, for each $z \in V(Z)$, and the probability distribution of ζ_{ij} is identical to that of χ_{ij}^f.

2.3 Soundness

Let us now consider the case when the bipartite distance of G is at least $(2+k)\varepsilon n^2$, and f be any bipartition of $V(G)$. To prove the soundness of our algorithm, we introduce the notion of DERIVED *bipartition* $\textsc{Der}_i^f : V(G) \to \{L, R\}$ with respect to f by f_{ij} (see Definition 4.4), such that $f(v)$, $f_{ij}(v)$ and $\textsc{Der}_i^f(v)$ are identical for each $v \in X_i$. Observe that the bipartite distance of G with respect to any DERIVED bipartition is at least $(2+k)\varepsilon n^2$ as well. Similar to the discussion of the completeness, if we generate a bipartition f''_{ij} of $V(G)$, f''_{ij} will be a DERIVED bipartition, with respect to f by f_{ij}. If we take $|Z| = \mathcal{O}(\frac{1}{k^5\varepsilon^2} \log \frac{1}{k\varepsilon})$ many random pairs of vertices and determine the fraction χ_{ij}^f of pairs that form monochromatic edges with respect to the DERIVED bipartition f''_{ij}, we can prove that $\chi_{ij}^f \leq \left(2 + \frac{k}{20}\right)\varepsilon$ holds, with

[5] An edge is said to be monochromatic with respect to f'_{ij} if both its endpoints have the same f'_{ij} values.

probability at most $2^{-\Omega(\frac{1}{k^3\varepsilon}\log\frac{1}{k\varepsilon})}$. We want to re-emphasize that we are not determining f''_{ij}, as well as χ^f_{ij} explicitly. The argument follows due to the facts that Z is chosen randomly and there exists an f''_{ij} such that $f'_{ij}(z) = f''_{ij}(z)$, for each $z \in V(Z)$, and the probability distribution of ζ_{ij} is identical to that of χ^f_{ij}. Using the union bound, we can say that the algorithm rejects with probability at least $\frac{9}{10}$.

3 Algorithm for Tolerant Bipartite Testing (Proof of Theorem 2.1)

In this section, we formalize the ideas discussed in Section 2, and prove Theorem 2.1.

Formal description of algorithm Tol-Bip-Dist(G, ε)

Step-1 Let C_1, C_2, C_3 be three suitably chosen large constants and $t := \lceil \log \frac{C_1}{k\varepsilon} \rceil$.

(i) We start by generating t many subset of vertices $X_1, \ldots, X_t \subset V(G)$, each with $\lceil \frac{C_2}{k^3\varepsilon} \log \frac{1}{k\varepsilon} \rceil$ many vertices, sampled randomly without replacement [6].

(ii) We sample $\lceil \frac{C_3}{k^5\varepsilon^2} \log \frac{1}{k\varepsilon} \rceil$ many random pairs of vertices, with replacement, and denote those sampled pairs of vertices as Z. Note that $X_1, \ldots, X_t, Z$ are generated independent of each other.

(iii) We find all the edges with one endpoint in $\mathcal{C} = X_1 \cup X_2 \cup \ldots X_t$ and the other endpoint in one of the vertices of $V(Z)$ [7], by performing $\mathcal{O}\left(\frac{1}{k^8\varepsilon^3}\log^2\frac{1}{k\varepsilon}\right)$ many adjacency queries.

Step-2 (i) Let $\{a_1, b_1\}, \ldots \{a_\lambda, b_\lambda\}$ be the pairs of vertices of Z, where $\lambda = \lceil \frac{C_3}{k^5\varepsilon^2} \log \frac{1}{k\varepsilon} \rceil$. Now we find the pairs of Z that are edges in G, by performing adjacency queries to all the pairs of vertices of Z (after this step, the algorithm does not make any query further).

(ii) For each $i \in [t]$, we do the following:

(a) Let $\mathcal{F}_i$ denote the set of all possible bipartitions of X_i, that is,

$$\mathcal{F}_i = \left\{ f_{ij} : X_i \to \{L, R\} : j \in \left[2^{|X_i|-2}\right] \right\}.$$

(b) For each bipartition f_{ij} (of X_i) in $\mathcal{F}_i$, we extend f_{ij} to $f'_{ij} : X_i \cup Z \to \{L, R\}$ to be a bipartition of $X_i \cup Z$, such that the mapping of each vertex of X_i are identical in f_{ij} and f'_{ij}, and is defined as follows:

$$f'_{ij}(z) = \begin{cases} f_{ij}(z), & z \in X_i \\ L, & z \notin X_i \text{ and } \left|N(z) \cap f_{ij}^{-1}(R)\right| > \left|N(z) \cap f_{ij}^{-1}(L)\right| + \frac{k\varepsilon|X_i|}{225000} \\ R, & z \notin X_i \text{ and } \left|N(z) \cap f_{ij}^{-1}(L)\right| > \left|N(z) \cap f_{ij}^{-1}(R)\right| + \frac{k\varepsilon|X_i|}{225000} \\ \text{L or } R \text{ arbitrarily}, & \text{otherwise} \end{cases}$$

Note that this step can be performed from the adjacency information between the vertices of $\mathcal{C}$ and Z, which have already been computed before.

(c) We now find the fraction of the vertex pairs of Z that are edges and have the same label with respect to f'_{ij}, that is,

$$\zeta_{ij} = 2 \cdot \frac{\left|\left\{\{a_\ell, b_\ell\} : \ell \in [\lambda], \{a_\ell, b_\ell\} \in E(G) \text{ and } f'_{ij}(a_\ell) = f'_{ij}(b_\ell)\right\}\right|}{\lambda}$$ [8].

(d) If $\zeta_{ij} \leq \left(2 + \frac{k}{20}\right)\varepsilon$, we Accept G as ε-close to being bipartite, and Quit the algorithm.

---

[6] Since we are assuming n is sufficiently large with respect to $\frac{1}{\varepsilon}$, sampling with and without replacement are the same.

[7] Recall that $V(Z)$ denotes the set of vertices present in at least one pair in Z.

(iii) If we arrive at this step, then $\zeta_{ij} > \left(2 + \frac{k}{20}\right)\varepsilon$, for each $i \in [t]$ and $f_{ij} \in \mathcal{F}_i$ in Step-(ii). We REJECT and declare that G is $(2+k)\varepsilon$-far from being bipartite.

We split the analysis of algorithm TOL-BIP-DIST(G, ε) into five parts:

Completeness: If G is ε-close to being bipartite, then TOL-BIP-DIST(G, ε) reports the same, with probability at least $\frac{9}{10}$.

Soundness: If G is $(2+k)\varepsilon$-far from being bipartite, then TOL-BIP-DIST(G, ε) reports the same, with probability at least $\frac{9}{10}$.

Sample Complexity: The sample complexity of TOL-BIP-DIST(G, ε) is $\mathcal{O}(\frac{1}{k^5\varepsilon^2}\log\frac{1}{k\varepsilon})$.

Query Complexity: The query complexity of TOL-BIP-DIST(G, ε) is $\mathcal{O}(\frac{1}{k^8\varepsilon^3}\log^2\frac{1}{k\varepsilon})$.

Time Complexity: The time complexity of TOL-BIP-DIST(G, ε) is $2^{\mathcal{O}(\frac{1}{k^3\varepsilon}\log\frac{1}{k\varepsilon})}$.

The last three quantities can be computed from the description of TOL-BIP-DIST(G, ε). In **Step-1(i)**, we sample vertices of G to generate $t = \lceil \log \frac{C_1}{k\varepsilon} \rceil$ subsets, each with $\lceil \frac{C_2}{k^3\varepsilon}\log\frac{1}{k\varepsilon} \rceil$ many vertices. Thereafter in **Step-1(ii)** and **Step-1(iii)**, we randomly choose $\lceil \frac{C_3}{k^5\varepsilon^2}\log\frac{1}{k\varepsilon} \rceil$ many pairs of vertices and perform adjacency queries for each vertex in any pair of Z to every X_i. Thus the sample complexity of TOL-BIP-DIST(G, ε) is $\mathcal{O}(\frac{1}{k^5\varepsilon^2}\log\frac{1}{k\varepsilon})$ and query complexity is $\mathcal{O}(\frac{1}{k^8\varepsilon^3}\log^2\frac{1}{k\varepsilon})$. The time complexity of the algorithm is $2^{\mathcal{O}(\frac{1}{k^3\varepsilon}\log\frac{1}{k\varepsilon})}$, which follows from **Step-2(ii)**, that dominates the running time.

4 Proof of Correctness of TOL-BIP-DIST(G, ε)

Before proceeding to the proof, we introduce some definitions for classifying the vertices of the graph, with respect to any particular bipartition, into two categories:

(i) ***heavy*** vertex, and

(ii) ***balanced*** vertex.

These definitions will be mostly used in the proof of completeness. Informally speaking, a vertex v is said to be **heavy** with respect to a bipartition f, if it has *substantially* large number of neighbors in one side of the bipartition (either L or R), as compared to the other side.

▶ **Definition 4.1** (Heavy vertex). *A vertex $v \in V$ is said to be L-**heavy*** with respect to a bipartition f, *if it satisfies two conditions:*

(i) $\left|N(v) \cap f^{-1}(L)\right| \geq \left|N(v) \cap f^{-1}(R)\right| + \frac{k\varepsilon n}{150}$;

(ii) *If* $\left|N(v) \cap f^{-1}(R)\right| \geq \frac{1}{(1+\frac{k}{200})}\frac{k\varepsilon n}{150}$, *then*

$$\left|N(v) \cap f^{-1}(L)\right| \geq (1 + \tfrac{k}{200})\left|N(v) \cap f^{-1}(R)\right|;$$

*We define R-**heavy** vertices analogously. The union of the set of L-**heavy** and R-**heavy** vertices, with respect to a bipartition f, is defined to be the set of heavy vertices (with respect to f), and is denoted by $\mathcal{H}_f$.*

*Similarly, a vertex v is said to be **balanced** if the number of neighbors of v are* similar *in both L and R, with respect to a bipartition f. We define it formally as follows:*

▶ **Definition 4.2** (Balanced vertex). *A vertex $v \in V$ is said to be* **balanced** with respect to a bipartition f, *if $v \notin \mathcal{H}_f$, that is, it satisfies at least one of the following conditions:*

(i) Type 1: $\left|\left|N(v) \cap f^{-1}(R)\right| - \left|N(v) \cap f^{-1}(L)\right|\right| < \frac{k\varepsilon n}{150}$;

(ii) Type 2: *Either*

$$\left|N(v)\cap f^{-1}(L)\right| \leq \left|N(v)\cap f^{-1}(R)\right| < (1+\frac{k}{200})\left|N(v)\cap f^{-1}(L)\right|,$$

or,

$$\left|N(v)\cap f^{-1}(R)\right| \leq \left|N(v)\cap f^{-1}(L)\right| < (1+\frac{k}{200})\left|N(v)\cap f^{-1}(R)\right|.$$

The set of balanced vertices of type 1 with respect to f is denoted as $\mathcal{B}_f^1$, and the set of balanced vertices of type 2 with respect to f is denoted as $\mathcal{B}_f^2$. The union of $\mathcal{B}_f^1$ and $\mathcal{B}_f^2$ is denoted by $\mathcal{B}_f$. Note that $\mathcal{B}_f^1$ and $\mathcal{B}_f^2$ may not be disjoint.

In order to prove the completeness (in Section 4.1), we also use a notion of SPECIAL *bipartition* to be defined below. The definition of SPECIAL bipartition is based on an optimal bipartition f of $V(G)$, and notions of heavy and balanced vertices. We would also like to note that, later in Lemma 4.6, we show that when $d_{bip}(G) \leq \varepsilon n^2$, the bipartite distance of G with respect to any SPECIAL bipartition is bounded by $(2+\frac{k}{50})\varepsilon n^2$.

▶ **Definition 4.3** (SPECIAL bipartition)**.** *Let $d_{bip}(G) \leq \varepsilon n^2$, and $f : V(G) \to \{L,R\}$ be an optimal bipartition of $V(G)$, that is, $d_{bip}(G,f) \leq \varepsilon n^2$, and there does not exist any bipartition g such that $d_{bip}(G,g) < d_{bip}(G,f)$. For an X_i selected in* **Step-1(i)** *of the algorithm, let $f_{ij} \in \mathcal{F}_i$ be the bipartition of X_i such that $f\mid_{X_i} = f_{ij}$. Then bipartition $\textsc{Spl}_i^f : V(G) \to \{L,R\}$ is said to be a* SPECIAL **bipartition** *with respect to f by f_{ij} such that*

- $\textsc{Spl}_i^f \mid_{X_i} = f\mid_{X_i} = f_{ij}$;
- *There exists a subset $\mathcal{H}'_f \subset \mathcal{H}_f$ such that $\left|\mathcal{H}'_f\right| \geq (1-o(k\varepsilon))\left|\mathcal{H}_f\right|$, and for each $v \in \mathcal{H}'_f$, $\textsc{Spl}_i^f(v)$ is defined as follows:*

$$\textsc{Spl}_i^f(v) = \begin{cases} R, & v \notin X_i \text{ and } v \text{ is } L-\text{heavy} \\ L, & v \notin X_i \text{ and } v \text{ is } R-\text{heavy} \end{cases}$$

- *For each $v \notin (\mathcal{H}'_f \cup X_i)$, $\textsc{Spl}_i^f(v)$ is set to L or R arbitrarily.*

In our proof of the soundness theorem (in Section 4.2), we need the notion of DERIVED *bipartition*. Unlike the definition of SPECIAL bipartition, the definition of DERIVED bipartition is more general, in the sense that it is not defined based on either any optimal bipartition, or on heavy or balanced vertices.

▶ **Definition 4.4** (DERIVED bipartition)**.** *Let $f : V(G) \to \{L,R\}$ be a bipartition of $V(G)$. For an X_i selected in* **Step-1(i)** *of the algorithm, let $f_{ij} \in \mathcal{F}_i$ be the bipartition of X_i such that $f\mid_{X_i} = f_{ij}$. A bipartition $\textsc{Der}_i^f : V(G) \to \{L,R\}$ is said to be* DERIVED *bipartition with respect to f by f_{ij}, if $\textsc{Der}_i^f \mid_{X_i} = f\mid_{X_i} = f_{ij}$.*

4.1 Proof of Completeness

In this section, we prove the following theorem:

▶ **Theorem 4.5.** *Let us assume G is ε-close to being bipartite. Then* TOL-BIP-DIST(G,ε) *reports the same, with probability at least* $9/10$.

The proof of Theorem 4.5 will cricially use the following lemma, which says that the bipartite distance of G with respect to any SPECIAL bipartition is bounded by a $\left(2+\frac{k}{50}\right)\varepsilon n^2$.

▶ **Lemma 4.6** (SPECIAL bipartition lemma). *Let f be a bipartition such that $d_{bip}(G,f) \leq \varepsilon n^2$ and there does not exist any bipartition g such that $d_{bip}(G,g) < d_{bip}(G,f)$. For any* SPECIAL *bipartition* $\textsc{Spl}_i^f$ *with respect to f, we have*

$$d_{bip}(G, \textsc{Spl}_i^f) \leq \left(2 + \frac{k}{50}\right) \varepsilon n^2.$$

We will prove the above lemma later. For now, we want to establish (in Lemma 4.8) that there exists an $i \in [t]$ and a$f_{ij} \in \mathcal{F}_i$ which can be thought of as a *random restriction* of some SPECIAL bipartition with respect to f by f_{ij}. In other words, Lemma 4.8 basically states that if G is ϵ-close to being bipartite, then the extension according to the rule in **Step-2(ii)(b)** of the mapping obtained by restricting an optimal bipartition to a random X_i is likely to correspond to a SPECIAL bipartition, and therefore, the number of monochromatic edges (with respect to a SPECIAL bipartition) in the randomly picked Z is likely to be low with respect to that bipartition. Thus, ζ_{ij} must be low for some i, j with high probability.

To prove Lemma 4.8, we need the following lemma (Lemma 4.7) about heavy vertices. In Lemma 4.7, we basically prove that a heavy vertex with respect to a bipartition f will have significantly more neighbors in the part of X_i, that corresponds to the heavy side of that vertex (with respect to f). Basically, if a vertex v is L-heavy with respect to f, it has more neighbors in the subset of X_i on the L-side as compared to the subset of X_i on the R-side of f.

▶ **Lemma 4.7** (Heavy vertex lemma). *Let f be a bipartition of G. Consider a vertex $v \in V$.*

- **(i)** *For every L-heavy vertex v, $\left|N(v) \cap f^{-1}(L) \cap X_i\right| - \left|N(v) \cap f^{-1}(R) \cap X_i\right| \geq \frac{k^2\varepsilon|X_i|}{225000}$ with probability at least $1 - o(k\varepsilon)$.*
- **(ii)** *For every R-heavy vertex v, $\left|N(v) \cap f^{-1}(L) \cap X_i\right| - \left|N(v) \cap f^{-1}(R) \cap X_i\right| \geq \frac{k^2\varepsilon|X_i|}{225000}$ with probability at least $1 - o(k\varepsilon)$.*

We would like to note that Lemma 4.7 holds for any bipartition. However, we will use it only for completeness with resepct to an optimal bipartition f.

▶ **Lemma 4.8.** *If $d_{bip}(G) \leq \varepsilon n^2$, then there exists an $i \in [t]$ and $f_{ij} \in \mathcal{F}_i$ such that $\zeta_{ij} \leq \left(2 + \frac{k}{20}\right)\varepsilon$ holds, with probability at least $1 - o(k\varepsilon)$.*

Proof. Let f be an optimal bipartition such that $d_{bip}(G,f) \leq \varepsilon n^2$. First, consider a SPECIAL bipartition $\textsc{Spl}_i^f$, and consider a set of random vertex pairs Y such that $|Y| = |Z|$. Now consider the fraction of monochromatic edges of Y, with respect to the bipartition $\textsc{Spl}_i^f$, that is,

$$\chi_{ij}^f = 2 \cdot \frac{\left|\left\{\{a,b\} \in Y : \{a,b\} \in E(G) \text{ and } \textsc{Spl}_i^f(a) = \textsc{Spl}_i^f(b)\right\}\right|}{|Y|}.$$

▶ **Observation 4.9.** *$\chi_{ij}^f \leq \left(2 + \frac{k}{20}\right)\varepsilon$ holds, with probability at least $9/10$.*

Proof. By Lemma 4.6, we know that when $d_{bip}(G) \leq \varepsilon n^2$, $d_{bip}(G, \textsc{Spl}_i^f) \leq \left(2 + \frac{k}{50}\right)\varepsilon n^2$. So, $\mathbb{E}[\chi_{ij}^f] \leq \left(2 + \frac{k}{50}\right)\varepsilon$. Using Chernoff bound (see Lemma B.1), we can say that

$$\mathbb{P}\left(\chi_{ij}^f \geq \left(2 + \frac{k}{20}\right)\varepsilon\right) \leq \frac{1}{2^{\Omega\left(\frac{1}{k^3\varepsilon}\log\frac{1}{k\varepsilon}\right)}} \leq \frac{1}{10}.$$ ◀

Now, we claim that bounding χ_{ij}^f is equivalent to bounding ζ_{ij}.

▷ Claim 4.10. For any $i \in [t]$, there exists a bipartition $f_{ij} \in \mathcal{F}_i$ such that the probability distribution of ζ_{ij} is identical to that of χ^f_{ij}, for some SPECIAL bipartition f with respect to f_{ij}, with probability at least $1/2$.

As $t = \mathcal{O}(\log \frac{1}{k\varepsilon})$, the above claim implies that there exists an $i \in [t]$ and $f_{ij} \in \mathcal{F}_i$ such that the probability distribution of ζ_{ij} is identical to that of χ^f_{ij}, with probability at least $1 - o(k\varepsilon)$.

Now we prove Claim 4.10. Recall the procedure of determining ζ_{ij} as described in **Step 2** of algorithm TOL-BIP-DIST(G, ε) presented in Section 3.

Fact 1: For any vertex $v \in \mathcal{H}_f \cap Z$, $\text{SPL}^f_i(v) = f'_{ij}(v)$, with probability at least $1 - o(k\varepsilon)$, where $\mathcal{H}_f$ denotes the set of heavy vertices of X_i with respect to the bipartition f. This follows according to Claim 4.7, along with the definition of $f'_{ij}(z)$.

Fact 2: Consider a bipartition $f_{ij} \in \mathcal{F}_i$ of X_i, and its extension $f^{'}_{ij}$ to $X_i \cup Z$, as considered in the algorithm. Assume a bipartition $f^{''}_{ij}$ of $V(G)$, constructed by extending f'_{ij} according to the rule of **Step-2(ii)(b)** of the algorithm. From Heavy vertex lemma (Lemma 4.7), we know that the expected number of vertices in $\mathcal{H}_f$ such that $f^{''}_{ij}(v) \neq f(v)$, is at most $o(k\varepsilon)\,|\mathcal{H}_f|$. Using Markov inequality, we can say that, with probability at least $\frac{1}{2}$, the number of vertices in $\mathcal{H}_f$ such that $f^{''}_{ij}(v) \neq f(v)$, is at most $o(k\varepsilon)\,|\mathcal{H}_f|$. Thus, with probability at least $\frac{1}{2}$, there exists a set of vertices $\mathcal{H}'_f$ such that $f^{''}_{ij}(v) = f(v)$ holds for at least $(1 - o(k\varepsilon))\left|\mathcal{H}'_f\right|$ vertices. Note that the bipartition $f^{''}_{ij}$ is a SPECIAL bipartition f with respect to f_{ij}.

From **Fact 1** and **Fact 2**, we can deduce that, there exists a SPECIAL bipartition SPL^f_i such that $\text{SPL}^f_i(v) = f'_{ij}(v)$ for each $z \in Z$.

Since we choose Z uniformly at random, Lemma 4.8 follows. ◀

According to the description of algorithm TOL-BIP-DIST(G, ε), the algorithm reports that $d_{bip}(G) \leq \varepsilon n^2$, if there exists a ζ_{ij} such that $\zeta_{ij} \leq \left(2 + \frac{k}{20}\right)\varepsilon$, for some $i \in [t]$ and $j \in [2^{|X_i|-2}]$. Hence, by Lemma 4.8, we are done with the proof of the completeness theorem (Theorem 4.5).

Now we focus on proving SPECIAL bipartition lemma (Lemma 4.6) and Heavy vertex lemma (Lemma 4.7), starting with the proof of SPECIAL bipartition lemma.

Proof of SPECIAL bipartition lemma (Lemma 4.6)

The idea of the proof relies on decomposing the bipartite distance with respect to a SPECIAL bipartition into a sum of three terms and then carefully bounding the cost of each of those parts individually.

Let us first recall the definition of bipartite distance of G with respect to a special bipartition SPL^f_i.

$$d_{bip}(G, \text{SPL}^f_i) = \left|\left\{(u,v) \in E(G) : \text{SPL}^f_i(u) = \text{SPL}^f_i(v)\right\}\right|. \tag{1}$$

By abuse of notation, here we are denoting $E(G)$ as the set of ordered edges.

We will upper bound $d_{bip}(G, \text{SPL}^f_i)$ as the sum of three terms defined below. Here $\mathcal{H}_f$ and $\mathcal{B}_f$ denote the set of heavy vertices and balanced vertices (with respect to f), as defined in Definition 4.1 and Definition 4.2, respectively. Also, $\mathcal{H}'_f \subseteq \mathcal{H}_f$ denotes the set of vertices of $\mathcal{H}_f$ that are mapped according to f, as defined in the definition of SPECIAL bipartition in Definition 4.3. The three terms that are used to upper bound $d_{bip}(G, \text{SPL}^f_i)$ are as follows:

(a) $D_{\mathcal{H}'_f \cup X_i, \mathcal{H}'_f \cup X_i} = \left|\left\{(u,v) \in E(G) : u \in \mathcal{H}'_f \cup X_i \ \& \ v \in \mathcal{H}'_f \cup X_i, \textsc{Spl}_i^f(u) = \textsc{Spl}_i^f(v)\right\}\right|$;

(b) $D_{\mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i), V(G)} =$
$\left|\left\{(u,v) \in E(G) : u \in \mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i), \& \ v \in V(G), \textsc{Spl}_i^f(u) = \textsc{Spl}_i^f(v)\right\}\right|$;

(c) $D_{\mathcal{B}_f \setminus X_i, V(G)} = \left|\left\{(u,v) \in E(G) : u \in \mathcal{B}_f \setminus X_i \text{ and } v \in V(G), \textsc{Spl}_i^f(u) = \textsc{Spl}_i^f(v)\right\}\right|$.

Now from Equation 1 along with the above definitions, we can upper bound $d_{bip}(G, \textsc{Spl}_i^f)$ as follows:

$$d_{bip}(G, \textsc{Spl}_i^f) \leq D_{\mathcal{H}'_f \cup X_i, \mathcal{H}'_f \cup X_i} + D_{\mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i), V(G)} + D_{\mathcal{B}_f \setminus X_i, V(G)}. \tag{2}$$

We now upper bound $d_{bip}(G, \textsc{Spl}_i^f)$ by bounding each term on the right hand side of the above expression separately, via the two following claims which we will prove later.

▷ Claim 4.11.

(i) $D_{\mathcal{H}'_f \cup X_i, \mathcal{H}'_f \cup X_i} \leq d_{bip}(G, f) - \Pi$, where

$$\Pi := \left[\sum_{v \in \mathcal{B}_f \setminus X_i : f(v) = L} |N(v) \cap f^{-1}(L)| + \sum_{v \in \mathcal{B}_f \setminus X_i : f(v) = R} |N(v) \cap f^{-1}(R)|\right];$$

(ii) $D_{\mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i), V(G)} \leq o(k\varepsilon)n^2$;

▷ Claim 4.12. $D_{\mathcal{B}_f \setminus X_i, V(G)} \leq 2\left(1 + \frac{k}{400}\right)\Pi + \frac{k\varepsilon n^2}{150}$.

Assuming Claim 4.11 and Claim 4.12 hold, along with Equation 2, $d_{bip}(G, \textsc{Spl}_i^f)$ can be upper bounded as follows:

$$\begin{aligned} d_{bip}(G, \textsc{Spl}_i^f) &\leq d_{bip}(G, f) - \Pi + o(k\varepsilon)n^2 + 2\left(1 + \frac{k}{400}\right)\Pi + \frac{k\varepsilon n^2}{150} \\ &\leq d_{bip}(G, f) + \Pi + \frac{k}{200}\Pi + \frac{k\varepsilon n^2}{100}. \end{aligned}$$

Note that $\Pi \leq d_{bip}(G, f)$ and $d_{bip}(G, f) \leq \varepsilon n^2$. Hence, we can say the following:

$$d_{bip}(G, \textsc{Spl}_i^f) \leq \left(2 + \frac{k}{50}\right)\varepsilon n^2.$$

So, we are done with the proof of the SPECIAL bipartition lemma. We are left with the proofs of Claim 4.11 and Claim 4.12.

Proof of Claim 4.11. (i) We use the following observation in our proof. The observation follows due to the fact that the bipartition f considered is an optimal bipartition.

▶ **Observation 4.13.** *Let v be a L-heavy vertex v with respect to f. Then $f(v) = R$. Similarly, for every R-heavy vertex v with respect to f, $f(v) = L$.*

Following the definition of SPECIAL bipartition, we know that there exists a set of vertices $\mathcal{H}'_f \subset \mathcal{H}_f$ such that $\left|\mathcal{H}'_f\right| \geq (1 - o(k\varepsilon))\left|\mathcal{H}_f\right|$, and for each $v \in \mathcal{H}'_f$, the following holds:

$$\textsc{Spl}_i^f(v) = \begin{cases} R, & v \notin X_i \text{ and } v \text{ is } L - \text{heavy} \\ L, & v \notin X_i \text{ and } v \text{ is } R - \text{heavy} \end{cases}$$

By Observation 4.13, we know that for every $v \in \mathcal{H}'_f$, $\textsc{Spl}_i^f(v) = f(v)$. Moreover, for each $v \in X_i$, $\textsc{Spl}_i^f(v) = f(v)$, following the definition of SPECIAL bipartition $\textsc{Spl}_i^f$. Thus, for every $v \in \mathcal{H}'_f \cup X_i$, we have $\textsc{Spl}_i^f(v) = f(v)$. Hence,

$$\begin{aligned}
&D_{\mathcal{H}'_f \cup X_i, \mathcal{H}'_f \cup X_i} \\
&= \left|\left\{(u,v) \in E(G) : u \in \mathcal{H}'_f \cup X_i \text{ and } v \in \mathcal{H}'_f \cup X_i, \textsc{Spl}_i^f(u) = \textsc{Spl}_i^f(v)\right\}\right| \\
&= \left|\left\{(u,v) \in E(G) : u \in \mathcal{H}'_f \cup X_i, \text{ and } v \in \mathcal{H}'_f \cup X_i, f(u) = f(v)\right\}\right| \\
&\qquad\qquad \left(\because \text{for every } v \in \mathcal{H}'_f \cup X_i,\ \textsc{Spl}_i^f(v) = f(v)\right) \\
&= d_{bip}(G,f) - \\
&\qquad \left[\sum_{v \in V \setminus (\mathcal{H}'_f \cup X_i) : f(v) = L} |N(v) \cap f^{-1}(L)| + \sum_{v \in V \setminus (\mathcal{H}'_f \cup X_i) : f(v) = R} |N(v) \cap f^{-1}(R)|\right] \\
&\leq d_{bip}(G,f) - \left[\sum_{v \in \mathcal{B}_f \setminus X_i : f(v) = L} |N(v) \cap f^{-1}(L)| + \sum_{v \in \mathcal{B}_f \setminus X_i : f(v) = R} |N(v) \cap f^{-1}(R)|\right] \\
&= d_{bip}(G,f) - \Pi.
\end{aligned}$$

(ii) By the definition of $\mathcal{H}'_f$, we know that $\left|\mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i)\right|$ is upper bounded by $o(k\varepsilon)\,|\mathcal{H}_f|$. Following the definition of $D_{\mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i), V(G)}$, we can say the following:

$$\begin{aligned}
&D_{\mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i), V(G)} \\
&= \left|\left\{(u,v) \in E(G) : u \in \mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i) \text{ and } v \in V(G), \textsc{Spl}_i^f(u) = \textsc{Spl}_i^f(v)\right\}\right| \\
&\leq \left|\mathcal{H}_f \setminus (\mathcal{H}'_f \cup X_i)\right| \times |V(G)| = o(k\varepsilon)\,|\mathcal{H}_f| \times n \leq o(k\varepsilon) n^2.
\end{aligned}$$

The last inequality follows as $|\mathcal{H}_f|$ is at most n. ◁

Proof of Claim 4.12. Observe that

$$\begin{aligned}
&D_{\mathcal{B}_f \setminus X_i, V(G)} \\
&= \left|\left\{(u,v) \in E(G) : u \in \mathcal{B}_f \setminus X_i \text{ and } v \in V(G), \textsc{Spl}_i^f(u) = \textsc{Spl}_i^f(v)\right\}\right| \\
&\leq |\{(u,v) \in E(G) : u \in \mathcal{B}_f \setminus X_i \text{ and } v \in V(G)\}| = \sum_{v \in \mathcal{B}_f \setminus X_i} |N(v)|
\end{aligned}$$

As $\mathcal{B}_f = \mathcal{B}_f^1 \cup \mathcal{B}_f^2$, we have

$$D_{\mathcal{B}_f \setminus X_i, V(G)} \leq \sum_{v \in \mathcal{B}_f^1 \setminus X_i} |N(v)| + \sum_{v \in \mathcal{B}_f^2 \setminus X_i} |N(v)|\,. \tag{3}$$

We will bound $D_{\mathcal{B}_f \setminus X_i, V(G)}$ by bounding $\sum_{v \in \mathcal{B}_f^1 \setminus X_i} |N(v)|$ and $\sum_{v \in \mathcal{B}_f^2 \setminus X_i} |N(v)|$ separately, which we prove in the following claim:

▷ Claim 4.14. Consider T_1 and T_2 defined as follows:

$$T_1 = 2\left(\sum_{v \in f^{-1}(L) \cap (\mathcal{B}_f^1 \setminus X_i)} \left|N(v) \cap f^{-1}(L)\right| + \sum_{v \in f^{-1}(R) \cap (\mathcal{B}_f^1 \setminus X_i)} \left|N(v) \cap f^{-1}(R)\right|\right) + \frac{k\varepsilon n^2}{150},$$

$$T_2 = \left(2 + \frac{k}{200}\right)\left(\sum_{v \in f^{-1}(L) \cap (\mathcal{B}_f^2 \setminus X_i)} \left|N(v) \cap f^{-1}(L)\right| + \sum_{v \in f^{-1}(R) \cap (\mathcal{B}_f^2 \setminus X_i)} \left|N(v) \cap f^{-1}(R)\right|\right).$$

Then

(i) For balanced vertices of **Type 1**, we have

$$\sum_{v \in \mathcal{B}_f^1 \setminus X_i} |N(v)| \leq T_1$$

(ii) For balanced vertices of **Type 2**, we have

$$\sum_{v \in \mathcal{B}_f^2 \setminus X_i} |N(v)| \leq T_2$$

See the full version of the paper [10] for the proof of the above claim. Using Claim 4.14 and Equation (3), we get

$$\begin{aligned} & D_{\mathcal{B}_f \setminus X_i, V(G)} \\ &= \sum_{v \in \mathcal{B}_f^1 \setminus X_i} |N(v)| + \sum_{v \in \mathcal{B}_f^2 \setminus X_i} |N(v)| \\ &\leq T_1 + T_2 \\ &\leq 2\left(1 + \frac{k}{400}\right)\Pi + \frac{k\varepsilon n^2}{150} \end{aligned}$$

The last inequality follows from the definitions of T_1, T_2 and Π. ◁

Proof of Heavy vertex lemma (Lemma 4.7)

Before proceeding to prove the Heavy vertex lemma, we will first prove two intermediate claims that will be crucially used in the proof of the lemma. The first claim states that when we consider a bipartition f of G, if a vertex $v \in G$ has a *large* number of neighbors on one side of the partition defined by f, the proportion of its neighbors in X_i on the same side of f will be approximately preserved, where X_i is a set of vertices picked at random in **Step-1(i)** of the algorithm Tol-Bip-Dist(G, ε). The result is formally stated as follows:

▷ Claim 4.15. Let f be a bipartition of G. Consider a vertex $v \in V$.

(i) Suppose $\left|N(v) \cap f^{-1}(L)\right| \geq \frac{k\varepsilon n}{150}$. Then, with probability at least $1 - o(k\varepsilon)$, we have

$$\left|N(v) \cap f^{-1}(L) \cap X_i\right| = \left(1 \pm \tfrac{k}{500}\right)\left|N(v) \cap f^{-1}(L)\right| \tfrac{|X_i|}{n}.$$

(ii) Suppose $\left|N(v) \cap f^{-1}(R)\right| \geq \frac{k\varepsilon n}{150}$. Then, with probability at least $1 - o(k\varepsilon)$, we have

$$\left|N(v) \cap f^{-1}(R) \cap X_i\right| = \left(1 \pm \tfrac{k}{500}\right)\left|N(v) \cap f^{-1}(R)\right| \tfrac{|X_i|}{n}.$$

The next claim is in similar spirit as that of Claim 4.15. Instead of considering vertices with large number of neighbors, it considers the case when a vertex has *small* number of neighbors on one side of a bipartition f.

▷ Claim 4.16. Let f be a bipartition of G. Consider a vertex $v \in V$.

(i) Suppose $\left|N(v) \cap f^{-1}(L)\right| \leq \frac{1}{1+\frac{k}{200}} \frac{k\varepsilon n}{150}$. Then, with probability at least $1 - o(k\varepsilon)$, we have

$$\left|N(v) \cap f^{-1}(L) \cap X_i\right| \leq \tfrac{1}{1+\frac{k}{300}} \tfrac{k\varepsilon |X_i|}{150}.$$

(ii) Suppose $\left|N(v) \cap f^{-1}(R)\right| \leq \frac{1}{1+\frac{k}{200}} \frac{k\varepsilon n}{150}$. Then, with probability at least $1 - o(k\varepsilon)$, we have

$$\left|N(v) \cap f^{-1}(R) \cap X_i\right| \leq \frac{1}{1+\frac{k}{300}} \frac{k\varepsilon |X_i|}{150}.$$

Claim 4.15 and Claim 4.16 can be proved by using large deviation inequalities (stated in Appendix B), and the proofs can be found in the full version of the paper [10].

Assuming Claim 4.15 and Claim 4.16 hold, we now prove the Heavy vertex lemma (Lemma 4.7).

Proof of Lemma 4.7. We will only prove (i) here, which concerns the L-heavy vertices. (ii) can be proved in similar fashion.

We first characterize L-heavy vertices into two categories:

(a) Both $\left|N(v) \cap f^{-1}(L)\right|$ and $\left|N(v) \cap f^{-1}(R)\right|$ are large, that is, $\left|N(v) \cap f^{-1}(L)\right| \geq \frac{k\varepsilon n}{150}$ and $\left|N(v) \cap f^{-1}(R)\right| \geq \frac{1}{1+\frac{k}{200}} \frac{k\varepsilon n}{150}$. Also, $\left|N(v) \cap f^{-1}(L)\right| \geq \left(1 + \frac{k}{200}\right) \left|N(v) \cap f^{-1}(R)\right|$.

(b) $\left|N(v) \cap f^{-1}(L)\right|$ is large and $\left|N(v) \cap f^{-1}(R)\right|$ is small, that is, $\left|N(v) \cap f^{-1}(L)\right| \geq \frac{k\varepsilon n}{150}$ and $\left|N(v) \cap f^{-1}(R)\right| \leq \frac{1}{1+\frac{k}{200}} \frac{k\varepsilon n}{150}$.

Case (a): Here $\left|N(v) \cap f^{-1}(L)\right| \geq \left(1 + \frac{k}{200}\right) \frac{k\varepsilon n}{150}$, and $\left|N(v) \cap f^{-1}(R)\right| \geq \frac{k\varepsilon n}{150}$. From Claim 4.15, the following hold, with probability at least $1 - o(k\varepsilon)$:

$$\left|N(v) \cap f^{-1}(L) \cap X_i\right| = \left(1 \pm \frac{k}{500}\right) \left|N(v) \cap f^{-1}(L)\right| \frac{|X_i|}{n}$$

and

$$\left|N(v) \cap f^{-1}(R) \cap X_i\right| = \left(1 \pm \frac{k}{500}\right) \left|N(v) \cap f^{-1}(R)\right| \frac{|X_i|}{n}.$$

So, with probability at least $1 - o(k\varepsilon)$, we have the following

$$\begin{aligned}
&\left|N(v) \cap f^{-1}(L) \cap X_i\right| - \left|N(v) \cap f^{-1}(R) \cap X_i\right| \\
&\geq \left(1 - \frac{k}{500}\right) \left|N(v) \cap f^{-1}(L)\right| \frac{|X_i|}{n} - \left(1 + \frac{k}{500}\right) \left|N(v) \cap f^{-1}(R)\right| \frac{|X_i|}{n} \\
&\geq \left(1 - \frac{k}{500} - \frac{1 + \frac{k}{500}}{1 + \frac{k}{200}}\right) \frac{\left|N(v) \cap f^{-1}(L)\right| |X_i|}{n} \\
&\qquad\qquad \left(\because \left|N(v) \cap f^{-1}(L)\right| \geq (1 + \frac{k}{200}) \left|N(v) \cap f^{-1}(R)\right|\right) \\
&\geq \frac{k}{1500} \times \frac{k\varepsilon\, |X_i|}{150} \\
&\geq \frac{k^2 \varepsilon\, |X_i|}{225000} \qquad\qquad (\because\ k \leq 100)
\end{aligned}$$

Case (b): Here $\left|N(v) \cap f^{-1}(L)\right| \geq \frac{k\varepsilon n}{150}$ and $\left|N(v) \cap f^{-1}(R)\right| \leq \left(\frac{1}{1+\frac{k}{200}}\right) \frac{k\varepsilon n}{150}$. So, from Claim 4.15 and Claim 4.16, the following hold, with probability at least $1 - o(k\varepsilon)$:

$$\left|N(v) \cap f^{-1}(L) \cap X_i\right| = \left(1 \pm \frac{k}{500}\right) \left|N(v) \cap f^{-1}(L)\right| \frac{|X_i|}{n}$$

and

$$\left|N(v) \cap f^{-1}(R) \cap X_i\right| \leq \frac{1}{1 + \frac{k}{300}} \frac{k\varepsilon\, |X_i|}{150}.$$

Thus, with probability at least $1 - o(k\varepsilon)$, we have the following:

$$\begin{aligned}
&\left|N(v) \cap f^{-1}(L) \cap X_i\right| - \left|N(v) \cap f^{-1}(R) \cap X_i\right| \\
&\geq (1 - \frac{k}{500}) \left|N(v) \cap f^{-1}(L)\right| \frac{|X_i|}{n} - \frac{1}{1 + \frac{k}{300}} \frac{k\varepsilon |X_i|}{150} \\
&= (1 - \frac{k}{500}) \frac{k\varepsilon |X_i|}{150} - \frac{1}{1 + \frac{k}{300}} \frac{k\varepsilon |X_i|}{150} \\
&\geq \frac{1}{1500} \left(2k - \frac{k^2}{100}\right) \frac{k\varepsilon |X_i|}{150} \\
&\geq \frac{k^2 \varepsilon |X_i|}{225000} \qquad (\because\ k \leq 100)
\end{aligned}$$

This completes the proof of part (i) of Lemma 4.7. ◀

4.2 Proof of Soundness

In this section, we prove the following theorem:

▶ **Theorem 4.17.** *Let us assume that G is $(2+k)\varepsilon$-far from being bipartite. Then* Tol-Bip-Dist(G, ε) *reports the same, with probability at least* $9/10$.

Assume f be a bipartition of $V(G)$. Now let us consider a DERIVED bipartition Der_i^f with respect to f by f_{ij}, and choose a set of random vertex pairs Y such that $|Y| = |Z|$. Let χ_{ij}^f denote the fraction of vertex pairs of Y that are monochromatic with respect to the bipartition Der_i^f, that is,

$$\chi_{ij}^f = 2 \cdot \frac{\left|\left\{\{a,b\} \in Y : \{a,b\} \in E(G) \text{ and } \text{Der}_i^f(a) = \text{Der}_i^f(b)\right\}\right|}{|Y|}.$$

▶ **Observation 4.18.** *$\chi_{ij}^f \leq \left(2 + \frac{k}{20}\right)\varepsilon$ holds with probability at most $\frac{1}{10N}$, where $N = 2^{\mathcal{O}(\frac{1}{k^3\varepsilon} \log \frac{1}{k\varepsilon})}$.*

Proof. Since G is $(2+k)\varepsilon$-far from being bipartite, the same holds for the bipartition Der_i^f as well, that is, $d_{bip}(G, \text{Der}_i^f) \geq (2+k)\varepsilon n^2$. So, $\mathbb{E}[\chi_{ij}^f] \geq (2+k)\,\varepsilon$. Using Chernoff bound (see Lemma B.1), we can say that, $\mathbb{P}\left(\chi_{ij}^f \leq \left(2 + \frac{k}{20}\right)\varepsilon\right) \leq \frac{1}{10N}$. Since $|Z| = \mathcal{O}\left(\frac{1}{k^5\varepsilon^2} \log \frac{1}{k\varepsilon}\right)$, the result follows. ◀

We will be done with the proof by proving the following claim, that says that bounding χ_{ij}^f is equivalent to bounding ζ_{ij}.

▷ Claim 4.19. For any $i \in [t]$, and any $f_{ij} \in \mathcal{F}_i$, the probability distribution of ζ_{ij} is identical to that of χ_{ij}^f for some DERIVED bipartition with respect to f by f_{ij}.

Proof. Consider a bipartition $f_{ij} \in \mathcal{F}_i$ of X_i, and the bipartition f_{ij}' of $X_i \cup Z$, constructed by extending f_{ij}, as described in the algorithm. For the sake of the argument, let us construct a new bipartition f_{ij}'' of $V(G)$ by extending the bipartition f_{ij}', following the same rule of **Step-2 (ii) (b)** of the algorithm. Observe that $f_{ij}''(v) = f_{ij}(v)$, for each $v \in X_i$. Thus f_{ij}'' is a DERIVED bipartition with respect to some f by f_{ij}. Hence, the claim follows according to the way we generate ζ_{ij}, along with the fact that Z is chosen uniformly at random by the algorithm in **Step-1 (ii)**. ◁

Let us now define a pair (X_i, f_{ij}), with $i \in [t]$ and $f_{ij} \in \mathcal{F}_i$ as a **configuration**. Now we make the following observation which follows directly from the description of the algorithm.

▶ **Observation 4.20.** *Total number of possible configurations is* $N = 2^{\mathcal{O}(\frac{1}{k^3\varepsilon}\log\frac{1}{k\varepsilon})}$.

Note that Claim 4.19 holds for a particular $f_{ij} \in \mathcal{F}_i$. Recall that in **Step-2(iii)**, our algorithm Tol-Bip-Dist(G, ε) reports that G is $(2+k)\varepsilon$-far if $\zeta_{ij} > \left(2 + \frac{k}{20}\right)\varepsilon$, for all $i \in [t]$ and $f_{ij} \in \mathcal{F}_i$. So, using the union bound, along with Observation 4.18, Claim 4.19 and Observation 4.20, we are done with the proof of Theorem 4.17.

5 Conclusion

We believe that our result will certainly improve the current understanding of (tolerant) bipartite testing in the dense graph model. However, one may wonder whether the analysis can be improved to show that the algorithm (presented in Section 3) can decide whether $d_{bip}(G) \leq \varepsilon n^2$ or $d_{bip}(G) \geq c\varepsilon n^2$ for any $c > 1$. There is a bottleneck in our technique as we are bounding error due to the balanced vertices by the sum of degrees of the balanced vertices (as done in Claim 4.12). Because of this reason, it is not obvious if our algorithm (and its analysis) can be used to get a result, like Theorem 2.1, for all $c > 1$ with the same query complexity.

On a different note, we can decide $d_{bip}(G) \leq \varepsilon n^2$ or $d_{bip}(G) \geq (1+k)\varepsilon n^2$ by using $\widetilde{\mathcal{O}}\left((1/k\varepsilon)^6\right)$ queries, which can be derived from the work of Alon, Vega, Kannan and Karpinski [1] (see Corollary A.3 in Appendix A). Hence, any algorithm that solves the general bipartite distance problem with query complexity $o\left((1/k\varepsilon)^6\right)$, will be of huge interest.

References

1 Noga Alon, Wenceslas Fernandez de la Vega, Ravi Kannan, and Marek Karpinski. Random Sampling and Approximation of MAX-CSPs. *Journal of Computer and System Sciences*, 67(2):212–243, 2003. doi:10.1016/S0022-0000(03)00008-4.

2 Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000. doi:10.1007/s004930070001.

3 Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: it's all about regularity. *SIAM Journal on Computing*, 39(1):143–167, 2009. doi:10.1137/060667177.

4 Noga Alon and Michael Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Mathematics*, 15(2):211–227, 2002. doi:10.1137/S0895480199358655.

5 Andrej Bogdanov and Fan Li. A better tester for bipartiteness? *arXiv preprint*, 2010. arXiv:1011.0531.

6 Andrej Bogdanov and Luca Trevisan. Lower Bounds for Testing Bipartiteness in Dense Graphs. In *CCC*, pages 75–81, 2004. doi:10.1109/CCC.2004.1313803.

7 Artur Czumaj, Morteza Monemizadeh, Krzysztof Onak, and Christian Sohler. Planar graphs: Random walks and bipartiteness testing. *Random Structures & Algorithms*, 55(1):104–124, 2019. doi:10.1002/rsa.20826.

8 Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

9 Eldar Fischer and Ilan Newman. Testing versus estimation of graph properties. *SIAM Journal on Computing*, 37(2):482–501, 2007. doi:10.1137/060652324.

10 Arijit Ghosh, Gopinath Mishra, Rahul Raychaudhury, and Sayantan Sen. Tolerant bipartiteness testing in dense graphs. *arXiv preprint*, 2022. arXiv:2204.12397.

11 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

12 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property Testing and its Connection to Learning and Approximation. *Journal of the ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

13 Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999. doi:10.1007/s004930050060.

14 Mira Gonen and Dana Ron. On the benefits of adaptivity in property testing of dense graphs. In *APPROX-RANDOM*, pages 525–539, 2007. doi:10.1007/978-3-540-74208-1_38.

15 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on computing*, 33(6):1441–1483, 2004. doi:10.1137/S0097539703436424.

16 Claire Mathieu and Warren Schudy. Yet Another Algorithm for Dense Max Cut: Go Greedy. In *SODA*, pages 176–182, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347102.

17 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

18 Christian Sohler. Almost Optimal Canonical Property Testers for Satisfiability. In *FOCS*, pages 541–550, 2012. doi:10.1109/FOCS.2012.59.

A Bipartite distance estimation with query complexity $\widetilde{\mathcal{O}}(1/\varepsilon^6)$

Formally, we state the following theorem.

▶ **Theorem A.1.** *Given an unknown graph G on n vertices and any approximation parameter $\varepsilon \in (0,1)$, there is an algorithm that performs $\widetilde{\mathcal{O}}(1/\varepsilon^6)$ adjacency queries, and outputs a number $\widehat{d}_{bip}(G)$ such that, with probability at least $9/10$, the following holds:*

$$d_{bip}(G) - \varepsilon n^2 \leq \widehat{d}_{bip}(G) \leq d_{bip}(G) + \varepsilon n^2,$$

where $d_{bip}(G)$ denotes the bipartite distance of G.

We have the following two corollaries of the above theorem.

▶ **Corollary A.2.** *There exists an algorithm that given adjacency query access to a graph G with n vertices and a parameter $\varepsilon \in (0,1)$ such that, with probability at least $9/10$, decides whether $d_{bip}(G) \leq \varepsilon n^2$ or $d_{bip}(G) \geq (2+\Omega(1))\varepsilon n^2$ using $\widetilde{\mathcal{O}}\left(1/\varepsilon^6\right)$ many queries to the adjacency matrix of G.*

▶ **Corollary A.3.** *There exists an algorithm that given adjacency query access to a graph G with n vertices and a parameter $\varepsilon \in (0,1)$ such that, with probability at least $9/10$, decides whether $d_{bip}(G) \leq \varepsilon n^2$ or $d_{bip}(G) \geq (1+k)\varepsilon n^2$ using $\widetilde{\mathcal{O}}\left((1/k\varepsilon)^6\right)$ many queries to the adjacency matrix of G.*

To prove Theorem A.1, we first discuss the connection between MaxCut and bipartite distance of a graph G. Then we use the result for MaxCut estimation by Alon, Vega, Kannan and Karpinski [1].

Connection between MaxCut and $d_{bip}(G)$

For a graph $G = (V, E)$ on the *vertex* set V and *edge* set E, let S be a subset of V. We define

$$\textsc{Cut}(S) := |\ \{\{u,v\} \in E \ \mid\ |\{u,v\} \cap S| = 1\}\ |$$

Maximum Cut (henceforth termed as MaxCut), denoted by $M(G)$, is a partition of the vertex set V of G into two parts such that the number of edges crossing the partition is maximized, that is,

$$M(G) := \max_{S \subseteq V} \textsc{Cut}(S).$$

The following equation connects MaxCut and the bipartite distance of a graph G:

$$d_{bip}(G) = |E(G)| - M(G). \tag{4}$$

So, $d_{bip}(G)$ can be estimated by estimating $|E(G)|$ and $M(G)$.

Result on edge estimation

Observe that estimating $|E(G)|$ with εn^2 additive error is equivalent to *parameter estimation problem* in probability theory, see Mitzenmacher and Upfal [17, Section 4.2.3].

▶ **Proposition A.4** (Folklore). *Given any graph G on n vertices and an input parameter $\varepsilon \in (0,1)$, the size of the edge set $E(G)$ can be estimated within an additive εn^2 error, with probability at least $9/10$, using $\mathcal{O}(1/\varepsilon^2)$ many adjacency queries to G.*

MaxCut estimation by using $\widetilde{\mathcal{O}}(1/\varepsilon^6)$ queries

Let $G = (V, E)$ be an n vertex graph. Both Alon, Vega, Kannan and Karpinski [1] and Mathieu and Schudy [16] showed that if S is a t-sized random subset of V, where $t = O\left(\frac{1}{\varepsilon^4} \log \frac{1}{\varepsilon}\right)$, then, with probability at least $\frac{9}{10}$, we have the following:

$$\left| \frac{M(G\,|_S)}{t^2} - \frac{M(G)}{n^2} \right| \leq \frac{\varepsilon}{2}$$

where $G\,|_S$ denotes the induced graph of G on the vertex set S. So, the above inequality tells us that if we can get an $\varepsilon t^2/2$ additive error to $M(G\,|_S)$, then we can get an εn^2 additive estimate for $M(G)$. Observation A.5 implies that using $O\left(t/\varepsilon^2\right) = O\left(\frac{1}{\varepsilon^6} \log \frac{1}{\varepsilon}\right)$ many adjacency queries to $G\,|_S$, we can get an $\varepsilon t^2/2$ additive estimate to $M(G\,|_S)$. Therefore, the query complexity of MaxCut algorithms of Alon, Vega, Kannan and Karpinski [1] and Mathieu and Schudy [16] is at most $O\left(\frac{1}{\varepsilon^6} \log \frac{1}{\varepsilon}\right)$.

Now we state and prove the following observation.

▶ **Observation A.5** (Folklore). *For a graph G with n vertices and an approximation parameter $\varepsilon \in (0,1)$, $\Theta\left(n/\varepsilon^2\right)$ many adjacency queries to G are sufficient to get an εn^2 additive approximation to* MaxCut *$M(G)$, with probability at least $9/10$.*

Proof. We sample t many pairs of vertices $\{a_1, b_1\}, \ldots, \{a_t, b_t\}$ uniformly at random and independent of each other, where $t = \Theta(n/\varepsilon^2)$. Thereafter, we perform t many adjacency queries to those sampled pairs of vertices. Now fix a subset $S \subset V(G)$ and let us denote $(S, \overline{S})$ to be the set of edges between S and $\overline{S}$.

Let us now define a set of random variables, one for each sampled pair of vertices as follows:

$$X_i = \begin{cases} 1, & \text{if } \{a_i, b_i\} \in (S, \overline{S}) \\ 0, & \text{Otherwise} \end{cases}$$

We will output $\max_{S \subset V(G)} \widehat{M_S}$ as our estimate of $M(G)$, where $\widehat{M_S} = \frac{\binom{n}{2}}{t} \sum_{i=1}^{t} X_i$.

Let us denote $X = \sum_{i=1}^{t} X_i$. Note that

$$\mathbb{E}[X_i] = \mathbb{P}(X_i = 1) = \frac{\left|(S, \overline{S})\right|}{\binom{n}{2}},$$

and hence

$$\mathbb{E}\left[\widehat{M}_S\right] = \frac{\binom{n}{2}}{t}\mathbb{E}\left[\sum_{i=1}^{t} X_i\right] = \left|(S, \overline{S})\right|.$$

Using Hoeffding's Inequality (see Lemma B.2), we can say that

$$\mathbb{P}\left(\left|\left|(S, \overline{S})\right| - \widehat{M}_S\right| \geq \frac{\varepsilon n^2}{10}\right) \leq \mathbb{P}\left(|X - \mathbb{E}[X]| \geq \frac{\varepsilon t}{10}\right) \leq 2e^{-\Theta(\varepsilon^2 t^2/t)} \leq 2e^{-\Theta(n)}.$$

Using union bound over all $S \subset V(G)$, we can show that with probability at least $3/4$, for each $S \subset V(G)$, $\widehat{M}_S$ approximates $\left|(S, \overline{S})\right|$ with εn^2 additive error. Therefore $\max_{S \subset V(G)} \widehat{M}_S$ estimates $M(G)$ with additive error εn^2, with probability at least $3/4$. ◀

B Large Deviation Inequalities

▶ **Lemma B.1** (Chernoff-Hoeffding bound, see [8]). *Let $X_1, \ldots, X_n$ be independent random variables such that $X_i \in [0, 1]$. For $X = \sum_{i=1}^{n} X_i$ and $\mu_l \leq \mathbb{E}[X] \leq \mu_h$, the followings hold for any $0 < \varepsilon < 1$:*

(i) $\mathbb{P}(X \geq (1+\varepsilon)\mu_h) \leq \exp\left(\frac{-\varepsilon^2 \mu_h}{3}\right)$.

(ii) $\mathbb{P}(X \leq (1-\varepsilon)\mu_l) \leq \exp\left(\frac{-\varepsilon^2 \mu_l}{3}\right)$.

▶ **Lemma B.2** (Hoeffding's Inequality, see [8]). *Let $X_1, \ldots, X_n$ be independent random variables such that $a_i \leq X_i \leq b_i$ and $X = \sum_{i=1}^{n} X_i$. Then, for all $\delta > 0$,*

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq \delta) \leq 2\exp\left(\frac{-2\delta^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right).$$

Homomorphism Tensors and Linear Equations

Martin Grohe
RWTH Aachen University, Germany

Gaurav Rattan
RWTH Aachen University, Germany

Tim Seppelt
RWTH Aachen University, Germany

Abstract

Lovász (1967) showed that two graphs G and H are isomorphic if and only if they are *homomorphism indistinguishable* over the class of all graphs, i.e. for every graph F, the number of homomorphisms from F to G equals the number of homomorphisms from F to H. Recently, homomorphism indistinguishability over restricted classes of graphs such as bounded treewidth, bounded treedepth and planar graphs, has emerged as a surprisingly powerful framework for capturing diverse equivalence relations on graphs arising from logical equivalence and algebraic equation systems.

In this paper, we provide a unified algebraic framework for such results by examining the linear-algebraic and representation-theoretic structure of tensors counting homomorphisms from labelled graphs. The existence of certain linear transformations between such homomorphism tensor subspaces can be interpreted both as homomorphism indistinguishability over a graph class and as feasibility of an equational system. Following this framework, we obtain characterisations of homomorphism indistinguishability over several natural graph classes, namely trees of bounded degree, graphs of bounded pathwidth (answering a question of Dell et al. (2018)), and graphs of bounded treedepth.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases homomorphisms, labelled graphs, treewidth, pathwidth, treedepth, linear equations, Sherali–Adams relaxation, Wiegmann–Specht Theorem, Weisfeiler–Leman

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.70

Category Track A: Algorithms, Complexity and Games

Related Version *Preprint*: `https://arxiv.org/abs/2111.11313` [22]

Funding *Gaurav Rattan*: DFG Research Grants Program RA 3242/1-1 via project number 411032549. *Tim Seppelt*: German Research Council (DFG) within Research Training Group 2236 (UnRAVeL).

Acknowledgements We thank Andrei Bulatov for many fruitful discussions on the foundations of theory developed here and its relation to the algebraic theory of valued constraint satisfaction problems. Moreover, we thank Jan Böker for discussions about linear systems of equations and basal graphs. Finally, we thank the anonymous reviewers for suggestions for improvement.

1 Introduction

Representations in terms of homomorphism counts provide a surprisingly rich view on graphs and their properties. Homomorphism counts have direct connections to logic [14, 17, 27], category theory [12, 34], the graph isomorphism problem [13, 14, 27], algebraic characterisations of graphs [13], and quantum groups [32]. Counting subgraph patterns in graphs has a wide range of applications, for example in graph kernels (see [24]) and motif counting (see [1, 33]). Homomorphism counts can be used as a flexible basis for counting all kinds of substructures [11], and their complexity has been studied in great detail (e.g. [8, 9, 11, 39]). It has been argued in [18] that homomorphism counts are well-suited as

a theoretical foundation for analysing graph embeddings and machine learning techniques on graphs, both indirectly through their connection with graph neural networks via the Weisfeiler–Leman algorithm [14, 35, 46] and directly as features for machine learning on graphs. The latter has also been confirmed experimentally [4, 25, 37].

The starting point of the theory is an old result due to Lovász [27]: two graphs G, H are isomorphic if and only if for every graph F, the number $\hom(F, G)$ of homomorphisms from F to G equals $\hom(F, H)$. For a class $\mathcal{F}$ of graphs, we say that G and H are *homomorphism indistinguishable* over $\mathcal{F}$ if and only if $\hom(F, G) = \hom(F, H)$ for all $F \in \mathcal{F}$. A beautiful picture that has only emerged in the last few years shows that homomorphism indistinguishability over natural graph classes, such as paths, trees, or planar graphs, characterises a variety of natural equivalence relations on graphs.

Broadly speaking, there are two types of such results, the first relating homomorphism indistinguishability to logical equivalence, and the second giving algebraic characterisations of homomorphism equivalence derived from systems of linear (in)equalities for graph isomorphism. Examples of logical characterisations of homomorphism equivalence are the characterisation of homomorphism indistinguishability over graphs of treewidth at most k in terms of the $(k + 1)$-variable fragment of first-order logic with counting [14] and the characterisation of homomorphism indistinguishability over graphs of treedepth at most k in terms of the quantifier-rank-k fragment of first-order logic with counting [17]. Results of this type have also been described in a general category theoretic framework [12, 34]. Examples of equational characterisations are the characterisation of homomorphism indistinguishability over trees in terms of fractional isomorphism [13, 14, 44], which may be viewed as the LP relaxation of a natural ILP for graph isomorphism, and a generalisation to homomorphism indistinguishability over graph of bounded treewidth in terms of the Sherali–Adams hierarchy over that ILP [3, 13, 21, 14, 31]. Further examples include a characterisation of homomorphism indistinguishability over paths in terms of the same system of equalities by dropping the non-negativity constraints of fractional isomorphism [13], and a characterisation of homomorphism indistinguishability over planar graphs in terms of quantum isomorphism [32]. Remarkably, quantum isomorphism is derived from interpreting the same system of linear equations over C*-algebras [2].

1.1 Results

Two questions that remained open in [13] are (1) whether the equational characterisation of homomorphism indistinguishability over paths can be generalised to graphs of bounded pathwidth in a similar way as the characterisation of homomorphism indistinguishability over trees can be generalised to graphs of bounded treewidth, and (2) whether homomorphism indistinguishability over graphs of bounded degree suffices to characterise graphs up to isomorphism. In this paper, we answer the first question affirmatively.

▶ **Theorem 1.** *For every $k \geq 1$, the following are equivalent for two graphs G and H:*

1. *G and H are homomorphism indistinguishable over graphs of pathwidth at most k.*
2. *The $(k+1)$-st level relaxation $\mathsf{L}^{k+1}_{\mathrm{iso}}(G, H)$ of the standard ILP for graph isomorphism has a rational solution.*

The detailed description of the system $\mathsf{L}^{k+1}_{\mathrm{iso}}(G, H)$ is provided in Section 5. In fact, we also devise an alternative system of linear equations $\mathsf{PW}^{k+1}(G, H)$ characterising homomorphism indistinguishability over graphs of pathwidth at most k. The definition of this system turns out to be very natural from the perspective of homomorphism counting, and as we explain later, it forms a fruitful instantiation of a more general representation-theoretic framework for homomorphism indistinguishability.

Moreover, we obtain an equational characterisation of homomorphism indistinguishability over graphs of bounded treedepth. The resulting system $\mathsf{TD}^k(G,H)$ is very similar to $\mathsf{L}^k_{\mathrm{iso}}(G,H)$ and $\mathsf{PW}^k(G,H)$, except that variables are indexed by (ordered) k-tuples of variables rather than sets of at most k variables, which reflects the order induced by the recursive definition of treedepth.

▶ **Theorem 2.** *For every $k \geq 1$, the following are equivalent for two graphs G and H:*

1. *G and H are homomorphism indistinguishable over graphs of treedepth at most k,*
2. *The linear systems of equations $\mathsf{TD}^k(G,H)$ has a non-negative rational solution,*
3. *The linear systems of equations $\mathsf{TD}^k(G,H)$ has a rational solution.*

Along with [17], the above theorem implies that the logical equivalence of two graphs G and H over the quantifier-rank-k fragment of first-order logic with counting can be characterised by the feasibility of the system $\mathsf{TD}^k(G,H)$ of linear equations.

We cannot answer the second open question from [13], but we prove a partial negative result: homomorphism indistinguishability over trees of bounded degree is strictly weaker than homomorphism indistinguishability over all trees.

▶ **Theorem 3.** *For every integer $d \geq 1$, there exist graphs G and H such that G and H are homomorphism indistinguishable over trees of degree at most d, but G and H are not homomorphism indistinguishable over the class of all trees.*

In conjunction with [13], the above theorem yields the following corollary: counting homomorphisms from trees of bounded degree is strictly less powerful than the classical Colour Refinement algorithm [20], in terms of their ability to distinguish non-isomorphic graphs.

To prove these results, we develop a general theory that enables us to derive some of the existing results as well as the new results in a unified algebraic framework exploiting a duality between algebraic varieties of "tensor maps" derived from homomorphism counts over families of rooted graphs and equationally defined equivalence relations, which are based on transformations of graphs in terms of unitary or, more often, pseudo-stochastic or doubly-stochastic matrices. (We call a matrix over the complex numbers *pseudo-stochastic* if its row and column sums are all 1, and we call it *doubly-stochastic* if it is pseudo-stochastic and all its entries are non-negative reals.) The foundations of this theory have been laid in [13] and, mainly, [32]. Some ideas can also be traced back to the work on homomorphism functions and connection matrices [15, 28, 29, 40], and a similar duality, called Galois connection there, that is underlying the algebraic theory of constraint satisfaction problems [6, 7, 42, 47].

1.2 Techniques

To explain our core new ideas, let us start from a simple and well-known result: two symmetric real matrices A, B are co-spectral if and only if for every $k \geq 1$ the matrices A^k and B^k have the same trace. If A, B are the adjacency matrices of two graph G, H, the latter can be phrased graph theoretically as: for every k, G and H have the same number of closed walks of length k, or equivalently, the numbers of homomorphisms from a cycle C_k of length k to G and to H are the same. Thus, G and H are homomorphism indistinguishable over the class of all cycles if and only if they are co-spectral. Note next that the graphs, or their adjacency matrices A, B, are co-spectral if and only if there is a unitary matrix U (or orthogonal matrix, but we need to work over the complex numbers) such that $UA = BU$. Now, in [14, 13] it was proved that G, H are homomorphism indistinguishable over the class of all paths if and only if there is a pseudo-stochastic matrix X such that $XA = BX$, and they are homomorphism indistinguishable over the class of all trees if and only if there is

a doubly-stochastic matrix X such that $XA = BX$. From an algebraic perspective, the transition from a unitary matrix in the cycle result to a pseudo-stochastic in the path result is puzzling: where unitary matrices are very natural, pseudo-stochastic matrices are much less so from an algebraic point of view. Moving on to the tree result, we suddenly add non-negativity constraints – where do they come from? Our theory presented in Section 3 provides a uniform and very transparent explanation for the three results. It also allows us to analyse homomorphism indistinguishability over d-ary trees, for every $d \geq 1$, and to prove that it yields a strict hierarchy of increasingly finer equivalence relations.

Now suppose we want to extend these results to edge coloured graphs. Each edge-coloured graph corresponds to a family of matrices, one for each colour. Theorems due to Specht [41] and Wiegmann [45] characterise families of matrices that are simultaneously equivalent with respect to a unitary transformation. Interpreted over coloured graphs, the criterion provided by these theorems can be interpreted as homomorphism indistinguishability over coloured cycles. One of our main technical contributions is a variant of these theorems that establishes a correspondence between simultaneous equivalence with respect to pseudo-stochastic transformations and homomorphism indistinguishability over coloured paths. The proof is based on basic representation theory, in particular the character theory of semisimple algebras.

Interpreting graphs of bounded pathwidth in a "graph-grammar style" over coloured paths using graphs of bounded size as building blocks, we give an equational characterisation of homomorphism indistinguishability over graphs of pathwidth at most k. After further manipulations, we even obtain a characterisation in terms of a system of equations that are derived by lifting the basic equations for paths in a Sherali–Adams style. (The basic idea of these lifted equations goes back to [3].) This answers the open question from [13] stated above. In the same way, we can lift the characterisations of homomorphism indistinguishability over trees to graphs of treewidth k, and we can also establish a characterisation of homomorphism indistinguishability over graphs of "cyclewidth" k, providing a uniform explanation for all these results. Finally, we combine these techniques to prove a characterisation of homomorphism indistinguishability over graphs of treedepth k in terms of a novel system of linear equations.

2 Preliminaries

We briefly state the necessary definitions and, along the way, introduce our notation. We assume familiarity with elementary definitions from graph theory and linear algebra. As usual, let $\mathbb{N} = \{1, 2, 3, \dots\}$, $[n] = \{1, \dots, n\}$, and $(n) = (1, \dots, n)$. All mentioned graphs are simple, loopless, and undirected.

2.1 Labelled Graphs and Tensor Maps

Labelled and Bilabelled Graphs. For $\ell \in \mathbb{N}$, an *ℓ-labelled* graph $\boldsymbol{F}$ is a tuple $\boldsymbol{F} = (F, \boldsymbol{v})$ where F is a graph and $\boldsymbol{v} \in V(F)^\ell$. The vertices in $\boldsymbol{v}$ are not necessarily distinct, i.e. vertices may have several labels.

The operation of *gluing* two ℓ-labelled graphs $\boldsymbol{F} = (F, \boldsymbol{u})$ and $\boldsymbol{F}' = (F', \boldsymbol{u}')$ yields the ℓ-labelled graph $\boldsymbol{F} \odot \boldsymbol{F}'$ obtained by taking the disjoint union of F and F' and pairwise identifying the vertices $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ to become the i-th labelled vertex, for $i \in [\ell]$, and removing any multiedges in the process. In fact, since we consider homomorphisms into simple graphs, multiedges can always be omitted. Likewise, self-loops can also be disregarded since the number of homomorphisms $F \to G$ where F has a self-loop and G does not is always zero. We henceforth tacitly assume that all graphs are simple.

For $\ell_1, \ell_2 \in \mathbb{N}$, an (ℓ_1, ℓ_2)-*bilabelled graph* $\boldsymbol{F}$ is a tuple $(F, \boldsymbol{u}, \boldsymbol{v})$ for $\boldsymbol{u} \in V(F)^{\ell_1}$, $\boldsymbol{v} \in V(F)^{\ell_2}$. If $\boldsymbol{u} = (u_1, \ldots, u_{\ell_1})$ and $\boldsymbol{v} = (v_1, \ldots, v_{\ell_2})$, it is usual to say that the vertex u_i, resp. v_i, is labelled with the i-th *in-label*, resp. *out-label*.

The *reverse* of an (ℓ_1, ℓ_2)-bilabelled graph $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ is defined to be the (ℓ_2, ℓ_1)-bilabelled graph $\boldsymbol{F}^* = (F, \boldsymbol{v}, \boldsymbol{u})$ with roles of in- and out-labels interchanged. The *concatenation* or *series composition* of an (ℓ_1, ℓ_2)-bilabelled graph $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ and an (ℓ_2, ℓ_3)-bilabelled graph $\boldsymbol{F}' = (F', \boldsymbol{u}', \boldsymbol{v}')$, $\ell_3 \in \mathbb{N}$, denoted by $\boldsymbol{F} \cdot \boldsymbol{F}'$ is the (ℓ_1, ℓ_3)-bilabelled graph obtained by taking the disjoint union of F and F' and identifying for all $i \in [\ell_2]$ the vertices $\boldsymbol{v}_i$ and $\boldsymbol{u}'_i$. The in-labels of $\boldsymbol{F} \cdot \boldsymbol{F}'$ lie on $\boldsymbol{u}$ while its out-labels are positioned on $\boldsymbol{v}'$. The *parallel composition* of (ℓ_1, ℓ_2)-bilabelled graphs $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ and $\boldsymbol{F}' = (F', \boldsymbol{u}', \boldsymbol{v}')$ denoted by $\boldsymbol{F} \odot \boldsymbol{F}'$ is obtained by taking the disjoint union of F and F' and identifying $\boldsymbol{u}_i$ with $\boldsymbol{u}'_i$, and $\boldsymbol{v}_j$ with with $\boldsymbol{v}'_j$ for $i \in [\ell_1]$ and $j \in [\ell_2]$.

Tensors and Tensor Maps. For a set V and $k \in \mathbb{N}$, the set of all functions $X \colon V^k \to \mathbb{C}$ forms a complex vector space denoted by $\mathbb{C}^{V^k}$. We call the elements of $\mathbb{C}^{V^k}$ the k-*dimensional tensors* over V. We identify 0-dimensional tensors with scalars, i.e. $\mathbb{C}^{V^0} = \mathbb{C}$. Furthermore, 1-dimensional tensors are vectors in $\mathbb{C}^V$, 2-dimensional tensors are matrices in $\mathbb{C}^{V \times V}$, et cetera.

A k-*dimensional tensor map on graphs* is a function φ that maps graphs G to k-dimensional tensors $\varphi_G \in \mathbb{C}^{V(G)^k}$. A k-dimensional tensor map φ is *equivariant* if for all isomorphic graphs G and H, all isomorphisms f from G to H, and all $\boldsymbol{v} \in V(G)^k$ it holds that $\varphi_G(\boldsymbol{v}) = \varphi_H(f(\boldsymbol{v}))$.

Homomorphism Tensors and Homomorphism Tensor Maps. For graphs F and G, let $\hom(F, G)$ denote the number of homomorphisms from F to G, i.e. the number of mappings $h \colon V(F) \to V(G)$ such that $v_1 v_2 \in E(F)$ implies $h(v_1)h(v_2) \in E(G)$. For an ℓ-labelled graph $\boldsymbol{F} = (F, \boldsymbol{v})$ and $\boldsymbol{w} \in V(G)^\ell$, let $\hom(\boldsymbol{F}, G, \boldsymbol{w})$ denote the number of homomorphisms h from F to G such that $h(\boldsymbol{v}_i) = \boldsymbol{w}_i$ for all $i \in [\ell]$. Analogously, for an (ℓ_1, ℓ_2)-bilabelled graph $\boldsymbol{F}' = (F', \boldsymbol{u}, \boldsymbol{v})$ and $\boldsymbol{x} \in V(G)^{\ell_1}$, $\boldsymbol{y} \in V(G)^{\ell_2}$, let $\hom(F', G, \boldsymbol{x}, \boldsymbol{y})$ denote the number of homomorphisms $h \colon F' \to G$ such that $h(\boldsymbol{u}_i) = \boldsymbol{x}_i$ and $h(\boldsymbol{v}_j) = \boldsymbol{y}_j$ for all $i \in [\ell_1]$, $j \in [\ell_2]$. More succinctly, we write $\boldsymbol{F}_G \in \mathbb{C}^{V(G)^\ell}$ for the *homomorphism tensor* defined by letting $\boldsymbol{F}_G(\boldsymbol{w}) \coloneqq \hom(\boldsymbol{F}, G, \boldsymbol{w})$ for all $\boldsymbol{w} \in V(G)^\ell$. Similarly, for a bilabelled graph $\boldsymbol{F}'$, $\boldsymbol{F}'_G \in \mathbb{C}^{V(G)^{\ell_1} \times V(G)^{\ell_2}}$ is the matrix defined as $\boldsymbol{F}'_G(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \hom(\boldsymbol{F}, G, \boldsymbol{x}, \boldsymbol{y})$ for all $\boldsymbol{x} \in V(G)^{\ell_1}$, $\boldsymbol{y} \in V(G)^{\ell_2}$.

Letting this construction range over all right-hand side graphs G, the map $G \mapsto \boldsymbol{F}_G$ becomes a tensor map, the *homomorphism tensor map* induced by $\boldsymbol{F}$. It is easy to see that homomorphism tensor maps are equivariant.

Homomorphism tensors give rise to the complex vector spaces of our main interest and their endomorphisms. For a set $\mathcal{R}$ of ℓ-labelled graphs, the tensors $\boldsymbol{R}_G$ for $\boldsymbol{R} \in \mathcal{R}$ span a subspace of $\mathbb{C}^{V(G)^\ell}$, which is denoted by $\mathbb{C}\mathcal{R}_G$. Moreover, the tensors $\boldsymbol{S}_G$ for an (ℓ, ℓ)-bilabelled graph $\boldsymbol{S}$ induces an endomorphisms of $\mathbb{C}^{V(G)^\ell}$.

▶ **Example 4.** For $k \geq 1$, let $\boldsymbol{1}^k$ denote the labelled graph consisting of k isolated vertices with distinct labels $(1, \ldots, k)$. Then, $\boldsymbol{1}^k_G$ is the uniform tensor in $\mathbb{C}^{V(G)^k}$ with every entry equal to 1. Let $\boldsymbol{A}$ denote the $(1,1)$-bilabelled graph $\left(\begin{smallmatrix}1\\ \bullet\end{smallmatrix}\!\!-\!\!\begin{smallmatrix}2\\ \bullet\end{smallmatrix}, (1), (2)\right)$. For every graph G, the matrix $\boldsymbol{A}_G$ is the adjacency matrix of G.

Algebraic and Combinatorial Operations on Homomorphism Tensor Maps. Tensor maps naturally admit a variety of algebraic operations. These include linear combination, complex conjugation, and permutation of coordinates, which are readily defined. Crucially, many operations when applied to homomorphism tensor maps correspond to operations on (bi)labelled graphs. This observation due to [30, 32] is illustrated by the following examples.

- *Sum of Entries = Dropping Labels.* Given a k-labelled graph $\boldsymbol{F} = (F, \boldsymbol{u})$, let $\mathrm{soe}(\boldsymbol{F})$ denote the 0-labelled graph $(F, ())$. Then, for every graph G, $\mathrm{soe}(\boldsymbol{F})_G = \hom(F, G) = \sum_{\boldsymbol{v} \in V(G)^k} \boldsymbol{F}_G(\boldsymbol{v}) =: \mathrm{soe}(\boldsymbol{F}_G)$.
- *Matrix Product = Series Composition.* Let an (ℓ_1, ℓ_2)-bilabelled graph $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ and an (ℓ_2, ℓ_3)-bilabelled graph $\boldsymbol{F}' = (F', \boldsymbol{u}', \boldsymbol{v}')$ be given. Then for every graph G, vertices $\boldsymbol{x} \in V(G)^{\ell_1}$, and $\boldsymbol{y} \in V(G)^{\ell_2}$, $(\boldsymbol{F} \cdot \boldsymbol{F}')_G(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{w} \in V(G)^{\ell_2}} \boldsymbol{F}_G(\boldsymbol{x}, \boldsymbol{w}) \boldsymbol{F}'_G(\boldsymbol{w}, \boldsymbol{y}) =: (\boldsymbol{F}_G \cdot \boldsymbol{F}'_G)(\boldsymbol{x}, \boldsymbol{y})$. A similar operation corresponds to the matrix-vector product, where $\boldsymbol{F}'$ is assumed to be ℓ_2-labelled.
- *Schur Product = Parallel Composition.* The parallel composition $\boldsymbol{F} \odot \boldsymbol{F}'$ of two k-labelled graphs $\boldsymbol{F} = (F, \boldsymbol{u})$ and $\boldsymbol{F}' = (F', \boldsymbol{u}')$ corresponds to the Schur product of the homomorphism tensors. That is, for every graph G and $\boldsymbol{v} \in V(G)^k$, $(\boldsymbol{F} \odot \boldsymbol{F})_G(\boldsymbol{v}) = \boldsymbol{F}_G(\boldsymbol{v}) \boldsymbol{F}'_G(\boldsymbol{v}) =: (\boldsymbol{F}_G \odot \boldsymbol{F}'_G)(\boldsymbol{v})$. Moreover, the *inner-product* of ℓ-labelled graphs $\boldsymbol{F}$, $\boldsymbol{F}'$ can be defined by $\langle \boldsymbol{F}, \boldsymbol{F}' \rangle := \mathrm{soe}(\boldsymbol{F} \odot \boldsymbol{F}')$. It corresponds to the standard inner-product on the tensor space.

2.2 Representation Theory of Involution Monoids

We recall standard notions from representation theory, cf. [26]. A *monoid* Γ is a possibly infinite set equipped with an associative binary operation and an identity element denoted by 1_Γ. An example for a monoid is the *endomorphism monoid* $\operatorname{End} V$ for a vector space V over $\mathbb{C}$ with composition as binary operation and id_V as identity element. A *monoid representation* of Γ is a map $\varphi \colon \Gamma \to \operatorname{End} V$ such that $\varphi(1_\Gamma) = \mathrm{id}_V$ and $\varphi(gh) = \varphi(g)\varphi(h)$ for all $g, h \in \Gamma$. The representation is *finite-dimensional* if V is finite-dimensional. For every monoid Γ, there exists a representation, for example the *trivial representation* $\Gamma \to \operatorname{End}\{0\}$ given by $g \mapsto \mathrm{id}_{\{0\}}$.

Let $\varphi \colon \Gamma \to \operatorname{End}(V)$ and $\psi \colon \Gamma \to \operatorname{End}(W)$ be two representations. Then φ and ψ are *equivalent* if there exists a vector space isomorphism $X \colon V \to W$ such that $X\varphi(g) = \psi(g)X$ for all $g \in \Gamma$. Moreover, φ is a *subrepresentation* of ψ if $V \leq W$ and $\psi(g)$ restricted to V equals $\varphi(g)$ for all $g \in \Gamma$. A representation φ is *simple* if its only subrepresentations are the trivial representation and φ itself. The *direct sum* of φ and ψ denoted by $\varphi \oplus \psi \colon \Gamma \to \operatorname{End}(V \oplus W)$ is the representation that maps $g \in \Gamma$ to $\varphi(g) \oplus \psi(g) \in \operatorname{End}(V) \oplus \operatorname{End}(W) \leq \operatorname{End}(V \oplus W)$. A representation φ is *semisimple* if it is the direct sum of simple representations.

Let $\varphi \colon \Gamma \to \operatorname{End} V$ be a representation with subrepresentations $\psi' \colon \Gamma \to \operatorname{End} V'$ and $\psi'' \colon \Gamma \to \operatorname{End} V''$. Then the restriction of φ to $V' \cap V''$ is a representation as well, called the *intersection of ψ' and ψ''*. For a set $S \subseteq V$, define the *subrepresentation of φ generated by S* as the intersection of all subrepresentations $\psi' \colon \Gamma \to \operatorname{End} V'$ of φ such that $S \subseteq V'$.

The *character* of a representation φ is the map $\chi_\varphi \colon \Gamma \to \mathbb{C}$ defined as $g \mapsto \operatorname{tr}(\varphi(g))$. Its significance stems from the following theorem, which can be traced back to Frobenius and Schur [16]. For a contemporary proof, consult [26] from whose Theorem 7.19 the statement follows.

▶ **Theorem 5** (Frobenius–Schur [16]). *Let Γ be a monoid. Let $\varphi \colon \Gamma \to \operatorname{End}(V)$ and $\psi \colon \Gamma \to \operatorname{End}(W)$ be finite-dimensional semisimple representations. Then φ and ψ are equivalent if and only if $\chi_\varphi = \chi_\psi$.*

The monoids studied in this work are equipped with an additional structure which ensures that their finite-dimensional representations are always semisimple: An *involution monoid* is a monoid Γ with a unary operation $^*\colon \Gamma \to \Gamma$ such that $(gh)^* = h^*g^*$ and $(g^*)^* = g$ for all $g, h \in \Gamma$. Note that $\operatorname{End} V$ is an involution monoid with the adjoint operation $X \mapsto X^*$. Representations of involution monoids must preserve the involution operations. Thereby, they correspond to representations of $*$-algebras.

▶ **Lemma 6.** *Let Γ be an involution monoid. Every finite-dimensional representation of Γ is semisimple.*

Proof. Let $\varphi\colon \Gamma \to \operatorname{End} V$ be a finite-dimensional representation of Γ. It suffices to show that for every subrepresentation $\psi\colon \Gamma \to \operatorname{End} W$ of φ there exists a subrepresentation $\psi'\colon \Gamma \to \operatorname{End} W'$ of φ such that $\varphi = \psi \oplus \psi'$, i.e. φ acts as ψ on W and as ψ' on W'. Set W' to be the orthogonal complement of W in V. It has to be shown that $\varphi(g) \in \operatorname{End} V$ for every $g \in \Gamma$ can be restricted to an endomorphism of W'. Let $w \in W$ and $w' \in W'$ be arbitrary. Then $\langle \varphi(g)w', w\rangle = \langle w', \varphi(g)^*w\rangle = \langle w', \varphi(g^*)w\rangle = 0$ since $\varphi(g^*)$ maps $W \to W$ and $W \perp W'$. Hence, the $\operatorname{im}\varphi(g)$ is contained in the orthogonal complement of W, which equals W'. Clearly, $\varphi = \psi \oplus \psi'$. ◀

2.3 Path and Cycle Decompositions of Bilabelled Graphs

We recall the well-studied notions of path and tree decompositions. For illustrating subsequent arguments, we introduce cycle decompositions.

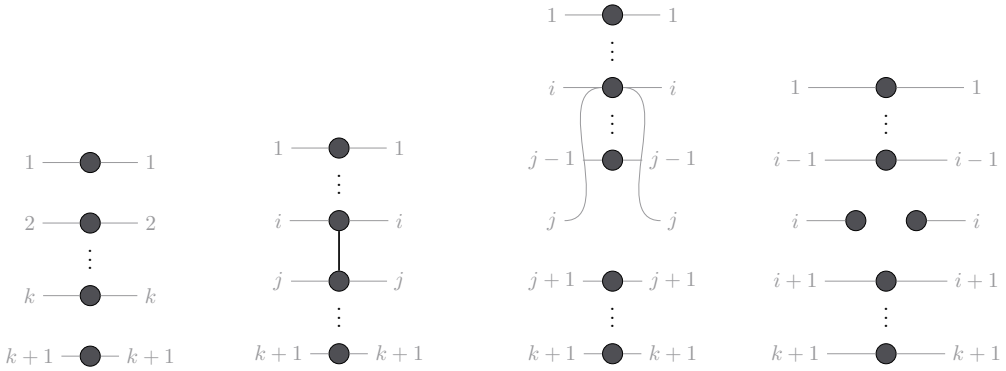
▶ **Definition 7.** *A* decomposition *of a graph G is a pair (F, β) where F is a graph and β is map $V(F) \to 2^{V(G)}$ such that*

1. *the union of the $\beta(v)$ for $v \in V(F)$ is equal to $V(G)$,*
2. *for every edge $e \in E(G)$ there exists $v \in V(F)$ such that $e \subseteq \beta(v)$,*
3. *for every vertex $u \in V(G)$ the set of vertices $v \in V(F)$ such that $u \in \beta(v)$ is connected in F.*

The sets $\beta(v)$ for $v \in V(F)$ are called the *bags of* (F, β). The *width* of (F, β) is the maximum over all $|\beta(v)| + 1$ for $v \in V(F)$. A decomposition (F, β) is called a *tree decomposition* if F is a tree, a *path decomposition* if F is a path, and a *cycle decomposition* if F is a cycle. The *tree- / path- / cyclewidth* of a graph G is the minimum width of a tree/path/cycle decomposition of G.

Let $k \in \mathbb{N}$. A *leaf bag* of a path decomposition (P, β) is a bag $\beta(v)$ such that $v \in V(P)$ has degree 1. A *path decomposition* of a $(k+1, k+1)$-bilabelled graph $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ is a path decomposition (P, β) of the underlying graph F such that the leaf bags consist precisely of the vertices occurring (possibly repeatedly) in $\boldsymbol{u}$ and in $\boldsymbol{v}$, respectively. A $(k+1, k+1)$-bilabelled graph $\boldsymbol{F}$ is said to be of *pathwidth at most k* if its underlying graph admits a path decomposition of width at most k with this property.

Let $\mathcal{PW}^k$ denote the set of all $(k+1, k+1)$-bilabelled graphs of pathwidth at most k. Every unlabelled graph F of pathwidth at most k can be turned into a $(k+1, k+1)$-bilabelled graph $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ of pathwidth at most k by assigning labels to the vertices $\boldsymbol{u}, \boldsymbol{v} \in V(F)^{k+1}$ in the leaf bags. The set $\mathcal{PW}^k$ is closed under concatenation and taking reverses. The *identity graph* $\boldsymbol{I} = (I, (1, \dots, k+1), (1, \dots, k+1))$ with $V(I) = [k+1]$, $E(I) = \emptyset$ is the multiplicative identity under concatenation. Hence, $\mathcal{PW}^k$ forms an involution monoid. A generating set for $\mathcal{PW}^k$ under these operations is called a k-basal set:



(a) Identity graph $\boldsymbol{I}$. **(b)** Adjacency graph $\boldsymbol{A}^{ij}$. **(c)** Identification graph $\boldsymbol{I}^{ij}$. **(d)** Forgetting graph $\boldsymbol{F}^{i}$.

Figure 1 Basal graphs from Lemma 9 in wire notation of [32]: A vertex carries in-label (out-label) i if it is connected to the number i on the left (right) by a wire. Actual edges and vertices of the graph are depicted in black.

▶ **Definition 8.** *A finite set $\mathcal{B}^k$ of $(k+1,k+1)$-bilabelled graphs is called a* k-basal set *if it satisfies the following properties:*

1. $\mathcal{B}^k \subseteq \mathcal{PW}^k$,
2. *the identity graph $\boldsymbol{I}$ is contained in $\mathcal{B}^k$,*
3. *for every $\boldsymbol{B} \in \mathcal{B}$, the reverse graph $\boldsymbol{B}^*$ also belongs to $\mathcal{B}$, and,*
4. *every $\boldsymbol{P} \in \mathcal{PW}^k$ can be obtained by concatenating a sequence of elements from $\mathcal{B}$.*

Concrete examples of k-basal sets can be constructed for every k as described in Lemma 9 and Figure 1. In fact, in all what follows, every k-basal set can be assumed to be this particular k-basal set.

▶ **Lemma 9.** *The set $\mathcal{B}^k$ consisting of the following $(k+1,k+1)$-bilabelled graphs is k-basal. For $1 \leq i \neq j \leq k+1$,*

- *the* identity graph $\boldsymbol{I} = (I, (1,\dots,k+1), (1,\dots,k+1))$ *with* $V(I) = [k+1]$, $E(I) = \emptyset$,
- *the* adjacency graphs $\boldsymbol{A}^{ij} = (A^{ij}, (k+1), (k+1))$ *with* $V(A^{ij}) = [k+1]$ *and* $E(A) = \{ij\}$,
- *the* identification graphs $\boldsymbol{I}^{ij} = (I^{ij}, (1,\dots,i,i+1,\dots,j-1,i,j+1,\dots,k+1), (1,\dots,i,i+1,\dots,j-1,i,j+1,\dots,k+1))$ *with* $V(I^{ij}) = [k+1] \setminus \{j\}$ *and* $E(I^{ij}) = \emptyset$, *and*
- *the* forgetting graphs $\boldsymbol{F}^{i} = (F^i, (1,\dots,k+1), (1,\dots,i-1,i',i+1,\dots,k+1))$ *with* $V(F^i) = [k+1] \cup \{i'\}$ *and* $E(F^i) = \emptyset$.

Proof. Items 1 and 3 of Definition 8 are clear. For Item 4, observe that every $\boldsymbol{P} = (P, \boldsymbol{v}, \boldsymbol{v}) \in \mathcal{PW}^k$ such that all vertices of $\boldsymbol{P}$ are labelled with corresponding in- and out-labels coinciding can be written as the concatenation of $\prod_{ij \in I} \boldsymbol{A}^{ij}$ for $I = E(P)$ with the $\boldsymbol{I}^{ij}$ for all $i \neq j$ such that $\boldsymbol{v}_i = \boldsymbol{v}_j$. Arbitrary $\boldsymbol{Q} \in \mathcal{PW}^k$ can then be obtained as the concatenation of such $\boldsymbol{P}$ interleaved with $\boldsymbol{F}^i$ for certain i. This corresponds to linking adjacent bags of the path decomposition together. ◀

Crucial is the following proposition which is immediate from the above observations:

▶ **Proposition 10.** *Let $\mathcal{B}^k$ denote a k-basal set. If F is a graph of pathwidth at most k then there exist $\boldsymbol{B}^1,\dots,\boldsymbol{B}^r \in \mathcal{B}^k$ such that* $\hom(F,G) = \operatorname{soe}(\boldsymbol{B}^1_G \cdots \boldsymbol{B}^r_G)$ *for all graphs G.*

The constructions for graphs of bounded pathwidth carry over to graphs of bounded cyclewidth (Definition 7). Let $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ be a $(k+1,k+1)$-bilabelled graph of pathwidth at most k. Let $\boldsymbol{F}^{\mathrm{id}}$ denote the $(k+1)$-labelled graph obtained by identifying the elements

of $\boldsymbol{u}$ and $\boldsymbol{v}$ element-wise. Every unlabelled graph C of cyclewidth k can be associated with a $(k+1, k+1)$-bilabelled graph $\boldsymbol{C}$ of pathwidth k such that C is the unlabelled graph underlying $\boldsymbol{C}^{\mathrm{id}}$. Observe that $\mathrm{soe}(\boldsymbol{F}^{\mathrm{id}}) = \mathrm{tr}(\boldsymbol{F})$.

▶ **Proposition 11.** *Let $\mathcal{B}^k$ denote a k-basal set. If F is a graph of cyclewidth at most k then there exist $\boldsymbol{B}^1, \dots, \boldsymbol{B}^r \in \mathcal{B}^k$ such that $\hom(F, G) = \mathrm{tr}(\boldsymbol{B}^1_G \cdots \boldsymbol{B}^r_G)$ for all graphs G.*

3 Homomorphisms from Trees, Paths, and Trees of Bounded Degree

Two graphs G, H with adjacency matrices $\boldsymbol{A}_G$, $\boldsymbol{A}_H$ are isomorphic if and only if there is a matrix X over the non-negative integers such that $X\boldsymbol{A}_G = \boldsymbol{A}_H X$ and $X\mathbf{1} = X^T\mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ is the all-ones vector. Writing the constraints as linear equations whose variables are the entries of X, we obtain a system $\mathsf{F}_{\mathrm{iso}}(G, H)$ that has a non-negative integer solution if and only if G and H are isomorphic. A combination of results from [44] and [14] shows that $\mathsf{F}_{\mathrm{iso}}(G, H)$ has a non-negative rational solution if and only if G and H are homomorphism indistinguishable over the class of trees, and by [13], $\mathsf{F}_{\mathrm{iso}}(G, H)$ has an arbitrary rational solution if and only if G and H are homomorphism indistinguishable over the class of paths. We devise a more general framework connecting homomorphism indistinguishability and $\mathsf{F}_{\mathrm{iso}}(G, H)$-style equations, with paths and trees as two special cases. On the way, we characterise homomorphism indistinguishability over trees of bounded degree.

The prime objects of our study are sets $\mathcal{R}$ of 1-labelled graphs. For a fixed target graph G, the corresponding homomorphism tensors yield a subspace $\mathbb{C}\mathcal{R}_G \leq \mathbb{C}^{V(G)}$. Since algebraic operations on 1-dimensional tensors (i.e. vectors) and combinatorial operations on 1-labelled graphs correspond to each other, cf. Section 2.1, the existence of *linear transformations* $X \colon \mathbb{C}\mathcal{R}_G \to \mathbb{C}\mathcal{R}_H$ respecting these algebraic operations is central to conceptualising homomorphism indistinguishability of G and H as solvability of *linear equations*.

▶ **Definition 12.** *Recall the definition of $\boldsymbol{A}$ from Example 4. Let $\mathcal{R}$ denote a set of 1-labelled graphs containing the one-vertex graph.*

1. *The set $\mathcal{R}$ is* $\boldsymbol{A}$-invariant *if for all $\boldsymbol{R} \in \mathcal{R}$ also $\boldsymbol{A} \cdot \boldsymbol{R} \in \mathcal{R}$. Combinatorially, $\boldsymbol{A}$-invariance means that for every labelled graph $\boldsymbol{R} = (R, u) \in \mathcal{R}$, the labelled graph $\boldsymbol{A} \cdot \boldsymbol{R}$ obtained by adding a fresh vertex u' to R, adding the edge uu', and placing the label on u', is also in $\mathcal{R}$.*
2. *The set $\mathcal{R}$ is* inner-product compatible *if for all $\boldsymbol{R}, \boldsymbol{S} \in \mathcal{R}$ there exists $\boldsymbol{T} \in \mathcal{R}$ such that $\langle \boldsymbol{R}, \boldsymbol{S} \rangle = \mathrm{soe}(\boldsymbol{T})$. Combinatorially, the homomorphism counts from the graph obtained by gluing $\boldsymbol{R}$ and $\boldsymbol{S}$ and forgetting labels, are equal to the homomorphism counts from another graph in $\mathcal{R}$.*

Examples of sets satisfying the above two properties include the set $\mathcal{P}$ of 1-labelled paths where the label is placed on a vertex of degree at most 1, the set $\mathcal{T}$ of 1-labelled trees where the label is placed on an arbitrary vertex, and the set $\mathcal{T}^d$ of 1-labelled d-ary trees with label on a vertex of degree at most 1, where a tree is *d-ary* if its vertices have degree at most $d+1$.

▶ **Theorem 13.** *Let $\mathcal{R}$ be an inner-product compatible set of 1-labelled graphs containing the one-vertex graph. Let G and H be two graphs. Then the following are equivalent:*

1. *G and H are homomorphism indistinguishable over $\mathcal{R}$, that is, for all $\boldsymbol{R} = (R, u) \in \mathcal{R}$, $\hom(R, G) = \hom(R, H)$.*
2. *There exists a unitary[1] map $U \colon \mathbb{C}\mathcal{R}_G \to \mathbb{C}\mathcal{R}_H$ such that $U\boldsymbol{R}_G = \boldsymbol{R}_H$ for all $\boldsymbol{R} \in \mathcal{R}$.*

[1] A map $U \colon V \to W$ is *unitary* if $U^*U = \mathrm{id}_V$ and $UU^* = \mathrm{id}_W$ for $U^* \colon W \to V$ the adjoint of U.

3. *There exists a pseudo-stochastic*[2] *map* $X\colon \mathbb{C}\mathcal{R}_G \to \mathbb{C}\mathcal{R}_H$ *such that* $X\boldsymbol{R}_G = \boldsymbol{R}_H$ *for all* $\boldsymbol{R} \in \mathcal{R}$.

If furthermore $\mathcal{R}$ *is* $\boldsymbol{A}$*-invariant then the conditions above are equivalent to the following:*

4. *There exists a pseudo-stochastic matrix* $X \in \mathbb{Q}^{V(H)\times V(G)}$ *such that* $X\boldsymbol{A}_G = \boldsymbol{A}_H X$ *and* $X\boldsymbol{R}_G = \boldsymbol{R}_H$ *for all* $\boldsymbol{R} \in \mathcal{R}$.

Proof. Suppose that Item 1 holds. Since $\mathcal{R}$ is inner-product compatible, for all $\boldsymbol{R}, \boldsymbol{S} \in \mathcal{R}$ there exists $\boldsymbol{T} = (T, v) \in \mathcal{R}$ such that

$$\langle \boldsymbol{R}_G, \boldsymbol{S}_G \rangle = \langle \boldsymbol{R}, \boldsymbol{S} \rangle_G = (\operatorname{soe} \boldsymbol{T})_G = \hom(T, G) = \hom(T, H) = \langle \boldsymbol{R}_H, \boldsymbol{S}_H \rangle .$$

Thus, by a Gram–Schmidt argument, there exists U with the properties in Item 2. Conversely, supposing that Item 2 holds, let $\boldsymbol{R} = (R, u) \in \mathcal{R}$. It holds that $U\mathbf{1}_G = \mathbf{1}_H$, because $\mathcal{R}$ contains the one-vertex graph $\mathbf{1}$. Since U is unitary, $\mathbf{1}_G = U^*\mathbf{1}_H$. Hence,

$$\hom(R, G) = \langle \mathbf{1}_G, \boldsymbol{R}_G \rangle = \langle \mathbf{1}_H, U\boldsymbol{R}_G \rangle = \langle \mathbf{1}_H, \boldsymbol{R}_H \rangle = \hom(R, H).$$

This shows that Items 1 and 2 are equivalent. Inspecting the above arguments more closely shows that Item 3 is also equivalent with these.

Now suppose additionally that $\mathcal{R}$ is $\boldsymbol{A}$-invariant. It remains to show that in this case Item 4 is equivalent with the first three assertions. For all graphs G, $\mathbb{C}\mathcal{R}_G$ is an $\boldsymbol{A}_G$-invariant subspace of $\mathbb{C}^{V(G)}$. Since $\boldsymbol{A}_G$ is symmetrical, it preserves the direct sum decomposition $\mathbb{C}^{V(G)} = \mathbb{C}\mathcal{R}_G \oplus (\mathbb{C}\mathcal{R}_G)^\perp$. Given U as in Item 2, define X as the map acting like U on $\mathbb{C}\mathcal{R}_G$ and annihilating $(\mathbb{C}\mathcal{R}_G)^\perp$. Let $\boldsymbol{R} \in \mathcal{R}$ be arbitrary. Then $\boldsymbol{A} \cdot \boldsymbol{R} \in \mathcal{R}$ and hence,

$$X\boldsymbol{A}_G\boldsymbol{R}_G = U\boldsymbol{A}_G\boldsymbol{R}_G = \boldsymbol{A}_H\boldsymbol{R}_H = \boldsymbol{A}_H U\boldsymbol{R}_G = \boldsymbol{A}_H X\boldsymbol{R}_G.$$

For $v \in (\mathbb{C}\mathcal{R}_G)^\perp$, $X\boldsymbol{A}_G v = 0 = \boldsymbol{A}_H X v$. Thus, $X\boldsymbol{A}_G = \boldsymbol{A}_H X$.

Finally, $X\mathbf{1}_G = U\mathbf{1}_G = \mathbf{1}_H$ since $\mathcal{R}$ contains the one-vertex graph, and $X^T\mathbf{1}_H = \overline{U^*\mathbf{1}_H} = \overline{\mathbf{1}_G} = \mathbf{1}_G$, so X is pseudo-stochastic. The just constructed matrix X may a priori have non-rational entries. However, since Item 4 is essentially a linear system of equations with rational coefficients, it holds that whenever it has a complex solution, it also has a solution over the rationals. This is a consequence of Cramer's rule. The converse, i.e. that Item 4 implies Item 1, follows analogously to the implication from Item 2 to Item 1. ◀

As an easy application of Theorem 13, we recover the characterisation of indistinguishability with respect to path homomorphisms [13].

▶ **Corollary 14.** *Two graphs* G *and* H *are homomorphism indistinguishable over the class of paths if and only if there exists a pseudo-stochastic* $X \in \mathbb{Q}^{V(H)\times V(G)}$ *such that* $X\boldsymbol{A}_G = \boldsymbol{A}_H X$.

The classical characterisation [43] of homomorphism indistinguishability over trees involves a non-negativity condition on the matrix X. While such an assumption appears natural from the viewpoint of solving the system of equations for fractional isomorphism, it lacks an algebraic or combinatorial interpretation. Using Theorem 13, we reprove this known characterisation and give an alternative description that emphasises its graph theoretic origin.

[2] Let I and J be finite sets. Fix vector spaces $V \leq \mathbb{C}^I$ and $W \leq \mathbb{C}^J$ such that the all-ones vectors $\mathbf{1}_I \in V$ and $\mathbf{1}_J \in W$. Then a map $X\colon V \to W$ is *pseudo-stochastic* if $X\mathbf{1}_I = \mathbf{1}_J$ and $X^*\mathbf{1}_J = \mathbf{1}_I$ for X^* the adjoint of X.

▶ **Corollary 15.** *Let G and H be two graphs. G and H are homomorphism indistinguishable over the class of trees if and only if there exists a pseudo-stochastic matrix $X \in \mathbb{Q}^{V(H)\times V(G)}$ such that $X\boldsymbol{A}_G = \boldsymbol{A}_H X$ and one of the following equivalent conditions holds:*

1. *$X \geq 0$ entry-wise,*
2. *$X\boldsymbol{T}_G = \boldsymbol{T}_H$ for all 1-labelled trees $\boldsymbol{T} \in \mathcal{T}$,*
3. *X preserves the Schur product on $\mathbb{C}\mathcal{T}_G$, i.e. $X(u \odot v) = (Xu) \odot (Xv)$ for all $u, v \in \mathbb{C}\mathcal{T}_G$.*

The key graph-theoretic observation is the following: Every labelled tree can be obtained from the one-vertex graph $\mathbf{1}$ by adding edges and identifying trees at their labels. Put algebraically, the set $\mathcal{T}$ of 1-labelled trees is the closure of $\{\mathbf{1}\}$ under Schur products and multiplication with $\boldsymbol{A}$. Hence, Items 2 and 3 are equivalent. Moreover, Theorem 13 implies the equivalence between Item 2 and homomorphism indistinguishability over the class of trees. The missing equivalence between Items 1 and 2 is deferred to the full version.

Finally, Theorem 13 also gives a characterisation of homomorphism indistinguishability over the class of bounded degree trees. Let $d \geq 1$. The set $\mathcal{T}^d$ of d-ary trees with label on a vertex of degree one or zero is closed under *guarded Schur products*, i.e. the d-ary operation $\circledast^d$ defined as $\circledast^d(\boldsymbol{R}^1, \dots, \boldsymbol{R}^d) \coloneqq \boldsymbol{A} \cdot (\boldsymbol{R}^1 \odot \cdots \odot \boldsymbol{R}^d)$ for $\boldsymbol{R}^1, \dots, \boldsymbol{R}^d \in \mathcal{T}^d$. This operation induces a d-ary operation on $\mathbb{C}\mathcal{T}_G^d$ for every graph G, i.e. $\circledast_G^d(u_1, \dots, u_d) \coloneqq \boldsymbol{A}_G(u_1 \odot \cdots \odot u_d)$ for $u_1, \dots, u_d \in \mathbb{C}\mathcal{T}_G^d$.

▶ **Corollary 16.** *Let $d \geq 1$. Let G and H be graphs. Then the following are equivalent:*

1. *G and H are homomorphism indistinguishable over the class of d-ary trees.*
2. *There exists a pseudo-stochastic matrix $X \in \mathbb{Q}^{V(H)\times V(G)}$ such that $X\boldsymbol{A}_G = \boldsymbol{A}_H X$ and $X\boldsymbol{T}_G = \boldsymbol{T}_H$ for all $\boldsymbol{T} \in \mathcal{T}^d$.*
3. *There exists a pseudo-stochastic matrix $X \in \mathbb{Q}^{V(H)\times V(G)}$ such that X preserves $\circledast^d$ on $\mathbb{C}\mathcal{T}_G^d$, i.e. $X(\circledast_G^d(u_1, \dots, u_d)) = \circledast_H^d(Xu_1, \dots, Xu_d)$ for all $u_1, \dots, u_d \in \mathbb{C}\mathcal{T}_G^d$.*

The systems of equations in Corollary 16 are parametrised by the nested subspaces $\mathbb{C}\mathcal{T}_G^d$ for $d \geq 1$. The following theorem asserts that there exist graphs G for which the inclusions in the chain

$$\mathbb{C}\mathcal{P}_G = \mathbb{C}\mathcal{T}_G^1 \subseteq \mathbb{C}\mathcal{T}_G^2 \subseteq \cdots \subseteq \mathbb{C}\mathcal{T}_G^d \subseteq \mathbb{C}\mathcal{T}_G^{d+1} \subseteq \cdots \subseteq \mathbb{C}\mathcal{T}_G \subseteq \mathbb{C}^{V(G)}$$

are strict. Conceptually, this is due to the fact that $\mathcal{T}^d$ is only closed under the guarded Schur product $\circledast^d$ while $\mathcal{T}$ is closed under arbitrary Schur products.

▶ **Theorem 17.** *For every integer $d \geq 1$, there exists a graph H such that $\mathbb{C}\mathcal{T}_H^d \neq \mathbb{C}\mathcal{T}_H^{d+1}$.*

The proof of the above theorem can be modified to show that homomorphism indistinguishability over trees of bounded degree is a strictly weaker notion than homomorphism indistinguishability over trees. As a consequence, it is not possible to simulate the 1-dimensional Weisfeiler–Leman algorithm (also known as Colour Refinement, [19]) by counting homomorphisms from trees of any fixed bounded degree.

▶ **Theorem 3.** *For every integer $d \geq 1$, there exist graphs G and H such that G and H are homomorphism indistinguishable over trees of degree at most d, but G and H are not homomorphism indistinguishable over the class of all trees.*

The key step underlying the proofs of Theorems 3 and 17 is the construction of graphs whose (adjacency matrix) eigenspaces behave nicely with respect to the Schur product. Both proofs are deferred to the full version.

4 Representations of Involution Monoids and Homomorphism Indistinguishability

Let $\boldsymbol{F}_1, \dots, \boldsymbol{F}_m$ be (ℓ, ℓ)-bilabelled graphs for some $\ell \in \mathbb{N}$. The closure of $\{\boldsymbol{F}_1, \dots, \boldsymbol{F}_m\}$ under concatenation and taking reverses gives rise to an involution monoid $\mathcal{F}$. If a target graph G is fixed, every bilabelled graph $\boldsymbol{F} \in \mathcal{F}$ yields a homomorphism tensor $\boldsymbol{F}_G$. The association $\boldsymbol{F} \mapsto \boldsymbol{F}_G$ is thus a representation[3] of the involution monoid $\mathcal{F}$. This representation-theoretic viewpoint constitutes a compelling framework for analysing homomorphism tensors.

Recall from Section 2.3 that for an (ℓ, ℓ)-bilabelled graph $\boldsymbol{F}$, the ℓ-labelled graph obtained from $\boldsymbol{F}$ by identifying the in- and out-labels pairwise is denoted by $\boldsymbol{F}^{\mathrm{id}}$. Let $\mathcal{F}^{\mathrm{id}}$ denote the set of all unlabelled graphs underlying graphs of the form $\boldsymbol{F}^{\mathrm{id}}$, $\boldsymbol{F} \in \mathcal{F}$. Then, the character χ_G of the representation of $\mathcal{F}$ induced by G tabulates all homomorphism numbers of the form $\hom(F, G)$ for $F \in \mathcal{F}^{\mathrm{id}}$. Given two target graphs G and H, the equality of characters χ_G and χ_H coincides thus with homomorphism indistinguishability over the class $\mathcal{F}^{\mathrm{id}}$.

On the other hand, equality of characters is, under mild representation-theoretic assumptions, a necessary and sufficient condition for two representations to be equivalent. The equivalence of the representation induced by G and H, when explicitly stated, yields a system of linear equations $X\boldsymbol{F}_G = \boldsymbol{F}_H X$ with $\boldsymbol{F} \in \{\boldsymbol{F}_1, \dots, \boldsymbol{F}_m\}$ where the desired solution X is a unitary matrix. This interpretation forms a useful template for homomorphism indistinguishability results: homomorphism indistinguishability over the class $\mathcal{F}^{\mathrm{id}}$ is equivalent to the existence of a unitary matrix satisfying a suitably defined system of linear equations. Indeed, this template yields Theorem 19 below, a characterisation of homomorphism indistinguishability over graphs of bounded cyclewidth, by setting the generators $\boldsymbol{F}_1, \dots, \boldsymbol{F}_m$ to form a k-basal set.

The following theorem about involution monoid representations due to Specht [41], in particular its generalisation due to Wiegmann [45], forms the main tool for obtaining the results of this section. Let $\mathsf{A} = (A_1, \dots, A_m)$ be a sequence of matrices in $\mathbb{C}^{I \times I}$ for some finite index set I. Let Σ_{2m} denote the finite alphabet $\{x_i, y_i \mid i \in [m]\}$. Let Γ_{2m}denote the infinite set of all words over Σ_{2m}. Equipped with the extension to Γ_{2m} of the map swapping x_i and y_i for all $i \in [m]$, Γ_{2m} can be thought of as a *free involution monoid*. For a word $w \in \Gamma_{2m}$, let w_{A} denote the matrix obtained by substituting $x_i \mapsto A_i$ and $y_i \mapsto A_i^*$ for all $i \in [m]$ and evaluating the matrix product. The substitution $w \mapsto w_{\mathsf{A}}$ is a representation of Γ_{2m}.

▶ **Theorem 18** (Specht [41], Wiegmann [45]). *Let I and J be finite index sets. Let $\mathsf{A} = (A_1, \dots, A_m)$ and $\mathsf{B} = (B_1, \dots, B_m)$ be two sequences of matrices such that $A_i \in \mathbb{C}^{I \times I}$ and $B_i \in \mathbb{C}^{J \times J}$ for $i \in [m]$. Then the following are equivalent:*

1. *There exists a unitary $U \in \mathbb{C}^{J \times I}$ such that $UA_i = B_i U$ and $UA_i^* = B_i^* U$ for every $i \in [m]$.*
2. *For every word $w \in \Gamma_{2m}$, $\operatorname{tr}(w_{\mathsf{A}}) = \operatorname{tr}(w_{\mathsf{B}})$.*

Note that the given matrices need not be symmetric. Moreover, it is easy to see that trace equality for words of length at most $\mathcal{O}(n^2)$ suffice to imply trace equality for all words in Γ_{2m}. See [38] for a tighter bound. Finally note that although Theorem 18 is stated as a result involving matrices, the underlying bases are in fact irrelevant.

[3] Phrased in the language of [28], this representation of an involution monoid is a representation of the *concatenation algebra* $\mathcal{F}$.

Our first result follows by applying Wiegmann's theorem to the $(k+1, k+1)$-bilabelled graphs of a k-basal set (Definition 8). This yields an equational characterisation of homomorphism indistinguishability over the class of graphs of cyclewidth at most k.

▶ **Theorem 19.** *Let $k \geq 1$. Let $\mathcal{B}^k$ denote a k-basal set. Let G and H be a graphs. Then the following are equivalent:*

1. *G and H are homomorphism indistinguishable over the class of cyclewidth at most k.*
2. *There exists a unitary matrix $U \in \mathbb{C}^{V(H)^{k+1} \times V(G)^{k+1}}$ such that $U\boldsymbol{B}_G = \boldsymbol{B}_H U$ for all $\boldsymbol{B} \in \mathcal{B}^k$.*

Proof. Let $\boldsymbol{B}^1, \dots, \boldsymbol{B}^m$ be an enumeration of the finite set $\mathcal{B}^k$. Define A and B by setting $A_i \coloneqq \boldsymbol{B}^i_G$ and $B_i \coloneqq \boldsymbol{B}^i_H$ for $i \in [m]$. Recall that the k-basal set $\mathcal{B}^k$ is closed under taking reverses. The theorem immediately follows by an application of Theorem 18 on the matrix sequences A and B, along with Proposition 11. ◀

In contrast to $\mathcal{F}^{\mathrm{id}}$, let $\mathcal{F}^{\mathrm{un}}$ denote the set of all unlabelled graphs underlying graphs $\boldsymbol{F} \in \mathcal{F}$. Although the class $\mathcal{F}^{\mathrm{un}}$ is combinatorially more natural than $\mathcal{F}^{\mathrm{id}}$, it invokes the operator soe on the representations instead of the tr operator, which is algebraically better understood. This technical difficulty is overcome by considering, instead of the original involution monoid representation, its subrepresentation generated by the all-ones vector. In this manner, the *useful spectrum* used in [13] to characterise homomorphism indistinguishability over paths receives an algebraic interpretation. The equivalence of these subrepresentations amounts to the desired solutions being pseudo-stochastic matrices instead of unitary matrices. Formally, we prove the following sum-of-entries analogue of Theorem 18.

▶ **Theorem 20.** *Let I and J be finite index sets. Let $\mathsf{A} = (A_1, \dots, A_m)$ and $\mathsf{B} = (B_1, \dots, B_m)$ be two sequences of matrices such that $A_i \in \mathbb{C}^{I \times I}$ and $B_i \in \mathbb{C}^{J \times J}$ for $i \in [m]$. Then the following are equivalent:*

1. *There exists a pseudo-stochastic matrix $X \in \mathbb{C}^{J \times I}$ such that $XA_i = B_i X$ and $XA_i^* = B_i^* X$ for all $i \in [m]$.*
2. *For every word $w \in \Gamma_{2m}$, $\operatorname{soe}(w_{\mathsf{A}}) = \operatorname{soe}(w_{\mathsf{B}})$.*

Theorem 20 is implied by Lemma 21, which provides a sum-of-entries analogue of Theorem 5. As it establishes a character-theoretic interpretation of the function soe, it may be of independent interest.

▶ **Lemma 21.** *Let Γ be an involution monoid. Let I and J be finite index sets. Let $\varphi\colon \Gamma \to \mathbb{C}^{I \times I}$ and $\varphi\colon \Gamma \to \mathbb{C}^{J \times J}$ be representations of Γ. Let $\varphi'\colon \Gamma \to V$ and $\psi'\colon \Gamma \to W$ denote the subrepresentations of φ and of ψ generated by $\mathbf{1}_I$ and $\mathbf{1}_J$, respectively. Then the following are equivalent:*

1. *For all $g \in \Gamma$, $\operatorname{soe}\varphi(g) = \operatorname{soe}\psi(g)$.*
2. *There exists a unitary pseudo-stochastic $U\colon V \to W$ such that $U\varphi'(g) = \psi'(g)U$ for all $g \in \Gamma$.*
3. *There exists a pseudo-stochastic $X \in \mathbb{C}^{J \times I}$ such that $X\varphi(g) = \psi(g)X$ for all $g \in \Gamma$.*

Proof. Suppose that Item 1 holds. The space V is spanned by the vectors $\varphi(g)\mathbf{1}_I$ for $g \in \Gamma$ while W is spanned by $\psi(g)\mathbf{1}_J$ for $g \in \Gamma$. For $g, h \in \Gamma$ it holds that

$$\langle \varphi(g)\mathbf{1}_I, \varphi(h)\mathbf{1}_I \rangle = \langle \mathbf{1}_I, \varphi(g^*h)\mathbf{1}_I \rangle = \operatorname{soe}\varphi(g^*h) = \operatorname{soe}\psi(g^*h) = \langle \psi(g)\mathbf{1}_J, \psi(h)\mathbf{1}_J \rangle .$$

Hence, V and W are spanned by vectors whose pairwise inner-products are respectively the same. Thus, by a Gram–Schmidt argument, there exists a unitary $U\colon V \to W$ such that $U\varphi(g)\mathbf{1}_I = \psi(g)\mathbf{1}_J$ for all $g \in \Gamma$. This immediately implies that $U\varphi'(g) = \psi'(g)U$ for $g \in \Gamma$. Furthermore, $U\mathbf{1}_I = U\varphi(1_\Gamma)\mathbf{1}_I = \psi(1_\Gamma)\mathbf{1}_J = \mathbf{1}_J$ and $U^*\mathbf{1}_J = \mathbf{1}_I$ since U is unitary. Thus, Item 2 holds.

Suppose now that Item 2 holds. By Lemma 6, write $\varphi = \varphi' \oplus \varphi''$ and $\psi = \psi' \oplus \psi''$. By assumption, there exists a unitary $U\colon V \to W$ such that $U\varphi'(g) = \psi'(g)U$ for all $g \in \Gamma$. Extend U to X by letting it annihilate $V^\perp$. Then $X\varphi(g) = (U \oplus 0)(\varphi' \oplus \varphi'')(g) = U\varphi'(g) \oplus 0 = \psi'(g)U \oplus 0 = \psi(g)X$ for all $g \in \Gamma$. Since U is pseudo-stochastic and $\mathbf{1}_I \in V$ and $\mathbf{1}_J \in W$, X is pseudo-stochastic as well. Hence, Item 3 holds. That Item 3 implies Item 1 is immediate. ◀

Paralleling Theorem 19, we now apply the sum-of-entries version of Specht's theorem to characterise homomorphism indistinguishability over the class of graphs of bounded pathwidth.

▶ **Theorem 22.** *Let $k \geq 1$. Let $\mathcal{B}^k$ denote a k-basal set. Let G and H be a graphs. Then the following are equivalent:*

1. *G and H are homomorphism indistinguishable over the class graphs of pathwidth at most k,*
2. *There exists a pseudo-stochastic matrix $X \in \mathbb{Q}^{V(H)^{k+1} \times V(G)^{k+1}}$ such that $X\boldsymbol{B}_G = \boldsymbol{B}_H X$ for all $\boldsymbol{B} \in \mathcal{B}^k$.*

Let $\mathsf{PW}^{k+1}(G, H)$ denote the system of linear equations in Item 2 above with the basal set from Lemma 9. It comprises $n^{\mathcal{O}(k^2)}$ variables and $\left|\mathcal{B}^k\right| \cdot n^{\mathcal{O}(k^2)} = \mathcal{O}(k^2 \cdot n^{\mathcal{O}(k^2)})$ equations.

5 Comparison with Known Systems of Linear Equations

Towards understanding the power and limitations of convex optimisation approaches to the graph isomorphism problem, the level-k Sherali–Adams relaxation of $\mathsf{F}_{\text{iso}}(G, H)$, denoted by $\mathsf{F}^k_{\text{iso}}(G, H)$, was studied in [3]. The system $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ is another closely related system of interest [21]. Every solution for $\mathsf{F}^{k+1}_{\text{iso}}(G, H)$ yields a solution to $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$, and every solution to $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ yields a solution to $\mathsf{F}^k_{\text{iso}}(G, H)$ [21]. In [3, 21], it was shown that the system $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ has a non-negative solution if and only if G and H are indistinguishable by the k-dimensional Weisfeiler–Leman algorithm. Following the results of [14, 13], the feasibility of $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ is thus equivalent to homomorphism indistinguishability over graphs of treewidth at most k.

Dropping non-negativity constraints in $\mathsf{F}_{\text{iso}}(G, H)$ yields a system of linear equations whose feasibility characterises homomorphism indistinguishability over the class of paths [13]. It was conjectured ibidem that dropping non-negativity constraints in $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ analogously characterises homomorphism indistinguishability over graphs of pathwidth at most k. One direction of this conjecture was shown in [13]: the existence of a rational solution to $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ implies homomorphism indistinguishability over graphs of pathwidth at most k.

We resolve the aforementioned conjecture by showing that the system of equations $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ is feasible if and only if the system of equations $\mathsf{PW}^{k+1}(G, H)$ stated in Theorem 22 is feasible. The proof repeatedly makes use of the observation that the equations in $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$ can be viewed as equations in $\mathsf{PW}^{k+1}(G, H)$ where specific k-basal graphs model the continuity and compatibility equations of $\mathsf{L}^{k+1}_{\text{iso}}(G, H)$. Building on Theorem 22, we obtain the following theorem implying Theorem 1.

▶ **Theorem 23.** *For $k \geq 1$ and graphs G and H, the following are equivalent:*

1. *G and H are homomorphism indistinguishable over the class of graphs of pathwidth at most k.*
2. *The system of equations $\mathsf{PW}^{k+1}(G,H)$ has a rational solution.*
3. *The system of equations $\mathsf{L}^{k+1}_{\mathrm{iso}}(G,H)$ has a rational solution.*

Moreover, we show that $\mathsf{PW}^{k+1}(G,H)$ has a non-negative rational solution if and only if $\mathsf{L}^{k+1}_{\mathrm{iso}}(G,H)$ has a non-negative rational solution. Consequently, the system of linear equations $\mathsf{PW}^{k+1}(G,H)$ has a non-negative rational solution if and only if G and H are homomorphism indistinguishable over graphs of treewidth at most k. Hence, the systems of equations $\mathsf{PW}^{k+1}(G,H)$, for $k \in \mathbb{N}$, form an alternative well-motivated hierarchy of linear programming relaxations of the graph isomorphism problem. The details are deferred to the full version.

6 Multi-Labelled Graphs and Homomorphisms from Graphs of Bounded Treewidth and -depth

By considering k-labelled graphs, we complete the picture emerging in Sections 3 and 4, where respectively 1-labelled and (k,k)-bilabelled graphs were considered. In virtue of a generalisation of Theorem 13, a representation-theoretic characterisation of indistinguishability with respect to the k-dimensional Weisfeiler–Leman algorithm (k-WL, see [19] for its definition) is obtained. Amounting [14] to a characterisation of homomorphism indistinguishability over the class of graphs of treewidth at most k, this goal is achieved by constructing, given a k-WL colouring, a representation-theoretic object, the *colouring algebra*, such that two graphs are not distinguished by k-WL if and only if the associated colouring algebras are isomorphic. It turns out that the well-known algebraic characterisation of 2-WL indistinguishability formulated in the language of coherent algebras [10] is a special case of this correspondence. Finally, a combination of the techniques developed in this article yields an equational characterisation of homomorphism indistinguishability over graphs of bounded treedepth. We set off by generalising Definition 12:

▶ **Definition 24.** *A set of k-labelled graphs $\mathcal{R}$ is* inner-product compatible *if $\mathbf{1}^k \in \mathcal{R}$ and for all $\boldsymbol{R}, \boldsymbol{S} \in \mathcal{R}$ there exists $\boldsymbol{T} \in \mathcal{R}$ such that $\langle \boldsymbol{R}, \boldsymbol{S} \rangle = \operatorname{soe} \boldsymbol{T}$.*

For example, the class $\mathcal{TW}^k$ of k-labelled graphs of treewidth k with all labels in a single bag is inner-product compatible. Another example is the class of 2-labelled planar graphs with labels placed on neighbouring vertices of the boundary of a single face. The following main theorem for k-labelled graphs can be derived analogously to Theorem 13.

▶ **Theorem 25.** *Let $k \geq 1$. Let $\mathcal{R}$ be an inner-product compatible set of k-labelled graphs. Let G and H be two graphs. Then the following are equivalent:*

1. *G and H are homomorphism indistinguishable over $\mathcal{R}$, that is for all $\boldsymbol{R} = (R, \boldsymbol{v}) \in \mathcal{R}$, $\hom(R,G) = \hom(R,H)$.*
2. *There exists a unitary $U \colon \mathbb{C}\mathcal{R}_G \to \mathbb{C}\mathcal{R}_H$ such that $U\boldsymbol{R}_G = \boldsymbol{R}_H$ for all $\boldsymbol{R} \in \mathcal{R}$.*
3. *There exists a pseudo-stochastic $X \in \mathbb{Q}^{V(H)^k \times V(G)^k}$ such that $X\boldsymbol{R}_G = \boldsymbol{R}_H$ for all $\boldsymbol{R} \in \mathcal{R}$.*

It turns out that Theorem 25 yields a characterisation of 2-WL indistinguishability in terms of coherent algebras (see [10, 23]): Given a graph G, let $X = (V(G); R_1, \dots, R_s)$ denote the binary relational structure encoding the 2-WL colouring of G. More precisely, each relation $R_i \subseteq V(G) \times V(G)$ corresponds to one of the 2-WL colour classes of G. The *adjacency algebra* $\mathbb{C}\mathcal{A}_G$ *of* G is the $\mathbb{C}$-span of the matrices A_i with $A_i(u,v) = 1$ iff $(u,v) \in R_i$ and zero otherwise. It follows from the properties of 2-WL [10, Theorem 2.3.6] that $\mathbb{C}\mathcal{A}_G$ is closed

under matrix products, Schur products, and Hermitian conjugations. In other words, it forms a *coherent algebra*. This construction yields the following algebraic characterisation of 2-WL indistinguishability [10, Proposition 2.3.17]: Two graphs G and H are 2-WL indistinguishable if and only if $\mathbb{C}\mathcal{A}_G$ and $\mathbb{C}\mathcal{A}_H$ are *isomorphic as coherent algebras*, i.e. there exists a vector space isomorphism $X\colon \mathbb{C}\mathcal{A}_G \to \mathbb{C}\mathcal{A}_H$ such that X respects the matrix and the Schur product. That is, for all $A, B \in \mathbb{C}\mathcal{A}_G$, $X(A \cdot B) = X(A) \cdot X(B)$ and $X(A \odot B) = X(A) \odot X(B)$.

Along these lines, it may be argued that adjacency algebras as coherent algebras are the adequate algebraic objects to capture 2-WL indistinguishability. For higher-dimensional WL we propose a similar construction: Informally, G and H are k-WL indistinguishable if and only if certain involution monoid representations closed under Schur products are isomorphic. The aforementioned characterisation of 2-WL will be recovered as a special case in Corollary 28.

More precisely, given a graph G with k-ary relational structure $X = (V(G); R_1, \dots, R_s)$ corresponding to its k-WL colouring, define its *k-WL colouring algebra* $\mathbb{C}\mathcal{A}^k_G$ as the $\mathbb{C}$-span of the tensors $A_i \in \mathbb{C}^{V(G)^k}$ with $A_i(\boldsymbol{u}) = 1$ iff $\boldsymbol{u} \in R_i$ and zero otherwise. The colouring algebra has a rich algebraic structure and is closed under various operations. In particular, it has an interpretation in terms of homomorphism tensors: Let $\mathcal{TW}^k$ denote the set of all k-labelled graphs of treewidth at most k where the labelled vertices all lie in the same bag. Furthermore, let $\mathcal{PWS}^k$ denote the set of (k,k)-bilabelled graphs $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ such that F has a path decomposition of width at most k with $\boldsymbol{u}$ and $\boldsymbol{v}$ representing respectively the vertices in the leaf bags.[4] As before, $\mathcal{PWS}^k$ forms an involution monoid under concatenation and taking reverses. These observations are summarised in the following Theorem 26.

▶ **Theorem 26.** *Let G be a graph and let $k \geq 1$. Then*
1. $\mathbb{C}\mathcal{TW}^k_G = \mathbb{C}\mathcal{A}^k_G$,
2. $\mathbb{C}\mathcal{TW}^k_G$ *is closed under Schur products,*
3. *The map $\mathcal{PWS}^k \to \mathrm{End}(\mathbb{C}\mathcal{A}^k_G)$ is a subrepresentation of the involution monoid representation $\mathcal{PWS}^k \to \mathbb{C}^{V(G)^k \times V(G)^k}$ defined as $\boldsymbol{P} \mapsto \boldsymbol{P}_G$.*

The involved proof of Theorem 26 is deferred to the full version. Finally, a representation-theoretic characterisation of k-WL indistinguishability extending [14] can be obtained.

▶ **Theorem 27.** *Let $k \geq 1$. Let G and H be two graphs. Then the following are equivalent:*
1. *G and H are k-WL indistinguishable.*
2. *G and H are homomorphism indistinguishable over the class of graphs of treewidth at most k.*
3. *There exists an isomorphism of $\mathcal{PWS}^k$-representations $X\colon \mathbb{C}\mathcal{A}^k_G \to \mathbb{C}\mathcal{A}^k_H$ respecting the Schur product. That is, for all $A, B \in \mathbb{C}\mathcal{A}^k_G$ and $\boldsymbol{P} \in \mathcal{PWS}^k$, $X(A \odot B) = X(A) \odot X(B)$ and $X(\boldsymbol{P}_G A) = \boldsymbol{P}_H X(A)$.*

To illustrate that the colouring algebra for 2-WL coincides with the coherent algebra, we conclude with inferring Corollary 28 from Theorem 27.

▶ **Corollary 28** (e.g. [10, Proposition 2.3.17])**.** *Let G and H be graphs. Then G and H are 2-WL indistinguishable if and only if there exists a vector space isomorphism $X\colon \mathbb{C}\mathcal{A}_G \to \mathbb{C}\mathcal{A}_H$ such that X respects the matrix and the Schur product. That is, for all $A, B \in \mathbb{C}\mathcal{A}_G$, $X(A \cdot B) = X(A) \cdot X(B)$ and $X(A \odot B) = X(A) \odot X(B)$.*

[4] Observe that this is in contrast to Section 2.3, where the set $\mathcal{PW}^k$ of $(k+1, k+1)$-bilabelled graphs with underlying graphs of pathwidth at most k was considered. There, the labels are carried by vertices in the intersection of two adjacent bags, while here the labelled vertices must only lie in the same bag.

As a final application of our theory, we infer an equational characterisation of homomorphism indistinguishability over graphs of bounded treedepth. The *treedepth* [36] of a graph F is defined as the minimum height of an elimination forest of F, i.e. of a rooted forest T with $V(T) = V(F)$ such that every edge in F connects vertices that are in an ancestor-descendent relationship in T. In [17], it was shown that homomorphism indistinguishable over graphs of treedepth at most k corresponds to equivalence over the quantifier-rank-k fragment of first order logic with counting quantifiers. We extend this characterisation by proposing a linear system of equations very similar to the one for bounded pathwidth.

Let $k \geq 1$. For graphs G and H, consider the following system of equations $\mathsf{TD}^k(G, H)$ with variables $X(\boldsymbol{w}, \boldsymbol{v})$ for every pair of tuples $\boldsymbol{w} \in V(H)^\ell$ and $\boldsymbol{v} \in V(G)^\ell$ for $0 \leq \ell \leq k$. A length-ℓ pair $(\boldsymbol{w}, \boldsymbol{v})$ is said to be a *partial pseudo-isomorphism* if $\boldsymbol{v}_i = \boldsymbol{v}_{i+1} \iff \boldsymbol{w}_i = \boldsymbol{w}_{i+1}$ for all $i \in [\ell - 1]$ and $\{\boldsymbol{v}_i, \boldsymbol{v}_j\} \in E(G) \iff \{\boldsymbol{w}_i, \boldsymbol{w}_j\} \in E(H)$ for all $i, j \in [\ell]$. Note that in contrary to the partial isomorphisms appearing in [13], partial pseudo-isomorphism only need to preserve the equality of consecutive vertices in the domain tuple.

$\mathsf{TD}^k(G, H)$

$$\sum_{v' \in V(G)} X(\boldsymbol{w}w, \boldsymbol{v}v') = X(\boldsymbol{w}, \boldsymbol{v}) \quad \text{for all } w \in V(H) \text{ and } \boldsymbol{v} \in V(G)^\ell,\ \boldsymbol{w} \in V(H)^\ell \text{ where } 0 \leq \ell < k. \tag{TD1}$$

$$\sum_{w' \in V(H)} X(\boldsymbol{w}w', \boldsymbol{v}v) = X(\boldsymbol{w}, \boldsymbol{v}) \quad \text{for all } v \in V(G) \text{ and } \boldsymbol{v} \in V(G)^\ell,\ \boldsymbol{w} \in V(H)^\ell \text{ where } 0 \leq \ell < k. \tag{TD2}$$

$$X((), ()) = 1 \tag{TD3}$$

$$X(\boldsymbol{w}, \boldsymbol{v}) = 0 \quad \text{whenever } (\boldsymbol{w}, \boldsymbol{v}) \text{ is not a partial pseudo-isomorphism} \tag{TD4}$$

The proof of the following theorem is deferred to the full version.

▶ **Theorem 2.** *For every $k \geq 1$, the following are equivalent for two graphs G and H:*

1. *G and H are homomorphism indistinguishable over graphs of treedepth at most k,*
2. *The linear systems of equations $\mathsf{TD}^k(G, H)$ has a non-negative rational solution,*
3. *The linear systems of equations $\mathsf{TD}^k(G, H)$ has a rational solution.*

7 Concluding Remarks

We have developed an algebraic theory of homomorphism indistinguishability that allows us to reprove known results in a unified way and derive new characterisations of homomorphism indistinguishability over bounded degree trees, graphs of bounded treedepth, graphs of bounded cyclewidth, and graphs of bounded pathwidth. The latter answers an open question from [13].

Homomorphism indistinguishabilities over various graph classes can be viewed as similarity measures for graphs, and our new results as well as many previous results show that these are natural and robust. Yet homomorphism indistinguishability only yields equivalence relations, or families of equivalence relations, and not a "quantitative" distance measure. For many applications of graph similarity, such quantitative measures are needed. Interestingly, we can derive distance measure both from homomorphism indistinguishability and from the equational characterisations we study here. For a class $\mathcal{F}$ of graphs, we can consider the *homomorphism embedding* that maps graphs G to the vector in $\mathbb{R}^{\mathcal{F}}$ whose entries are the numbers $\hom(F, G)$ for graphs $F \in \mathcal{F}$. Then a norm on the space $\mathbb{R}^{\mathcal{F}}$ induces a graph (pseudo)metric. Such metrics give a generic family of graph kernels (see [18]). On the

equational side, a notion like fractional isomorphism induces a (pseudo)metric on graphs where the distance between graphs G and H is $\min_X \|X\boldsymbol{A}_G - \boldsymbol{A}_H X\|$, where X ranges over all doubly-stochastic matrices. It is a very interesting question whether the correspondence between the equivalence relations for homomorphism indistinguishability and feasibility of the systems of equations can be extended to the associated metrics. In the special case of isomorphism and homomorphism indistinguishability over all graphs, the theory of graph limits provides some answers [28]. This has recently been extended to fractional isomorphism and homomorphism indistinguishability over trees [5].

References

1 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and Süleyman Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. In *Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, 2008*, pages 241–249, 2008. doi:10.1093/bioinformatics/btn163.

2 Albert Atserias, Laura Mančinska, David E. Roberson, Robert Sámal, Simone Severini, and Antonios Varvitsiotis. Quantum and non-signalling graph isomorphisms. *J. Comb. Theory, Ser. B*, 136:289–328, 2019. doi:10.1016/j.jctb.2018.11.002.

3 Albert Atserias and Elitza N. Maneva. Sherali–Adams Relaxations and Indistinguishability in Counting Logics. *SIAM J. Comput.*, 42(1):112–137, 2013. doi:10.1137/120867834.

4 Paul Beaujean, Florian Sikora, and Florian Yger. Scaling up graph homomorphism for classification via sampling. *ArXiv*, 2104.04040, 2021. doi:10.48550/arXiv.2104.04040.

5 Jan Böker. Graph Similarity and Homomorphism Densities. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 32:1–32:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.32.

6 A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005. doi:10.1137/S0097539700376676.

7 A.A. Bulatov. A Dichotomy Theorem for Nonuniform CSPs. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

8 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34:1–34:41, 2013. doi:10.1145/2528400.

9 Andrei A. Bulatov and Stanislav Zivný. Approximate counting CSP seen from the other side. *ACM Trans. Comput. Theory*, 12(2):11:1–11:19, 2020. doi:10.1145/3389390.

10 Gang Chen and Ilia Ponomarenko. *Lectures on Coherent Configurations*. Central China Normal University Press, Wuhan, 2018. URL: https://www.pdmi.ras.ru/~inp/ccNOTES.pdf.

11 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

12 Anuj Dawar, Tomás Jakl, and Luca Reggio. Lovász-type theorems and game comonads. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470609.

13 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Leman. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.40.

14 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, August 2010. doi:10.1002/jgt.20461.

15 M. Freedman, L. Lovász, and A. Schrijver. Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society*, 20:37–51, 2007. doi:10.1090/S0894-0347-06-00529-7.

16 G. Frobenius and I. Schur. Über die Äquivalenz der Gruppen linearer Substitutionen. *Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu Berlin*, 1:209–217, 1906. URL: https://www.biodiversitylibrary.org/item/93364.

17 Martin Grohe. Counting Bounded Tree Depth Homomorphisms. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 507–520. ACM, 2020. doi:10.1145/3373718.3394739.

18 Martin Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 1–16. ACM, 2020. doi:10.1145/3375395.3387641.

19 Martin Grohe. The Logic of Graph Neural Networks. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–17. IEEE, 2021. doi:10.1109/LICS52264.2021.9470677.

20 Martin Grohe, Kristian Kersting, Martin Mladenov, and Pascal Schweitzer. Color Refinement and Its Applications. In *An Introduction to Lifted Probabilistic Inference*. The MIT Press, August 2021. doi:10.7551/mitpress/10548.003.0023.

21 Martin Grohe and Martin Otto. Pebble Games and Linear Equations. *J. Symb. Log.*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.

22 Martin Grohe, Gaurav Rattan, and Tim Seppelt. Homomorphism tensors and linear equations. *ArXiv*, 2111.11313, 2021. doi:10.48550/ARXIV.2111.11313.

23 Donald G. Higman. Coherent algebras. *Linear Algebra and its Applications*, 93:209–239, August 1987. doi:10.1016/S0024-3795(87)90326-0.

24 Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1):6, 2020. doi:10.1007/s41109-019-0195-3.

25 Pascal Kühner. Graph embeddings based on homomorphism counts. Master's thesis, Department of Computer Science, RWTH Aachen University, 2021.

26 Tsit-Yuen Lam. *A First Course in Noncommutative Rings*. Number 131 in Graduate texts in mathematics. Springer, New York, NY, 2. ed edition, 2001. doi:10.1007/978-1-4419-8616-0.

27 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3):321–328, September 1967. doi:10.1007/BF02280291.

28 László Lovász. *Large Networks and Graph Limits*. American Mathematical Society, 2012. doi:10.1090/coll/060.

29 László Lovász and Alexander Schrijver. Graph parameters and semigroup functions. *Eur. J. Comb.*, 29(4):987–1002, 2008. doi:10.1016/j.ejc.2007.11.008.

30 László Lovász and Balázs Szegedy. Contractors and connectors of graph algebras. *Journal of Graph Theory*, 60(1):11–30, 2009. doi:10.1002/jgt.20343.

31 Peter N. Malkin. Sherali–Adams Relaxations of Graph Isomorphism Polytopes. *Discrete Optimization*, 12:73–97, 2014. doi:10.1016/j.disopt.2014.01.004.

32 Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 661–672. IEEE, 2020. doi:10.1109/FOCS46700.2020.00067.

33 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

34 Yoàv Montacute and Nihil Shah. The pebble-relation comonad in finite model theory. *ArXiv*, 2110.08196, 2021. doi:10.48550/arXiv.2110.08196.

35 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33014602.

36 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: graphs, structures, and algorithms*. Number 28 in Algorithms and combinatorics. Springer, Heidelberg ; New York, 2012. OCLC: ocn773019512. doi:10.1007/978-3-642-27875-4.

37 Hoang Nguyen and Takanori Maehara. Graph homomorphism convolution. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7306–7316. PMLR, 2020. URL: http://proceedings.mlr.press/v119/nguyen20c.html.

38 Christopher J Pappacena. An Upper Bound for the Length of a Finite-Dimensional Algebra. *Journal of Algebra*, 197(2):535–545, 1997. doi:10.1006/jabr.1997.7140.

39 Marc Roth and Philip Wellnitz. Counting and finding homomorphisms is universal for parameterized complexity theory. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2161–2180. SIAM, 2020. doi:10.1137/1.9781611975994.133.

40 Alexander Schrijver. Graph invariants in the spin model. *Journal of Combinatorial Theory, Series B*, 99:502–511, 2009. doi:10.1016/j.jctb.2008.10.003.

41 Wilhelm Specht. Zur Theorie der Matrizen. II. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 50:19–23, 1940. URL: http://gdz.sub.uni-goettingen.de/dms/load/toc/?PPN=PPN37721857X_0050&DMDID=dmdlog6.

42 J. Thapper and S. Živný. The complexity of finite-valued csps. *Journal of the ACM*, 63(4), 2016. doi:10.1145/2974019.

43 Gottfried Tinhofer. Graph isomorphism and theorems of Birkhoff type. *Computing*, 36(4):285–300, December 1986. doi:10.1007/BF02240204.

44 Gottfried Tinhofer. A note on compact graphs. *Discret. Appl. Math.*, 30(2-3):253–264, 1991. doi:10.1016/0166-218X(91)90049-3.

45 N. A. Wiegmann. Necessary and sufficient conditions for unitary similarity. *Journal of the Australian Mathematical Society*, 2(1):122–126, 1961. Edition: 2009/04/09 Publisher: Cambridge University Press. doi:10.1017/S1446788700026422.

46 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

47 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, pages 331–342, 2017. doi:10.1109/FOCS.2017.38.

Downsampling for Testing and Learning in Product Distributions

Nathaniel Harms
University of Waterloo, Canada

Yuichi Yoshida
National Institute of Informatics, Tokyo, Japan

Abstract

We study distribution-free property testing and learning problems where the unknown probability distribution is a product distribution over $\mathbb{R}^d$. For many important classes of functions, such as intersections of halfspaces, polynomial threshold functions, convex sets, and k-alternating functions, the known algorithms either have complexity that depends on the support size of the distribution, or are proven to work only for specific examples of product distributions. We introduce a general method, which we call *downsampling*, that resolves these issues. Downsampling uses a notion of "rectilinear isoperimetry" for product distributions, which further strengthens the connection between isoperimetry, testing and learning. Using this technique, we attain new efficient distribution-free algorithms under product distributions on $\mathbb{R}^d$:

1. A simpler proof for non-adaptive, one-sided monotonicity testing of functions $[n]^d \to \{0,1\}$, and improved sample complexity for testing monotonicity over unknown product distributions, from $O(d^7)$ [Black, Chakrabarty, & Seshadhri, SODA 2020] to $\widetilde{O}(d^3)$.
2. Polynomial-time agnostic learning algorithms for functions of a constant number of halfspaces, and constant-degree polynomial threshold functions;
3. An $\exp(O(d\log(dk)))$-time agnostic learning algorithm, and an $\exp(O(d\log(dk)))$-sample tolerant tester, for functions of k convex sets; and a $2^{\widetilde{O}(d)}$ sample-based one-sided tester for convex sets;
4. An $\exp\left(\widetilde{O}(k\sqrt{d})\right)$-time agnostic learning algorithm for k-alternating functions, and a sample-based tolerant tester with the same complexity.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms; Theory of computation → Machine learning theory; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases property testing, learning, monotonicity, halfspaces, intersections of halfspaces, polynomial threshold functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.71

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2007.07449`

Funding *Nathaniel Harms*: Research funded partly by NSERC. Some of this work was done while the author was visiting NII, Tokyo.
Yuichi Yoshida: Supported in part by JSPS KAKENHI Grant Number 18H05291 and 20H05965.

1 Introduction

In property testing and learning, the goal is to design algorithms that use as little information as possible about the input while still being correct (with high probability). This includes using as little information as possible about the probability distribution against which correctness is measured. Information about the probability distribution could be in the form

of guarantees on this distribution (e.g. it is guaranteed to be uniform, or Gaussian), or in the form of samples from the distribution. So we want to minimize the requirements on this distribution, as well as the number of samples used by the algorithm.

Progress on high-dimensional property testing and learning problems is usually made by studying algorithms for the uniform distribution over the hypercube $\{\pm 1\}^d$, or the standard Gaussian distribution over $\mathbb{R}^d$, as the simplest case. For example, efficiently learning intersections of halfspaces is a major open problem in learning theory [23, 33], and progress on this problem has been made by studying the uniform distribution over the hypercube $\{\pm 1\}^d$ and the Gaussian as special cases [30, 34, 38]. Another important example is the class of degree-k polynomial threshold functions (PTFs). Unlike intersections of halfspaces, these can be efficiently learned in the PAC model [33], but *agnostic* learning is more challenging. Again, progress has been made by studying the hypercube [22]. An even more extreme example is the class of convex sets, which are not learnable in the distribution-free PAC model, because they have infinite VC dimension, but which become learnable under the Gaussian [34]. The uniform distribution over the hypercube and the Gaussian are both examples of *product distributions*, so the next natural question to ask is, can these results be generalized to any *unknown* product distribution? A partial answer was given by Blais, O'Donnell, & Wimmer [10] for some of these classes; in this paper we resolve this question.

Similar examples appear in the property testing literature. Distribution-free property testing and testing functions with domain $\mathbb{R}^d$ are emerging trends in the field (e.g. [2, 21, 29, 19, 26, 9]). Testing monotonicity is one of the most well-studied problems in property testing, and recent work [6] has extended this study to product distributions over domain $\mathbb{R}^d$. Work of Chakrabarty & Seshadhri [17], Khot, Minzer, & Safra [32], and Black, Chakrabarty, & Seshadhri [5, 6] has resulted in efficient $o(d)$-query algorithms for the hypercube $\{\pm 1\}^d$ [32] and the hypergrid $[n]^d$. Black, Chakrabarty, & Seshadhri [6] showed that testing monotonicity over unknown product distributions on $\mathbb{R}^d$ could be done with $\widetilde{O}(d^{5/6})$ queries and $O(d^7)$ samples. Their "domain reduction" method is intricate and specialized for the problem of testing monotonicity. We improve[1] the sample complexity to $\widetilde{O}(d^3)$ using a much simpler proof. We also generalize the testers of [18, 15] for convex sets and k-alternating functions, respectively, and provide new testers for arbitrary functions of convex sets.

This paper provides a general framework for designing distribution-free testing and learning algorithms under product distributions on $\mathbb{R}^d$, which may be finite or continuous. An algorithm is *distribution-free under product distributions* if it does not require any prior knowledge of the probability distribution, except the guarantee that it is a product distribution. The technique in this paper, which we call *downsampling*, improves upon previous methods (in particular, [6, 10]), in a few ways. It is more general and does not apply only to a specific type of algorithm [10] or a specific problem [6], and we use it to obtain many other results. It is conceptually simpler. And it allows quantitative improvements over both [10] and [6].

Organization

This paper is presented as an extended abstract, with the results, techniques, and definitions described in the main text, and most of the proofs given in the full version of the paper. We present our result for testing monotonicity in this extended abstract, as an example application of our techniques. In Section 1.1, we describe the main results of this paper in

[1] An early version of this paper proved a weaker result, with two-sided error and worse sample complexity.

context of the related work. In Section 1.2, we briefly describe the main techniques in the paper. Section 2 presents the definitions and lemmas required by the main results. Section 3 gives the proofs for our results on testing monotonicity. The remaining proofs are in the full version. For simplicity, continuous distributions are treated in the main text and the method for extending the results to finite distributions are handled separately.

1.1 Results

See Table 1 for a summary of our results on property testing, and Table 2 for a summary of our results on learning. Some standard definitions are as follows.

For a set $\mathcal{P}$ of distributions over X and a set $\mathcal{H}$ of functions $X \to \{\pm 1\}$, a *distribution-free* property testing algorithm for $\mathcal{H}$ under $\mathcal{P}$ is a randomized algorithm that is given a parameter $\epsilon > 0$. It has access to the input probability distribution $\mathcal{D} \in \mathcal{P}$ via a *sample oracle*, which returns an independent sample from $\mathcal{D}$. It has access to the input function $f : X \to \{\pm 1\}$ via a *query oracle*, which given query $x \in X$ returns the value $f(x)$. A *two-sided* distribution-free testing algorithm must satisfy:

1. If $f \in \mathcal{H}$ then the algorithm accepts with probability at least $2/3$;
2. If f is ϵ-far from $\mathcal{H}$ with respect to μ then the algorithm rejects with probability at least $2/3$.

A *one-sided* algorithm must accept with probability 1 when $f \in \mathcal{H}$. An (ϵ_1, ϵ_2)-*tolerant* tester must accept with probability at least $2/3$ when $\exists h \in \mathcal{H}$ such that $\mathbb{P}_{x \sim \mu}[f(x) \neq h(x)] \leq \epsilon_1$ and reject when f is ϵ_2-far from $\mathcal{H}$ with respect to μ.

In the *query model*, the queries to the query oracle can be arbitrary. In the *sample model*, the tester queries a point $x \in X$ if and only if x was obtained from the sample oracle. A tester in the query model is *adaptive* if it makes its choice of query based on the answers to previous queries. It is *non-adaptive* if it chooses its full set of queries in advance, before obtaining any of the answers. The *sample complexity* of an algorithm is the number of samples requested from the sample oracle. The *query complexity* of an algorithm is the number of queries made to the query oracle.

Let $\mathcal{H}$ be a set of functions $X \to \{\pm 1\}$ and let $\mathcal{P}$ be a set of probability distributions over X. A learning algorithm for $\mathcal{H}$ under $\mathcal{P}$ (in the *non-agnostic* or *realizable*) model is a randomized algorithm that receives a parameter $\epsilon > 0$ and has *sample access* to an input function $f \in \mathcal{H}$. Sample access means that the algorithm may request an independent random example $(x, f(x))$ where x is sampled from some input distribution $\mathcal{D} \in \mathcal{P}$. The algorithm is required to output a function $g : X \to \{\pm 1\}$ that, with probability $2/3$, satisfies the condition $\mathbb{P}_{x \sim \mathcal{D}}[f(x) \neq g(x)] \leq \epsilon$.

In the *agnostic* setting, the algorithm instead has sample access to an input distribution $\mathcal{D}$ over $X \times \{0, 1\}$ whose marginal over X is in $\mathcal{P}$ (i.e. it receives samples of the form $(x, b) \in X \times \{0, 1\}$). The algorithm is required to output a function $g : X \to \{\pm 1\}$ that, with probability $2/3$, satisfies the following condition: $\forall h \in \mathcal{H}$,

$$\mathbb{P}_{(x,b) \sim \mathcal{D}}[g(x) \neq b] \leq \mathbb{P}_{(x,b) \sim \mathcal{D}}[h(x) \neq b] + \epsilon \,.$$

A *proper* learning algorithm is one whose output must also satisfy $g \in \mathcal{H}$; otherwise it is *improper*.

1.1.1 Testing Monotonicity

Testing monotonicity is the problem of testing whether an unknown function $f : X \to \{0, 1\}$ is monotone, where X is a partial order. It is one of the most commonly studied problems in the field of property testing. Previous work on this problem has mostly focused on uniform

probability distributions (exceptions include [1, 28, 16, 9]) and finite domains. However, there is growing interest in property testing for functions on domain $\mathbb{R}^d$ ([2, 21, 29, 19, 26, 9]) and [6] generalized the problem to this domain.

Testing monotonicity under product distributions has been studied a few times. Ailon & Chazelle [1] gave a distribution-free monotonicity tester for real-valued functions under product distributions on $[n]^d$, with query complexity $O(\frac{1}{\epsilon}d2^d \log n)$. Chakrabarty *et al.* [16] improved this to $O(\frac{1}{\epsilon}d \log n)$ and gave a matching lower bound. This lower bound applies to the *real-valued* case. For the *boolean-valued* case, monotonicity testers under the uniform distribution on $\{\pm 1\}^d$ [17, 32] and $[n]^d$ [5, 6] are known with query complexity $o(d)$. In [6], an $o(d)$-query tester was given for domain $\mathbb{R}^d$. That paper showed that there is a one-sided, non-adaptive, distribution-free monotonicity tester under product distributions on $\mathbb{R}^d$, with query complexity $O\left(\frac{d^{5/6}}{\epsilon^{4/3}} \operatorname{poly}\log(d/\epsilon)\right)$ and sample complexity $O((d/\epsilon)^7)$. In this paper we improve the sample complexity to $\widetilde{O}((d/\epsilon)^3)$, while greatly simplifying the proof.

▶ **Theorem 1.1.** *There is a one-sided, non-adaptive ϵ-tester for monotonicity of functions $\mathbb{R}^d \to \{0,1\}$ that is distribution-free under (finite or continuous) product distributions, using*

$$O\left(\frac{d^{5/6}}{\epsilon^{4/3}} \operatorname{poly}\log(d/\epsilon)\right)$$

queries and $O(\frac{d^3}{\epsilon^3}\log(d/\epsilon))$ samples.

The main result of [6] is a "domain reduction" lemma, which shows that for any function $f : [n]^d \to \{0,1\}$, the distance to monotonicity (under the uniform distribution) is not significantly reduced by sampling a random subgrid S of $[n]^d$ with sides of length $k = O(d^7)$ and restricting f to the domain S. To prove this lemma, [6] develops specialized structural tools for analyzing the "violation graph" of f. The violation graph is a standard object in the study of testing monotonicity. Its vertices are points in the domain, and its edges are "violations of monotonicity": pairs of points $x \prec y$ in the partial order where $f(x) > f(y)$. The distance of f to monotonicity is related to the size of the maximum matching in this graph (due to a result of [25]). The main technical challenge of [6] is to show how to find large matchings in the violation graph under the random restriction to a subgrid, for which they do a "line-by-line analysis" to show how to preserve many of the matched endpoints on each line in the grid. Compared to the technique in our paper, their proof is highly specialized to testing monotone functions, and requires a much more technical analysis. Our result replaces this domain reduction method with a simpler and more general 2-page argument, and gives a different generalization to the distribution-free case. See Section 3 for the proofs.

1.1.2 Learning Functions of Halfspaces

Intersections of k halfspaces have VC dimension $\Theta(dk \log k)$ [14, 20], so the sample complexity of learning is known, but it is not possible to efficiently find k halfspaces whose intersection is correct on the sample, unless $\mathsf{P} = \mathsf{NP}$ [13]. Therefore the goal is to find efficient "improper" algorithms that output a function other than an intersection of k halfspaces. Several learning algorithms for intersections of k halfspaces actually work for arbitrary functions of k halfspaces. We will write $\mathcal{B}_k$ for the set of all functions $\{0,1\}^k \to \{0,1\}$, and for any class $\mathcal{F}$ of functions we will write $\mathcal{B}_k \circ \mathcal{F}$ as the set of all functions $x \mapsto g(f_1(x), \ldots, f_k(x))$ where $g \in \mathcal{B}_k$ and each $f_i \in \mathcal{F}$. Then for $\mathcal{H}$ the class of halfspaces, Klivans, O'Donnell, & Servedio [33] gave a (non-agnostic) learning algorithm for $\mathcal{B}_k \circ \mathcal{H}$ over the uniform distribution on $\{\pm 1\}^d$ with complexity $d^{O(k^2/\epsilon^2)}$, Kalai, Klivans, Mansour, & Servedio [30] presented an agnostic algorithm with complexity $d^{O(k^2/\epsilon^4)}$ in the same setting using "polynomial regression".

Table 1 Testing results.

	$\mathsf{unif}(\{\pm 1\}^d)$	$\mathsf{unif}([n]^d)$	Gaussian	$\forall$ Products
1-Sided Testing Monotonicity (Query model)	$\widetilde{O}\left(\frac{\sqrt{d}}{\epsilon^2}\right)$ [32]	$\widetilde{O}\left(\frac{d^{5/6}}{\epsilon^{4/3}}\right)$ [6]	$\widetilde{O}\left(\frac{d^{5/6}}{\epsilon^{4/3}}\right)$ [6]	$\widetilde{O}\left(\frac{d^{5/6}}{\epsilon^{4/3}}\right)$ queries, $\widetilde{O}\left(\left(\frac{d}{\epsilon}\right)^3\right)$ samples (Thm. 1.1)
1-Sided Testing Convex Sets (Sample model)	–	–	$\left(\frac{d}{\epsilon}\right)^{(1+o(1))d}$ $2^{\Omega(d)}$ [18]	$\left(\frac{d}{\epsilon}\right)^{(1+o(1))d}$ (Thm. 1.4)
Tolerant Testing Functions of k Convex Sets (Sample model)	–	–	–	$\left(\frac{dk}{\epsilon}\right)^{O(d)}$ (Thm. 1.5)
Tolerant Testing k-Alternating Functions (Sample model)	–	$\left(\frac{dk}{\tau}\right)^{O\left(\frac{k\sqrt{d}}{\tau^2}\right)}$ $\tau = \epsilon_2 - 3\epsilon_1$ [15]	–	$\left(\frac{dk}{\tau}\right)^{O\left(\frac{k\sqrt{d}}{\tau^2}\right)}$ $\tau = \epsilon_2 - \epsilon_1$ (Thm. 1.8)

Polynomial regression is a powerful technique, so it is important to understand when it can be applied. Blais, O'Donnell, & Wimmer [10] studied how to generalize it to arbitrary product distributions. With their method, they obtained an agnostic learning algorithm for $\mathcal{B}_k \circ \mathcal{H}$ with complexity $(dn)^{O(k^2/\epsilon^4)}$ for product distributions $X_1 \times \cdots \times X_d$ where each $|X_i| = n$, and complexity $d^{O(k^2/\epsilon^4)}$ for the "polynomially bounded" continuous distributions. This is not a complete generalization, because, for example, on the grid $[n]^d$ its complexity depends on n. This prevents a full generalization to the domain $\mathbb{R}^d$. Their algorithm also requires some prior knowledge of the support or support size. We use a different technique and fully generalize the polynomial regression algorithm to arbitrary product distributions. See the full version for the proof.

▶ **Theorem 1.2.** *There is an improper agnostic learning algorithm for $\mathcal{B}_k \circ \mathcal{H}$, which is distribution-free under (continuous or finite) product distributions over $\mathbb{R}^d$, with time complexity*

$$\min\left\{\left(\frac{dk}{\epsilon}\right)^{O\left(\frac{k^2}{\epsilon^4}\right)}, O\left(\frac{1}{\epsilon^2}\left(\frac{3dk}{\epsilon}\right)^d\right)\right\}.$$

1.1.3 Learning Polynomial Threshold Functions

Degree-k PTFs are another generalization of halfspaces. A function $f : \mathbb{R}^d \to \{\pm 1\}$ is a degree-k PTF if there is a degree-k polynomial $p : \mathbb{R}^d \to \mathbb{R}$ such that $f(x) = \operatorname{sign}(p(x))$. Degree-$k$ PTFs can be PAC learned in time $d^{O(k)}$ using linear programming [33], but agnostic learning is more challenging. Diakonikolas *et al.* [22] previously gave an agnostic learning algorithm for degree-k PTFs in the uniform distribution over $\{\pm 1\}^d$ with time complexity $d^{\psi(k,\epsilon)}$, where

$$\psi(k,\epsilon) := \min\left\{O(\epsilon^{-2^{k+1}}), 2^{O(k^2)}\left(\log(1/\epsilon)/\epsilon^2\right)^{4k+2}\right\}.$$

The main result of that paper is an upper bound on the noise sensitivity of PTFs. Combined with the reduction of [10], this implies an algorithm for the uniform distribution over $[n]^d$ with complexity $(dn)^{\psi(k,\epsilon)}$ and for the Gaussian distribution with complexity $d^{\psi(k,\epsilon)}$.

Table 2 *Learning results. All learning algorithms are agnostic except that of [38]. The PTF result for the Gaussian follows from the two cited works but is not stated in either. All statements are informal, see references for restrictions and qualifications. For PTFs,* $\psi(k,\epsilon) := \min\left\{O(\epsilon^{-2^{k+1}}), 2^{O(k^2)}\left(\log(1/\epsilon)/\epsilon^2\right)^{4k+2}\right\}$.

	$\mathsf{unif}(\{\pm1\}^d)$	$\mathsf{unif}([n]^d)$	Gaussian	$\forall$ Products
Functions of k Convex Sets	$\Omega(2^d)$	–	$d^{O\left(\frac{\sqrt{d}}{\epsilon^4}\right)}, 2^{\Omega(\sqrt{d})}$ [34]	$O\left(\frac{1}{\epsilon^2}\left(\frac{6dk}{\epsilon}\right)^d\right)$ (Thm. 1.6)
Functions of k Halfspaces	$d^{O\left(\frac{k^2}{\epsilon^4}\right)}$ [30]	$(dn)^{O\left(\frac{k^2}{\epsilon^4}\right)}$ [10]	$d^{O\left(\frac{\log k}{\epsilon^4}\right)}$, poly $\left(d, \left(\frac{k}{\epsilon}\right)^k\right)$ [34, 38] (Intersections only)	$\left(\frac{dk}{\epsilon}\right)^{O\left(\frac{k^2}{\epsilon^4}\right)}$ (Thm. 1.2)
Degree-k PTFs	$d^{\psi(k,\epsilon)}$ [22]	$(dn)^{\psi(k,\epsilon)}$ [22, 10]	$d^{\psi(k,\epsilon)}$ [22, 10]	$\left(\frac{dk}{\epsilon}\right)^{\psi(k,\epsilon)}$ (Thm. 1.3)
k-Alternating Functions	$2^{\Theta\left(\frac{k\sqrt{d}}{\epsilon}\right)}$ [8]	$\left(\frac{dk}{\tau}\right)^{O\left(\frac{k\sqrt{d}}{\tau^2}\right)}$ (Testing) [15]	–	$\left(\frac{dk}{\epsilon}\right)^{O\left(\frac{k\sqrt{d}}{\epsilon^2}\right)}$ (Thm. 1.7)

Our agnostic learning algorithm for degree-k PTFs eliminates the dependence on n and works for any unknown product distribution over $\mathbb{R}^n$, while matching the complexity of [22] for the uniform distribution over the hypercube. See the full version for the proof.

▶ **Theorem 1.3.** *There is an improper agnostic learning algorithm for degree-k PTFs, which is distribution-free under (finite or continuous) product distributions over* $\mathbb{R}^d$*, with time complexity*

$$\min\left\{\left(\frac{kd}{\epsilon}\right)^{\psi(k,\epsilon)}, O\left(\frac{1}{\epsilon^2}\left(\frac{9dk}{\epsilon}\right)^d\right)\right\}.$$

1.1.4 Testing & Learning Convex Sets

One of the first properties (sets) of functions $\mathbb{R}^d \to \{0,1\}$ to be studied in the property testing literature is the set of indicator functions of convex sets, i.e. functions $f : \mathbb{R}^d \to \{0,1\}$ where $f^{-1}(1)$ is convex. Write $\mathcal{C}$ for this class of functions. This problem has been studied in various models of testing [36, 35, 18, 4, 7]. In this paper we consider the *sample-based* model of testing, where the tester receives only random examples of the function and cannot make queries. This model of testing has received a lot of recent attention (e.g. [2, 4, 12, 18, 27, 29, 37, 9]), partly because it matches the standard sample-based model for learning algorithms.

Chen *et al.* [18] gave a sample-based tester for $\mathcal{C}$ under the Gaussian distribution on $\mathbb{R}^d$ with one-sided error and sample complexity $(d/\epsilon)^{O(d)}$, along with a lower bound (for one-sided testers) of $2^{\Omega(d)}$. We match their upper bound while generalizing the tester to be distribution-free under product distributions. See the full version for proofs.

▶ **Theorem 1.4.** *There is a sample-based one-sided* ϵ*-tester for* $\mathcal{C}$ *which is distribution-free under (finite or continuous) product distributions that uses at most* $O\left(\left(\frac{6d}{\epsilon}\right)^d\right)$ *samples.*

A more powerful kind of tester is an (ϵ_1, ϵ_2)-*tolerant* tester, which must accept (with high probability) any function that is ϵ_1-close to the property, while rejecting functions that are ϵ_2-far. Tolerantly testing convex sets has been studied by [3] for the uniform distribution

over the 2-dimensional grid, but not (to the best of our knowledge) in higher dimensions. We obtain a sample-based tolerant tester (and distance) approximator for convex sets in high dimension. In fact, recall that $\mathcal{B}_k$ is the set of all functions $\{0,1\}^k \to \{0,1\}$ and $\mathcal{B}' \subset \mathcal{B}_k$ any subset, so $\mathcal{B}' \circ \mathcal{C}$ is any property of functions of convex sets. We obtain a distance approximator for any such property:

▶ **Theorem 1.5.** *Let $\mathcal{B}' \subset \mathcal{B}_k$. There is a sample-based algorithm, which is distribution-free under (finite or continuous) product distributions, that approximates distance to $\mathcal{B}' \circ \mathcal{C}$ up to additive error ϵ using $O\left(\frac{1}{\epsilon^2}\left(\frac{3dk}{\epsilon}\right)^d\right)$ samples. Setting $\epsilon = (\epsilon_2 - \epsilon_1)/2$ we obtain an (ϵ_1, ϵ_2)-tolerant tester with sample complexity $O\left(\frac{1}{(\epsilon_2-\epsilon_1)^2}\left(\frac{6dk}{\epsilon_2-\epsilon_1}\right)^d\right)$.*

General distribution-free learning of convex sets is not possible, since this class has infinite VC dimension. However, they can be learned under the Gaussian distribution. Non-agnostic learning under the Gaussian was studied by Vempala [38, 39]. Agnostic learning under the Gaussian was studied by Klivans, O'Donnell, & Servedio [34] who presented a learning algorithm with complexity $d^{O(\sqrt{d}/\epsilon^4)}$, and a lower bound of $2^{\Omega(\sqrt{d})}$.

Unlike the Gaussian, there is a trivial lower bound of $\Omega(2^d)$ in arbitrary product distributions, because any function $f : \{\pm 1\}^d \to \{0,1\}$ belongs to this class. However, unlike the general distribution-free case, we show that convex sets (or any functions of convex sets) can be learned under unknown product distributions.

▶ **Theorem 1.6.** *There is an agnostic learning algorithm for $\mathcal{B}_k \circ \mathcal{C}$, which is distribution-free under (finite or continuous) product distributions over $\mathbb{R}^d$, with time complexity $O\left(\frac{1}{\epsilon^2} \cdot \left(\frac{6dk}{\epsilon}\right)^d\right)$.*

1.1.5 Testing & Learning k-Alternating Functions

A k-alternating function $f : X \to \{\pm 1\}$ on a partial order X is one where for any chain $x_1 < \cdots < x_m$ in X, f changes value at most k times. Learning k-alternating functions on domain $\{\pm 1\}^d$ was studied by Blais *et al.* [8], motivated by the fact that these functions are computed by circuits with few negation gates. They show that $2^{\Theta(k\sqrt{d}/\epsilon)}$ samples are necessary and sufficient in this setting. Canonne *et al.* [15] later obtained an algorithm for (ϵ_1, ϵ_2)-tolerant testing k-alternating functions, when $\epsilon_2 > 3\epsilon_1$, in the uniform distribution over $[n]^d$, with query complexity $(kd/\tau)^{O(k\sqrt{d}/\tau^2)}$, where $\tau = \epsilon_2 - 3\epsilon_1$.

We obtain an agnostic learning algorithm for k-alternating functions that matches the query complexity of the tester in [15], and nearly matches the complexity of the (non-agnostic) learning algorithm of [8] for the uniform distribution over the hypercube. See the full version for proofs.

▶ **Theorem 1.7.** *There is an agnostic learning algorithm for k-alternating functions, which is distribution-free under (finite or continuous) product distributions over $\mathbb{R}^d$, that runs in time at most*

$$\min\left\{\left(\frac{dk}{\epsilon}\right)^{O\left(\frac{k\sqrt{d}}{\epsilon^2}\right)}, O\left(\frac{1}{\epsilon^2}\left(\frac{3kd}{\epsilon}\right)^d\right)\right\}.$$

We also generalize the tolerant tester of [15] to be distribution-free under product distributions, and eliminate the condition $\epsilon_2 > 3\epsilon_1$.

▶ **Theorem 1.8.** *For any $\epsilon_2 > \epsilon_1 > 0$, let $\tau = (\epsilon_2 - \epsilon_1)/2$, there is a sample-based (ϵ_1, ϵ_2)-tolerant tester for k-alternating functions using $\left(\frac{dk}{\tau}\right)^{O\left(\frac{k\sqrt{d}}{\tau^2}\right)}$ samples, which is distribution-free under (finite or continuous) product distributions over $\mathbb{R}^d$.*

1.2 Techniques

What connects these diverse problems is a notion of rectilinear surface area or isoperimetry that we call "block boundary size". There is a close connection between learning & testing and various notions of isoperimetry or surface area (e.g. [17, 33, 34, 32]). We show that testing or learning a class $\mathcal{H}$ on product distributions over $\mathbb{R}^d$ can be reduced to testing and learning on the *uniform* distribution over $[r]^d$, where r is determined by the block boundary size, and we call this reduction "downsampling". The name *downsampling* is used in image and signal processing for the process of reducing the resolution of an image or reducing the number of discrete samples used to represent an analog signal. We adopt the name because our method can be described by analogy to image or signal processing as the following 2-step process:

1. Construct a "digitized" or "pixellated" image of the function $f : \mathbb{R}^d \to \{\pm 1\}$ by sampling from the distribution and constructing a grid in which each cell has roughly equal probability mass; and
2. Learn or test the "low-resolution" pixellated function.

As long as the function f takes a constant value in the vast majority of "pixels", the low resolution version seen by the algorithm is a good enough approximation for testing or learning. The block boundary size is, informally, the number of pixels on which f is not constant.

This technique reduces distribution-free testing and learning problems to the uniform distribution in a way that is conceptually simpler than in the prior work [10, 6]. However, some technical challenges remain. The first is that it is not always easy to bound the number of "pixels" on which a function f is not constant – for example, for PTFs. Second, unlike in the uniform distribution, the resulting downsampled function class on $[r]^d$ is not necessarily "the same" as the original class – for example, halfspaces on $\mathbb{R}^d$ are not downsampled to halfspaces on $[r]^d$, since the "pixels" are not of equal size. Thus, geometric arguments may not work, unlike the case for actual images.

A similar technique of constructing "low-resolution" representations of the input has been used and rediscovered ad-hoc a few times in the property testing literature, but only for the uniform distribution over $[n]^d$ [31, 36, 24, 12, 15], or the Gaussian in [18]. With this paper, we aim to provide a unified and generalized study of this simple and powerful technique.

1.3 Block Boundary Size

Informally, we define the *r-block boundary size* $\mathsf{bbs}(\mathcal{H}, r)$ of a class $\mathcal{H}$ of functions $\mathbb{R}^d \to \{0, 1\}$ as the maximum number of grid cells on which a function $f \in \mathcal{H}$ is non-constant, over all possible $r \times \cdots \times r$ grid partitions of $\mathbb{R}^d$ (which are not necessarily evenly spaced) – see Section 2 for formal definitions. Whether downsampling can be applied to $\mathcal{H}$ depends on whether

$$\lim_{r \to \infty} \frac{\mathsf{bbs}(\mathcal{H}, r)}{r^d} \to 0\,,$$

and the complexity of the algorithms depends on how large r must be for the non-constant blocks to vanish relative to the whole r^d grid. A general observation is that any function class $\mathcal{H}$ where downsampling can be applied can be learned under unknown product distributions with a finite number of samples; for example, this holds for convex sets even though the VC dimension is infinite.

▶ **Proposition 1.9.** *Let $\mathcal{H}$ be any set of functions $\mathbb{R}^d \to \{0,1\}$ (measurable with respect to continuous product distributions) such that*

$$\lim_{r\to\infty} \frac{\mathsf{bbs}(\mathcal{H}, r)}{r^d} = 0\,.$$

Then there is some function $\sigma(d,\epsilon)$ such that $\mathcal{H}$ is distribution-free learnable under product distributions, up to error ϵ, with $\sigma(d,\epsilon)$ samples.

For convex sets, monotone functions, k-alternating functions, and halfspaces, $\mathsf{bbs}(\mathcal{H}, r)$ is easy to calculate. For degree-k PTFs, it is more challenging – it requires proving a bound on the number of unevenly-spaced grid cells in $\mathbb{R}^d$ in which a degree-k multivariate polynomial might take the value 0; this result may be of independent interest.

We obtain this result by proving a more general lemma. We say that a function $f : \mathbb{R}^d \to \{0,1\}$ induces a connected component S if for every $x, y \in S$ there is a continuous curve in $\mathbb{R}^d$ from x to y such that $f(z) = f(x) = f(y)$ for all z on the curve, and S is a maximal such set. Then we prove a general lemma that bounds the block boundary size by the number of connected components induced by functions $f \in \mathcal{H}$.

▶ **Lemma 1.10** (Informal, see full version). *Suppose that for any axis-aligned affine subspace A of affine dimension $n \le d$, and any function $f \in \mathcal{H}$, f induces at most k^n connected components in A. Then for $r = \Omega(dk/\epsilon)$, $\mathsf{bbs}(\mathcal{H}, r) \le \epsilon \cdot r^d$.*

This lemma in fact generalizes all computations of block boundary size in this paper (up to constant factors in r). Using a theorem of Warren [40], we get the following corollary:

▶ **Corollary 1.11** (Informal, see full version). *Let $p : \mathbb{R}^d \to \mathbb{R}$ be a degree-k polynomial, and let $\epsilon > 0$. For $r \ge 3\sqrt{24}dk/\epsilon$ and any $r \times \cdots \times r$ grid partition of $\mathbb{R}^d$, p takes value 0 in at most ϵr^d grid cells.*

1.4 Polynomial Regression

The second step of downsampling is to find a testing or learning algorithm that works for the uniform distribution over the (not necessarily evenly-spaced) hypergrid. Most of our learning results use *polynomial regression*. This is a powerful technique introduced in [30] that performs linear regression over a vector space of functions that approximately spans the hypothesis class. This method is usually applied by using Fourier analysis to construct such an approximate basis for the hypothesis class [10, 22, 15]. This was the method used, for example, by Blais, O'Donnell, & Wimmer [10] to achieve the poly(dn)-time algorithms for intersections of halfspaces.

We take the same approach but we use the Walsh basis for functions on domain $[n]^d$ (see e.g. [11]) instead of the bases used in the prior works. We show that if one can establish bounds on the noise sensitivity in the Fourier basis for the hypothesis class restricted to the uniform distribution over $\{\pm 1\}^d$, then one gets a bound on the number of Walsh functions required to approximately span the "downsampled" hypothesis class. In this way, we establish that if one can apply standard Fourier-analytic techniques to the hypothesis class over the *uniform* distribution on $\{\pm 1\}^d$ and calculate the block boundary size, then the results for the hypercube essentially carry over to product distributions on $\mathbb{R}^d$.

An advantage of this technique is that both noise sensitivity and block boundary size grow at most linearly during function composition: for functions $f(x) = g(h_1(x), \ldots, h_k(x))$ where each h_i belongs to the class $\mathcal{H}$, the noise sensitivity and block boundary size grow at most linearly in k. Therefore learning results for $\mathcal{H}$ obtained in this way are easy to extend to arbitrary compositions of $\mathcal{H}$, which is how we get our result for intersections of halfspaces.

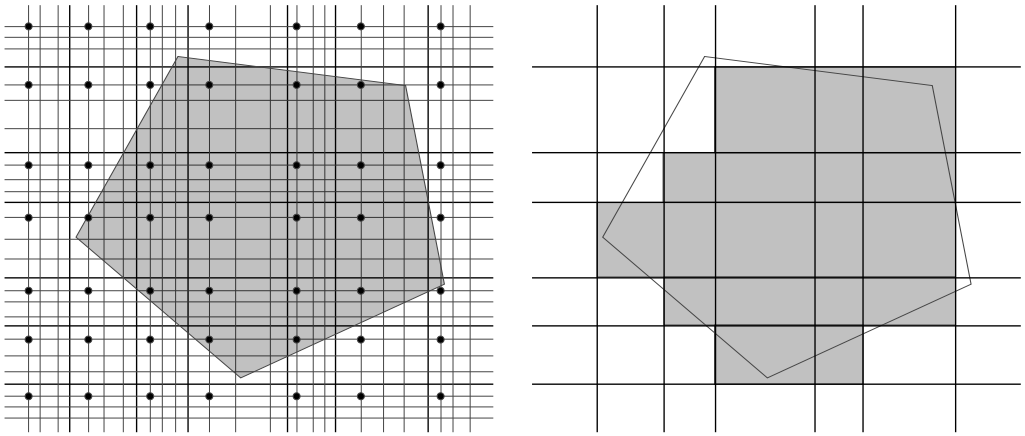


Figure 1 Left: Random grid X (pale lines) with induced block partition (thick lines) and blockpoint values (dots), superimposed on $f^{-1}(1)$ (gray polygon). Right: f^{coarse} (grey) compared to f (polygon outline).

2 Downsampling

We will now introduce the main definitions, notation, and lemmas required by our main results. The purpose of this section is to establish the main conceptual component of the downsampling technique: that functions with small enough block boundary size can be efficiently well-approximated by a "coarsened" version of the function that is obtained by random sampling. See Figure 1 for an illustration of the following definitions.

▶ **Definition 2.1** (Block Partitions). *An* r-block partition *of* $\mathbb{R}^d$ *is a pair of functions* $\mathsf{block} : \mathbb{R}^d \to [r]^d$ *and* $\mathsf{blockpoint} : [r]^d \to \mathbb{R}^d$ *obtained as follows. For each* $i \in [d], j \in [r-1]$ *let* $a_{i,j} \in \mathbb{R}$ *such that* $a_{i,j} < a_{i,j+1}$ *and define* $a_{i,0} = -\infty, a_{i,r} = \infty$ *for each* i*. For each* $i \in [d], j \in [r]$ *define the interval* $B_{i,j} = (a_{i,j-1}, a_{i,j}]$ *and a point* $b_{i,j} \in B_{i,j}$*. The function* $\mathsf{block} : \mathbb{R}^d \to [r]^d$ *is defined by setting* $\mathsf{block}(x)$ *to be the unique vector* $v \in [r]^d$ *such that* $x_i \in B_{i,v_i}$ *for each* $i \in [d]$*. The function* $\mathsf{blockpoint} : [r]^d \to \mathbb{R}^d$ *is defined by setting* $\mathsf{blockpoint}(v) = (b_{1,v_1}, \ldots, b_{d,v_d})$*; note that* $\mathsf{blockpoint}(v) \in \mathsf{block}^{-1}(v)$ *where* $\mathsf{block}^{-1}(v) = \{x \in \mathbb{R}^d : \mathsf{block}(x) = v\}$.

▶ **Definition 2.2** (Block Functions and Coarse Functions). *For a function* $f : \mathbb{R}^d \to \{\pm 1\}$*, we define* $f^{\mathsf{block}} : [r]^d \to \{\pm 1\}$ *as* $f^{\mathsf{block}} := f \circ \mathsf{blockpoint}$ *and* $f^{\mathsf{coarse}} : \mathbb{R}^d \to \mathbb{R}$ *as* $f^{\mathsf{coarse}} := f^{\mathsf{block}} \circ \mathsf{block} = f \circ \mathsf{blockpoint} \circ \mathsf{block}$*. For any set* $\mathcal{H}$ *of functions* $\mathbb{R}^d \to \{\pm 1\}$*, we define* $\mathcal{H}^{\mathsf{block}} := \{f^{\mathsf{block}} \mid f \in \mathcal{H}\}$*. For a distribution* μ *over* $\mathbb{R}^d$ *and an* r*-block partition* $\mathsf{block} : \mathbb{R}^d \to [r]^d$ *we define the distribution* $\mathsf{block}(\mu)$ *over* $[r]^d$ *as the distribution of* $\mathsf{block}(x)$ *for* $x \sim \mu$.

▶ **Definition 2.3** (Induced Block Partitions). *When* μ *is a product distribution over* $\mathbb{R}^d$*, a* random grid X *of length* m *is the grid obtained by sampling* m *points* $x_1, \ldots, x_m \in \mathbb{R}^d$ *independently from* μ *and for each* $i \in [d], j \in [m]$ *defining* $X_{i,j}$ *to be the* j^{th}*-smallest coordinate in dimension* i *among all sampled points. For any* r *that divides* m *we define an* r*-block partition depending on* X *by defining for each* $i \in [d], j \in [r-1]$ *the point* $a_{i,j} = X_{i,mj/r}$ *so that the intervals are* $B_{i,j} := (X_{i,m(j-1)/r}, X_{i,mj/r}]$ *when* $j \in \{2, \ldots, r-1\}$ *and* $B_{i,1} = (-\infty, X_{i,m/r}], B_{i,r} = (X_{i,m(r-1)/r}, \infty)$*; we let the points* $b_{i,j}$ *defining* blockpoint *be arbitrary. This is the* r*-block partition* induced *by* X.

▶ **Definition 2.4** (Block Boundary Size). *For a block partition* $\mathsf{block} : \mathbb{R}^d \to [r]^d$*, a distribution* μ *over* $\mathbb{R}^d$*, and a function* $f : \mathbb{R}^d \to \{\pm 1\}$*, we say* f *is* non-constant *on a block* $v \in [r]^d$ *if there are sets* $S, T \subset \mathsf{block}^{-1}(v)$ *such that* $\forall s \in S, t \in T : f(s) = 1, f(t) = -1$*; and* S, T *have positive measure (in the product of Lebesgue measures). For a function* $f : \mathbb{R}^d \to \{\pm 1\}$ *and a number* r*, we define the* r*-block boundary size* $\mathsf{bbs}(f, r)$ *as the maximum number of blocks on which* f *is non-constant, where the maximum is taken over all* r*-block partitions* $\mathsf{block} : \mathbb{R}^d \to [r]^d$*. For a set* $\mathcal{H}$ *of functions* $\mathbb{R}^d \to \{\pm 1\}$*, we define* $\mathsf{bbs}(\mathcal{H}, r) := \max\{\mathsf{bbs}(f, r) \mid f \in \mathcal{H}\}$.

The *total variation distance* between two distributions μ, ν over a finite domain $\mathcal{X}$ is defined as

$$\|\mu - \nu\|_{\mathsf{TV}} := \frac{1}{2} \sum_{x \in \mathcal{X}} |\mu(x) - \nu(x)| = \max_{S \subseteq \mathcal{X}} |\mu(S) - \nu(S)| .$$

The essence of downsampling is apparent in the next proposition. It shows that the distance of f to its coarsened version f^{coarse} is bounded by two quantities: the fraction of blocks in the r-block partition on which f is not constant, and the distance of the distribution $\mathsf{block}(\mu)$ to uniform. When both quantities are small, testing or learning f can be done by testing or learning f^{coarse} instead. The uniform distribution over a set S is denoted $\mathsf{unif}(S)$:

▶ **Proposition 2.5.** *Let* μ *be a continuous product distribution over* $\mathbb{R}^d$*, let* X *be a random grid, and let* $\mathsf{block} : \mathbb{R}^d \to [r]^d$ *be the induced* r*-block partition. Then, for any measurable* $f : \mathbb{R}^d \to \{\pm 1\}$*, the following holds with probability 1 over the choice of* X*:*

$$\mathop{\mathbb{P}}_{x \sim \mu} [f(x) \neq f^{\mathsf{coarse}}(x)] \leq r^{-d} \cdot \mathsf{bbs}(f, r) + \|\mathsf{block}(\mu) - \mathsf{unif}([r]^d)\|_{\mathsf{TV}} .$$

Proof. We first establish that, with probability 1 over X and $x \sim \mu$, if $f(x) \neq f^{\mathsf{coarse}}(x)$ then f is non-constant on $\mathsf{block}(x)$. Fix X and suppose there exists a set Z of positive measure such that for each $x \in Z, f(x) \neq f^{\mathsf{coarse}}(x)$ but f is not non-constant on $\mathsf{block}(x)$, i.e. for $V = \mathsf{block}^{-1}(\mathsf{block}(x))$, either $\mu(V \cap f^{-1}(1)) = \mu(V)$ or $\mu(V \cap f^{-1}(-1)) = \mu(V)$. Then there is $v \in [r]^d$ such that for $V = \mathsf{block}^{-1}(v)$, $\mu(Z \cap V) > 0$. Let $y = \mathsf{blockpoint}(v)$. If $\mu(V \cap f^{-1}(f(y)) = \mu(V)$ then $\mu(Z \cap V) = 0$, so $\mu(V \cap f^{-1}(f(y)) = 0$. But for random X, the probability that there exists $v \in [r]^d$ such that $\mu(V \cap f^{-1}(\mathsf{blockpoint}(v))) = 0$ is 0, since $\mathsf{blockpoint}(v)$ is random within V.

Assuming that the above event occurs,

$$\begin{aligned}\mathop{\mathbb{P}}_{x \sim \mu} [f(x) \neq f^{\mathsf{coarse}}(x)] &\leq \mathop{\mathbb{P}}_{x \sim \mu} [f \text{ is non-constant on } \mathsf{block}(x)] \\ &\leq \mathop{\mathbb{P}}_{v \sim [r]^d} [f \text{ is non-constant on } v] + \|\mathsf{block}(\mu) - \mathsf{unif}([r]^d)\|_{\mathsf{TV}} .\end{aligned}$$

Since $v \sim [r]^d$ is uniform, the probability of hitting a non-constant block is at most $r^{-d} \cdot \mathsf{bbs}(f, r)$. ◀

Next we give a bound on the number of samples required to ensure that $\mathsf{block}(\mu)$ is close to uniform. We need the following lemma.

▶ **Lemma 2.6.** *Let* μ *be continuous probability distribution over* $\mathbb{R}$*,* $m, r \in \mathbb{N}$ *such that* r *divides* m*, and* $\delta \in (0, 1/2)$*. Let* X *be a set of* m *points sampled independently from* μ*. Write* $X = \{x_1, \ldots, x_m\}$ *labeled such that* $x_1 < \cdots < x_m$ *(and write* $x_0 = -\infty$*). Then for any* $i \in [r]$*,*

$$\mathbb{P}\left[\mu\left(x_{(i-1)(m/r)}, x_{i(m/r)}\right] < \frac{1-\delta}{r}\right] \leq 4 \cdot e^{-\frac{\delta^2 m}{32r}} .$$

Proof. We assume that $i-1 \le r/2$. If $i-1 > r/2$ then we can repeat the following analysis with the opposite ordering on the points in X. Write $x^* = x_{(i-1)\frac{m}{r}}$ and $\beta = \mu(-\infty, x^*]$. First suppose that $(1-\delta/2)\frac{i-1}{r} < \beta < (1+\delta/2)\frac{i-1}{r} \le (1+\delta/2)/2$; we will bound the probability of this event later.

Let $t \in \mathbb{R}$ be the point such that $\mu(x^*, t] = (1-\delta)/r$ (which must exist since μ is continuous). Let $\eta = \frac{\delta}{1-\delta} \ge \delta$. Write $X^* = \{x \in X : x > x^*\}$. The expected value of $|X^* \cap (x^*, t]|$ is $|X^*|\frac{1-\delta}{r(1-\beta)} = \left(1 - \frac{i-1}{r}\right)\frac{1-\delta}{r(1-\beta)}$, where the factor $1-\beta$ in the denominator is due to the fact that each element of X^* is sampled from μ conditional on being larger than x^*. The event $\mu(x^*, x_{i(m/r)}] < (1-\delta)/r$ occurs if and only if $|X^* \cap (x^*, t]| > m/r$, which occurs with probability

$$\mathbb{P}\left[|X^* \cap (x^*, t]| > \frac{m}{r}\right] = \mathbb{P}\left[|X^* \cap (x^*, t]| > m\left(1 - \frac{(i-1)}{r}\right)\frac{1-\delta}{r(1-\beta)}(1+\eta)\right]$$

where

$$\begin{aligned} 1+\eta = \frac{(1-\beta)}{(1-\delta)\left(1-\frac{i-1}{r}\right)} &\ge \frac{\left(1-(1+\delta/2)\frac{i-1}{r}\right)}{(1-\delta)\left(1-\frac{i-1}{r}\right)} = \frac{1}{1-\delta}\left(1 - \frac{(\delta/2)(i-1)}{r-(i-1)}\right) \\ &\ge \frac{1-\delta/2}{1-\delta} = 1 + \frac{\delta}{2(1-\delta)} \ge 1+\delta/2\,. \end{aligned}$$

Since the expected value satisfies

$$|X^*|\frac{1-\delta}{r(1-\beta)} \ge \frac{m}{r}(1-\frac{i-1}{r})\frac{2(1-\delta)}{1-\delta/2} \ge \frac{m}{r}(1-\delta/2) \ge \frac{m}{2r}\,,$$

the Chernoff bound gives

$$\mathbb{P}\left[|X^* \cap (x^*, t]| > \frac{m}{r}\right] \le \exp\left(-\frac{\delta^2|X^*|(1-\delta)}{3\cdot 4\cdot r(1-\beta)}\right) \le e^{-\frac{\delta^2 m}{3\cdot 4\cdot 2r}}\,.$$

Now let $t \in \mathbb{R}$ be the point such that $\mu(x^*, t] = (1+\delta)/r$. The expected value of $|X^* \cap (x^*, t]|$ is now $|X^*|\frac{1+\delta}{r(1-\beta)}$. The event $\mu(x^*, x_{i(m/r)}] > (1+\delta)/r$ occurs if and only if $|X^* \cap (x^*, t]| < m/r$, which occurs with probability

$$\mathbb{P}\left[|X^* \cap (x^*, t]| < \frac{m}{r}\right] = \mathbb{P}\left[|X^* \cap (x^*, t]| < m\left(1 - \frac{i-1}{r}\right)\frac{1+\delta}{r(1-\beta)}(1-\eta)\right]$$

where

$$\begin{aligned} 1-\eta = \frac{1-\beta}{(1+\delta)(1-\frac{i-1}{r})} &\le \frac{1-(1+\delta/2)\frac{i-1}{r}}{(1+\delta)\left(1-\frac{i-1}{r}\right)} = \frac{1}{1+\delta}\left(1 + \frac{(\delta/2)(i-1)}{r-(i-1)}\right) \\ &\le \frac{1+\delta/2}{1+\delta} = 1 - \frac{\delta/2}{1+\delta} \le 1 - \frac{\delta}{4}\,. \end{aligned}$$

The expected value satisfies $|X^*|\frac{1+\delta}{r(1-\beta)} > m/r$, so the Chernoff bound gives

$$\mathbb{P}\left[|X^* \cap (x^*, t]| < \frac{m}{r}\right] \le \exp\left(-\frac{\delta^2|X^*|(1+\delta)}{2\cdot 4^2\cdot r(1-\beta)}\right) \le e^{-\frac{\delta^2 m}{2\cdot 4^2}}\,.$$

It remains to bound the probability that $(1-\delta/2)\frac{i-1}{r} < \beta < (1+\delta/2)\frac{i-1}{r}$. Define $t \in \mathbb{R}$ such that $\mu(-\infty, t] = (1+\delta/2)\frac{i-1}{r}$. $\beta = \mu(-\infty, x^*] \ge (1+\delta/2)\frac{i-1}{r}$ if and only if $x^* > t$, i.e. $|X \cap (-\infty, t]| < \frac{i-1}{r}$. The expected value of $|X \cap (-\infty, t]|$ is $m\frac{(1+\delta/2)(i-1)}{r}$, so for $\eta = \frac{\delta/2}{1+\delta/2} \ge \delta/3$, the Chernoff bound implies

$$\begin{aligned} \mathbb{P}\left[|X \cap (-\infty, t]| < m\frac{i-1}{r}\right] &= \mathbb{P}\left[|X \cap (-\infty, t]| < m\frac{(1+\delta/2)(i-1)}{r}(1-\eta)\right] \\ &\le e^{-\frac{\delta^2 m(1+\delta/2)(i-1)}{18r}} \le e^{-\frac{\delta^2 m}{18r}}\,. \end{aligned}$$

Now define $t \in \mathbb{R}$ such that $\mu(-\infty, t] = (1-\delta/2)\frac{i-1}{r}$. $\beta = \mu(-\infty, x^*] \leq (1-\delta/2)\frac{i-1}{r}$ if and only if $x^* < t$, i.e. $|X \cap (-\infty, t]| > \frac{i-1}{r}$. The expected value of $|X \cap (-\infty, t]|$ is $m\frac{(1-\delta/2)(i-1)}{r}$, so for $\eta = \frac{\delta}{2-\delta} \geq \delta/2$,

$$\mathbb{P}\left[|X \cap (-\infty, t]| > m\frac{i-1}{r}\right] = \mathbb{P}\left[|X \cap (-\infty, t]| > m\frac{(1-\delta/2)(i-1)}{r}(1+\eta)\right]$$
$$\leq e^{-\frac{\delta^2 m(1-\delta/2)(i-1)}{2\cdot 4r}} \leq e^{-\frac{\delta^2 m}{4^2 r}}.$$

The conclusion then follows from the union bound over these four events. ◀

▶ **Lemma 2.7.** *Let $\mu = \mu_1 \times \cdots \times \mu_d$ be a product distribution over $\mathbb{R}^d$ where each μ_i is continuous. Let X be a random grid with length m sampled from μ, and let $\mathsf{block} : \mathbb{R}^d \to [r]^d$ be the r-block partition induced by X. Then*

$$\mathop{\mathbb{P}}_{X}\left[\|\mathsf{block}(\mu) - \mathsf{unif}([r]^d)\|_{\mathsf{TV}} > \epsilon\right] \leq 4rd \cdot e^{-\frac{\epsilon^2 m}{18rd^2}}$$

Proof. For a fixed grid X and each $i \in [d]$, write $p_i : [r] \to [0,1]$ be the probability distribution on $[r]$ with $p_i(z) = \mu_i(B_{i,z})$. Then $\mathsf{block}(\mu) = p_1 \times \cdots \times p_d$.

Let $\delta = \frac{4\epsilon}{3d}$. Suppose that for every $i, j \in [d] \times [r]$ it holds that $\frac{1+\delta}{r} \leq p_i(j) \geq \frac{1-\delta}{r}$. Note that $d\delta = \frac{4\epsilon}{3} \leq \ln(1+2\epsilon) \leq 2\epsilon$. Then for every $v \in [r]^d$,

$$\mathop{\mathbb{P}}_{u\sim\mu}[\mathsf{block}(u) = v] = \prod_{i=1}^{d} p_i(v_i) \begin{cases} \leq (1+\delta)^d r^{-d} \leq e^{d\delta} r^{-d} \leq (1+2\epsilon) r^{-d} \\ \geq (1-\delta)^d r^{-d} \geq (1-d\delta) r^{-d} \geq (1-2\epsilon) r^{-d}. \end{cases}$$

So

$$\|\mathsf{block}(\mu) - \mathsf{unif}([r]^d)\|_{\mathsf{TV}} = \frac{1}{2}\sum_{v\in[r]^d} |\mathop{\mathbb{P}}_{u\sim\mu}[\mathsf{block}(u) = v] - r^{-d}| \leq \frac{1}{2}\sum_{v\in[r]^d} 2\epsilon r^{-d} = \epsilon.$$

By Lemma 2.6 and the union bound, the probability that there is some $i \in [d], j \in [r]$ that satisfies $p_i(j) < (1-\delta)/r$ is at most $4rd \cdot e^{-\frac{\epsilon^2 m}{18rd^2}}$. ◀

3 Testing Monotonicity

3.1 Testing Monotonicity on the Hypergrid

A good introduction to downsampling is the following short proof of the main result of Black, Chakrabarty, & Seshadhri [6]. In an earlier work, [5], they gave an $O((d^{5/6}/\epsilon^{4/3}) \operatorname{poly}\log(dn))$ tester for the domain $[n]^d$, and in the later work they showed how to reduce the domain $[n]^d$ to $[r]^d$ for $r = \operatorname{poly}(d/\epsilon)$.

Our monotonicity tester will use as a subroutine the following tester for *diagonal* functions. For a hypergrid $[n]^d$, a *diagonal* is a subset of points $\{x \in [n]^d : x = v + \lambda\vec{1}, \lambda \in \mathbb{Z}\}$ defined by some $v \in [n]^d$. A function $f : [n]^d \to \{0,1\}$ is a *diagonal function* if it has at most one 1-valued point in each diagonal.

▶ **Lemma 3.1.** *There is an ϵ-tester with one-sided error and query complexity $O\left(\frac{1}{\epsilon}\log^2(1/\epsilon)\right)$ for diagonal functions on $[n]^d$.*

Proof. For each $t \in [n]$ let D_t be the set of diagonals with length t. For any $x \in [n]^d$ let $\mathrm{diag}(x)$ be the unique diagonal that contains x. For input $f : [n]^d \to \{0,1\}$ and any $x \in [n]^d$, let $R(x) = \frac{|\{y \in \mathrm{diag}(x) : f(y)=1\}|}{|\mathrm{diag}(x)|}$.

Suppose that f is ϵ-far from diagonal. Then f must have at least ϵn^d 1-valued points; otherwise we could set each 1-valued point to 0 to obtain the constant 0 function. Now observe

$$\begin{aligned}\mathop{\mathbb{E}}_{x\sim[n]^d}[R(x)] &= \mathop{\mathbb{E}}_{x\sim[n]^d}\left[\sum_{t=1}^{n}\sum_{L\in D_t}\mathbf{1}\left[\mathrm{diag}(x)=L\right]\frac{|\{y\in L : f(y)=1\}|}{t}\right]\\ &= \sum_{t=1}^{n}\sum_{L\in D_t}\mathop{\mathbb{P}}_{x\sim[n]^d}[x\in L]\frac{|\{y\in L : f(y)=1\}|}{t} = \sum_{t=1}^{n}\sum_{L\in D_t}\frac{t}{n^d}\frac{|\{y\in L : f(y)=1\}|}{t}\\ &= \frac{1}{n^d}|\{y\in[n]^d : f(y)=1\}| \geq \epsilon\,.\end{aligned}$$

For each i, define $A_i = \left\{x \in [n]^d : \frac{1}{2^i} < R(x) \leq \frac{1}{2^{i-1}}\right\}$. Let $k = \log(4/\epsilon)$. Then

$$\begin{aligned}\epsilon \leq \mathbb{E}\left[R(x)\right] &\leq \sum_{i=1}^{\infty}\frac{|A_i|}{n^d}\max_{x\in A_i}R(x) \leq \sum_{i=1}^{\infty}\frac{|A_i|}{n^d 2^{i-1}} \leq \sum_{i=1}^{k}\frac{|A_i|}{n^d 2^{i-1}} + \sum_{i=k+1}^{\infty}\frac{1}{2^{i-1}}\\ &\leq \sum_{i=1}^{k}\frac{|A_i|}{n^d 2^{i-1}} + \frac{1}{2^{k-1}} \leq \sum_{i=1}^{k}\frac{|A_i|}{n^d 2^{i-1}} + \frac{\epsilon}{2}\\ \implies \frac{\epsilon}{2} &\leq \sum_{i=1}^{k}\frac{|A_i|}{n^d 2^{i-1}}\,.\end{aligned}$$

Therefore there is some $\ell \in [k]$ such that $|A_\ell| \geq \frac{\epsilon n^d 2^{\ell-1}}{2k}$.

The tester is as follows. For each $i \in [k]$:

1. Sample $p = \frac{k}{\epsilon 2^{i-2}}\ln(6)$ points $x_1, \ldots, x_p \sim [n]^d$.
2. For each $j \in [p]$, sample $q = 2^{i+2}\ln(12)$ points $y_1, \ldots, y_q$ from $\mathrm{diag}(x_i)$ and reject if there are two distinct 1-valued points in the sample.

The query complexity of the tester is $\sum_{i=1}^{k} 4^2\ln(6)\ln(12)\frac{k}{\epsilon 2^i}2^i = O\left(\frac{1}{\epsilon}\log^2(1/\epsilon)\right)$.

The tester will clearly accept any diagonal function. Now suppose that f is ϵ-far from having this property, and let $\ell \in [k]$ be such that $|A_\ell| \geq \frac{\epsilon n^d 2^{\ell-2}}{k}$. On iteration $i = \ell$, the algorithm samples $p = \frac{k}{\epsilon 2^{\ell-2}}\ln(6)$ points $x_1, \ldots, x_p$. The probability that $\forall j \in [p], x_j \notin A_\ell$ is at most

$$\left(1 - \frac{|A_\ell|}{n^d}\right)^p \leq \left(1 - \frac{\epsilon 2^{\ell-2}}{k}\right)^p \leq \exp\left(-\frac{\epsilon p 2^{\ell-2}}{k}\right) \leq 1/6\,.$$

Now assume that there is some $x_j \in A_\ell$, so that $R(x_j) > 2^{-\ell}$. Let $A, B \subset \mathrm{diag}(x_j)$ be disjoint subsets that partition the 1-valued points in $\mathrm{diag}(x_i)$ into equally-sized parts. Then for y sampled uniformly at random from $\mathrm{diag}(x_j)$, $\mathbb{P}\left[y \in A\right], \mathbb{P}\left[y \in B\right] \geq 2^{-(\ell+1)}$. The probability that there are at least 2 distinct 1-valued points in $y_1, \ldots, y_q$ sampled by the algorithm is at least the probability that one of the first $q/2$ samples is in A and one of the last $q/2$ samples is in B. This fails to occur with probability at most $2(1 - 2^{-(\ell+1)})^{q/2} \leq 2e^{-q2^{-(\ell+2)}} \leq 1/6$. So the total probability of failure is at most $2/6 = 1/3$. ◀

▶ **Theorem 3.2.** *There is a non-adaptive monotonicity tester on domain $[n]^d$ with one-sided error and query complexity $\widetilde{O}\left(\frac{d^{5/6}}{\epsilon^{4/3}}\right)$.*

Proof. Set $r = \lceil 4d/\epsilon \rceil$, and assume without loss of generality that r divides n. Partition $[n]$ into r intervals $B_i = \{(i-1)(n/r)+1, \ldots, i(n/r)\}$. For each $v \in [r]^d$ write $B_v = B_{v_1} \times \cdots \times B_{v_d}$. Define $\mathsf{block} : [n]^d \to [r]^d$ where $\mathsf{block}(x)$ is the unique vector $v \in [r]^d$ such that $x \in B_v$. Define $\mathsf{block}^{-\downarrow}(v) = \min\{x \in B_v\}$ and $\mathsf{block}^{-\uparrow}(v) = \max\{x \in B_v\}$, where the minimum and maximum are with respect to the natural ordering on $[n]^d$. For $f : [n]^d \to \{0,1\}$, write $f^{\mathsf{block}} : [r]^d \to \{0,1\}$, $f^{\mathsf{block}}(v) = f(\mathsf{block}^{-\downarrow}(v))$. We may simulate queries v to f^{block} by returning $f(\mathsf{block}^{-\downarrow}(v))$. We will call $v \in [r]^d$ a *boundary block* if $f(\mathsf{block}^{-\downarrow}(v)) \neq f(\mathsf{block}^{-\uparrow}(v))$.

The test proceeds as follows: On input $f : [n]^d \to \{0,1\}$ and a block $v \in [r]^d$, define the following functions:

$$g : [n]^d \to \{0,1\}, \qquad g(x) = \begin{cases} f^{\mathsf{block}}(\mathsf{block}(x)) & \text{if } \mathsf{block}(x) \text{ is not a boundary block} \\ f(x) & \text{if } \mathsf{block}(x) \text{ is a boundary block.} \end{cases}$$

$$b : [r]^d \to \{0,1\}, \qquad b(v) = \begin{cases} 0 & \text{if } v \text{ is not a boundary block} \\ 1 & \text{if } v \text{ is a boundary block.} \end{cases}$$

$$h : [r]^d \to \{0,1\}, \qquad h(v) = \begin{cases} f^{\mathsf{block}}(v) & \text{if } v \text{ is not a boundary block} \\ 0 & \text{if } v \text{ is a boundary block.} \end{cases}$$

Queries to each of these functions can be simulated by 2 or 3 queries to f. The tester performs:

1. Test whether $g = f$, or whether $\mathsf{dist}(f,g) > \epsilon/4$, using $O(1/\epsilon)$ queries.
2. Test whether b is diagonal, or is $\epsilon/4$-far from diagonal, using Lemma 3.1, with $O\left(\frac{1}{\epsilon}\log^2(1/\epsilon)\right)$ queries.
3. Test whether h is monotone or $\epsilon/4$-far from monotone, using the tester of Black, Chakrabarty, & Seshadhri with $\widetilde{O}\left(\frac{d^{5/6}}{\epsilon^{4/3}}\right)$ queries.

▷ Claim 3.3. If f is monotone, the tester passes all 3 tests with probability 1.

Proof of claim. To see that $g = f$, observe that if $v = \mathsf{block}(x)$ is not a boundary block then $f(\mathsf{block}^{-\downarrow}(v)) = f(\mathsf{block}^{-\uparrow}(v))$. If $f(x) \neq f^{\mathsf{block}}(\mathsf{block}(x))$ then $f(x) \neq f(\mathsf{block}^{-\downarrow}(v))$ and $f(x) \neq f(\mathsf{block}^{-\uparrow}(v))$ while $\mathsf{block}^{-\downarrow}(v) \preceq x \preceq \mathsf{block}^{-\uparrow}(v)$, and this is a violation of the monotonicity of f. Therefore f will pass the first test with probability 1.

To see that f passes the second test with probability 1, observe that if f had 2 boundary blocks in some diagonal, then there are boundary blocks $u, v \in [r]^d$ such that $\mathsf{block}^{-\uparrow}(u) \prec \mathsf{block}^{-\downarrow}(v)$. But then there is $x, y \in [n]^d$ such that $\mathsf{block}(x) = u, \mathsf{block}(y) = v$ and $f(x) = 1, f(y) = 0$; since $x \preceq \mathsf{block}^{-\uparrow}(u) \prec \mathsf{block}^{-\downarrow}(v) \preceq y$, this contradicts the monotonicity of f. So f has at most 1 boundary block in each diagonal.

To see that h is monotone, it is sufficient to consider the boundary blocks, since all other values are the same as f^{block}. Let $v \in [r]^d$ be a boundary block, so there exist $x, y \in [n]^d$ such that $\mathsf{block}(x) = \mathsf{block}(y)$ and $f(x) = 1, f(y) = 0$. Suppose $u \prec v$ is not a boundary block (if it is a boundary block then $h(u) = h(v) = 0$). If $h(u) = 1$ then $f(\mathsf{block}^{-\downarrow}(u)) = 1$, but $\mathsf{block}^{-\downarrow}(u) \prec \mathsf{block}^{-\downarrow}(v) \preceq y$ while $f(\mathsf{block}^{-\downarrow}(u)) > f(y)$, a contradiction. So it must be that $h(u) = 0$ whenever $u \prec v$. For any block $u \in [r]^d$ such that $v \prec u$, we have $0 = h(v) \leq h(u)$, so monotonicity holds. Since the tester of Black, Chakrabarty, & Seshadhri has one-sided error, the test passes with probability 1. ◁

▷ Claim 3.4. If g is $\epsilon/4$-close to f, b is $\epsilon/4$-close to diagonal, and h is $\epsilon/4$-close to monotone, then f is ϵ-close to monotone.

Proof of claim. Let $h^{\mathsf{coarse}} : [n]^d \to \{0,1\}$ be the function $h^{\mathsf{coarse}}(x) = h(\mathsf{block}(x))$. Suppose that $f(x) \neq h^{\mathsf{coarse}}(x)$. If $v = \mathsf{block}(x)$ is not a boundary block of f then $h^{\mathsf{coarse}}(x) = h(v) = f^{\mathsf{block}}(v) = g(x)$, so $f(x) \neq g(x)$. If v is a boundary block then $h^{\mathsf{coarse}}(x) = h(v) = 0$ so $f(x) = 1$, and $b(v) = 1$.

Suppose for contradiction that there are more than $\frac{\epsilon}{2}r^d$ boundary blocks $v \in [r]^d$, so there are more than $\frac{\epsilon}{2}r^d$ 1-valued points of b. Any diagonal function has at most dr^{d-1} 1-valued points. Therefore the distance of b to diagonal is at least

$$r^{-d}\left(\frac{\epsilon}{2}r^d - dr^{d-1}\right) = \frac{\epsilon}{2} - \frac{d}{r} = \frac{\epsilon}{2} - \frac{\epsilon}{4} = \frac{\epsilon}{4}\,,$$

a contradiction. So f has at most $\frac{\epsilon}{2}r^d$ boundary blocks. Now

$$\mathsf{dist}(f, h^{\mathsf{coarse}}) = \mathsf{dist}(f,g) + \mathop{\mathbb{P}}_{x\sim[n]^d}[f(x) = 1, \mathsf{block}(x) \text{ is a boundary block}] \leq \frac{\epsilon}{4} + r^{-d}\cdot\frac{\epsilon r^d}{2} = \frac{3}{4}\epsilon\,.$$

Let $p : [r]^d \to \{0,1\}$ be a monotone function minimizing the distance to h, and let $p^{\mathsf{coarse}} : [n]^d \to \{0,1\}$ be the function $p^{\mathsf{coarse}}(x) = p(\mathsf{block}(x))$. Then

$$\mathsf{dist}(h^{\mathsf{coarse}}, p^{\mathsf{coarse}}) = \mathop{\mathbb{P}}_{x\sim[n]^d}[h(\mathsf{block}(x)) \neq p(\mathsf{block}(x))] = \mathop{\mathbb{P}}_{v\sim[r]^d}[h(v) \neq p(v)] \leq \epsilon/4\,.$$

Finally, the distance of f to the nearest monotone function is at most

$$\mathsf{dist}(f, p^{\mathsf{coarse}}) \leq \mathsf{dist}(f, h^{\mathsf{coarse}}) + \mathsf{dist}(h^{\mathsf{coarse}}, p^{\mathsf{coarse}}) \leq \frac{3}{4}\epsilon + \frac{1}{4}\epsilon = \epsilon\,. \qquad \lhd$$

These two claims suffice to establish the theorem. ◀

3.2 Monotonicity Testing for Product Distributions

The previous section used a special case of downsampling, tailored for the uniform distribution over $[n]^d$. We will call a product distribution $\mu = \mu_1 \times \cdots \times \mu_d$ over $\mathbb{R}^d$ *continuous* if each of its factors μ_i are continuous (i.e. absolutely continuous with respect to the Lebesgue measure). The proof for discrete distributions is in the full version.

▶ **Theorem 1.1.** *There is a one-sided, non-adaptive ϵ-tester for monotonicity of functions $\mathbb{R}^d \to \{0,1\}$ that is distribution-free under (finite or continuous) product distributions, using*

$$O\left(\frac{d^{5/6}}{\epsilon^{4/3}}\operatorname{poly}\log(d/\epsilon)\right)$$

queries and $O(\frac{d^3}{\epsilon^3}\log(d/\epsilon))$ samples.

Proof. We follow the proof of Theorem 3.2, with some small changes. Let $r = \lceil 16d/\epsilon \rceil$. The tester first samples a grid X with length $m = O\left(\frac{rd^2}{\epsilon^2}\log(rd)\right)$ and constructs the induced $(r+2)$-block partition, with cells labeled $\{0,\ldots,r+1\}^d$. We call a block $v \in \{0,\ldots,r+1\}^d$ *upper extreme* if there is some $i \in [d]$ such that $v_i = r+1$, and we call it *lower extreme* if there is some $i \in [d]$ such that $v_i = 0$ but v is not upper extreme. Call the upper extreme blocks U and the lower extreme blocks L. Note that $[r]^d = \{0,\ldots,r+1\}^d \setminus (U \cup L)$.

For each $v \in [r]^d$, we again define $\mathsf{block}^{-\uparrow}(v), \mathsf{block}^{-\downarrow}(v)$ as, respectively, the supremal and infimal point $x \in \mathbb{R}^d$ such that $\mathsf{block}(x) = v$. The algorithm will ignore the extreme blocks $U \cup L$, which do not have a supremal or an infimal point. Therefore it is not defined whether these blocks are boundary blocks.

By Lemma 2.7, with probability at least 5/6, we will have $\|\mathsf{block}(\mu) - \mathsf{unif}(\{0,\dots,r+1\})\|_{\mathsf{TV}} \le \epsilon/8$. We define b, h as before, with domain $[r]^d$. Define g similarly but with domain $\mathbb{R}^d$ and values

$$g(x) = \begin{cases} 1 & \text{if } \mathsf{block}(x) \in U \\ 0 & \text{if } \mathsf{block}(x) \in L \\ f(x) & \text{if } \mathsf{block}(x) \in [n]^d \text{ is a boundary block} \\ f^{\mathsf{block}}(\mathsf{block}(x)) & \text{otherwise.} \end{cases}$$

If f is monotone, it may now be the case $f \neq g$, but we will have $f(x) = g(x)$ for all x with $\mathsf{block}(x) \in [r]^d$, where the algorithm will make its queries. The algorithm will test whether $f(x) = g(x)$ on all x with $\mathsf{block}(x) \in [r]^d$, or $\epsilon/8$-far from this property, which can be again done with $O(1/\epsilon)$ samples. Note that if f is $\epsilon/8$-close to having this property, then

$$\begin{aligned} \mathsf{dist}_\mu(f,g) &\le \mathop{\mathbb{P}}_{x\sim\mu}\left[\mathsf{block}(x) \notin [n]^d\right] + \epsilon/8 \\ &\le \frac{d(r+2)^{d-1}}{(r+2)^d} + \epsilon/8 + \|\mathsf{block}(\mu) - \mathsf{unif}([r]^d \cup U \cup L)\|_{\mathsf{TV}} \\ &\le \frac{\epsilon}{16} + \frac{\epsilon}{8} + \frac{\epsilon}{4} \le \frac{\epsilon}{2}\,. \end{aligned}$$

The algorithm then procedes as before, with error parameter $\epsilon/2$. To test whether $g = f$, the algorithm samples from μ and throws away any sample $x \in \mathbb{R}^d$ with $\mathsf{block}(x) \notin [r]^d$. It then tests b and h using the uniform distribution on $[r]^d$. It suffices to prove the following claim, which replaces Claim 3.4.

▷ Claim 3.5. If g is $\epsilon/2$-close to f, b is $\epsilon/16$-close to diagonal, and h is $\epsilon/8$-close to monotone, then f is ϵ-close to monotone.

Proof of claim. Let $p : [r]^d \to \{0,1\}$ be a monotone function minimizing the distance to h. Then $p(v) \neq h(v)$ on at most $\frac{\epsilon r^d}{8}$ blocks $v \in [r]^d$. Define $p^{\mathsf{coarse}} : \mathbb{R}^d \to \{0,1\}$ as $p^{\mathsf{coarse}}(x) = p(\mathsf{block}(x))$ when $\mathsf{block}(x) \in [r]^d$, and $p^{\mathsf{coarse}}(x) = g(x)$ when $\mathsf{block}(x) \in U \cup L$. Note that p^{coarse} is monotone.

By the triangle inequality,

$$\mathsf{dist}_\mu(f, p^{\mathsf{coarse}}) \le \mathsf{dist}_\mu(f,g) + \mathsf{dist}_\mu(g, p^{\mathsf{coarse}})\,.$$

From above, we know $\mathsf{dist}_\mu(f,g) \le \epsilon/2$. To bound the second term, observe that since b is $\epsilon/16$-close to diagonal, there are at most

$$\frac{\epsilon}{16}r^d + dr^{d-1} \le \frac{\epsilon}{16}r^d + \frac{d}{r}r^d \le \frac{\epsilon}{16}r^d + \frac{\epsilon}{16}r^d = \frac{\epsilon}{8}r^d$$

boundary blocks. Then observe that if $g(x) \neq p^{\mathsf{coarse}}(x)$ then $\mathsf{block}(x) \in [r]^d$ and either $\mathsf{block}(x)$ is a boundary block, or $g(x) = f^{\mathsf{block}}(\mathsf{block}(x)) = h(\mathsf{block}(x))$ and $h(\mathsf{block}(x)) \neq p(\mathsf{block}(x))$. Then

$$\begin{aligned} \mathsf{dist}_\mu(g, p^{\mathsf{coarse}}) &\le \left(\frac{1}{(r+2)^d}\sum_{v\in[r]^d} \mathbf{1}\left[v \text{ is a boundary block, or } h(v) \neq p(v)\right]\right) \\ &\quad + \|\mathsf{block}(\mu) - \mathsf{unif}(\{0,\dots,r+1\}^d)\|_{\mathsf{TV}} \\ &\le \frac{\epsilon r^d}{8r^d} + \frac{\epsilon r^d}{8r^d} + \frac{\epsilon}{4} \le \frac{\epsilon}{2}\,. \end{aligned}$$

◁

◀

References

1. Nir Ailon and Bernard Chazelle. Information theory in property testing and monotonicity testing in higher dimension. *Information and Computation*, 204(11):1704–1717, 2006.
2. Maria-Florina Balcan, Eric Blais, Avrim Blum, and Liu Yang. Active property testing. In *Proceedings of the IEEE 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 21–30, 2012.
3. Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova. Tolerant testers of image properties. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
4. Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova. Testing convexity of figures under the uniform distribution. *Random Structures & Algorithms*, 54(3):413–443, 2019.
5. Hadley Black, Deeparnab Chakrabarty, and C Seshadhri. A $o(d) \cdot \operatorname{poly} \log n$ monotonicity tester for boolean functions over the hypergrid $[n]^d$. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2133–2151, 2018.
6. Hadley Black, Deeparnab Chakrabarty, and C Seshadhri. Domain reduction for monotonicity testing: A $o(d)$ tester for boolean functions in d-dimensions. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1975–1994, 2020.
7. Eric Blais and Abhinav Bommireddi. On testing and robust characterizations of convexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
8. Eric Blais, Clément Canonne, Igor Oliveira, Rocco Servedio, and Li-Yang Tan. Learning circuits with few negations. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, page 512, 2015.
9. Eric Blais, Renato Ferreira Pinto Jr, and Nathaniel Harms. VC dimension and distribution-free sample-based testing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 504–517, 2021.
10. Eric Blais, Ryan O'Donnell, and Karl Wimmer. Polynomial regression under arbitrary product distributions. *Machine Learning*, 80(2-3):273–294, 2010.
11. Eric Blais, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Lower bounds for testing properties of functions over hypergrid domains. In *Proceedings of the IEEE 29th Conference on Computational Complexity (CCC)*, pages 309–320, 2014.
12. Eric Blais and Yuichi Yoshida. A characterization of constant-sample testable properties. *Random Structures & Algorithms*, 55(1):73–88, 2019.
13. Avrim L Blum and Ronald L Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992.
14. Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.
15. Clément L. Canonne, Elena Grigorcscu, Siyao Guo, Akash Kumar, and Karl Wimmer. Testing k-monotonicity: The rise and fall of boolean functions. *Theory of Computing*, 15(1):1–55, 2019.
16. Deeparnab Chakrabarty, Kashyap Dixit, Madhav Jha, and C Seshadhri. Property testing on product distributions: Optimal testers for bounded derivative properties. *ACM Transactions on Algorithms (TALG)*, 13(2):1–30, 2017.
17. Deeparnab Chakrabarty and Comandur Seshadhri. An $o(n)$ monotonicity tester for boolean functions over the hypercube. *SIAM Journal on Computing*, 45(2):461–472, 2016.
18. Xi Chen, Adam Freilich, Rocco A Servedio, and Timothy Sun. Sample-based high-dimensional convexity testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
19. Xue Chen, Anindya De, and Rocco A Servedio. Testing noisy linear functions for sparsity. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 610–623, 2020.

20 Mónika Csikós, Nabil H Mustafa, and Andrey Kupavskii. Tight lower bounds on the VC-dimension of geometric set systems. *Journal of Machine Learning Research*, 20(81):1–8, 2019.

21 Anindya De, Elchanan Mossel, and Joe Neeman. Is your function low-dimensional? In *Conference on Learning Theory*, pages 979–993, 2019.

22 Ilias Diakonikolas, Prahladh Harsha, Adam Klivans, Raghu Meka, Prasad Raghavendra, Rocco A Servedio, and Li-Yang Tan. Bounding the average sensitivity and noise sensitivity of polynomial threshold functions. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 533–542, 2010.

23 Ilias Diakonikolas, Daniel M. Kane, and Alistair Stewart. Learning geometric concepts with nasty noise. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1061–1073, 2018.

24 Shahar Fattal and Dana Ron. Approximating the distance to monotonicity in high dimensions. *ACM Transactions on Algorithms (TALG)*, 6(3):1–37, 2010.

25 Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th annual ACM symposium on Theory of Computing (STOC)*, pages 474–483, 2002.

26 Noah Fleming and Yuichi Yoshida. Distribution-free testing of linear functions on $\mathbb{R}^n$. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

27 Oded Goldreich and Dana Ron. On sample-based testers. *ACM Transactions on Computation Theory (TOCT)*, 8(2):1–54, 2016.

28 Shirley Halevy and Eyal Kushilevitz. Distribution-free property-testing. *SIAM Journal on Computing*, 37(4):1107–1138, 2007.

29 Nathaniel Harms. Testing halfspaces over rotation-invariant distributions. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 694–713, 2019.

30 Adam T. Kalai, Adam R. Klivans, Yishay Mansour, and Rocco A Servedio. Agnostically learning halfspaces. *SIAM Journal on Computing*, 37(6):1777–1805, 2008.

31 Michael Kearns and Dana Ron. Testing problems with sublearning sample complexity. *Journal of Computer and System Sciences*, 61(3):428–456, 2000.

32 Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric-type theorems. *SIAM Journal on Computing*, 47(6):2238–2276, 2018.

33 Adam R Klivans, Ryan O'Donnell, and Rocco A Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer and System Sciences*, 68(4):808–840, 2004.

34 Adam R Klivans, Ryan O'Donnell, and Rocco A Servedio. Learning geometric concepts via gaussian surface area. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 541–550, 2008.

35 Luis Rademacher and Santosh Vempala. Testing geometric convexity. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 469–480. Springer, 2004.

36 Sofya Raskhodnikova. Approximate testing of visual properties. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 370–381. Springer, 2003.

37 Dana Ron and Asaf Rosin. Optimal distribution-free sample-based testing of subsequence-freeness. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 337–256. SIAM, 2021.

38 Santosh Vempala. Learning convex concepts from gaussian distributions with PCA. In *Proceedings of the IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–130, 2010.

39 Santosh Vempala. A random-sampling-based algorithm for learning intersections of halfspaces. *Journal of the ACM*, 57(6):1–14, 2010.

40 Hugh E. Warren. Lower bounds for approximation by nonlinear manifolds. *Transactions of the American Mathematical Society*, 133(1):167–178, 1968.

A Fixed-Parameter Algorithm for the Kneser Problem

Ishay Haviv
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Abstract

The Kneser graph $K(n,k)$ is defined for integers n and k with $n \geq 2k$ as the graph whose vertices are all the k-subsets of $\{1,2,\ldots,n\}$ where two such sets are adjacent if they are disjoint. A classical result of Lovász asserts that the chromatic number of $K(n,k)$ is $n-2k+2$. In the computational Kneser problem, we are given an oracle access to a coloring of the vertices of $K(n,k)$ with $n-2k+1$ colors, and the goal is to find a monochromatic edge. We present a randomized algorithm for the Kneser problem with running time $n^{O(1)} \cdot k^{O(k)}$. This shows that the problem is fixed-parameter tractable with respect to the parameter k. The analysis involves structural results on intersecting families and on induced subgraphs of Kneser graphs.

We also study the Agreeable-Set problem of assigning a small subset of a set of m items to a group of ℓ agents, so that all agents value the subset at least as much as its complement. As an application of our algorithm for the Kneser problem, we obtain a randomized polynomial-time algorithm for the Agreeable-Set problem for instances that satisfy $\ell \geq m - O(\frac{\log m}{\log\log m})$. We further show that the Agreeable-Set problem is at least as hard as a variant of the Kneser problem with an extended access to the input coloring.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph coloring; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Probabilistic algorithms

Keywords and phrases Kneser graph, Fixed-parameter tractability, Agreeable Set

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.72

Category Track A: Algorithms, Complexity and Games

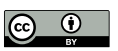
Related Version *Full Version*: `http://arxiv.org/abs/2204.06761`

Funding *Ishay Haviv*: Research supported in part by the Israel Science Foundation (grant No. 1218/20).

Acknowledgements We are grateful to Andrey Kupavskii for helpful discussions and to the anonymous reviewers for their very useful suggestions.

1 Introduction

The Kneser graph $K(n,k)$ is defined for integers n and k with $n \geq 2k$ as the graph whose vertices are all the k-subsets of $[n] = \{1,2,\ldots,n\}$ where two such sets are adjacent if they are disjoint. In 1955, Kneser [17] observed that there exists a proper coloring of the vertices of $K(n,k)$ with $n-2k+2$ colors and conjectured that fewer colors do not suffice, that is, that its chromatic number satisfies $\chi(K(n,k)) = n-2k+2$. The conjecture was proved more than two decades later by Lovász [19] as a surprising application of the Borsuk-Ulam theorem from algebraic topology [2]. Following this result, topological methods have become a common and powerful tool in combinatorics, discrete geometry, and theoretical computer science (see, e.g., [22]). Several alternative proofs of Kneser's conjecture were provided in the literature over the years (see, e.g., [23]), and despite the combinatorial nature of the conjecture, all of them essentially rely on the topological Borsuk-Ulam theorem. One exception is a proof by Matoušek [21], which is presented in a combinatorial form, but is yet inspired by a discrete variant of the Borsuk-Ulam theorem known as Tucker's lemma.




In the computational KNESER problem, we are given an access to a coloring of the vertices of $K(n,k)$ with $n-2k+1$ colors, and the goal is to find a monochromatic edge, i.e., two vertices with the same color that correspond to disjoint sets. Since the number of colors used by the input coloring is strictly smaller than the chromatic number of $K(n,k)$ [19], it follows that every instance of the problem has a solution. However, the topological proofs of the lower bound on the chromatic number of $K(n,k)$ are not constructive, in the sense that they do not supply an efficient algorithm for finding a monochromatic edge. By an efficient algorithm we mean that its running time is polynomial in n, whereas the number of vertices $\binom{n}{k}$ might be exponentially larger. Hence, it is natural to assume that the input coloring is given as an access to an oracle that given a vertex of $K(n,k)$ returns its color. Alternatively, the coloring can be given by some succinct representation, e.g., a Boolean circuit that computes the color of any given vertex.

The question of determining the complexity of the KNESER problem was proposed by Deng, Feng, and Kulkarni [6], who asked whether it is complete in the complexity class PPA. This complexity class belongs to a family of classes that were introduced by Papadimitriou [26] in the attempt to characterize the mathematical arguments that lie behind the existence of solutions to search problems of TFNP. The complexity class TFNP, introduced in [25], is the class of all total search problems in NP, namely, the search problems in which a solution is guaranteed to exist and can be verified in polynomial running time. Papadimitriou has introduced in [26] several subclasses of TFNP, each of which consists of the total search problems that can be reduced to a problem that represents some mathematical argument. In particular, the class PPA (Polynomial Parity Argument) corresponds to the simple fact that every (undirected) graph with maximum degree 2 that has a vertex of degree 1 must have another degree 1 vertex. Hence, PPA is defined as the class of all problems in TFNP that can be efficiently reduced to the LEAF problem, in which given a succinct representation of a graph with maximum degree 2 and given a vertex of degree 1 in the graph, the goal is to find another such vertex.

In recent years, it has been shown that the complexity class PPA perfectly captures the complexity of several total search problems for which the existence of the solution relies on the Borsuk-Ulam theorem. Indeed, Filos-Ratsikas and Goldberg [9, 10] proved that the Consensus Halving problem with inverse-polynomial precision parameter is PPA-complete and derived the PPA-completeness of some classical problems, such as the Splitting Necklace problem with two thieves and the Discrete Sandwich problem. The PPA-hardness of the Consensus Halving problem was further extended to a constant precision parameter in a recent work of Deligkas, Fearnley, Hollender, and Melissourgos [4]. Another PPA-complete problem, studied in [15] and closely related to the KNESER problem, is the SCHRIJVER problem which given a coloring of the Schrijver graph $S(n,k)$ with fewer colors than its chromatic number asks to find a monochromatic edge. Note that $S(n,k)$ is the induced subgraph of $K(n,k)$ on the collection of all k-subsets of $[n]$ with no two consecutive elements modulo n, and that its chromatic number is equal to that of $K(n,k)$ [27]. Despite the progress in understanding the complexity of total search problems related to the Borsuk-Ulam theorem, the question of [6] on the complexity of the KNESER problem is still open. The question of determining the complexity of its extension to Kneser hypergraphs was recently raised by Filos-Ratsikas, Hollender, Sotiraki, and Zampetakis [11].

The study of the KNESER problem has been recently motivated by its connection to a problem called AGREEABLE-SET that was introduced by Manurangsi and Suksompong [20] and further studied by Goldberg, Hollender, Igarashi, Manurangsi, and Suksompong [14]. This problem falls into the category of resource allocation problems, where one assigns items

from a given collection $[m]$ to ℓ agents that have different preferences. The preferences are given by monotone utility functions that associate a non-negative value to each subset of $[m]$. In the Agreeable-Set setting, the agents act as a group, and the goal is to collectively allocate a subset of items that is agreeable to all of them, in the sense that every agent likes it at least as much as it likes the complement set. The authors of [20] proved that for every ℓ agents with monotone utility functions defined on the subsets of $[m]$, there exists a subset $S \subseteq [m]$ of size

$$|S| \leq \min\left(\left\lfloor \frac{m+\ell}{2} \right\rfloor, m\right) \tag{1}$$

that is agreeable to all agents, and that this bound is tight in the worst case. They initiated the study of the Agreeable-Set problem that given an oracle access to the utility functions of the ℓ agents, asks to find a subset $S \subseteq [m]$ that satisfies the worst-case bound given in (1) and that is agreeable to all agents. Note that for instances with $\ell \geq m$, the collection $[m]$ forms a proper solution.

Interestingly, the proof of [20] for the existence of a solution to the Agreeable-Set problem relies on the chromatic number of Kneser graphs. In fact, the proof implicitly shows that the Agreeable-Set problem is efficiently reducible to the Kneser problem, hence the Kneser problem is at least as hard as the Agreeable-Set problem. An alternative existence proof, based on the Consensus Halving theorem [28], was given by the authors of [14]. Their approach was applied there to show that the Agreeable-Set problem can be solved in polynomial time when the utility functions of the agents are additive and when the number ℓ of agents is a fixed constant. The complexity of the problem in the general case is still open.

1.1 Our Contribution

Our main result concerns the parameterized complexity of the Kneser problem. We prove that the problem is fixed-parameter tractable with respect to the parameter k, namely, it can be solved on an input coloring of a Kneser graph $K(n,k)$ in running time $n^{O(1)} \cdot f(k)$ for some function f.

▶ **Theorem 1.** *There exists a randomized algorithm that given integers n and k with $n \geq 2k$ and an oracle access to a coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of the vertices of the Kneser graph $K(n,k)$, runs in time $n^{O(1)} \cdot k^{O(k)}$ and returns a monochromatic edge with probability $1 - 2^{-\Omega(n)}$.*

It should be mentioned that the notion of fixed-parameter tractability is used here in a slightly different manner than in the parameterized complexity literature (see, e.g., [3]). This is because Theorem 1 deals with the parameterized complexity of a total search problem, rather than a decision problem, and because its input is given as an oracle access. In fact, borrowing the terminology of this area, our algorithm for the Kneser problem can be viewed as a randomized polynomial Turing kernelization algorithm for the problem (see, e.g., [12, Chapter 22]). Namely, we show that the problem of finding a monochromatic edge in a Kneser graph $K(n,k)$ can essentially be reduced by a randomized efficient algorithm to finding a monochromatic edge in a Kneser graph $K(n',k)$ for $n' = O(k^4)$ (see Section 3.4 for the precise details).

The analysis of the algorithm given in Theorem 1 relies on properties of induced subgraphs of Kneser graphs (see Section 3.1). The proofs involve a stability result due to Hilton and Milner [16] of the celebrated Erdös-Ko-Rado theorem [8] (see Theorem 9) and an idea recently applied by Frankl and Kupavskii [13] in the study of maximal degrees in induced subgraphs of Kneser graphs. An overview of the proof of Theorem 1 is given in Section 1.2.

Our next result provides a simple deterministic algorithm for the KNESER problem that is particularly useful for Kneser graphs $K(n,k)$ with k close to $n/2$. Its analysis is based on the chromatic number of Schrijver graphs [27].

▶ **Theorem 2.** *There exists an algorithm that given integers n and k with $n \geq 2k$ and an oracle access to a coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of the vertices of the Kneser graph $K(n,k)$, returns a monochromatic edge in running time polynomial in $\binom{n-k+1}{k} \leq n^{\min(k,n-2k+1)}$.*

We proceed by presenting our results on the AGREEABLE-SET problem. First, as applications of Theorems 1 and 2, using the relation from [20] between the AGREEABLE-SET and KNESER problems, we obtain the following algorithmic results.

▶ **Theorem 3.** *There exists a randomized algorithm for the AGREEABLE-SET problem that given an oracle access to an instance with m items and ℓ agents ($\ell < m$), runs in time $m^{O(1)} \cdot k^{O(k)}$ for $k = \lceil \frac{m-\ell}{2} \rceil$ and returns a proper solution with probability $1 - 2^{-\Omega(m)}$.*

▶ **Theorem 4.** *There exists an algorithm for the AGREEABLE-SET problem that given an oracle access to an instance with m items and ℓ agents ($\ell < m$), returns a proper solution in running time polynomial in $\binom{m-k+1}{k} \leq m^{\min(k,m-2k+1)}$ for $k = \lceil \frac{m-\ell}{2} \rceil$.*

We apply Theorems 3 and 4 to show that the AGREEABLE-SET problem can be solved in polynomial time for certain families of instances. By Theorem 3, we obtain a randomized efficient algorithm for instances in which the number of agents ℓ is not much smaller than the number of items m, namely, for $\ell \geq m - O(\frac{\log m}{\log \log m})$. By Theorem 4, we obtain an efficient algorithm for instances with a constant number of agents, providing an alternative proof for a result of [14].

We finally explore the relations between the AGREEABLE-SET and KNESER problems. As already mentioned, there exists an efficient reduction from the AGREEABLE-SET problem to the KNESER problem [20]. Here we provide a reduction in the opposite direction. However, for the reduction to be efficient we reduce from a variant of the KNESER problem with an extended type of queries, which we call *subset queries* and define as follows. For an input coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of a Kneser graph $K(n,k)$, a subset query is a pair (i,B) of a color $i \in [n-2k+1]$ and a set $B \subseteq [n]$, and the answer on the query (i,B) determines whether B contains a vertex colored i, that is, whether there exists a k-subset $A \subseteq B$ satisfying $c(A) = i$. We prove the following result (see Section 2.1 for the computational input model of the problems).

▶ **Theorem 5.** *There exists a polynomial-time reduction from the KNESER problem with subset queries to the AGREEABLE-SET problem.*

1.2 Proof Overview of Theorem 1

We present here the main ideas of our fixed-parameter algorithm for the KNESER problem. Suppose that we are given, for $n \geq 2k$, an oracle access to a coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of the vertices of the Kneser graph $K(n,k)$ with $n-2k+1$ colors. As mentioned before, since the chromatic number of $K(n,k)$ is $n-2k+2$ [19], the coloring c must have a monochromatic edge. Such an edge can clearly be found by an algorithm that queries the oracle for the colors of all the vertices. However, the running time of such an algorithm is polynomial in $\binom{n}{k}$, so it is not fixed-parameter with respect to the parameter k.

A natural attempt to improve on this running time is to consider a randomized algorithm that picks uniformly and independently at random polynomially many vertices of $K(n,k)$ and checks if any two of them form a monochromatic edge. However, it is not difficult to see that

there exist colorings of $K(n,k)$ with $n-2k+1$ colors for which a small fraction of the vertices are involved in all the monochromatic edges, implying that the success probability of this randomized algorithm is negligible on them. To see this, consider the canonical coloring of $K(n,k)$, in which for every $i \in [n-2k+1]$ the vertices $A \in \binom{[n]}{k}$ with $\min(A) = i$ are colored i, and the remaining vertices, those contained in $[n] \setminus [n-2k+1]$, are colored $n-2k+2$. By recoloring the vertices of the last color class with arbitrary colors from $[n-2k+1]$, we get a coloring of $K(n,k)$ with $n-2k+1$ colors such that every monochromatic edge has an endpoint in a collection of $\binom{2k-1}{k}$ vertices. This implies that the probability that two random vertices chosen uniformly and independently from $\binom{[n]}{k}$ form a monochromatic edge may go to zero faster than any inverse polynomial in n.

While the above coloring shows that all the monochromatic edges can involve vertices from a small set, one may notice that this coloring is very well structured, in the sense that each color class is quite close to a trivial intersecting family, i.e., an intersecting family all of whose members share a common element. This is definitely not a coincidence. It is known that large intersecting families of k-subsets of $[n]$ are "essentially" contained in trivial intersecting families. Indeed, the classical Erdös-Ko-Rado theorem [8] asserts that the largest size of an intersecting family of k-subsets of $[n]$ is $\binom{n-1}{k-1}$, attained by, and only by, the n largest trivial families (for $n > 2k$). Moreover, Hilton and Milner [16] proved a stability result for the Erdös-Ko-Rado theorem, saying that if an intersecting family of sets from $\binom{[n]}{k}$ is not trivial then its size cannot exceed $\binom{n-1}{k-1} - \binom{n-k-1}{k-1} + 1$, which is much smaller than the largest possible size of an intersecting family when n is sufficiently larger than k (see Theorem 9 and Remark 10). More recently, it was shown by Dinur and Friedgut [7] that every intersecting family can be made trivial by removing not more than $\tilde{c} \cdot \binom{n-2}{k-2}$ of its members for a constant $\tilde{c}$ (see [18] for an exact $\binom{n-3}{k-2}$ bound on the number of sets that should be removed, provided that $n \geq \tilde{c} \cdot k$ for a constant $\tilde{c}$). Hence, our strategy for finding a monochromatic edge in $K(n,k)$ is to learn the structure of the large color classes which are close to being intersecting. We use random samples from the vertex set of the graph in order to identify the common elements of the trivial families that "essentially" contain these color classes. Roughly speaking, this allows us to repeatedly reduce the size of the ground set $[n]$ of the given Kneser graph and to obtain a small subgraph, whose size depends only on k, that is expected to contain a monochromatic edge. Then, such an edge can simply be found by querying the oracle for the colors of all the vertices of this subgraph.

With the above idea in mind, let us consider an algorithm that starts by selecting uniformly and independently at random polynomially many vertices of $K(n,k)$ and queries the oracle for their colors. Among the $n-2k+1$ color classes, there must be a non-negligible one that includes at least $\frac{1}{n-2k+1} \geq \frac{1}{n}$ fraction of the vertices. It can be shown, using the Chernoff-Hoeffding bound, that the samples of the algorithm can be used to learn a color $i \in [n-2k+1]$ of a quite large color class. Moreover, the samples can be used to identify an element $j \in [n]$ that is particularly popular on the vertices colored i (say, that belongs to a constant fraction of them) in case that such an element exists. Now, if the coloring satisfies that (a) there exists an element j that is popular on the vertices colored i and, moreover, (b) this element j belongs to *all* the vertices that are colored i, then it suffices to focus on the subgraph of $K(n,k)$ induced by the vertices of $\binom{[n]\setminus\{j\}}{k}$. Indeed, condition (b) implies that the restriction of the given coloring to this subgraph uses at most $n-2k$ colors. Since this subgraph is isomorphic to $K(n-1,k)$, its chromatic number is $n-2k+1$, hence it has a monochromatic edge. By repeatedly applying this procedure, assuming that the conditions (a) and (b) hold in all iterations, we can eliminate elements from the ground set $[n]$ and obtain smaller and smaller Kneser graphs that still have a monochromatic edge.

When the ground set becomes sufficiently small, one can go over all the remaining vertices and efficiently find the required edge. We now turn to address the case where at least one of the conditions (a) and (b) does not hold.

For condition (a), suppose that in some iteration the algorithm identifies a color i that appears on a significant fraction of vertices, but no element of $[n]$ is popular on these vertices. In this case, one might expect the color class of i to be so far from being intersecting, so that the polynomially many samples would include with high probability a monochromatic edge of vertices colored i. It can be observed that the aforementioned stability results of the Erdös-Ko-Rado theorem yield that a non-negligible fraction of the vertices of a large color class with no popular element lie on monochromatic edges. However, in order to catch a monochromatic edge with high probability we need here the stronger requirement, saying that a non-negligible fraction of the *pairs* of vertices from this color class form monochromatic edges. We prove that this indeed holds for color classes of $K(n,k)$ that include at least, say, $\frac{1}{n}$ fraction of the vertices, provided that $n \geq \tilde{c} \cdot k^4$ for some constant $\tilde{c}$. The proof uses the Hilton-Milner theorem and borrows an idea of [13] (see Lemma 12 and Corollary 13).

For condition (b), suppose that in some iteration the algorithm identifies a color i of a large color class and an element j that is popular in its sets but does not belong to all of them. In this case, we can no longer ensure that the final Kneser graph obtained after all iterations has a monochromatic edge, as some of its vertices might be colored i. To handle this situation, we show that every vertex $A \in \binom{[n]}{k}$ with $j \notin A$ has a lot of neighbors colored i. Hence, if the algorithm finds at its final step, or even earlier, a vertex A colored i satisfying $j \notin A$, where i and j are the color and element chosen by the algorithm in a previous iteration, then it goes back to the ground set of this iteration and finds using additional polynomially many random vertices from it a neighbor of A colored i, and thus a monochromatic edge (see Lemma 14 and Corollary 15).

To summarize, our algorithm for the Kneser problem repeatedly calls an algorithm, which we refer to as the "element elimination" algorithm, that uses polynomially many random vertices to identify a color i of a large color class. If no element of $[n]$ is popular on this color class then the random samples provide a monochromatic edge with high probability and we are done. Otherwise, the algorithm finds such a popular $j \in [n]$ and focuses on the subgraph obtained by eliminating j from the ground set. This is done as long as the size of the ground set is larger than $\tilde{c} \cdot k^4$ for a constant $\tilde{c}$. After all iterations, the remaining vertices induce a Kneser graph $K(n',k)$ for $n' \leq \tilde{c} \cdot k^4$, and the algorithm queries for the colors of all of its vertices in time polynomial in $\binom{n'}{k} \leq k^{O(k)}$. If the colors that were chosen through the iterations of the "element elimination" algorithm do not appear in this subgraph, then it must contain a monochromatic edge which can be found by an exhaustive search. Otherwise, this search gives us a vertex A satisfying $c(A) = i$ and $j \notin A$ for a color i associated with an element j by one of the calls to the "element elimination" algorithm. As explained above, given such an A it is possible to efficiently find with high probability a vertex that forms with A a monochromatic edge. This gives us a randomized algorithm with running time $n^{O(1)} \cdot k^{O(k)}$ that finds a monochromatic edge with high probability. The full description of the algorithm and its analysis are presented in Section 3.

1.3 Outline

The rest of the paper is organized as follows. In Section 2, we gather several definitions and results that will be used throughout the paper. In Section 3, we present and analyze our randomized fixed-parameter algorithm for the Kneser problem and prove Theorem 1. In Section 4, we present a simple deterministic algorithm for the Kneser problem and prove Theorem 2. Due to space limitation, we omit the proofs of our results on the Agreeable-Set problem, which can be found in the full version of the paper.

2 Preliminaries

2.1 Computational Models

In this work we consider total search problems whose inputs involve functions that are defined on domains of size exponential in the parameters of the problems. For example, the input of the KNESER problem is a coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of the vertices of the Kneser graph $K(n,k)$ for $n \geq 2k$. For such problems, one has to specify how the input is given. We consider the following two input models.

- In the *black-box input model*, an input function is given as an oracle access, so that an algorithm can query the oracle for the value of the function on any element of its domain. This input model is used in the current work to present our algorithmic results, reflecting the fact that the algorithms do not rely on the representation of the input functions.
- In the *white-box input model*, an input function is given by a succinct representation that can be used to efficiently determine the values of the function, e.g., a Boolean circuit or an efficient Turing machine. This input model is appropriate to study the computational complexity of problems, and in particular, to show membership and hardness results with respect to the complexity class PPA.

Reductions form a useful tool to show relations between problems. Let P_1 and P_2 be total search problems. We say that P_1 is (polynomial-time) reducible to P_2 if there exist (polynomial-time) computable functions f, g such that f maps any input x of P_1 to an input $f(x)$ of P_2, and g maps any pair (x, y) of an input x of P_1 and a solution y of $f(x)$ with respect to P_2 to a solution of x with respect to P_1. For problems P_1 and P_2 in the black-box input model, one has to use the notion of *black-box reductions*. A (polynomial-time) black-box reduction satisfies that the oracle access needed for the input $f(x)$ of P_2 can be simulated by a (polynomial-time) procedure that has an oracle access to the input x. In addition, the solution $g(x, y)$ of x in P_1 can be computed (in polynomial time) given the solution y of $f(x)$ and the oracle access to the input x. In the current work we will use black-box reductions to obtain algorithmic results for problems in the black-box input model. For more details on these concepts, we refer the reader to [1, Section 2.2] (see also [5, Sections 2 and 4] for related discussions).

2.2 Kneser and Schrijver Graphs

Consider the following definition.

▶ **Definition 6.** *For a family $\mathcal{F}$ of non-empty sets, let $K(\mathcal{F})$ denote the graph on the vertex set $\mathcal{F}$ in which two vertices are adjacent if they represent disjoint sets.*

For a set X and an integer k, let $\binom{X}{k}$ denote the family of all k-subsets of X. Equipped with Definition 6, the *Kneser graph* $K(n,k)$ can be defined for integers n and k with $n \geq 2k$ as the graph $K(\binom{[n]}{k})$. A set $A \subseteq [n]$ is said to be *stable* if it includes no two consecutive elements modulo n, that is, it forms an independent set in the cycle C_n with the numbering from 1 to n along the cycle. For integers n and k with $n \geq 2k$, let $\binom{[n]}{k}_{\text{stab}}$ denote the collection of all stable k-subsets of $[n]$. The *Schrijver graph* $S(n,k)$ is defined as the graph $K(\binom{[n]}{k}_{\text{stab}})$. Equivalently, it is the induced subgraph of $K(n,k)$ on the vertex set $\binom{[n]}{k}_{\text{stab}}$.

The chromatic numbers of the graphs $K(n,k)$ and $S(n,k)$ were determined, respectively, by Lovász [19] and by Schrijver [27] as follows.

▶ **Theorem 7** ([19, 27])**.** *For all integers $n \geq 2k$, $\chi(K(n,k)) = \chi(S(n,k)) = n - 2k + 2$.*

The computational search problem associated with the Kneser graph is defined as follows.

▶ **Definition 8.** *In the computational* KNESER *problem, the input is a coloring* $c : \binom{[n]}{k} \to [n-2k+1]$ *of the vertices of the Kneser graph* $K(n,k)$ *with* $n-2k+1$ *colors for integers* n *and* k *with* $n \geq 2k$*, and the goal is to find a monochromatic edge, i.e.,* $A, B \in \binom{[n]}{k}$ *satisfying* $A \cap B = \emptyset$ *and* $c(A) = c(B)$*. In the black-box input model, the input coloring is given as an oracle access that for a vertex* A *returns its color* $c(A)$*. In the white-box input model, the input coloring is given by a Boolean circuit that for a vertex* A *computes its color* $c(A)$*.*

The existence of a solution to every instance of the KNESER problem follows from Theorem 7.

2.3 Intersecting Families

For integers n and k with $n \geq 2k$, let $\mathcal{F} \subseteq \binom{[n]}{k}$ be a family of k-subsets of $[n]$. We call $\mathcal{F}$ *intersecting* if for every two sets $F_1, F_2 \in \mathcal{F}$ it holds that $F_1 \cap F_2 \neq \emptyset$. The Erdös-Ko-Rado theorem [8] asserts that every intersecting family $\mathcal{F} \subseteq \binom{[n]}{k}$ satisfies $|\mathcal{F}| \leq \binom{n-1}{k-1}$. This bound is tight and is attained, for each $i \in [n]$, by the family $\{A \in \binom{[n]}{k} \mid i \in A\}$. An intersecting family of sets is said to be *trivial* if its members share a common element. Hilton and Milner [16] proved the following stability result for the Erdös-Ko-Rado theorem, providing an upper bound on the size of any non-trivial intersecting family.

▶ **Theorem 9** (Hilton-Milner Theorem [16])**.** *For all integers* $k \geq 2$ *and* $n \geq 2k$*, every non-trivial intersecting family of* k*-subsets of* $[n]$ *has size at most* $\binom{n-1}{k-1} - \binom{n-k-1}{k-1} + 1$*.*

▶ Remark 10. The bound given in Theorem 9 is tight. To see this, for an arbitrary k-subset F of $[n]$ and for an arbitrary element $i \notin F$, consider the family $\mathcal{F} = \{A \in \binom{[n]}{k} \mid A \cap F \neq \emptyset,\ i \in A\} \cup \{F\}$. The family $\mathcal{F}$ is intersecting and non-trivial, and its size coincides with the bound given in Theorem 9. Note that $|\mathcal{F}| \leq k \cdot \binom{n-2}{k-2}$, provided that $k \geq 3$.

2.4 Chernoff-Hoeffding Bound

We need the following concentration result (see, e.g., [24, Theorem 2.1]).

▶ **Theorem 11** (Chernoff-Hoeffding Bound)**.** *Let* $0 < p < 1$*, let* $X_1, \ldots, X_m$ *be* m *independent binary random variables satisfying* $\Pr[X_i = 1] = p$ *and* $\Pr[X_i = 0] = 1 - p$ *for all* i*, and put* $\overline{X} = \frac{1}{m} \cdot \sum_{i=1}^{m} X_i$*. Then, for any* $\mu \geq 0$*,*

$$\Pr\left[|\overline{X} - p| \geq \mu\right] \leq 2 \cdot e^{-2m\mu^2}.$$

3 A Fixed-Parameter Algorithm for the Kneser Problem

In this section we present and analyze our randomized fixed-parameter algorithm for the KNESER problem. We start with a couple of lemmas on induced subgraphs of Kneser graphs that will play a central role in the analysis of the algorithm. We then describe an algorithm, called "element elimination", which forms a main ingredient in our algorithm for the KNESER problem, and then use it to present the final algorithm and to prove Theorem 1.

3.1 Induced Subgraphs of Kneser Graphs

The following lemma shows that in a large induced subgraph of $K(n,k)$ whose vertices do not have a popular element, a random pair of vertices forms an edge with a non-negligible probability. Its proof uses the Hilton-Milner theorem (Theorem 9) and can be found in the full version of the paper. Recall that for a family $\mathcal{F} \subseteq \binom{[n]}{k}$, we let $K(\mathcal{F})$ stand for the subgraph of $K(n,k)$ induced by $\mathcal{F}$ (see Definition 6).

▶ **Lemma 12.** *For integers $k \geq 3$ and $n \geq 2k$, let $\mathcal{F}$ be a family of k-subsets of $[n]$ of size $|\mathcal{F}| \geq k^2 \cdot \binom{n-2}{k-2}$ and let $\gamma \in (0,1]$. Suppose that every element of $[n]$ belongs to at most γ fraction of the sets of $\mathcal{F}$. Then, the probability that two random sets chosen uniformly and independently from $\mathcal{F}$ are adjacent in $K(\mathcal{F})$ is at least*

$$\frac{1}{2}\cdot\left(1-\gamma-\frac{k}{|\mathcal{F}|}\cdot\binom{n-2}{k-2}\right)\cdot\left(1-\frac{k^2}{|\mathcal{F}|}\cdot\binom{n-2}{k-2}\right).$$

As a corollary of Lemma 12, we obtain the following.

▶ **Corollary 13.** *For integers $k \geq 3$ and $n \geq 8k^4$, let $\mathcal{F}$ be a family of k-subsets of $[n]$ of size $|\mathcal{F}| \geq \frac{1}{2n}\cdot\binom{n}{k}$ and let $\gamma \in (0,1]$. Suppose that every element of $[n]$ belongs to at most γ fraction of the sets of $\mathcal{F}$. Then, the probability that two random sets chosen uniformly and independently from $\mathcal{F}$ are adjacent in $K(\mathcal{F})$ is at least $\frac{3}{8}\cdot(\frac{3}{4}-\gamma)$.*

Proof. Observe that the assumptions $|\mathcal{F}| \geq \frac{1}{2n}\cdot\binom{n}{k}$ and $n \geq 8k^4$ imply that

$$\frac{k^2}{|\mathcal{F}|}\cdot\binom{n-2}{k-2} \leq \frac{2nk^2}{\binom{n}{k}}\cdot\binom{n-2}{k-2} = \frac{2k^3(k-1)}{n-1} \leq \frac{1}{4}.$$

Applying Lemma 12, we obtain that the probability that two random sets chosen uniformly and independently from $\mathcal{F}$ are adjacent in $K(\mathcal{F})$ is at least $\frac{1}{2}\cdot(1-\gamma-\frac{1}{4})\cdot(1-\frac{1}{4}) = \frac{3}{8}\cdot(\frac{3}{4}-\gamma)$. ◀

The following lemma shows that if a large collection of vertices of $K(n,k)$ has a quite popular element, then every vertex that does not include this element is adjacent to many of the vertices in the collection.

▶ **Lemma 14.** *For integers $k \geq 2$ and $n \geq 2k$, let $X \subseteq [n]$ be a set, let $\mathcal{F}$ be a family of k-subsets of X, and let $\gamma \in (0,1]$. Let $j \in X$ be an element that belongs to at least γ fraction of the sets of $\mathcal{F}$, and suppose that $A \in \binom{[n]}{k}$ is a set satisfying $j \notin A$. Then, the probability that a random set chosen uniformly from $\mathcal{F}$ is disjoint from A is at least*

$$\gamma - \frac{k}{|\mathcal{F}|}\cdot\binom{|X|-2}{k-2}.$$

Proof. Let $\mathcal{F} \subseteq \binom{X}{k}$ be a family as in the lemma, and put $\mathcal{F}' = \{F \in \mathcal{F} \mid j \in F\}$. By assumption, it holds that $|\mathcal{F}'| \geq \gamma\cdot|\mathcal{F}|$. Suppose that $A \in \binom{[n]}{k}$ is a set satisfying $j \notin A$. Observe that the number of sets $B \in \binom{X}{k}$ with $j \in B$ that intersect A does not exceed $|A \cap X|\cdot\binom{|X|-2}{k-2} \leq k\cdot\binom{|X|-2}{k-2}$. It thus follows that the number of sets of $\mathcal{F}$ that are disjoint from A is at least

$$|\mathcal{F}'| - k\cdot\binom{|X|-2}{k-2} \geq \gamma\cdot|\mathcal{F}| - k\cdot\binom{|X|-2}{k-2}.$$

Hence, a random set chosen uniformly from $\mathcal{F}$ is disjoint from A with the desired probability. ◀

As a corollary of Lemma 14, we obtain the following.

▶ **Corollary 15.** *For integers $k \geq 2$ and n, let $X \subseteq [n]$ be a set of size $|X| \geq 8k^3$, let $\mathcal{F}$ be a family of k-subsets of X of size $|\mathcal{F}| \geq \frac{1}{2|X|}\cdot\binom{|X|}{k}$, and let $\gamma \in (0,1]$. Let $j \in X$ be an element that belongs to at least γ fraction of the sets of $\mathcal{F}$, and suppose that $A \in \binom{[n]}{k}$ is a set satisfying $j \notin A$. Then, the probability that a random set chosen uniformly from $\mathcal{F}$ is disjoint from A is at least $\gamma - \frac{1}{4}$.*

Proof. Observe that the assumptions $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \binom{|X|}{k}$ and $|X| \geq 8k^3$ imply that

$$\frac{k}{|\mathcal{F}|} \cdot \binom{|X|-2}{k-2} \leq \frac{2|X|k}{\binom{|X|}{k}} \cdot \binom{|X|-2}{k-2} = \frac{2k^2(k-1)}{|X|-1} \leq \frac{1}{4}.$$

Applying Lemma 14, we obtain that the probability that a random set chosen uniformly from $\mathcal{F}$ is disjoint from A is at least $\gamma - \frac{1}{4}$. ◀

3.2 The Element Elimination Algorithm

A main ingredient in our fixed-parameter algorithm for the KNESER problem is the "element elimination" algorithm given by the following theorem. It will be used to repeatedly reduce the size of the ground set of a Kneser graph while looking for a monochromatic edge.

▶ **Theorem 16.** *There exists a randomized algorithm that given integers n and k, a set $X \subseteq [n]$ of size $|X| \geq 8k^4$, a set of colors $C \subseteq [n-2k+1]$ of size $|C| = |X| - 2k + 1$, and an oracle access to a coloring $c : \binom{X}{k} \to [n-2k+1]$ of the vertices of $K(\binom{X}{k})$, runs in time polynomial in n and returns, with probability $1 - 2^{-\Omega(n)}$,*

- **(a)** *a monochromatic edge of $K(\binom{X}{k})$, or*
- **(b)** *a vertex $A \in \binom{X}{k}$ satisfying $c(A) \notin C$, or*
- **(c)** *a color $i \in C$ and an element $j \in X$ such that for every $A \in \binom{[n]}{k}$ with $j \notin A$, a random vertex B chosen uniformly from $\binom{X}{k}$ satisfies $c(B) = i$ and $A \cap B = \emptyset$ with probability at least $\frac{1}{16n}$.*

Proof. For integers n and k, let $X \subseteq [n]$, $C \subseteq [n-2k+1]$, and $c : \binom{X}{k} \to [n-2k+1]$ be an input satisfying $|X| \geq 8k^4$ and $|C| = |X| - 2k + 1$ as in the statement of the theorem. It can be assumed that $k \geq 3$. Indeed, Theorem 7 guarantees that $K(\binom{X}{k})$ has either a monochromatic edge or a vertex whose color does not belong to C, hence for $k \leq 2$, an output of type (a) or (b) can be found by querying the oracle for the colors of all the vertices in time polynomial in n. For $k \geq 3$, consider the algorithm that given an input as above acts as follows (see Algorithm 1).

The algorithm first selects uniformly and independently m random sets $A_1, \ldots, A_m \in \binom{X}{k}$ for $m = n^3$ (see lines 1–2) and queries the oracle for their colors. If the sampled sets include two vertices that form a monochromatic edge, then the algorithm returns such an edge (output of type (a); see line 5). If they include a vertex whose color does not belong to C, then the algorithm returns it (output of type (b); see line 10). Otherwise, the algorithm defines $i^* \in C$ as a color that appears on a largest number of sampled sets A_t (see lines 13–16). It further defines $j^* \in X$ as an element that belongs to a largest number of sampled sets A_t with $c(A_t) = i^*$ (see lines 17–20). Then, the algorithm returns the pair (i^*, j^*) (output of type (c); see line 21).

It is clear that the algorithm runs in time polynomial in n. We turn to prove that for every input, the algorithm returns a valid output, of type (a), (b), or (c), with probability $1 - 2^{-\Omega(n)}$. We start with the following lemma that shows that if the input coloring has a large color class with no popular element, then with high probability the algorithm returns a valid output of type (a).

▶ **Lemma 17.** *Suppose that the input coloring c has a color class $\mathcal{F} \subseteq \binom{X}{k}$ of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \binom{|X|}{k}$ such that every element of X belongs to at most half of the sets of $\mathcal{F}$. Then, Algorithm 1 returns a monochromatic edge with probability $1 - 2^{-\Omega(n)}$.*

Algorithm 1 Element Elimination Algorithm (Theorem 16).

Input: Integers n and $k \geq 3$, a set $X \subseteq [n]$ of size $|X| \geq 8k^4$, a set of colors $C \subseteq [n-2k+1]$ of size $|C| = |X| - 2k + 1$, and an oracle access to a coloring $c : \binom{X}{k} \to [n-2k+1]$ of $K(\binom{X}{k})$.
Output: (a) A monochromatic edge of $K(\binom{X}{k})$, or (b) a vertex $A \in \binom{X}{k}$ satisfying $c(A) \notin C$, or (c) a color $i \in C$ and an element $j \in X$ such that for every $A \in \binom{[n]}{k}$ with $j \notin A$, a random vertex B chosen uniformly from $\binom{X}{k}$ satisfies $c(B) = i$ and $A \cap B = \emptyset$ with probability at least $\frac{1}{16n}$.

1: $m \leftarrow n^3$
2: pick uniformly and independently at random sets $A_1, \ldots, A_m \in \binom{X}{k}$
3: **for all** $t, t' \in [m]$ **do**
4: **if** $c(A_t) = c(A_{t'})$ and $A_t \cap A_{t'} = \emptyset$ **then**
5: **return** $\{A_t, A_{t'}\}$ ▷ output of type (a)
6: **end if**
7: **end for**
8: **for all** $t \in [m]$ **do**
9: **if** $c(A_t) \notin C$ **then**
10: **return** A_t ▷ output of type (b)
11: **end if**
12: **end for**
13: **for all** $i \in C$ **do**
14: $\widetilde{\alpha}_i \leftarrow \frac{1}{m} \cdot |\{t \in [m] \mid c(A_t) = i\}|$
15: **end for**
16: $i^* \leftarrow$ an $i \in C$ with largest value of $\widetilde{\alpha}_i$
17: **for all** $j \in X$ **do**
18: $\widetilde{\gamma}_{i^*,j} \leftarrow \frac{1}{m} \cdot |\{t \in [m] \mid c(A_t) = i^* \text{ and } j \in A_t\}|$
19: **end for**
20: $j^* \leftarrow$ a $j \in X$ with largest value of $\widetilde{\gamma}_{i^*,j}$
21: **return** (i^*, j^*) ▷ output of type (c)

Proof. Let $\mathcal{F}$ be as in the lemma. Applying Corollary 13 with $\gamma = \frac{1}{2}$, using the assumptions $k \geq 3$ and $|X| \geq 8k^4$, we obtain that two random sets chosen uniformly and independently from $\mathcal{F}$ are adjacent in $K(\mathcal{F})$ with probability at least $\frac{3}{8} \cdot (\frac{3}{4} - \gamma) = \frac{3}{32}$. Further, the fact that $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \binom{|X|}{k}$ implies that a random vertex chosen uniformly from $\binom{X}{k}$ belongs to $\mathcal{F}$ with probability at least $\frac{1}{2|X|}$. Hence, for two random vertices chosen uniformly and independently from $\binom{X}{k}$, the probability that they both belong to $\mathcal{F}$ is at least $(\frac{1}{2|X|})^2$, and conditioned on this event, their probability to form an edge in $K(\mathcal{F})$ is at least $\frac{3}{32}$. This implies that the probability that two random vertices chosen uniformly and independently from $\binom{X}{k}$ form a monochromatic edge in $K(\binom{X}{k})$ is at least $(\frac{1}{2|X|})^2 \cdot \frac{3}{32} = \frac{3}{128|X|^2}$.

Now, by considering $\lfloor m/2 \rfloor$ pairs of the random sets chosen by Algorithm 1 (line 2), it follows that the probability that no pair forms a monochromatic edge does not exceed

$$\left(1 - \frac{3}{128|X|^2}\right)^{\lfloor m/2 \rfloor} \leq e^{-3 \cdot \lfloor m/2 \rfloor / (128|X|^2)} \leq 2^{-\Omega(n)},$$

where the last inequality follows by $|X| \leq n$ and $m = n^3$. It thus follows that with probability $1 - 2^{-\Omega(n)}$, the algorithm returns a monochromatic edge, as required. ◀

We next handle the case in which every large color class of the input coloring has a popular element. To do so, we first show that the samples of the algorithm provide a good estimation for the fraction of vertices in each color class as well as for the fraction of vertices that share any given element in each color class. For every color $i \in C$, let α_i denote the fraction of vertices of $K(\binom{X}{k})$ colored i, that is,

$$\alpha_i = \frac{|\{A \in \binom{X}{k} \mid c(A) = i\}|}{\binom{|X|}{k}},$$

and let $\widetilde{\alpha}_i$ denote the fraction of the vertices sampled by the algorithm that are colored i (see line 14). Similarly, for every $i \in C$ and $j \in X$, let $\gamma_{i,j}$ denote the fraction of vertices of $K(\binom{X}{k})$ colored i that include j, that is,

$$\gamma_{i,j} = \frac{|\{A \in \binom{X}{k} \mid c(A) = i \text{ and } j \in A\}|}{\binom{|X|}{k}},$$

and let $\widetilde{\gamma}_{i,j}$ denote the fraction of the vertices sampled by the algorithm that are colored i and include j. Let E denote the event that

$$|\alpha_i - \widetilde{\alpha}_i| \leq \frac{1}{2|X|} \text{ and } |\gamma_{i,j} - \widetilde{\gamma}_{i,j}| \leq \frac{1}{2|X|} \text{ for all } i \in C,\ j \in X. \tag{2}$$

By a standard concentration argument, we obtain the following lemma.

▶ **Lemma 18.** *The probability of the event E is $1 - 2^{-\Omega(n)}$.*

Proof. By the Chernoff-Hoeffding bound (Theorem 11) applied with $\mu = \frac{1}{2|X|}$, the probability that an inequality from (2) does not hold is at most

$$2 \cdot e^{-2m/(4|X|^2)} \leq 2 \cdot e^{-n/2},$$

where the inequality follows by $|X| \leq n$ and $m = n^3$. By the union bound over all the colors $i \in C$ and all the pairs $(i, j) \in C \times X$, that is, over $|C| \cdot (1 + |X|) \leq n^2$ events, we get that all the inequalities in (2) hold with probability at least $1 - 2n^2 \cdot e^{-n/2} = 1 - 2^{-\Omega(n)}$, as required. ◀

We now show that if every large color class of the input coloring has a popular element and the event E occurs, then the algorithm returns a valid output.

▶ **Lemma 19.** *Suppose that the coloring c satisfies that for every color class $\mathcal{F} \subseteq \binom{X}{k}$ of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \binom{|X|}{k}$ there exists an element of X that belongs to more than half of the sets of $\mathcal{F}$. Then, if the event E occurs, Algorithm 1 returns a valid output.*

Proof. Assume that the event E occurs. If Algorithm 1 returns an output of type (a) or (b), i.e., a monochromatic edge or a vertex whose color does not belong to C, then the output is verified before it is returned and is thus valid. So suppose that the algorithm returns a pair $(i^*, j^*) \in C \times X$. Recall that the color i^* is defined by Algorithm 1 as an $i \in C$ with largest value of $\widetilde{\alpha}_i$ (see line 16). Since the colors of all the sampled sets belong to C, it follows that $\sum_{i \in C} \widetilde{\alpha}_i = 1$, and thus

$$\widetilde{\alpha}_{i^*} \geq \frac{1}{|C|} \geq \frac{1}{|X|}, \tag{3}$$

where the last inequality follows by $|C| = |X| - 2k + 1 \leq |X|$.

Let $\mathcal{F}$ be the family of vertices of $K(\binom{X}{k})$ colored i^*, i.e., $\mathcal{F} = \{A \in \binom{X}{k} \mid c(A) = i^*\}$. Since the event E occurs (see (2)), it follows from (3) that

$$|\mathcal{F}| = \alpha_{i^*} \cdot \binom{|X|}{k} \geq \left(\widetilde{\alpha}_{i^*} - \frac{1}{2|X|}\right) \cdot \binom{|X|}{k} \geq \frac{1}{2|X|} \cdot \binom{|X|}{k}.$$

Hence, by the assumption of the lemma, there exists an element $j \in X$ that belongs to more than half of the sets of $\mathcal{F}$, that is, $\gamma_{i^*,j} > \frac{1}{2}$. Since the event E occurs, it follows that this j satisfies $\widetilde{\gamma}_{i^*,j} > \frac{1}{2} - \frac{1}{2|X|}$. Recalling that the element j^* is defined by Algorithm 1 as a $j \in X$ with largest value of $\widetilde{\gamma}_{i^*,j}$ (see line 20), it must satisfy $\widetilde{\gamma}_{i^*,j^*} > \frac{1}{2} - \frac{1}{2|X|}$, and using again the fact that the event E occurs, we derive that $\gamma_{i^*,j^*} \geq \widetilde{\gamma}_{i^*,j^*} - \frac{1}{2|X|} > \frac{1}{2} - \frac{1}{|X|} \geq \frac{3}{8}$ (using $|X| \geq 8$).

By $k \geq 3$ and $|X| \geq 8k^4$, we can apply Corollary 15 with $\mathcal{F}$, j^*, and $\gamma = \frac{3}{8}$ to obtain that for every set $A \in \binom{[n]}{k}$ with $j^* \notin A$, the probability that a random set chosen uniformly from $\mathcal{F}$ is disjoint from A is at least $\gamma - \frac{1}{4} = \frac{1}{8}$. Since the probability that a random set chosen uniformly from $\binom{X}{k}$ belongs to $\mathcal{F}$ is at least $\frac{1}{2|X|}$, it follows that the probability that a random set B chosen uniformly from $\binom{X}{k}$ satisfies $c(B) = i^*$ and $A \cap B = \emptyset$ is at least $\frac{1}{2|X|} \cdot \frac{1}{8} = \frac{1}{16|X|} \geq \frac{1}{16n}$. This implies that (i^*, j^*) is a valid output of type (c), as required. ◀

Equipped with Lemmas 17, 18, and 19, we are ready to derive the correctness of Algorithm 1 and to complete the proof of Theorem 16. If the input coloring c has a color class $\mathcal{F} \subseteq \binom{X}{k}$ of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \binom{|X|}{k}$ such that every element of X belongs to at most half of the sets of $\mathcal{F}$, then, by Lemma 17, the algorithm returns with probability $1 - 2^{-\Omega(n)}$ a monochromatic edge, i.e., a valid output of type (a). Otherwise, the input coloring c satisfies that for every color class $\mathcal{F} \subseteq \binom{X}{k}$ of size $|\mathcal{F}| \geq \frac{1}{2|X|} \cdot \binom{|X|}{k}$ there exists an element of X that belongs to more than half of the sets of $\mathcal{F}$. By Lemma 18, the event E occurs with probability $1 - 2^{-\Omega(n)}$, implying by Lemma 19 that with such probability, the algorithm returns a valid output. It thus follows that for every input coloring the algorithm returns a valid output with probability $1 - 2^{-\Omega(n)}$, and we are done. ◀

3.3 The Fixed-Parameter Algorithm for the Kneser Problem

We turn to present our fixed-parameter algorithm for the KNESER problem and to complete the proof of Theorem 1.

Proof of Theorem 1. Suppose that we are given, for integers $n \geq 2k$, an oracle access to a coloring $c : \binom{[n]}{k} \to [n - 2k + 1]$ of the vertices of the Kneser graph $K(n, k)$. Our algorithm has two phases, as described below (see Algorithm 2).

In the first phase, the algorithm repeatedly applies the "element elimination" algorithm given in Theorem 16 (Algorithm 1). Initially, we define

$$s = \max(n - 8k^4, 0), \quad X_0 = [n], \quad \text{and} \quad C_0 = [n - 2k + 1].$$

In the lth iteration, $0 \leq l < s$, we call Algorithm 1 with n, k, X_l, C_l, and with the restriction of the given coloring c to the vertices of $\binom{X_l}{k}$ to obtain with probability $1 - 2^{-\Omega(n)}$,

(a) a monochromatic edge $\{A, B\}$ of $K(\binom{X_l}{k})$, or

(b) a vertex $A \in \binom{X_l}{k}$ satisfying $c(A) \notin C_l$, or

(c) a color $i_l \in C_l$ and an element $j_l \in X_l$ such that for every $A \in \binom{[n]}{k}$ with $j_l \notin A$, a random vertex B chosen uniformly from $\binom{X_l}{k}$ satisfies $c(B) = i_l$ and $A \cap B = \emptyset$ with probability at least $\frac{1}{16n}$.

Algorithm 2 The Algorithm for the KNESER Problem (Theorem 1).

Input: Integers n, k with $n \geq 2k$ and an oracle access to a coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of $K(n,k)$.
Output: A monochromatic edge of $K(n,k)$.

1: $s \leftarrow \max(n - 8k^4, 0)$, $X_0 \leftarrow [n]$, $C_0 \leftarrow [n-2k+1]$ ▷ $|C_0| = |X_0| - 2k + 1$
2: **for all** $l = 0, 1, \ldots, s-1$ **do** ▷ first phase
3: **call** Algorithm 1 with n, k, X_l, C_l, and with the restriction of c to $\binom{X_l}{k}$
4: **if** Algorithm 1 returns an edge $\{A, B\}$ with $c(A) = c(B)$ **then** ▷ output of type (a)
5: **return** $\{A, B\}$
6: **end if**
7: **if** Algorithm 1 returns a vertex $A \in \binom{X_l}{k}$ with $c(A) = i_r \notin C_l$ **then** ▷ output of type (b)
8: **for all** $t \in [n^2]$ **do**
9: pick uniformly at random a set $B_t \in \binom{X_r}{k}$
10: **if** $c(B_t) = i_r$ and $A \cap B_t = \emptyset$ **then**
11: **return** $\{A, B_t\}$
12: **end if**
13: **end for**
14: **return** "failure"
15: **end if**
16: **if** Algorithm 1 returns a pair $(i_l, j_l) \in C_l \times X_l$ **then** ▷ output of type (c)
17: $X_{l+1} \leftarrow X_l \setminus \{j_l\}$, $C_{l+1} \leftarrow C_l \setminus \{i_l\}$ ▷ $|C_{l+1}| = |X_{l+1}| - 2k + 1$
18: **end if**
19: **end for**
20: query the oracle for the colors of all the vertices of $K(\binom{X_s}{k})$ ▷ second phase
21: **if** there exists a vertex $A \in \binom{X_s}{k}$ of color $c(A) = i_r \notin C_s$ **then**
22: **for all** $t \in [n^2]$ **do**
23: pick uniformly at random a set $B_t \in \binom{X_r}{k}$
24: **if** $c(B_t) = i_r$ and $A \cap B_t = \emptyset$ **then**
25: **return** $\{A, B_t\}$
26: **end if**
27: **end for**
28: **return** "failure"
29: **else**
30: find $A, B \in \binom{X_s}{k}$ satisfying $c(A) = c(B)$ and $A \cap B = \emptyset$ ▷ exist by Theorem 7 [19]
31: **return** $\{A, B\}$
32: **end if**

As will be explained shortly, if the output of Algorithm 1 is of type (a) or (b) then we either return a monochromatic edge or declare "failure", and if the output is a pair (i_l, j_l) of type (c) then we define $X_{l+1} = X_l \setminus \{j_l\}$ and $C_{l+1} = C_l \setminus \{i_l\}$ and, as long as $l < s$, proceed to the next call of Algorithm 1. Note that the sizes of the sets X_l and C_l are reduced by 1 in every iteration, hence we maintain the equality $|C_l| = |X_l| - 2k + 1$ for all l. We now describe how the algorithm acts in the lth iteration for each type of output returned by Algorithm 1.

If the output is of type (a), then the returned monochromatic edge of $K(\binom{X_l}{k})$ is also a monochromatic edge of $K(n,k)$, so we return it (see lines 4–6).

If the output is of type (b), then we are given a vertex $A \in \binom{X_l}{k}$ satisfying $c(A) = i_r \notin C_l$ for some $r < l$. Since $i_r \notin C_l$, it follows that $j_r \notin X_l$, and thus $j_r \notin A$. In this case, we pick uniformly and independently n^2 random sets from $\binom{X_r}{k}$ and query the oracle for their colors. If we find a vertex B that forms together with A a monochromatic edge in $K(n,k)$, we return the monochromatic edge $\{A, B\}$, and otherwise we declare "failure" (see lines 7–15).

If the output of Algorithm 1 is a pair (i_l, j_l) of type (c), then we define, as mentioned above, the sets $X_{l+1} = X_l \setminus \{j_l\}$ and $C_{l+1} = C_l \setminus \{i_l\}$ (see lines 16–18). Observe that for $0 \leq l < s$, it holds that $|X_l| = n - l > n - s = 8k^4$, allowing us, by Theorem 16, to call Algorithm 1 in the lth iteration.

In case that all the s calls to Algorithm 1 return an output of type (c), we arrive to the second phase of the algorithm. Here, we are given the sets X_s and C_s that satisfy $|X_s| = n - s \leq 8k^4$ and $|C_s| = |X_s| - 2k + 1$, and we query the oracle for the colors of each and every vertex of the graph $K(\binom{X_s}{k})$. If we find a vertex $A \in \binom{X_s}{k}$ satisfying $c(A) = i_r \notin C_s$ for some $r < s$, then, as before, we pick uniformly and independently n^2 random sets from $\binom{X_r}{k}$ and query the oracle for their colors. If we find a vertex B that forms together with A a monochromatic edge in $K(n,k)$, we return the monochromatic edge $\{A, B\}$, and otherwise we declare "failure" (see lines 21–28). Otherwise, all the vertices of $K(\binom{X_s}{k})$ are colored by colors from C_s. By Theorem 7, the chromatic number of $K(\binom{X_s}{k})$ is $|X_s| - 2k + 2 > |C_s|$. Hence, there must exist a monochromatic edge in $K(\binom{X_s}{k})$, and by checking all the pairs of its vertices we find such an edge and return it (see lines 30–31).

We turn to analyze the probability that Algorithm 2 returns a monochromatic edge. Note that whenever the algorithm returns an edge, it checks that it is monochromatic and thus ensures that it forms a valid solution. Hence, it suffices to show that the algorithm declares "failure" with probability $2^{-\Omega(n)}$. To see this, recall that the algorithm calls Algorithm 1 at most $s < n$ times, and that by Theorem 16 the probability that its output is not valid is $2^{-\Omega(n)}$. By the union bound, the probability that any of the calls to Algorithm 1 returns an invalid output is $2^{-\Omega(n)}$ too. The only situation in which Algorithm 2 declares "failure" is when it finds, for some $r < s$, a vertex $A \in \binom{[n]}{k}$ with $c(A) = i_r$ and $j_r \notin A$, and none of the n^2 sampled sets $B \in \binom{X_r}{k}$ satisfies $c(B) = i_r$ and $A \cap B = \emptyset$ (see lines 7–15, 21–28). However, assuming that all the calls to Algorithm 1 return valid outputs, the rth run guarantees, by Theorem 16, that a random vertex B uniformly chosen from $\binom{X_r}{k}$ satisfies $c(B) = i_r$ and $A \cap B = \emptyset$ for the given A with probability at least $\frac{1}{16n}$. Hence, the probability that the algorithm declares "failure" does not exceed $(1 - \frac{1}{16n})^{n^2} \leq e^{-n/16} = 2^{-\Omega(n)}$. Using again the union bound, it follows that the probability that Algorithm 2 either gets an invalid output from Algorithm 1 or fails to find a vertex that forms a monochromatic edge with a set A as above is $2^{-\Omega(n)}$. Therefore, the probability that Algorithm 2 successfully finds a monochromatic edge is $1 - 2^{-\Omega(n)}$, as desired.

We finally analyze the running time of Algorithm 2. In its first phase, the algorithm calls Algorithm 1 at most $s < n$ times, where the running time needed for each call is, by Theorem 16, polynomial in n. It is clear that the other operations made throughout this phase can also be implemented in time polynomial in n. In its second phase, the algorithm enumerates all the vertices of $K(\binom{X_s}{k})$. This phase can be implemented in running time polynomial in n and in the number of vertices of this graph. The latter is $\binom{|X_s|}{k} \leq |X_s|^k \leq (8k^4)^k = k^{O(k)}$. It thus follows that the total running time of Algorithm 2 is $n^{O(1)} \cdot k^{O(k)}$, completing the proof. ◀

3.4 Turing Kernelization for the Kneser Problem

As mentioned in the introduction, our algorithm for the Kneser problem can be viewed as a randomized polynomial Turing kernelization algorithm for the problem. In what follows we extend on this aspect of the algorithm.

We start with the definition of a Turing kernelization algorithm as it is used in the standard context of parameterized complexity (see, e.g., [12, Chapter 22]). A decision parameterized problem P is a language of pairs (x, k) where k is an integer referred to as the parameter of P. For a decision parameterized problem P and a computable function f, a Turing kernelization algorithm of size f for P is an algorithm that decides whether an input (x, k) belongs to P in polynomial time given an oracle access that decides membership in P for instances (x', k') with $|x'| \leq f(k)$ and $k' \leq f(k)$. The kernelization algorithm is said to be polynomial if f is a polynomial function.

The Kneser problem, however, is a total search problem whose input is given as an oracle access. Hence, for an instance of the Kneser problem associated with integers n and k, we require a Turing kernelization algorithm to find a solution using an oracle that finds solutions for instances associated with integers n' and k' that are bounded by a function of k. We further require the algorithm to be able to simulate the queries of the produced instances using queries to the oracle associated with the original input. Note that we have here two different types of oracles: the oracle that solves the Kneser problem on graphs $K(n', k')$ with bounded n', k' and the oracle that supplies an access to the instance of the Kneser problem.

We claim that the proof of Theorem 1 shows that the Kneser problem admits a randomized polynomial Turing kernelization algorithm in the following manner. Given an instance of the Kneser problem, a coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of the vertices of a Kneser graph $K(n, k)$ for integers n and k with $n \geq 2k$, the first phase of Algorithm 2 runs in time polynomial in n and either finds a monochromatic edge or produces a ground set $X_s \subseteq [n]$ and a set of colors $C_s \subseteq [n-2k+1]$ satisfying $|C_s| = |X_s| - 2k + 1$ and $|X_s| = O(k^4)$ (see lines 2–19). In the latter case, the restriction of the input coloring c to the vertices of the graph $K(\binom{X_s}{k})$ is not guaranteed to use only colors from C_s, as required by the definition of the Kneser problem. Yet, by applying the oracle to this restriction of c, simulating its queries using the access to the coloring c of $K(n, k)$, we either get a monochromatic edge in $K(\binom{X_s}{k})$ or find a vertex whose color does not belong to C_s. In both cases, a solution to the original instance can be efficiently found with high probability. Indeed, if a monochromatic edge is returned then it forms a monochromatic edge of $K(n, k)$ as well. Otherwise, as shown in the analysis of Algorithm 2, a vertex $A \in \binom{X_s}{k}$ whose color does not belong to C_s can be used to efficiently find with high probability a vertex $B \in \binom{[n]}{k}$ such that $\{A, B\}$ is a monochromatic edge (see lines 21–28).

4 An Algorithm for the Kneser Problem Based on Schrijver Graphs

In this section we present the simple deterministic algorithm for the Kneser problem given in Theorem 2. Let us first state a simple fact on the number of vertices in Schrijver graphs (see Section 2.2). We include a quick proof for completeness.

▶ **Fact 20.** *For integers $k \geq 2$ and $n \geq 2k$, the number of vertices in $S(n, k)$ is*

$$\binom{n-k+1}{k} - \binom{n-k-1}{k-2}.$$

Proof. Recall that the vertex set $\binom{[n]}{k}_{\text{stab}}$ of $S(n, k)$ is the collection of all k-subsets of $[n]$ with no two consecutive elements modulo n. We first observe that the number of k-subsets of $[n]$ with no two consecutive elements, allowing both 1 and n to be in the subsets, is $\binom{n-k+1}{k}$. To see this, identify the subsets of $[n]$ with their characteristic vectors in $\{0, 1\}^n$, and in every such vector, interpret the zeros as balls and the ones as separations between bins. It

follows that every such set corresponds to a partition of $n-k$ identical balls into $k+1$ bins, where no bin but the first and last is empty. The number of those partitions is equal to the number of partitions of $n-2k+1$ identical balls into $k+1$ bins, which is $\binom{n-k+1}{k}$. Finally, to obtain the number of vertices of $S(n,k)$, one has to subtract the number of k-subsets of $[n]$ with no two consecutive elements that include both 1 and n. The latter is equal to the number of $(k-2)$-subsets of $[n-4]$ with no two consecutive elements, which by the above argument equals $\binom{n-k-1}{k-2}$, so we are done. ◀

Equipped with Fact 20, we are ready to prove Theorem 2.

Proof of Theorem 2. Consider the algorithm that given integers n and k with $n \geq 2k$ and an oracle access to a coloring $c : \binom{[n]}{k} \to [n-2k+1]$ of the vertices of the Kneser graph $K(n,k)$, enumerates all vertices of the Schrijver graph $S(n,k)$, i.e., the sets of $\binom{[n]}{k}_{\text{stab}}$, and queries the oracle for their colors. Then, the algorithm goes over all pairs of those vertices and returns a pair that forms a monochromatic edge. The existence of such an edge follows from Theorem 7, which asserts that $\chi(S(n,k)) = n-2k+2$. The running time of the algorithm is polynomial in the number of vertices of $S(n,k)$, which by Fact 20 does not exceed $\binom{n-k+1}{k} = \binom{n-k+1}{n-2k+1} \leq n^{\min(k,n-2k+1)}$. ◀

References

1. Paul Beame, Stephen A. Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *J. Comput. Syst. Sci.*, 57(1):3–19, 1998. Preliminary version in STOC'95.
2. Karol Borsuk. Drei Sätze über die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 20(1):177–190, 1933.
3. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
4. Argyrios Deligkas, John Fearnley, Alexandros Hollender, and Themistoklis Melissourgos. Constant inapproximability for PPA. In *Proc. of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC'22)*, 2022.
5. Argyrios Deligkas, Aris Filos-Ratsikas, and Alexandros Hollender. Two's company, three's a crowd: Consensus-halving for a constant number of agents. In *Proc. of the 22nd ACM Conference on Economics and Computation (EC'21)*, pages 347–368, 2021.
6. Xiaotie Deng, Zhe Feng, and Rucha Kulkarni. Octahedral Tucker is PPA-complete. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:118, 2017.
7. Irit Dinur and Ehud Friedgut. Intersecting families are essentially contained in juntas. *Comb. Probab. Comput.*, 18(1–2):107–122, 2009.
8. Paul Erdös, Chao Ko, and Richard Rado. Intersection theorems for systems of finite sets. *Quart. J. Math.*, 12(1):313–320, 1961.
9. Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is PPA-complete. In *Proc. of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC'18)*, pages 51–64, 2018.
10. Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proc. of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC'19)*, pages 638–649, 2019.
11. Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. A topological characterization of modulo-p arguments and implications for necklace splitting. In *Proc. of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*, pages 2615–2634, 2021.
12. Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

13 Peter Frankl and Andrey Kupavskii. Maximal degrees in subgraphs of Kneser graphs. *arXiv*, abs/2004.08718, 2020. `arXiv:2004.08718`.

14 Paul W. Goldberg, Alexandros Hollender, Ayumi Igarashi, Pasin Manurangsi, and Warut Suksompong. Consensus halving for sets of items. In *Proc. 16th Web and Internet Economics International Conference (WINE'20)*, pages 384–397, 2020.

15 Ishay Haviv. The complexity of finding fair independent sets in cycles. In *12th Innovations in Theoretical Computer Science Conference (ITCS'21)*, pages 4:1–4:14, 2021.

16 Anthony J. W. Hilton and Eric Charles Milner. Some intersection theorems for systems of finite sets. *Quart. J. Math.*, 18(1):369–384, 1967.

17 Martin Kneser. Aufgabe 360. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 58(2):27, 1955.

18 Andrey Kupavskii. Diversity of uniform intersecting families. *Eur. J. Comb.*, 74:39–47, 2018.

19 László Lovász. Kneser's conjecture, chromatic number, and homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978.

20 Pasin Manurangsi and Warut Suksompong. Computing a small agreeable set of indivisible items. *Artif. Intell.*, 268:96–114, 2019. Preliminary versions in IJCAI'16 and IJCAI'17.

21 Jiří Matoušek. A combinatorial proof of Kneser's conjecture. *Combinatorica*, 24(1):163–170, 2004.

22 Jiří Matoušek. *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*. Springer Publishing Company, Incorporated, 2007.

23 Jiří Matoušek and Günter M. Ziegler. Topological lower bounds for the chromatic number: A hierarchy. *Jahresbericht der DMV*, 106(2):71–90, 2004.

24 Colin McDiarmid. Concentration. In *Probabilistic methods for algorithmic discrete mathematics*, volume 16 of *Algorithms Combin.*, pages 195–248. Springer, Berlin, 1998.

25 Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.

26 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

27 Alexander Schrijver. Vertex-critical subgraphs of Kneser graphs. *Nieuw Arch. Wiskd.*, 26(3):454–461, 1978.

28 Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Math. Soc. Sci.*, 45(1):15–25, 2003.

Delegation for Search Problems

Justin Holmgren
NTT Research, Sunnyvale, CA, USA

Andrea Lincoln
University of California Berkeley, CA, USA

Ron D. Rothblum ✉
Technion, Haifa, Israel

Abstract

The theory of proof systems in general, and interactive proofs in particular, has been immensely influential. Such proof systems allow a prover to convince a verifier whether a given statement is true or not – namely to solve a decision problem. In this work we initiate a study of interactive proofs for *search problems*.

More precisely, we consider a setting in which a client C, given an input x, would like to find a solution y satisfying $(x, y) \in R$, for a given relation R. The client wishes to delegate this work to an (untrusted) advisor A, who has more resources than C. We seek solutions in which the communication from A is short, and, in particular, shorter than the length of the output y. (In particular, this precludes the trivial solution of the advisor sending y and then proving that $(x, y) \in R$ using a standard interactive proof.)

We show that such search delegation schemes exist for several problems of interest including (1) longest common subsequence (LCS) and edit distance, (2) parsing context-free grammars and (3) k-SAT.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Interactive Proofs, Fine-Grained Complexity, Delegation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.73

Category Track A: Algorithms, Complexity and Games

Funding *Ron D. Rothblum*: Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We thank Benny Applebaum for helpful discussions.

1 Introduction

Interactive proofs, conceived by Goldwasser, Micali and Rackoff [13], allow a prover to convince a verifier that a given computational statement of the form $x \in L$ is true, where L is a language. They require that if the statement is true, then there is a strategy that will convince the verifier to accept with high probability; whereas if the statement is false, then the verifier should reject with high probability no matter what the prover does. Interactive proofs have had an incredible and enduring impact on complexity theory, cryptography, and beyond.

While interactive proofs help the verifier to determine whether a given statement is true or not, often we are interested in actually finding a solution: fixing some relation R of interest, we would like given an input x to find a solution $y \in R(x)$, where $R(x) := \{y : (x, y) \in R\}$. In other words, we are interested in solving *search problems*.


In this work, we consider a setting in which a computationally bounded client C is given an input x and would like to find some $y \in R(x)$ with help from a powerful advisor A.[1] We refer to a protocol for this setting as a "search delegation scheme" and note that the notion has immediate connections to cloud computing. Continuing the parallel to interactive proofs, our main focus is on protocols in which the advisor is viewed as untrusted. However, as discussed below, protocols that assume a trusted advisor are also non-trivial, and arguably even more natural.

One trivial approach to designing search delegation schemes is for the advisor to simply compute some $y \in R(x)$, send y to the client, and then prove that indeed $y \in R(x)$. The downside of this solution is that y may be so long that just sending y may already be extremely costly. Thus, in this work we are interested in search delegation schemes with *laconic* advisors, i.e. advisors that are restricted to sending a short (h-bit) hint, where $h \ll |y|$ is a parameter. Note that an h-bit hint can at most increase the probability with which the client successfully computes some $y \in R(x)$ by a multiplicative factor of 2^h. We seek to understand how much of this increase is actually realizable, and for which values of h.

The restriction that the advisor can only send a short hint makes the construction of search delegation schemes, even with a trusted advisor, algorithmically non-trivial. Moreover, given a search delegation scheme with a trusted advisor, we can in many cases upgrade the scheme (using interactive proofs or arguments generically and in a black-box way) to provide correctness guarantees to the client even when interacting with an untrusted advisor. In fact protocols for trusted advisors comprise the technical heart of our constructions, even those for untrusted advisors.

Before proceeding, we mention several prior works that can be cast as studying proof-systems for search problems, but with communication that is as long as the solution. First, the search version of NP (e.g., finding a satisfying solution to a formula, or a 3-coloring of a graph), often referred to as FNP or PC [11], consists of search problems whose solutions are efficiently verifiable. However, no distinction is made between the hint length vs. the length of the solution. There are also several works in cryptographic contexts that consider using interaction for solving search problems (e.g., [10, 8, 4]). Additionally, there has been a study of fine-grained complexity and approximation that connects the fine-grained hardness of approximation to interactive proof systems for space-bounded computation [7]. However, in these works the communication is as large as the output y (similarly to the canonical solution in which the result is sent by the advisor and then verified).

1.1 Search Delegation

We define a search delegation scheme as follows:

▶ **Definition 1.** *A* search delegation scheme *for a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ consists of a client C and an advisor A. The two parties interact (possibly using randomness) on common input x and at the end of the interaction the client outputs a string denoted by $(C, A)(x)$. The delegation scheme has completeness error $c = c(|x|)$ and soundness error $s = s(|x|)$ if:*

- *(Completeness:) For every input x such that $R(x) \neq \emptyset$, it holds that $(C, A)(x) \in R(x)$ with probability at least $1 - c(|x|)$.*
- *(Soundness:) For every input x and every (potentially malicious) advisor A^*, it holds that $(C, A^*)(x) \in R(x) \cup \{\perp\}$ with probability at least $1 - s(|x|)$.*

[1] We use the nomenclature of client/advisor rather than verifier/prover since the goal of the interaction is not to prove correctness but rather to find a solution. Nevertheless, we emphasize that the role of the client (resp., advisor) is analogous to that of the verifier (resp., prover).

Here and throughout this work the $\perp$ symbol represents a "reject" by the client. In case the errors are not specified explicitly we default to *perfect completeness* (i.e., $c \equiv 0$) and *negligible soundness error* (i.e. $s(n) = n^{-\omega(1)}$).

The main resources that we focus on are the running time of the client, and the amount of communication between the parties (a secondary goal is bounding the running time of the (honest) advisor).

1.1.1 Variant Definitions

As discussed above, we will mainly be interested in search delegation schemes in which the communication complexity is less than the output length of the relation. Given this, Definition 1 is interesting even if we only require completeness to hold (without any guarantees against cheating advisors). We refer to schemes satisfying this weaker definition as *honest-advisor search delegation schemes*.

Actually, it will be convenient to present many of the protocols in this work as such honest-advisor search delegation schemes. Note that in case the relevant relation R has an efficient interactive proof for proving that $(x, y) \in R$ (with communication $\ll |y|$), then we can easily "bootstrap" an honest-advisor scheme into a full-fledged one. See Section 5 for details.

We will also sometimes consider a relaxation of the soundness condition of Definition 1 in which soundness is only required to hold only against malicious advisors A^* that are polynomially bounded. This is similar to the notion of computationally sound proofs, aka arguments, from the literature. We refer to protocols satisfying this weaker notion as computational search delegation schemes.

Lastly we remark that in contrast to standard interactive proofs, it is not clear how to reduce the completeness and soundness errors in search delegation schemes. The problem is that if the basic protocol is repeated, each invocation may yield a different candidate solution. While in general we do not know how to check which (if any) of the solutions is valid, in cases for which there *is* an (efficient) method of doing so, we can reduce the errors by repetition.

1.2 Our Results

We construct search delegation schemes for a number of problems of interest from the literature. In all of our schemes the client is given a hint whose length is sub-linear in the output length of the problem, and runs in significantly less time than the best known algorithms for the problem.

1.2.1 Longest Common Subsequence and Edit Distance

A *common subsequence* between strings $x \in \Sigma^n$ and $y \in \Sigma^m$, over an alphabet Σ, is a string z that appears in both x and y as a (possibly non-contiguous) sub-string. The *longest common subsequence* (LCS) problem asks, given x and y, to find a common subsequence of maximal length. When $n = m$, it is known that LCS requires $n^{2-o(1)}$ time if the strong exponential time hypothesis (SETH) holds [1].

The *edit distance* (Edit) problem is a related and central problem with important applications to computational biology. We consider a search version of the problem in which the goal is, given strings x and y, to find a minimal sequence of operations for transforming x into y, where the allowed operations are single-character insertions, deletions, and substitutions. Similar to LCS, there is a known running time lower bound for Edit of $n^{2-o(1)}$ based on SETH [5].

Our first result is an honest-advisor search delegation schemes for solving both LCS and Edit. We show how to solve both problems given a sub-linear hint string and so that the client runs in $o(n \cdot m)$ time. More generally, for every parameter a, we construct a protocol with an a-bit hint and running time $O\big(n+m+a+\frac{n \cdot m}{a} \cdot \log(n \cdot m)\big)$. For example, specializing to the case that $n = m$ and taking $a = n/\mathsf{polylog}(n)$ we get sub-linear communication $n/\mathsf{polylog}(n)$ with quasi-linear client running time.

Furthermore, using cryptographic techniques we can bootstrap the protocols to hold against computationally bounded cheating advisors, with essentially the same parameters (assuming the existence of collision-resistant hash functions and using additional rounds of interaction).

1.2.2 Parsing Context-Free Grammars

Context-free grammars are a computational model which is particular well suited for describing programming language structure. A context-free grammar is composed of production rules of the form $A \to \alpha$, where A is a variable and α is a string composed of variables and "terminals" (aka alphabet symbols). A word is derived by a grammar by applying production rules starting with some initial variable by replacing each occurrence of a variable using a corresponding production rule for that variable. The final word, composed only of terminals (i.e., after all variables have been replaced), is said to be derived by the grammar. We follow the convention that variables are described in upper case and terminals in lowercase.

A derivation tree (aka parse tree) is a tree describing the derivation of the word - namely, the root of the tree is labeled by the initial variable and the children of each vertex are labeled based on a production rule applied to the label of the parent. Thus, the leaves of the tree are labeled by the terminals of the derived word. The important computational task of *parsing* is, given a description of a grammar and a word w, to output a derivation tree describing the derivation of w using the grammar (or to output $\perp$ if the word cannot be derived by the grammar).

The best known algorithm for parsing context-free grammars (in Chomsky[2] normal form), due to Valiant [20], takes time $O(n^{\omega_b})$, where $\omega_b \geq 2$ is the exponent for Boolean matrix multiplication[3]. There is also evidence that a substantially faster algorithm does not exist [17, 2].

As our second main result, we construct a search delegation scheme for context-free parsing (of Chomsky normal form grammars) with sub-linear communication and quasi-linear work for the client. In more detail, for every parameter a, we construct a full-fledged search delegation scheme (even against computationally unbounded cheating advisors) in which the advisor sends a bits and the clients running time is $\tilde{O}(a + n^{\omega_b}/a^{\omega_b - 1})$. For example with $a = n/\mathsf{polylog}(n)$ we get quasi-linear client time, and for $a = n^{\frac{\omega_b - 2}{\omega_b - 1}} < \sqrt{n}$ we obtain a quadratic-time client with $\tilde{O}(\sqrt{n})$ communication.

---

[2] A grammar is in Chomsky normal form if the production rules only have one of the following three forms: $A \to BC$, $A \to a$, or $A \to \varepsilon$, where ε denotes the empty string. Recall that any grammar can be readily modified to be in Chomsky normal form.

[3] Boolean matrix multiplication is defined similarly to standard matrix multiplication except that the inner product operation is replaced by an OR of ANDs. The boolean matrix multiplication exponent is defined as the minimum number, ω_b, such that a $n^{\omega_b + o(1)}$ algorithm exists. The best known upper bound on ω_b is roughly 2.37286 [3].

1.2.3 k-SAT

The search version of the k-SAT problem is to find a satisfying assignment for a given k-CNF formula (i.e., a CNF boolean formula whose clauses contain at most k literals). The problem is well-known to be NP complete when $k \geq 3$. Moreover if SETH holds, then for all $\epsilon > 0$ there exists k such that k-SAT cannot be solved in time $O(2^{(1-\epsilon)n})$.

Nevertheless, there has been a fascinating line of work studying non-trivial algorithms for k-SAT for fixed constant k. The current fastest algorithm for k-SAT builds on PPSZ [14]. The PPSZ algorithm [19] is another classical, albeit slightly slower, algorithm whose running time is $\approx 2^{c_k \cdot n}$, where c_k tends to $1 - \Theta(1/k)$.

A naive approach for a search delegation scheme for k-SAT would be for the advisor to simply specify the value of a of the variables and then having the client solve the residual formula using a generic k-SAT algorithm. For example, using PPSZ we would obtain a client run-time of $\approx 2^{c_k \cdot (n-a)}$.

We give a non-trivial improvement on this scheme, in which given an a-bit hint, the client runs in time $2^{c_k \cdot n - a + o(n)}$. Denoting the run-time of the best known algorithm for k-SAT by $2^{c'_k \cdot n + o(n)}$, we get that the client beats the performance of the naive solution whenever $a > \frac{c_k - c'_k}{1 - c'_k} \cdot n$. We remark that for this result we assume the server and client have access to shared randomness of length $\Omega(n)$.

1.2.4 Separations

In addition to these positive results, we also give negative results indicating that solving a search problem (finding $y \in R(x)$) with a short hint can be much harder than solving the corresponding decision problem (i.e. checking whether a given y satisfies $y \in R(x)$). Our separation results all depend on cryptographic assumptions and are based on the elementary fact that an a-bit hint can boost an algorithm's success probability by at most a multiplicative factor of 2^a.

1.3 Organization

In Section 2 we describe our delegation protocols for LCS and Edit. We continue in Section 3 to describe our protocol for parsing context-free grammars, and in Section 4 we describe our protocol for k-SAT.

2 LCS and Edit Distance

The Longest Common Subsequence (LCS) problem asks, given strings $x \in \Sigma^n$ and $y \in \Sigma^m$ (over a common alphabet Σ), to find increasing sequences $i_1, \ldots, i_\ell \in [n]$ and $j_1, \ldots, j_\ell \in [m]$, for ℓ as big as possible, such that $x_{i_k} = y_{j_k}$ for all $k \in [\ell]$. The best known algorithms for LCS run in worst-case $O(n \cdot m)$ time, and when $m = n$ this is nearly optimal – LCS requires $n^{2-o(1)}$ time if the strong exponential time hypothesis (SETH) holds [1].

The search version of the Edit Distance problem, which we denote by Edit, asks, given strings $\mathbf{x} \in \Sigma^n$ and $\mathbf{y} \in \Sigma^m$, to find a minimum length sequence of operations that transforms $\mathbf{x}$ into $\mathbf{y}$, where allowed operations are single character insertions, deletions, and substitutions. Edit is also known to be solvable in $O(n \cdot m)$ time, and to require at least $n^{2-o(1)}$ time if SETH holds [5].

In this section we construction search delegation protocols for both LCS and Edit.

▶ **Theorem 2.** *For every $a = a(n, m)$, there is an honest-advisor search delegation scheme for* LCS *and for* Edit *in which the advisor sends an a-bit hint, and the client runs in time $O\big(n + m + a + \frac{nm \log(nm)}{a}\big)$.*

Furthermore assuming collision-resistant hash functions, the protocols can be improved to an untrusted-advisor search delegation scheme with computational soundness.

We start in Section 2.1 with an overview for the protocol for LCS. The remaining sections are devoted to the formal proof of Theorem 2. Since the protocols for LCS and Edit are similar, we present a common framework that captures both problems and is described in Section 2.2.

2.1 Protocol Overview

The idea underlying both protocols is to use trusted advice to facilitate a "divide-and-conquer" approach. This yields an honest advisor delegation protocol which in turn can be transformed, via Corollary 16, to a full-fledged protocol with computational soundness (assuming the existence of collision resistant hash functions).

For sake of this overview, we restrict our attention to LCS and describe a simple protocol with short advice ($O(\log(n) + \log(m))$ bits) that reduces the client's work by a factor of 2. The protocol for Edit is very similar and for both protocols the client's work can be further reduced by recursion, see the subsequent subsections for details.

Given input $(\mathbf{x}, \mathbf{y}) \in \{0,1\}^n \times \{0,1\}^m$, the trusted advisor first computes a longest common subsequence $\mathbf{z}$. Fix a partial mapping $\mu : [n] \to [m]$ from the coordinates of $\mathbf{x}$ to those of $\mathbf{y}$ that correspond to the subsequence $\mathbf{z}$ (i.e., $x_i = y_{\mu(i)}$, for every i for which $\mu(i)$ is defined).

The advisor finds and sends indices $i^\star \in [n]$ and $j^\star \in [m]$ such that no index $i < i^\star$ of $\mathbf{x}$ is mapped to an index $j > j^\star$ of $\mathbf{y}$. One option is that this is due to the fact that $\mu(i^\star) = j^\star$, but this does not have to be the case (e.g., if i^* is not mapped at all).[4] The specific choice of $i^\star$ and $j^\star$ will be crucial for bounding the client's runtime, but we defer the discussion of how they are chosen for the moment.

The indices $i^\star$ and $j^\star$ now define two LCS instances that the client solves directly (i.e., using the standard dynamic programming based algorithm for LCS): the first is $\mathbf{x}[1, i^\star - 1]$ vs. $\mathbf{y}[1, j^\star - 1]$ and the second is $\mathbf{x}[i^\star + 1, n]$ vs. $\mathbf{y}[j^\star + 1, n]$. Denote the solution found for the former by $\mathbf{z}_L$ and for the latter by $\mathbf{z}_R$.

Assuming that $i^\star$ was not mapped to $j^\star$, the client simply outputs $\mathbf{z}_L \circ \mathbf{z}_R$ and in case $i^\star$ was mapped to $j^\star$ it outputs $\mathbf{z}_L \circ \mathbf{x}[i^\star] \circ \mathbf{z}_R$. (Since we assume the advisor is trusted, it can simply specify whether or not $i^\star$ is mapped to $j^\star$.) It is straightforward to see that the output of the client is a valid solution – namely, a longest common subsequence for $\mathbf{x}$ and $\mathbf{y}$. The more important challenge is analyzing the efficiency of the protocol.

First note that if we use a time $c \cdot n \cdot m$ algorithm for LCS, for some constant c, we obtain a client that runs in time roughly: $c \cdot i^\star \cdot j^\star + c \cdot (n - i^\star) \cdot (m - j^\star)$. Note that for some values of $i^\star$ and $j^\star$ (e.g., $i^\star = j^\star = 0$) this gives no saving! Fortunately, we are able to show that there always exist $i^\star$ and $j^\star$ such that:

$$i^\star \cdot j^\star + (n - i^\star) \cdot (m - j^\star) \leq \frac{nm}{2}, \tag{1}$$

which in turn implies a factor 2 saving for the client.

To see that Equation (1) holds we consider two cases:

[4] Actually, it will be convenient to allow $i^\star$ and $j^\star$ to be real numbers rather than integers, but the reader can ignore this subtlety for the overview.

Case 1

There is an index $i < n/2$ that is mapped, via μ, to an index $j > n/2$. We fix $i^\star$ and $j^\star$ to be such indices. Note that since $i^\star$ is mapped to $j^\star$ they satisfy that $i < i^\star$ cannot be mapped to $j > j^\star$. Moreover:

$$i^\star \cdot j^\star + (n - i^\star) \cdot (m - j^\star) = \frac{1}{2}\big(n \cdot m + (n - 2i^\star) \cdot (m - 2j^\star)\big) \leq \frac{nm}{2},$$

where the equality can be verified by elementary arithmetic manipulations and the inequality follows from the fact that $n - 2i^\star < 0$ and $m - 2j^\star > 0$.

Case 2

There is no index $i < n/2$ that is mapped to $j^\star > m/2$. In such case, we define $i^\star = n/2$ and $j = n/2$. Note that by definition, there is no $i < i^\star$ that is mapped to $j > j_\star$. It is now straightforward that:

$$i^\star \cdot j^\star + (n - i^\star) \cdot (m - j^\star) = \frac{nm}{2}.$$

Thus, in both cases we obtain a factor 2 savings in the client's runtime as desired. In the full protocol, described below, we obtain a further savings by essentially recursing on both LCS instances rather than solving them directly.

2.2 The Minimum Cost String Alignment Problem

A cost function is a function $c : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$. The two cost functions that we focus on are $c_{\mathsf{LCS}}(a, b) \stackrel{\text{def}}{=} a + b$ and $c_{\mathsf{Edit}}(a, b) \stackrel{\text{def}}{=} \max(a, b)$.

▶ **Definition 3.** *The* Minimum Cost String Alignment problem *with respect to a cost function c, denoted* MCSA$[c]$*, is a relation consisting of all pairs $\big((\mathbf{x}, \mathbf{y}), (\mathbf{i}, \mathbf{j})\big)$ for which, if $\mathbf{x} \in \Sigma^n$ and $\mathbf{y} \in \Sigma^m$,*

- $\mathbf{i} \subseteq [n]^\ell$, $\mathbf{j} \subseteq [m]^\ell$ *for some* $\ell \in \mathbb{Z}^+$*;*
- $i_1 < \cdots < i_\ell$ *and* $j_1 < \cdots < j_\ell$*;*
- $x_{i_k} = y_{j_k}$ *for all* $k \in [\ell]$*; and*
- $\mathbf{i}$ *and* $\mathbf{j}$ *minimize* $\sum_{k=1}^{\ell+1} c(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1)$ *subject to the prior three constraints, where we define* $i_0 = j_0 = 0$*,* $i_{\ell+1} = n + 1$*, and* $j_{\ell+1} = m + 1$*.*

▶ Remark 4. To facilitate recursion, we will often consider MCSA instances where the strings $\mathbf{x}$ and $\mathbf{y}$ are most naturally viewed as substrings of larger strings, i.e. $\mathbf{x} = \mathbf{X}_I$ and $\mathbf{y} = \mathbf{Y}_J$. In these cases, we will view the symbols of $\mathbf{x}$ and $\mathbf{y}$ as indexed not by some $[n]$ and $[m]$, but by I and J respectively. The definition naturally extends to this setting, and we will frequently use this generalization.

The following two propositions show that LCS and Edit are special cases of MCSA with respect to different cost functions.

▶ **Proposition 5.** LCS *is the same as* MCSA$[c_{\mathsf{LCS}}]$.

Proof. For any common subsequence $x_{i_1} = y_{j_1}, \ldots, x_{i_\ell} = y_{j_\ell}$ of $\mathbf{x} \in \Sigma^n$ and $\mathbf{y} \in \Sigma^m$, define $i_0 = j_0 = 0$, $i_{\ell+1} = n + 1$, and $j_{\ell+1} = m + 1$ as in Definition 3

We have

$$\begin{aligned}
&\sum_{k=1}^{\ell+1} c_{\mathsf{LCS}}(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1) \\
=&\sum_{k=1}^{\ell+1} i_k - i_{k-1} + j_k - j_{k-1} - 2 \\
=&i_{\ell+1} - i_0 + j_{\ell+1} - j_0 - 2\ell - 2 \\
=&n + m - 2\ell,
\end{aligned}$$

which is minimized when ℓ is maximized. ◀

▶ **Proposition 6.** Edit *is the same as* $\mathsf{MCSA}[c_{\mathsf{Edit}}]$.

Proof. Suppose $x_{i_1} = y_{j_1}, \ldots, x_{i_\ell} = y_{j_\ell}$ is a common subsequence of $\mathbf{x}, \mathbf{y}$. Define $i_0 = j_0 = 0$, $i_{\ell+1} = n+1$, and $j_{\ell+1} = m+1$ as in Definition 3. Then one way of editing $\mathbf{x}$ into $\mathbf{y}$ is by editing $\mathbf{x}_{(i_{k-1}, i_k)}$ into $\mathbf{y}_{(j_{k-1}, j_k)}$ for each $k \in [\ell+1]$, which takes at most $\max(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1)$ operations.

On the other hand, suppose we are given a minimal sequence of operations $\mathcal{O} = (\mathsf{op}_1, \ldots, \mathsf{op}_d)$ that edits $\mathbf{x}$ into $\mathbf{y}$. This leads to a categorization of the symbols of $\mathbf{x}$ as unchanged, modified, or deleted; and a similar categorization of the symbols of $\mathbf{y}$ as unchanged from $\mathbf{x}$, modified from $\mathbf{x}$, or inserted. The unchanged symbols of $\mathbf{x}$ and of $\mathbf{y}$ form a common subsequence $x_{i_1} = y_{j_1}, \ldots, x_{i_\ell} = y_{j_\ell}$ and partition the original operation sequence into $\ell + 1$ subsequences $\mathcal{O}_1, \ldots, \mathcal{O}_{\ell+1}$ such that for each $k \in [\ell + 1]$, $\mathcal{O}_k$ edits $\mathbf{x}_{(i_{k-1}, i_k)}$ into $\mathbf{y}_{(j_{k-1}, j_k)}$. By the minimality of $\mathcal{O}$, each $\mathcal{O}_k$ must contain either only insertions and modifications or only deletions and modifications. Thus we have $\left|\mathcal{O}_k\right| = \max\left(i_k - i_{k-1} - 1, j_k - j_{k-1} - 1\right)$. ◀

The following proposition is a standard exercise in dynamic programming, whose proof we omit (see, e.g., [9]).

▶ **Proposition 7.** LCS *and* Edit *can be solved on input* $\mathbf{x}, \mathbf{y} \in \Sigma^n \times \Sigma^m$ *in* $O(n \cdot m)$ *time.*

The main technical lemma of Section 2 is a search delegation scheme for $\mathsf{MCSA}[c]$. running time.

▶ **Lemma 8.** *For every cost function* c *such that* $\mathsf{MCSA}[c]$ *is solvable in* $O(n \cdot m)$ *time and for every* $a = a(n, m)$*, there is an honest-advisor search delegation scheme for* $\mathsf{MCSA}[c]$ *that uses* a *bits of advice, and runs in time* $O\left(n + m + a + \frac{nm \log(nm)}{a}\right)$.

Furthermore assuming collision-resistant hash functions, this can be improved to an untrusted-advisor search delegation scheme with computational soundness.

2.3 Solution-Dependent Optimal Substructure

From this point on, we fix some cost function c such that $\mathsf{MCSA}[c]$ is solvable in $O(n \cdot m)$ time, and simply write MCSA in place of $\mathsf{MCSA}[c]$. We fix a specific c here because not all functions c necessarily have a $O(n \cdot m)$ time algorithm, however, both LCS and Edit do (see Proposition 7).

The main lemma in this section is that if some optimal alignment of $\mathbf{x}$ to $\mathbf{y}$ matches x_i with y_j, then any alignment of $\mathbf{x}$ to $\mathbf{y}$ that matches x_i with y_j is optimal if and only if it optimally aligns $\mathbf{x}_{<i}$ to $\mathbf{y}_{<j}$ and $\mathbf{x}_{>i}$ to $\mathbf{y}_{>j}$.

▶ **Proposition 9.** *Let* $\big((\mathbf{x},\mathbf{y}),(\mathbf{i},\mathbf{j})\big) \in \mathsf{MCSA}$ *with* $\mathbf{i} = (i_1,\ldots,i_\ell)$ *and* $\mathbf{j} = (j_1,\ldots,j_\ell)$.

For any $k \in [\ell]$, *any* $\ell_L, \ell_R \in \mathbb{Z}^+$, *any* $(\mathbf{i}'_L,\mathbf{j}'_L) \in [i_k-1]^{\ell_L} \times [j_k-1]^{\ell_L}$, *and any* $(\mathbf{i}'_R,\mathbf{j}'_R) \in [i_k+1,n]^{\ell_R} \times [j_k+1,m]^{\ell_R}$, *we have*

$$\big(\mathbf{i}'_L \circ i_k \circ \mathbf{i}'_R,\ \mathbf{j}'_L \circ j_k \circ \mathbf{j}'_R\big) \in \mathsf{MCSA}\big(\mathbf{x},\mathbf{y}\big)$$

if and only if $(\mathbf{i}'_L,\mathbf{j}'_L) \in \mathsf{MCSA}(\mathbf{x}_{<i_k},\mathbf{y}_{<j_k})$ *and* $(\mathbf{i}'_R,\mathbf{j}'_R) \in \mathsf{MCSA}(\mathbf{x}_{>i_k},\mathbf{y}_{>j_k})$.

Proof. By definition, the "total cost" of $\big(\mathbf{i}'_L \circ i_k \circ \mathbf{i}'_R,\ \mathbf{j}'_L \circ j_k \circ \mathbf{j}'_R\big)$ with respect to $(\mathbf{x},\mathbf{y})$ is the sum of the cost of $(\mathbf{i}'_L,\mathbf{j}'_L)$ with respect to $(\mathbf{x}_{<i_k},\mathbf{y}_{<j_k})$ and the cost of $(\mathbf{i}'_R,\mathbf{j}'_R)$ with respect to $(\mathbf{x}_{>i_k},\mathbf{y}_{>j_k})$. ◀

2.4 Our Protocol

Following the discussion in the introduction, we assume first that the advisor is honest. Suppose we are given an input $(\mathbf{x},\mathbf{y}) \in \Sigma^n \times \Sigma^m$ and have an advice budget of $a = a' \cdot \log(nm)$.

The advice in our protocol consists of an ordered rooted binary tree $\mathcal{T}$ with $O(a)$ vertices, such that:

- Every vertex v of $\mathcal{T}$ is labeled with a pair of intervals $(v.I, v.J)$ with $v.I \subseteq [n]$ and $v.J \subseteq [m]$. The root v_0 has $v_0.I = [n]$ and $v_0.J = [m]$.
- Every internal vertex v of $\mathcal{T}$ is additionally labeled with either:
  - (Case 1) $(v.i, v.j) \in v.I \times v.J$; or
  - (Case 2) $(v.i_1, v.i_2, v.j_1, v.j_2) \in v.I^2 \times v.J^2$.

  such that if $v.L$ and $v.R$ denote the left and right children of v, and if $(\mathbf{i}_L,\mathbf{j}_L)$ and $(\mathbf{i}_R,\mathbf{j}_R)$ are arbitrary elements of $\mathsf{MCSA}(\mathbf{x}_{v.L.I},\mathbf{y}_{v.L.J})$ and $\mathsf{MCSA}(\mathbf{x}_{v.R.I},\mathbf{y}_{v.R.J})$ respectively, then either:
  - (Case 1) $(\mathbf{i}_L \circ v.i \circ \mathbf{i}_R,\ \ \mathbf{j}_L \circ v.j \circ \mathbf{j}_R) \in \mathsf{MCSA}(\mathbf{x}_{v.I},\mathbf{y}_{v.J})$; or
  - (Case 2) $(\mathbf{i}_L \circ v.i_1 \circ v.i_2 \circ \mathbf{i}_R,\ \ \mathbf{j}_L \circ v.j_1 \circ v.j_2 \circ \mathbf{j}_R) \in \mathsf{MCSA}(\mathbf{x}_{v.I},\mathbf{y}_{v.J})$.

  This property gives a means to compute a solution to $\mathsf{MCSA}(\mathbf{x}_{v.I},\mathbf{y}_{v.J})$ in $O(1)$ time given any solutions to $\mathsf{MCSA}(\mathbf{x}_{v.L.I},\mathbf{y}_{v.L.J})$ and $\mathsf{MCSA}(\mathbf{x}_{v.R.I},\mathbf{y}_{v.R.J})$. Thus if we have solutions to $\mathsf{MCSA}(\mathbf{x}_{\ell.I},\mathbf{y}_{\ell.J})$ for all leaf nodes $\ell \in \mathcal{T}$, then we can compute a solution to $\mathsf{MCSA}(\mathbf{x},\mathbf{y})$ in time $O(|\mathcal{T}|)$.
- For every internal vertex v of $\mathcal{T}$, $v.L.I$ and $v.R.I$ are disjoint subsets of $v.I$. Similarly $v.L.J$ and $v.R.J$ are disjoint subsets of $v.J$, such that

$$|v.L.I| \cdot |v.L.J| + |v.R.I| \cdot |v.R.J| \le \frac{|v.I| \cdot |v.J|}{2}. \tag{2}$$

- Every vertex v of $\mathcal{T}$ with depth less than $\lfloor \log a' \rfloor$ satisfies $\min\big(|v.I|, |v.J|\big) = O(1)$.

2.4.1 Solving MCSA(x, y) given $\mathcal{T}$

As alluded to in the second property of $\mathcal{T}$, the client directly solves $\mathsf{MCSA}\big(\mathbf{x}_{\ell.I},\mathbf{y}_{\ell.J}\big)$ for every leaf node ℓ in $\mathcal{T}$. This takes time

$$O\Big(\sum_{\ell} |\ell.I| \cdot |\ell.J|\Big).$$

We split this sum into two sums: one over leaves of depth less than $\lfloor \log a' \rfloor$ and the other over leaves of depth at least $\lfloor \log a' \rfloor$.

- (Low-Depth Leaves) For low depth leaves ℓ, we bound $|\ell.I| \cdot |\ell.J|$ by $O(|\ell.I| + |\ell.J|)$. This is justified by the last property of $\mathcal{T}$, which asserts that either $|\ell.I|$ or $|\ell.J|$ is a constant. Now, we bound

$$\sum_{\text{low-depth } \ell} |\ell.I| + |\ell.J| \leq n + m,$$

  using the fact that $v.L.I$ and $v.R.I$ (respectively $v.L.J$ and $v.R.J$) are disjoint subsets of $v.I$ (respectively $v.J$).
- (High-Depth Leaves) By Equation (2), we know that

$$\sum_{\text{depth-}d \text{ leaves } \ell} |\ell.I| \cdot |\ell.J| \leq n \cdot m \cdot 2^{-d},$$

  and thus

$$\sum_{d \geq \lfloor \log a \rfloor} \sum_{\text{depth-}d \text{ leaves } \ell} |\ell.I| \cdot |\ell.J| \leq \frac{2nm}{a'}.$$

Finally, using the second property of $\mathcal{T}$, the client processes its leaf solutions into a solution to $\mathsf{MCSA}(\mathbf{x}, \mathbf{y})$. This takes time $O(|\mathcal{T}|) = O(a)$.

In total, the client's runtime is

$$O\left(n + m + a + \frac{nm}{a \log(nm)}\right).$$

2.4.2 Constructing $\mathcal{T}$

We construct $\mathcal{T}$ iteratively. On input $\mathbf{x} \in \Sigma^n, \mathbf{y} \in \Sigma^m$, start with a tree that consists only of a root vertex labeled with $I = [n]$ and $J = [m]$.

We say that a leaf vertex ℓ is $\mathsf{expandable}$ if either $\min(|\ell.I|, |\ell.J|) \geq 2$. For any interval $[a, b] \subseteq \mathbb{Z}$, let $\mathsf{middle}(I)$ denote $(a + b)/2$.

While there is an expandable leaf of $\mathcal{T}$:

1. Pick an expandable leaf of minimum-depth, and call it ℓ.
2. Compute $(\mathbf{i}, \mathbf{j}) \in \mathsf{MCSA}(\mathbf{x}_{\ell.I}, \mathbf{y}_{\ell.J})$.
3. Do either of the following (at least one will be applicable), defining i_0 and j_0 as $\min(I) - 1$ and $\min(J) - 1$ respectively, and defining $i_{|\mathbf{i}|+1}$ and $j_{|\mathbf{i}|+1}$ as $\max(I) + 1$ and $\max(J) + 1$ respectively:
   - **Case 1**: For some k, (i_k, j_k) "crosses" $(\mathsf{middle}(\ell.I), \mathsf{middle}(\ell.J))$. That is, either $i_k \leq \mathsf{middle}(\ell.I)$ and $j_k \geq \mathsf{middle}(\ell.J)$, or $i_k \geq \mathsf{middle}(\ell.I)$ and $j_k \leq \mathsf{middle}(\ell.J)$.
     In this case, add child nodes $v.L$ and $v.R$ to v, with $v.L.I = v.I \cap (-\infty, i_k)$, $v.L.J = v.J \cap (-\infty, j_k)$, $v.R.I = v.I \cap (i_k, \infty)$, and $v.R.J = v.J \cap (j_k, \infty)$. Set $v.i = i_k$ and $v.j = j_k$.
   - **Case 2:** For some k, $i_k \leq \mathsf{middle}(v.I) < i_{k+1}$ and $j_k \leq \mathsf{middle}(v.J) < j_{k+1}$.
     In this case, add child nodes $v.L$ and $v.R$ to v, with $v.L.I = v.I \cap (-\infty, i_k)$, $v.L.J = v.J \cap (-\infty, j_k)$, $v.R.I = v.I \cap (i_{k+1}, \infty)$, and $v.R.J = v.J \cap (j_{k+1}, \infty)$. Set $v.i_1 = i_k$, $v.i_2 = i_{k+1}$, $v.j_1 = j_k$, and $v.j_2 = j_{k+1}$.

   To see that one of the above cases will always be possible, let k^* and $\hat{k}$ be such that $i_{k^*} \leq \mathsf{middle}(I) < i_{k^*+1}$ and $j_{\hat{k}} \leq \mathsf{middle}(J) < j_{\hat{k}+1}$. If $k^* = \hat{k}$ then we are in case 2. Otherwise we are in case 1.

   The labelings of the children of v ensure the second property of $\mathcal{T}$ by Proposition 9. The last property of $\mathcal{T}$ is guaranteed by the fact that we always expand a lowest-depth expandable leaf.

We now turn to the third property of $\mathcal{T}$, i.e. establishing that Equation (2) holds. This is easy to see in Case 2. In Case 1, observe that either $|v.L.I| \leq |v.I|/2$ and $|v.L.J| \geq |v.J|/2$ or $|v.L.I| \geq |v.I|/2$ and $|v.L.J| \geq |v.J|/2$. We then have

$$\begin{aligned}|v.L.I| \cdot |v.L.J| + |v.R.I| \cdot |v.R.J| &\leq |v.L.I| \cdot |v.L.J| + \big(|v.I| - |v.L.I|\big) \cdot \big(|v.J| - |v.L.J|\big) \\ &= \frac{|v.I| \cdot |v.J|}{2} + \frac{1}{2} \cdot \big(|v.I| - 2|v.L.I|\big) \cdot \big(|v.J| - 2|v.L.J|\big) \\ &\leq \frac{|v.I| \cdot |v.J|}{2},\end{aligned}$$

as desired.

2.4.3 Advisor Efficiency

We note that the advisor's messages can be computed in linear time given any $(\mathbf{i}, \mathbf{j}) \in \mathsf{MCSA}(\mathbf{x}, \mathbf{y})$. Thus the advisor's running time is dominated by the cost of finding such a $(\mathbf{i}, \mathbf{j})$, which by Proposition 7 takes time $O(n \cdot m)$.

3 Parsing Context-Free Grammars

We start with background on context-free grammars in Section 3.1. Then, in Section 3.2 we state our main result and prove it in Section 3.3.

3.1 Context-Free Grammars

We first define context-free grammars. Parts of the following overview are directly based on [12]. See the standard textbook [15] for more detailed background.

▶ **Definition 10** (Context-free grammar). *A* context-free grammar *is a tuple $G = (V, \Sigma, R, A_{\mathsf{start}})$ such that V is a (finite) set of symbols, referred to as* variables*; Σ is a (finite) set of symbols, referred to as* terminals*; $R \subseteq V \times (V \cup \Sigma)^*$ is a (finite) relation, where each $(A, \alpha) \in R$ is referred to as a* production rule *and is denoted by $A \to \alpha$; $A_{\mathsf{start}} \in V$ is a variable that is referred to as the* start variable*.*

Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar, and let $\alpha, \beta \in (V \cup \Sigma)^*$ be strings of terminals and variables. We say that α directly yields β, denoted by $\alpha \Rightarrow \beta$, if there exists a production rule $A \to \gamma$ in R such that β is obtained from α by replacing exactly one occurrence of the variable A in α with the string $\gamma \in (V \cup \Sigma)^*$. We say that α yields β, denoted $\alpha \overset{*}{\Rightarrow} \beta$ if there exists a finite sequence of strings $\alpha_0, \ldots, \alpha_k \in (V \cup \Sigma)^*$ such that $\alpha_0 = \alpha$, $\alpha_k = \beta$, and $\alpha_0 \Rightarrow \ldots \Rightarrow \alpha_k$.

A grammar is said to be in *Chomsky normal form* if all of the production rules are of the form: (1) $A \to B \circ C$, where A, B and C are variables, (2) $A \to a$, where A is a variable and a is a terminal, or (3) $A \to \varepsilon$, where A is a variable and ε denotes the empty string. We note that every context-free grammar can be transformed into Chomsky normal form (with at most a quadratic blowup in the size of the grammar).

3.1.1 Derivation Tree

Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar. For $A \in V$ and $\alpha \in (V \cup \Sigma)^*$, a derivation tree, corresponding to the derivation[5] $A \overset{*}{\Rightarrow} \alpha$, is a rooted, directed, ordered, and labeled tree T (with edges oriented away from the root) that satisfies the following properties:

[5] The literature usually focuses on derivations trees for words composed of terminals only whereas we allow for a mix of variables and terminals.

- Each internal vertex is labeled by some variable, and the root is labeled by the variable A.
- Each leaf is labeled by a terminal or variable, where the i^{th} leaf is labeled by the i^{th} symbol of α.
- For every internal vertex v, if v is labeled by A' and its children are labeled by $\alpha_1, \ldots, \alpha_d \in (V \cup \Sigma)$ (where d denotes the number of children of v), then the production rule $A' \to \alpha_1 \circ \cdots \circ \alpha_k$ must belong to R, where $\circ$ denotes concatenation.

Note that for every derivation $A \stackrel{*}{\Rightarrow} \alpha$ there exists (at least one) corresponding derivation tree.

3.1.2 Parsing

The *parsing* problem is, given a grammar G and a word $w \in \Sigma^n$, to find a derivation tree corresponding to the fact that G derives w, or output $\perp$ if no such tree exists. A more general problem also considered in the literature is outputting *all* of the derivation trees corresponding to w but we focus here on the simpler task of finding *some* derivation tree. Thus, a parser for a grammar G is an algorithm that given as input a word w outputs a corresponding derivation tree, or $\perp$ if no such tree exists. As customary, we view the grammar as constant size and measure the complexity as a function of the input length.

We rely on the following result due to Valiant [20]:

▶ **Theorem 11** ([20]). *Every context-free grammar in Chomsky normal form has a time $O(n^{\omega_b})$ parser.*

Here and throughout, ω_b is the exponent for Boolean matrix multiplication (i.e., matrix multiplication in which the inner product operations is replaced with an OR of ANDs). The current known upper bound on ω_b is equal to the best known bound on ω [3] , the standard exponent for matrix multiplication.

Trees and the Lewis-Stearns-Hartmanis Lemma

In this section we only consider trees that are rooted, directed, and ordered (such as derivation trees defined above). Thus, throughout this section, whenever we say tree, we mean a rooted, directed, and ordered tree (with edges oriented away from the root). Note that the fact that the tree is ordered induces an ordering of its leaves. We define the arity of a tree to be the maximal number of children of any vertex in the tree. We follow the data-structure literature and define a subtree of a tree T as a tree consisting of a node in T and all of its descendants in T.[6] We use $L(T)$ to denote the number of leaves in the tree T.

The classic Lewis-Stearns-Hartmanis Lemma [18] shows that every binary tree on n leaves has a subtree with between $n/3$ and $2n/3$ leaves. We use here a more general form of this lemma, given by Goldreich *et al.* [12] who show that for every desired parameter t, any binary tree has a subtree with approximately t leaves (actually [12] further generalize to trees of different constant arity):

▶ **Lemma 12** ([12, Lemma 2.5]). *Let T be a binary tree and let $t \in [L(T)]$. Then, there exists a subtree T' of T with $L(T') \in [t/2, t]$ leaves.*

(The Lewis-Stearns-Hartmanis lemma corresponds to the special case when $t = 2n/3$.)

[6] This definition differs from the graph-theoretic definition that defines a subtree as any connected subgraph of a tree. For example, the root of a tree is a subtree in the graph theoretic sense but not according to our definition (unless the tree has exactly one vertex).

3.2 Delegating Context-Free Parsing

▶ **Theorem 13.** *Let G be a context-free grammar in Chomsky normal form. Then, for every parameter $a = a(n)$, there exists a search delegation scheme for the parsing problem for G, where the client runs in time $O(n^{\omega_b}/a^{\omega_b-1} + a \cdot \log n)$ and the communication complexity is $O(a \cdot \log n)$.*

For example, taking $a = n/\mathsf{polylog}(n)$, we obtain a quasi-linear time client with sub-linear communication. Alternatively, taking $a = n^{\frac{\omega_b-2}{\omega_b-1}} < \sqrt{n}$ we obtain a quadratic-time client with $\tilde{O}(\sqrt{n})$ communication.

3.3 Proof of Theorem 13

Let $G = (V, \Sigma, R, A_{\mathsf{start}})$ be a context-free grammar in Chomsky normal form. Recall that our task is to construct a delegation scheme for finding a derivation tree for a given word w derived by G. However, to facilitate recursion, we will actually solve a more general problem where the word $w \in (\Sigma \cup V)^*$ can be a mix of terminals and variables.

Consider a derivation tree T for $w \in (\Sigma \cup V)^*$. Since the grammar is in Chomsky normal form, the tree is binary and so for a given parameter t, Lemma 12 guarantees that T has a subtree T' of size roughly t. More precisely, a subtree T' with between $t/2$ and t leaves. Let x' be the substring of x corresponding to the tree T'.

We first construct an honest-advisor delegation scheme and then show how to deal with an untrusted advisor. In the scheme, the advisor first finds x' within x – that is, indices i and j such that $x' = x[i+1, \ldots, j]$). The advisor also finds the variable A associated with the root of T' in the derivation tree T. The advisor sends (i, j, A) to the client. The client is now left with two tasks:

- Find the derivation tree T' corresponding to $A \stackrel{*}{\Rightarrow} x'$.
- Find a derivation tree T'' corresponding to $A_{\mathsf{start}} \stackrel{*}{\Rightarrow} x[1, \ldots, i-1] \circ S \circ x[j+1, \ldots, n]$.

We solve the first of these directly, that is, by invoking a context-free parser (as in Theorem 11). The second problem is solved recursively. Note that two resulting derivation trees T' and T'' can be easily grafted together to construct a derivation tree corresponding to $A_{\mathsf{start}} \stackrel{*}{\Rightarrow} x$. However, some care needs to be given also for this task, since we do not want to process the entire tree T' again just for grafting. Rather, we ensure that the tree is represented using a data structure so that the grafting takes $O(1)$ time (e.g., using pointers). We also note that the base case of the recursion is solved by directly invoking a context-free parser.

Denote by $W(n, a)$ the running time of the client if we allow a recursive calls (and note that the corresponding communication complexity is $O(a \cdot \log n)$. Given the time $c \cdot n^{\omega_b}$ context-free parser of Theorem 11, where c is some constant, we have:

$$W(n, a) \leq c \cdot t^{\omega_b} + W(n - t/2, a - 1) + O(\log n). \tag{3}$$

Expanding, we see that any solution to Equation (3) must satisfy

$$W(n, a) \leq c \cdot a \cdot t^{\omega_b} + c \cdot (n - a \cdot t/2)^{\omega_b} + O(a \cdot \log n).$$

Setting the parameter $t = 2n/a$, we obtain:

$$W(n, a) \leq c \cdot a \cdot (2n/a)^{\omega_b} + O(a \cdot \log n) = O(n^{\omega_b}/a^{\omega_b-1} + a \cdot \log n).$$

3.3.1 Coping with cheating advisors

In order to deal with a dishonest advisor, we note that given a candidate derivation tree, there is a linear (in the size of the tree) time algorithm that checks that its validity (by simply checking that each vertex is consistent with the grammar). Thus, we can easily transform our protocol to handle dishonest advisors by having the client check its answer and outputting $\perp$ in case the generated tree is invalid.

4 k-SAT

In this section, we construct a search delegation scheme for k-SAT, where k is some constant. Recall that an instance of k-SAT is a CNF formula on n variables $(x_1, \ldots, x_n)$ with m clauses, each of which is a disjunction of at most k literals (variables or their negations).

A trivial delegation scheme for k-SAT on a given formula $\varphi : \{0,1\}^n \to \{0,1\}$ is to simply send the first a bits of a satisfying assignment for φ. This reduces the number of variables to $n - a$, so if $T_k(n)$ denotes the time to solve an n-variable k-SAT instance, this hint reduces the client's running time to $T_k(n-a)$. In particular, if $T_k(n) = 2^{c_k \cdot n}$ for some $c_k < 1$, then a bits of hint yield a factor $2^{c_k \cdot a}$ speedup.

We achieve an improved factor 2^a speedup for a specific k-SAT algorithm due to [19], which we henceforth refer to as PPSZ. PPSZ is relatively simple, and has a running time that is close to the state of the art algorithm due to Hansen *et al.* [14].

▶ **Theorem 14.** *Let $(c_k)_{k \in \mathbb{Z}^+}$ be constants such that the analyzed running time of the PPSZ algorithm for k-SAT, with success probability*[7] *$\frac{1}{2}$, is $2^{c_k n + o(n)}$. Then, for every $a = a(n)$ and $\kappa = \kappa(n) \leq n^{O(1)}$, there exists a search delegation scheme in which the advisor sends a bits and the client's run-time is $2^{c_k \cdot n - a + o(n)}$, assuming the advisor and client have access to a shared $\mathsf{poly}(n)$-length random string. The completeness error is $2^{-\kappa}$ and the soundness error is 0.*

To prove Theorem 14, roughly speaking, we observe that PPSZ can be viewed as a nondeterministic polynomial-time algorithm that uses $c_k n$ bits of nondeterminism - in other words, an exhaustive search for a $c_k n$-bit string. Such an emulation clearly takes $\tilde{O}(2^{c_k n})$ time. The hint in our protocol then consists of the first a bits of this nondeterminism rather than the first a bits of a satisfying assignment for φ.

This description is not completely accurate – PPSZ is actually a randomized algorithm. Each choice of randomness defines an exhaustive search problem such such that the exhaustive search has noticeable probability of yielding a satisfying assignment for φ. We describe PPSZ in a way that elucidates this structure in Appendix A.

Finally, we can deal with an untrusted advisor by simply verifying that at the end we have actually found a satisfying assignment to φ.

5 From Honest Advisor to Dishonest Advisor

In this section we describe a simple transformation from an honest-advisor search delegation to a full-fledged one (i.e., secure against an untrusted advisor).

[7] Since we can amplify success probability by repetition, c_k would be the same if we we required success probability as small as $1/\mathsf{poly}(n)$ or as large as $1 - 2^{-\mathsf{poly}(n)}$.

▶ **Lemma 15.** *Suppose that the relation $R \subseteq \{0,1\}^n \times \{0,1\}^m$ has an honest-advisor search delegation scheme with communication $c_{\mathsf{srch}} = c_{\mathsf{srch}}(n,m)$ and client running time $t_{\mathsf{srch}} = t_{\mathsf{srch}}(n,m)$. Suppose also that membership in the relation R can be verified by an interactive proof with communication $c_{\mathsf{prf}} = c_{\mathsf{prf}}(n,m)$ and verifier running time $t_{\mathsf{prf}} = t_{\mathsf{prf}}(n,m)$. Then, R has a full-fledged search delegation scheme with $c_{\mathsf{srch}} + c_{\mathsf{prf}}$ communication and $t_{\mathsf{srch}} + t_{\mathsf{prf}}$ client run-time.*

In case the interactive proof for R is only computationally sound, then the resulting delegation scheme is also only computationally sound.

Proof. Follows immediately by first running the honest-advisor protocol and then checking the solution using the interactive proof. ◀

For example, using Kilian's [16] celebrated argument-system we obtain the following corollary:

▶ **Corollary 16.** *Suppose that the relation $R \subseteq \{0,1\}^n \times \{0,1\}^m$ is decidable in polynomial-time and has an honest-advisor search delegation scheme with communication $c_{\mathsf{srch}} = c_{\mathsf{srch}}(n,m)$ and client running time $t_{\mathsf{srch}} = t_{\mathsf{srch}}(n,m)$. Then, assuming that there exist collision-resistant hash functions, the relation R has a computationally sound delegation scheme with communication $c_{\mathsf{srch}} + \mathsf{poly}(\kappa, \log(n+m))$ and verifier runtime $\tilde{O}(n+m) + \mathsf{poly}(\kappa, \log(n+m))$, where κ denotes a cryptographic security parameter.*

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
3. Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
4. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010. doi:10.1007/978-3-642-14165-2_14.
5. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.
6. Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.6.
7. Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. *CoRR*, abs/1805.02351, 2018. arXiv:1805.02351.

8 Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2010. doi:10.1007/978-3-642-14623-7_26.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

10 Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. *IACR Cryptol. ePrint Arch.*, page 547, 2009. URL: http://eprint.iacr.org/2009/547.

11 Oded Goldreich. *Computational complexity – A conceptual perspective*. Cambridge University Press, 2008. doi:10.1017/CBO9780511804106.

12 Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs. *Inf. Comput.*, 261:175–201, 2018. doi:10.1016/j.ic.2018.02.003.

13 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. doi:10.1137/0218012.

14 Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. Faster k-sat algorithms using biased-ppsz. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 578–589. ACM, 2019. doi:10.1145/3313276.3316359.

15 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

16 Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992. doi:10.1145/129712.129782.

17 Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, 2002. doi:10.1145/505241.505242.

18 Philip M. Lewis, Richard Edwin Stearns, and Juris Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *SWCT (FOCS)*, pages 191–202, 1965. doi:10.1109/FOCS.1965.14.

19 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, 2005. doi:10.1145/1066100.1066101.

20 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

A PPSZ with Hints: Details

In this section we provide details about PPSZ and its analysis. We then discuss how a hint can be used to achieve optimal savings up to an additive log factor. We focus our attention on k-SAT instances that are *sparse* (the number of clauses is linear in the number of variables) and have *unique solutions* because when $k \geq 5$ there is a randomized reduction from general k-SAT to this restricted subset of instances [6].

PPSZ Overview

PPSZ has the following high-level structure. It first performs a polynomial-time "conditioning" step on the input formula k-CNF formula φ that produces an equivalent formula φ_s that has more clauses. Then PPSZ runs Search (see Algorithm 1) on φ_s. This produces I random candidate values of $(\pi, y) \in S_n \times \{0,1\}^n$ (where, as usual, S_n denotes the group of

permutations over $[n]$). Each pair is fed into the algorithm MODIFY (see Algorithm 2) until MODIFY produces a satisfying assignment to φ_s (and hence to φ). At a high level, MODIFY goes through the variables of φ_s one by one (in an order specified by π), and iteratively simplifies φ_s by setting the i^{th} variable to y_i. If at any stage there is a "unital" clause (a clause with exactly one variable x), then x is set to 1 if the clause is (x) and to 0 if the clause is $(\neg x)$. When this happens, we say that x is forced.

PPSZ Analysis

Let z denote the unique satisfying assignment to φ, and let $Forced(\varphi, \pi, y)$ denote the set of forced variables when running MODIFY(φ, π, y).

PPSZ prove that for $\pi \leftarrow S_n$, we have

$$\mathbb{E}_\pi\left[\left|Forced(\varphi_s, \pi, z)\right|\right] \geq (1 - c_k)n,$$

where c_k is some constant.

Call $\pi \in S_n$ good if $\left|Forced(\varphi_s, \pi, z)\right| \geq (1 - c_k)n - 1$. If SEARCH happens to sample a good π, then there is a $\approx 2^{-c_k n}$ probability that y will be sampled to agree with z outside of $Forced(\varphi_s, \pi, z)$, which causes MODIFY to terminate with a satisfying assignment for φ.

Finally, a random π is good with probability at least $1/n$. To see this, let F denote $\left|Forced(\varphi_s, \pi, z)\right|$. Markov's inequality then implies (because F is $[0, n]$-valued) that $\Pr\left[F \geq \mathbb{E}[F] - 1\right] \geq 1/n$.

Thus after $I = 2^{c_k n} \cdot \mathsf{poly}(n)$ trials, MODIFY will with overwhelming probability output a satisfying assignment for φ.

Algorithm 1 The SEARCH Algorithm from [19].

```
SEARCH(formula φ, integer I):
for i ∈ [1, I] do
    pick y uniformly at random {0,1}^n
    pick π uniformly at random
    z=MODIFY(φ, π, y)
    if z satisfies φ then
        return z
    end if
end for
return Unsatisfied
```

▶ **Lemma 17.** *Let c_k denoting the constant in the exponent of the running time of the PPSZ algorithm for k-SAT. Let $\kappa = \kappa(n) \leq n^{O(1)}$ denote a statistical security parameter.*

There for any $a = a(n)$, there is an honest-advisor search delegation scheme for k-SAT with:

- *completeness error $2^{-\kappa}$;*
- *a bits of communication;*
- *$\mathsf{poly}(n)$ bits of shared randomness; and*
- *client running time of $2^{c_k n - a} \cdot \mathsf{poly}(n)$.*

Proof Sketch. In our protocol, we assume that the advisor and client have access to shared randomness, which determines a sequence of permutations $\pi_1, \pi_2, \ldots \in S_n$. The advisor first sends the client an i such that π_i is good. With all but $2^{-\kappa}$ probability, i is $O(\kappa \log n)$, so this constitutes just $\log \kappa + \log \log n + O(1)$ bits of communication.

Algorithm 2 The MODIFY algorithm from [19].

MODIFY(formula φ, permutation π, assignment y):
$\varphi_0 = \varphi$
for $i \in [1, n]$ **do**
  if φ_{i-1} contains the unit clause $(x_{\pi[i]})$ **then**
    $z_{\pi[i]} = 1$
  else if φ_{i-1} contains the unit clause $(\bar{x}_{\pi[i]})$ **then**
    $z_{\pi[i]} = 0$
  else
    $x_i = y_i$
  end if
  $\varphi_i = \varphi_{i-1}$ with $x_{\pi_i} = z_{\pi_i}$
end for
return z

Algorithm 3 The MODIFY' algorithm.

MODIFY'(formula φ, permutation π, settings y):
$\varphi_0 = \varphi$
$j = 0$
for $i \in [1, n]$ **do**
  if φ_{i-1} contains the unit clause $(x_{\pi[i]})$ **then**
    $z_{\pi[i]} = 1$
  else if φ_{i-1} contains the unit clause $(\bar{x}_{\pi[i]})$ **then**
    $z_{\pi[i]} = 0$
  else
    $x_i = y_j$
    $j = j + 1$
  end if
  $\varphi_i = \varphi_{i-1}$ with $x_{\pi_i} = z_{\pi_i}$
end for
return z

Next, let $j_1, \ldots, j_{n'} \in [n]$ denote indices such that on input (φ_s, π_i, z), MODIFY assigns unforced values to $x_{j_1}, \ldots, x_{j_{n'}}$ (in that order). Because π_i is good, we have $n' \leq c_k n + 1$. If the total remaining advice budget is a, then the advisor sends $z_{j_1}, \ldots, z_{j_a}$ and n'.

Upon receiving $\big((y_1, \ldots, y_a), n'\big)$, the client then repeatedly samples $y_{a+1}, \ldots, y_{n'}$, computes $z := \text{MODIFY}'\big(\varphi_s, \pi_i, (y_1, \ldots, y_{n'})\big)$ and hopes for a successful trial, i.e. one in which z is a satisfying assignment for φ (this happens with probability at least $2^{-c_k n - 1}$ over the choice of $y_{a+1}, \ldots, y_{n'}$). After $\approx \kappa \cdot 2^{c_k n}$ trials, the client will have at least one successful trial with all but $2^{-\kappa}$ probability. ◀

Understanding the Moments of Tabulation Hashing via Chaoses

Jakob Bæk Tejs Houen
BARC, Department of Computer Science, University of Copenhagen, Denmark

Mikkel Thorup
BARC, Department of Computer Science, University of Copenhagen, Denmark

Abstract

Simple tabulation hashing dates back to Zobrist in 1970 and is defined as follows: Each key is viewed as c characters from some alphabet Σ, we have c fully random hash functions $h_0, \ldots, h_{c-1} : \Sigma \to \{0, \ldots, 2^l - 1\}$, and a key $x = (x_0, \ldots, x_{c-1})$ is hashed to $h(x) = h_0(x_0) \oplus \ldots \oplus h_{c-1}(x_{c-1})$ where $\oplus$ is the bitwise XOR operation. The previous results on tabulation hashing by Pătraşcu and Thorup [J.ACM'11] and by Aamand et al. [STOC'20] focused on proving Chernoff-style tail bounds on hash-based sums, e.g., the number keys hashing to a given value, for simple tabulation hashing, but their bounds do not cover the entire tail. Thus their results cannot bound moments. The paper Dahlgaard et al. [FOCS'15] provides a bound on the moments of certain hash-based sums, but their bound only holds for constant moments, and we need logarithmic moments.

Chaoses are random variables of the form $\sum a_{i_0,\ldots,i_{c-1}} X_{i_0} \cdot \ldots \cdot X_{i_{c-1}}$ where X_i are independent random variables. Chaoses are a well-studied concept from probability theory, and tight analysis has been proven in several instances, e.g., when the independent random variables are standard Gaussian variables and when the independent random variables have logarithmically convex tails. We notice that hash-based sums of simple tabulation hashing can be seen as a sum of chaoses that are not independent. This motivates us to use techniques from the theory of chaoses to analyze hash-based sums of simple tabulation hashing.

In this paper, we obtain bounds for all the moments of hash-based sums for simple tabulation hashing which are tight up to constants depending only on c. In contrast with the previous attempts, our approach will mostly be analytical and does not employ intricate combinatorial arguments. The improved analysis of simple tabulation hashing allows us to obtain bounds for the moments of hash-based sums for the mixed tabulation hashing introduced by Dahlgaard et al. [FOCS'15]. With simple tabulation hashing, there are certain inputs for which the concentration is much worse than with fully random hashing. However, with mixed tabulation, we get logarithmic moment bounds that are only a constant factor worse than those with fully random hashing for any possible input. This is a strong addition to other powerful probabilistic properties of mixed tabulation hashing proved by Dahlgaard et al.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases hashing, concentration bounds, moment bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.74

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2205.01453`

Funding Research supported by Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

Acknowledgements We thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 74; pp. 74:1–74:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Hashing is a ubiquitous tool of randomized algorithms which dates all the way back to the 1950s [12]. A hash function is a random function, $h\colon U \to R$, that assigns a random hash value, $h(x) \in R$, to every key, $x \in U$. When designing algorithms and data structures, it is often assumed that one has access to a uniformly random hash function that can be evaluated in constant time. Even though this assumption is very useful and convenient, it is unfortunately also unrealistic. It is thus a natural goal to find practical and efficient constructions of hash functions that provably have guarantees akin to those of uniformly random hashing.

If we want implementable algorithms with provable performance similar to that proven assuming uniformly random hashing, then we have to find practical and efficient constructions of hash functions with guarantees akin to those of uniformly random hashing. An example of this is simple tabulation hashing introduced by Zobrist in 1970 [27]. The scheme is efficient and easy to implement, and Pătraşcu and Thorup [22] proved that it could replace uniformly random hashing in many algorithmic contexts. The versatility of simple tabulation does not stem from a single probabilistic power like k-independence (it is only 3-independent), but from an array of powers that have different usages in different applications. Having one hash function with multiple powers has many advantages. One is that we can use the same hash function implementation for many purposes. Another is that hash functions are often an inner-loop bottleneck, and then it is an advantage if the same hash value can be used for multiple purposes. Also, if we have proved that a simple hash function has some very different probabilistic properties, then, morally, we would expect it to possess many other properties to be uncovered as it has happened over the years for simple tabulation (see, e.g., [3, 4]). Finally, when we hash a key, we may not even know what property is needed, e.g., with weighted keys, we may need one property to deal with a few heavy keys, and another property to deal with the many light keys, but when we hash the key, we may not know if it is heavy or light.

One of the central powers proved for simple tabulation in [22] is that it has strong concentration bounds for hash-based sums (will be defined shortly in Section 1.1). The concentration holds only for quite limited expected values, yet this suffices for important applications in classic hash tables. Recently, Aamand et al. [2] introduced tabulation-permutation, which is only about twice as slow as simple tabulation, and which offers general concentration bounds that hold for all hash-based sums regardless of the expected size. An issue with tabulation-permutation is that it is not clear if it possesses the other strong powers of simple tabulation.

A different way to go is to construct increasingly strong schemes, each inheriting all the nice properties of its predecessors. In this direction, [21] introduced twisted tabulation strengtening simple tabulation, and [9] introduced mixed tabulation strengthening twisted tabulation. Each new scheme was introduced to get some powers not available with the predecessor. In particular, mixed tabulation has some selective full-randomness that is needed for aggregating statistics over hash-based k-partitions. These applications also needed concentration bounds for hash-based sums, but [9] only provided some specialized suboptimal concentration bounds.

In this paper, we do provide strong concentration bounds for mixed tabulation hashing which can then be used in tandem with all the other strong properties of simple, twisted, and mixed tabulation. In fact our bounds are more general than the strong concentration bounds proved in [2] for tabulation-permutation. More precisely, the concentration bounds in [2]

are Chernoff-style tail bounds that hold with high probability, while what we do is to show moment bounds that imply such tail bounds as special cases. Indeed the key to our results for mixed tabulation is a much stronger understanding of the moments of simple tabulation.

Below we proceed to describe our new mathematical understanding, including the relevance of chaoses. We will contextualize this with other work later in Section 1.6.

1.1 Moment bounds for hash-based sums

In this paper, we will focus on analyzing hash-based sums. More precisely, we consider a fixed *value function*, $v\colon U \times R \to \mathbb{R}$, and define the random variable $X_x = v(x, h(x))$ for every key $x \in U$. We are then interested in proving concentration bounds for the sum $X = \sum_{x\in U} X_x = \sum_{x\in U} v(x, h(x))$. It should be noted that the randomness of X derives from the hash function h, thus the results will depend on the strength of h.

This is quite a general problem, and at first glance, it might not be obvious why this is a natural construction to consider, but it does generalize a variety of well-studied constructions:

1. Let $S \subseteq U$ be a set of balls and assign a weight, $w_x \in \mathbb{R}$, for every ball, $x \in S$. The goal is to distribute the balls, S, into a set of bins $R = [m]$.[1] For a bin, $y \in [m]$, we define the value function $v_y\colon U \times [m] \to \mathbb{R}$ by $v_y(x, j) = w_x\,[j = y]\,[x \in S]$, then $X = \sum_{x\in U} v_y(x, h(x)) = \sum_{x\in S} w_x\,[h(x) = y]$ will be the weight of the balls hashing to bin y.[2]
2. Instead of concentrating on a single bin, we might be interested in the total weight of the balls hashing below some threshold l. This is useful for sampling, for if $h(x)$ is uniform in $[m]$, then $\Pr[h(x) < l] = l/m$. We then define the value function $v\colon U \times [m] \to \mathbb{R}$ by $v(x, j) = w_x\,[j < l]\,[x \in S]$, then $X = \sum_{x\in U} v(x, h(x)) = \sum_{x\in S} w_x\,[h(x) < l]$ will be precisely the total weight of the balls hashing below l.

The first case appears when one tries to allocate resources, and the second case arises in streaming algorithms, see, e.g., [1]. In any case, X ought to be concentrated around the mean $\mu = \mathrm{E}[X]$. If h is a uniformly random hash function then this will be the case under mild assumptions about v but it cannot otherwise be assumed a priori to be the case.

There are two natural ways to quantify the concentration of X, either we bound the tail of X, i.e., we bound $\Pr[|X - \mu| \geq t]$ for all $t \geq 0$, or we bound the central moments of X, i.e., we bound the p-th moment $\mathrm{E}[|X - \mu|^p]$ for all $p \geq 2$. If we have a bound on the tail that is exponentially decreasing, we can bound the central moments of X for all $p \geq 2$. Unfortunately, some of the prior works [2, 11, 25] prove bounds on the tail that are exponentially decreasing but also has an additive term of the form $n^{-\gamma}$ where $\gamma = O(1)$. It will then only be possible to give strong bounds for the central moments of X for $p = O(1)$. This is not necessarily a fault of the hash function but a defect of the analysis. In contrast, if we prove strong bounds for the central moments of X for $p = O(\log n)$ then we can use Markov's inequality to prove a bound the tail that is exponentially decreasing but with an additive term of the form $n^{-\gamma}$ where $\gamma = O(1)$. Thus in some sense, it is more robust to bound the moments compared to bounding the tail.

We can use the classic k-independent hashing framework of Wegman and Carter [26] as an easy way to obtain a hash function that has bounds on the central moments as a uniformly random hash function. A random hash function, $h\colon U \to R$, is k-independent if $(h(x_0), \ldots, h(x_{k-1}))$ is uniformly distributed in R^k for any k distinct keys $x_0, \ldots, x_{k-1} \in U$.

[1] For a positive integer $m \in \mathbb{N}$ we define $[m] = \{0, \ldots, m-1\}$.
[2] For a statement P we let $[P]$ be 1 if P is true and 0 otherwise.

The p-th central moment $\mathrm{E}[(X-\mu)^p]$ of X for a k-independent hash function h is the same as the p-th central moment of X for a fully random hash function when p is an even integer less than k. We shall, however, focus on simple and fast hashing schemes that are not even 4-independent, and yet we will show strong moment bounds.

1.2 Tabulation Hashing

Simple tabulation hashing dates back to 1970 and was first introduced by Zobrist for optimizing chess computers [27]. In simple tabulation hashing, we view the universe, U, to be of the form $U = \Sigma^c$ for some alphabet, Σ, and a positive integer c. Let $T\colon \{0,\dots,c-1\}\times\Sigma \to [2^l]$ be a uniformly random table, i.e., each value is chosen independently and uniformly at random from the set $[2^l]$. A simple tabulation hash function, $h\colon \Sigma^c \to [2^l]$, is then defined by

$$h(\alpha_0,\dots,\alpha_{c-1}) = \bigoplus_{i=0}^{c-1} T(i,\alpha_i)\ ,$$

where $\oplus$ is the bitwise XOR-operation, i.e., addition when $[2^l]$ is identified with the Abelian group $(\mathbb{Z}/2\mathbb{Z})^l$. We say that h is a simple tabulation hash function with c characters. With 8- or 16-bit characters, the random table T fits in cache, and then simple tabulation is very fast, e.g., in experiments, [22] found it to be as fast as two to three multiplications.

The moments of simple tabulation hashing have been studied in multiple papers. Braverman et al. [7] showed that for a fixed bin the 4th central moment is close to that achieved by truly random hashing. Dahlgaard et al. [10] generalized this to any constant moment p. Their proof works for any p but with a doubly exponential dependence on p, so their bound is only useful for $p = O(1)$. In this paper, we obtain bounds for all the moments of hash-based sums for simple tabulation hashing which are tight up to constants depending only on c.

Previous work has just treated c as a constant, hidden in O-notation. However, c does provide a fundamental trade-off between evaluation time with c lookups and the space $cU^{1/c}$. We therefore find it relevant to elucidate how our moment bounds depend on c even though we typically choose $c = 4$.

Mixed tabulation hashing was introduced by Dahlgaard et al. [9]. As in simple tabulation hashing, we view the universe, U, to be of the form $U = \Sigma^c$ for some alphabet, Σ, and a positive integer c. We further assume that the alphabet, Σ, has the form $\Sigma = [2^k]$. Let $h_1\colon \Sigma^c \to [2^l]$, $h_2\colon \Sigma^c \to \Sigma^d$, and $h_3\colon \Sigma^d \to [2^l]$ be independent simple tabulation hash functions. A mixed tabulation hash function, $h\colon \Sigma^c \to [2^l]$, is then defined by

$$h(x) = h_1(x) \oplus h_3(h_2(x))\ .$$

As in simple tabulation hashing, $\oplus$ is the bitwise XOR-operation. We call h a mixed tabulation hash function with c characters and d derived characters. We note that h_1 and h_2 can be combined in a single simple tabulation hash function $\Sigma^c \to [2^l]\times\Sigma^d$, and then h is implemented with only $c+d$ lookups.

With simple tabulation hashing, there are certain inputs for which the concentration is much worse than with fully random hashing. However, with mixed tabulation, even if we have just $d = 1$ derived character, we get logarithmic moment bounds that, for $c = O(1)$, are only a constant factor worse than those with fully-random hashing for any input assuming that hash range at most polynomial in the key universe.

Getting within a constant factor is very convenient within algorithm analysis, where we typically only aim for O-bounds that are tight within a constant factor.

1.3 Relation between Simple Tabulation and Chaoses

A *chaos* of order c is a random variable of the form

$$\sum_{0\leq i_0<\ldots<i_{c-1}<n} a_{i_0,\ldots,i_{c-1}} \prod_{j\in[c]} X_{i_j} \ ,$$

where $(X_i)_{i\in[n]}$ are independent random variables and $(a_{i_0,\ldots,i_{c-1}})_{0\leq i_0<\ldots<i_{c-1}<n}$ is a multi-indexed array of real numbers. And a *decoupled chaos* of order c is a random variable of the form

$$\sum_{i_0,\ldots,i_{c-1}\in[n]} a_{i_0,\ldots,i_{c-1}} \prod_{j\in[c]} X_{i_j}^{(j)} \ ,$$

where $(X_i^{(j)})_{i\in[n],j\in[c]}$ are independent random variables and $(a_{i_0,\ldots,i_{c-1}})_{i_0,\ldots,i_{c-1}\in[n]}$ is a multiindexed array of real numbers. Chaoses have been studied in different settings, e.g., when the variables are standard Gaussian variables [17, 18], when the variables have logarithmically concave tails [5], and when the variables have logarithmically convex tails [16].

From the definition of a chaos and simple tabulation hashing it might not be immediately clear that there is connection between the two. But we can rewrite the expression for hash-based sums of simple tabulation hashing as follows

$$\begin{aligned}\sum_{x\in\Sigma^c} v(x,h(x)) &= \sum_{\alpha_0,\ldots,\alpha_{c-1}\in\Sigma} v((\alpha_0,\ldots,\alpha_{c-1}),h(\alpha_0,\ldots,\alpha_{c-1}))\\ &= \sum_{j_0,\ldots,j_{c-1}\in[m]} \sum_{\alpha_0,\ldots,\alpha_{c-1}\in\Sigma} v\left((\alpha_0,\ldots,\alpha_{c-1}),\bigoplus_{i\in[c]} j_i\right) \prod_{i\in[c]} [T(i,\alpha_i)=j_i] \ .\end{aligned}$$

We then notice that $\sum_{\alpha_0,\ldots,\alpha_{c-1}\in\Sigma} v\left((\alpha_0,\ldots,\alpha_{c-1}),\bigoplus_{i\in[c]} j_i\right) \prod_{i\in[c]} [T(i,\alpha_i)=j_i]$ is a decoupled chaos of order c for any $(j_i)_{i\in[c]}$, thus hash-based sums of simple tabulation hashing can be seen as a sum of chaoses. Now since the random variables, $([T(i,\alpha_i)=j])_{j\in[m]}$, are not independent then the chaoses are not independent either which complicates the analysis. Nonetheless, this realization inspires us to use techniques from the study of chaoses to analyze the moments of tabulation hashing, in particular, our approach will be analytical in contrast with the combinatorial approach of the previous papers. We will expand further on the techniques in Section 1.5.

1.4 Our Results

When proving and stating bounds for the p-th moment of a random variable it is often more convenient and more instructive to do it in terms of the p-norm of the random variable. The p-norm of a random variable is the p-th root of the p-th moment of the random variable and is formally defined as follows:

▶ **Definition 1** (p-norm). *Let $p\geq 1$ and X be a random variable with $\mathrm{E}[|X|^p]<\infty$. We then define the p-norm of X by $\|X\|_p = \mathrm{E}[|X|^p]^{1/p}$.*

Our main contributions of this paper are analyses of the moments of hash-based sums of simple tabulation hashing and mixed tabulation hashing. To do this we first had to analyze the moments of hash-based sums of fully random hashing which as far as we are aware have not been analyzed tightly before.

1.4.1 The Moments of Fully Random Hashing

Previously, the focus has been on proving Chernoff-like bounds by using the moment generating function but a natural, different approach would be to use moments instead. Both the Chernoff bounds [8] and the more general Bennett's inequality [6] bound the tail using the Poisson distribution. More precisely, let $v\colon U \times [m] \to \mathbb{R}$ be a value function that satisfies that $\sum_{j\in[m]} v(x,j) = 0$ and define the following two parameters M_v and σ_v^2 which will be important throughout the paper as follows:

$$M_v = \max_{x\in U, j\in[m]} |v(x,j)| \ , \tag{1}$$

$$\sigma_v^2 = \frac{\sum_{x\in U, j\in[m]} v(x,j)^2}{m} \ . \tag{2}$$

Bennett's inequality specialized to our setting then says that for a fully random hash function h

$$\begin{aligned}\Pr\left[\left|\sum_{x\in U} v(x,h(x))\right| \geq t\right] &\leq 2\exp\Big(-\tfrac{\sigma_v^2}{M_v^2}\mathcal{C}\Big(\tfrac{tM_v}{\sigma_v^2}\Big)\Big) \\ &\leq \begin{cases} 2\exp\Big(-\frac{t^2}{3\sigma_v^2}\Big) & \text{if } t \leq \frac{\sigma_v^2}{M_v} \\ 2\exp\Big(-\frac{t}{2M_v}\log\Big(1+\frac{tM_v}{\sigma_v^2}\Big)\Big) & \text{if } t > \frac{\sigma_v^2}{M_v} \end{cases} \ , \end{aligned} \tag{3}$$

where $\mathcal{C}(x) = (x+1)\log(x+1) - x$.[3]

This inspires us to try to bound the p-norms of X_v with the p-norms of the Poisson distribution. To do this we will introduce the function $\Psi_p(M,\sigma^2)$ which is quite technical but we will prove that $\Psi_p(1,\lambda)$ is equal up to a constant factor to the central p-norm of a Poisson distributed variable with mean λ. One should think of $\Psi_p(M,\sigma^2)$ as a p-norm version of $\frac{\sigma_v^2}{M_v^2}\mathcal{C}\Big(\frac{tM_v}{\sigma_v^2}\Big)$ which appears in Bennett's inequality.

▶ **Definition 2.** *For $p \geq 2$ we define the function $\Psi_p\colon \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ as follows*

$$\Psi_p(M,\sigma^2) = \begin{cases} \left(\frac{\sigma^2}{pM^2}\right)^{1/p} M & \text{if } p < \log\frac{pM^2}{\sigma^2} \\ \frac{1}{2}\sqrt{p}\sigma & \text{if } p < e^2\frac{\sigma^2}{M^2} \\ \frac{p}{e\log\frac{pM^2}{\sigma^2}} M & \text{if } \max\Big\{\log\frac{pM^2}{\sigma^2}, e^2\frac{\sigma^2}{M^2}\Big\} \leq p \end{cases} .$$

▶ Remark 3. When p is *small* then case 1 and 2 apply while for *large* p case 3 applies. If $2 < e^2\frac{\sigma^2}{M^2}$ then we always have that $p > \log\frac{pM^2}{\sigma^2}$ for $2 \leq p$, hence only case 2 and 3 apply. Similarly, if $e^2\frac{\sigma^2}{M^2} \leq 2$ then $p \geq e^2\frac{\sigma^2}{M^2}$ for all $2 \leq p$, hence only case 1 and 3 apply. This shows that the cases disjoint and cover all parameter configurations.

The definition $\Psi_p(M,\sigma^2)$ might appear strange but it does in fact capture the central p-norms of Poisson distributed random variables. This is stated more formally in the following lemma.

▶ **Lemma 4.** *There exist universal constants K_1 and K_2 satisfying that for a Poisson distributed random variable, X, with $\lambda = \mathrm{E}[X]$*

$$K_2\Psi_p(1,\lambda) \leq \|X-\lambda\|_p \leq K_1\Psi_p(1,\lambda) \ ,$$

for all $p \geq 2$.

[3] Here and throughout the paper $\log(x)$ will refer to the natural logarithm.

Bennett's inequality shows that we can bound the tail of $\sum_{x\in U} v(x,h(x))$ and Lemma 4 shows that $\Psi_p(M,\sigma^2)$ captures the central p-norms of the Poisson distribution. It is therefore not so surprising that we are to bound the p-norms of $\sum_{x\in U} v(x,h(x))$ using $\Psi_p(M,\sigma^2)$.

▶ **Theorem 5.** *Let $h\colon U \to [m]$ be a uniformly random function, let $v\colon U\times[m]\to\mathbb{R}$ be a fixed value function, and assume that $\sum_{j\in[m]} v(x,j) = 0$ for all keys $x\in U$. Define the random variable $X_v = \sum_{x\in U} v(x,h(x))$. Then for all $p\geq 2$*

$$\|X_v\|_p \leq L\Psi_p\left(M_v,\sigma_v^2\right) \ ,$$

where $L\leq 16e$ is a universal constant.

To get a further intuition for $\Psi_p(M,\sigma^2)$ is is instructive to apply Markov's inequality and compare the tail bound to Bennett's inequality. More precisely, assume that $\|Y-\mathrm{E}[Y]\|_p \leq L\Psi_p(M,\sigma^2)$ for a constant L and for all $p\geq 2$. Then we can use Markov's inequality to get the following tail bound for all $t>0$

$$\begin{aligned}\Pr\Big[\Big|Y-\mathrm{E}[Y]\Big|\geq t\Big] &\leq \left(\frac{\|Y-\mathrm{E}[Y]\|_p}{t}\right)^p \\ &\leq \begin{cases} \frac{L^2\sigma^2}{2t^2} & \text{if } t\leq L\max\left\{M,\frac{e\sigma}{\sqrt{2}}\right\} \\ \exp\left(-\frac{4t^2}{e^2L^2\sigma^2}\right) & \text{if } L\frac{e\sigma}{\sqrt{2}}\leq t\leq L\frac{e^2\sigma^2}{2M} \\ \exp\left(-\frac{t}{LM}\log\left(\frac{2tM}{L\sigma^2}\right)\right) & \text{if } L\max\left\{\frac{e^2\sigma^2}{2M},M\right\}\leq t \end{cases} \ . \end{aligned} \tag{4}$$

In order to obtain these bounds p is chosen as follows: If $t\leq \max\left\{M,\frac{e\sigma}{\sqrt{2}}\right\}$ then $p=2$ and otherwise p is chosen such that $\|Y-\mathrm{E}[Y]\|_p \leq e^{-1}t$. More precisely, we have that

$$p = \begin{cases} 2 & \text{if } t\leq L\max\left\{M,\frac{e\sigma}{\sqrt{2}}\right\} \\ \frac{4t^2}{e^2L^2\sigma^2} & \text{if } L\frac{e\sigma}{\sqrt{2}}\leq t\leq L\frac{e^2\sigma^2}{2M} \\ \frac{t}{LM}\log\left(\frac{2tM}{L\sigma^2}\right) & \text{if } L\max\left\{\frac{e^2\sigma^2}{2M},M\right\}\leq t \end{cases} \ .$$

We see that Equation (4) gives the same tail bound as Bennett's inequality, Equation (3), up to a constant in the exponent.

We also prove a matching lower bound to Theorem 5 which shows that $\Psi_p(M,\sigma^2)$ is the correct function to consider.

▶ **Theorem 6.** *Let $h\colon U\to[m]$ be a uniformly random function, then there exists a value function, $v\colon U\times[m]\to\mathbb{R}$, where $\sum_{j\in[m]} v(x,j)=0$ for all keys $x\in U$, such that the random variable $X_v=\sum_{x\in U} v(x,h(x))$ satisfies that for all $p\leq L_1|U|\log(m)$*

$$\left\|\sum_{x\in U} v(x,h(x))\right\|_p \geq L_2\Psi_p\left(M_v,\sigma_v^2\right) \ ,$$

where L_1 and L_2 are a universal constant.

1.4.2 The Moments of Tabulation Hashing

We analyze the p-norms of hash-based sums for simple tabulation hashing, and our analysis is the first that provides useful bounds for non-constant moments. Furthermore, it is also the first analysis of simple tabulation hashing that does not assume that c is constant. We

obtain an essentially tight understanding of this problem and show that simple tabulation hashing only works well when the range is large. This was also noted by Aamand et al. [2] and they solve this deficiency of simple tabulation hashing by introducing a new hashing scheme, tabulation-permutation hashing. We show that it is also possible to break the bad instances of simple tabulation hashing by using mixed tabulation hashing.

We introduce a bit of notation to make the theorems cleaner. We will view a value function $v\colon \Sigma^c \times [m] \to \mathbb{R}$ as a vector, more precisely, we let

$$\|v\|_q = \left(\sum_{x\in\Sigma^c}\sum_{j\in[m]} |v(x,j)|^q\right)^{1/q}$$

for all $q \in [1,\infty]$. For every key $x \in \Sigma^c$ we define $v[x]$ to be the sub-vector v restricted to x, more precisely, we let

$$\|v[x]\|_q = \left(\sum_{j\in[m]} |v(x,j)|^q\right)^{1/q}$$

for all $q \in [1,\infty]$.

1.4.2.1 Simple Tabulation Hashing

Our main result for simple tabulation hashing is a version of Theorem 5.

▶ **Theorem 7.** *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function, $v\colon \Sigma^c \times [m] \to \mathbb{R}$ a value function, and assume that $\sum_{j\in[m]} v(x,j) = 0$ for all keys $x \in \Sigma^c$. Define the random variable $V_v^{\text{simple}} = \sum_{x\in\Sigma^c} v(x,h(x))$. Then for all $p \geq 2$*

$$\left\|V_v^{\text{simple}}\right\|_p \leq L_1\Psi_p\left(K_c\gamma_p^{c-1}M_v, K_c\gamma_p^{c-1}\sigma_v^2\right) ,$$

where $K_c = (L_2c)^{c-1}$, L_1 and L_2 are universal constants, and

$$\gamma_p = \frac{\max\left\{\log(m) + \log\left(\frac{\sum_{x\in\Sigma^c}\|v[x]\|_2^2}{\max_{x\in\Sigma^c}\|v[x]\|_2^2}\right)/c, p\right\}}{\log\left(e^2m\left(\max_{x\in\Sigma^c}\frac{\|v[x]\|_1^2}{\|v[x]\|_2^2}\right)^{-1}\right)}$$

It is instructive to compare this result to Theorem 5 for fully random hashing. Ignoring the constant K_c, the result for simple tabulation hashing corresponds to the result for fully random hashing if we group keys into groups of size γ_p^{c-1}.

The definition of γ_p is somewhat complicated because of the generality of the theorem, but we will try to explain the intuition behind it. The expression $\max_{x\in\Sigma^c}\frac{\|v[x]\|_1^2}{\|v[x]\|_2^2}$ measures how spread out the mass of the value function is. It was also noted in the previous analysis by Aamand et al. [2] that this measure is naturally occurring. In fact, their result needs that $\max_{x\in\Sigma^c}\frac{\|v[x]\|_1^2}{\|v[x]\|_2^2} \leq m^{1/4}$. If we consider the example from the introduction of hashing below a threshold $l \leq m$ where each key, $x \in \Sigma^c$, has weight w_x, then the value function, v, will be $v(x,j) = w_x\left([j<l] - \frac{l}{m}\right)$ for $x \in \Sigma^c, j \in [m]$, and we then get that

$$\max_{x\in\Sigma^c}\frac{\|v[x]\|_1^2}{\|v[x]\|_2^2} = 4l\left(1-\frac{l}{m}\right) \leq 4l .$$

This correctly measures that the mass of the value function is mostly concentrated to the l positions of $[m]$.

The expression $\frac{\sum_{x\in\Sigma^c}\|v[x]\|_2^2}{\max_{x\in\Sigma^c}\|v[x]\|_2^2}$ is a measure for how many keys that have significant weight. This also showed up in the previous analyses of simple tabulation hashing [2, 22]. If we again consider the example from before, we get that

$$\frac{\sum_{x\in\Sigma^c}\|v[x]\|_2^2}{\max_{x\in\Sigma^c}\|v[x]\|_2^2} = \frac{\sum_{x\in\Sigma^c} w_x^2}{\max_{x\in\Sigma^c} w_x^2} \ .$$

We can summarize the example in the following corollary.

▶ **Corollary 8.** *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function, assign a weight, $w_x \in \mathbb{R}$, to every key, $x \in \Sigma^c$, and consider a threshold $l \leq m$. Define the random variable $V_v^{\text{simple}} = \sum_{x\in\Sigma^c} w_x\left([h(x) < l] - \frac{l}{m}\right)$. Then for all $p \geq 2$*

$$\left\|V_v^{\text{simple}}\right\|_p \leq \Psi_p\left(K_c\gamma_p^{c-1}\max_{x\in\Sigma^c}|w_x|\,, K_c\gamma_p^{c-1}\left(\sum_{x\in\Sigma^c} w_x^2\right)\frac{l}{m}\left(1-\frac{l}{m}\right)\right) \ ,$$

where $K_c = L_1\left(L_2 c\right)^{c-1}$, L_1 and L_2 are universal constants, and

$$\gamma_p = \frac{\max\left\{\log(m) + \log\left(\frac{\sum_{x\in\Sigma^c} w_x^2}{\max_{x\in\Sigma^c} w_x^2}\right)/c, p\right\}}{\log\left(\frac{e^2 m}{4l}\right)}$$

A natural question is how close Theorem 7 is to being tight. We show that if $\log(m) + \log\left(\frac{\sum_{x\in\Sigma^c}\|v[x]\|_2^2}{\max_{x\in\Sigma^c}\|v[x]\|_2^2}\right)/c = O\left(\log\left(1 + m\left(\max_{x\in\Sigma^c}\frac{\|v[x]\|_1^2}{\|v[x]\|_2^2}\right)^{-1}\right)\right)$ then the result is tight up to a universal constant depending only c. Formally, we prove the following lemma.

▶ **Theorem 9.** *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function, and $2 \leq p \leq L_1|\Sigma|\log(m)$, then there exists a value function, $v\colon U\times[m]\to\mathbb{R}$, where $\sum_{j\in[m]} v(x,j) = 0$ for all keys $x \in \Sigma^c$, and for which*

$$\left\|\sum_{x\in\Sigma^c} v(x,h(x))\right\|_p \geq K_c'\Psi_p\left(\gamma_p^{c-1}M_v, \gamma_p^{c-1}\sigma_v^2\right) \ ,$$

where $K_c' = L_1^c$ and L_1 is a universal constant, and

$$\gamma_p = \max\left\{1, \frac{p}{\log\left(e^2 m\left(\max_{x\in\Sigma^c}\frac{\|v[x]\|_1^2}{\|v[x]\|_2^2}\right)^{-1}\right)}\right\}$$

1.4.2.2 Mixed Tabulation Hashing

The results of simple tabulation hashing work well when the range is large and when the mass of the value function is on few coordinates. We show that mixed tabulation hashing works well even if the range is small.

▶ **Theorem 10.** *Let $h\colon \Sigma^c \to [m]$ be a mixed tabulation function with $d \geq 1$ derived characters, $v\colon \Sigma^c\times[m]\to\mathbb{R}$ a value function, and assume that $\sum_{j\in[m]} v(x,j) = 0$ for all keys $x \in \Sigma^c$. Define the random variable $V_v^{\text{mixed}} = \sum_{x\in\Sigma^c} v(x,h(x))$. For all $p \geq 2$ then*

$$\left\|V_v^{\text{mixed}}\right\|_p \leq \Psi_p\left(K_c\gamma_p^c M_v, K_c\gamma_p^c\sigma_v^2\right) \tag{5}$$

where $K_c = L_1\left(L_2 c\right)^c$, L_1 and L_2 are universal constants, and

$$\gamma_p = \max\left\{1, \frac{\log(m)}{\log(|\Sigma|)}, \frac{p}{\log(|\Sigma|)}\right\} \ .$$

Usually, in hashing contexts, we do not map to a much larger domain, i.e., we will usually have that $m \leq |U|^\gamma$ for some constant $\gamma \geq 1$. If this is the case then we can obtain the following nice tail bound for mixed tabulation hashing by using Markov's inequality.

▶ **Corollary 11.** *Let $h\colon \Sigma^c \to [m]$ be a mixed tabulation function with $d \geq 1$ derived characters, $v\colon \Sigma^c \times [m] \to \mathbb{R}$ a value function, and assume that $\sum_{j\in[m]} v(x,j) = 0$ for all keys $x \in \Sigma^c$. Define the random variable $V_v^{\text{mixed}} = \sum_{x\in\Sigma^c} v(x,h(x))$. If $m \leq |U|^\gamma$ for a value $\gamma \geq 1$ then for all $t \geq 0$*

$$\Pr\left[\left|V_v^{\text{mixed}}\right| \geq t\right] \leq \exp\left(-\frac{\sigma_v^2}{M_v^2}\mathcal{C}\left(\frac{tM_v}{\sigma_v^2}\right)/K_{c,\gamma}\right) + |U|^{-\gamma} \ ,$$

where $\mathcal{C}(x) = (x+1)\log(x+1) - x$, $K_{c,\gamma} = L_1\left(L_2c^2\gamma\right)^c$, and L_1 and L_2 are universal constants.

Proof. The idea is to combine Theorem 10 and Markov's inequality. We use Theorem 10 for $2 \leq p \leq \gamma\log|U|$ to get that

$$\left\|V_v^{\text{mixed}}\right\|_p \leq \Psi_p\left(K_c\gamma_p^c M_v, K_c\gamma_p^c\sigma_v^2\right) \ ,$$

where we can bound γ_p by

$$\gamma_p = \max\left\{1, \frac{\log(m)}{\log(|\Sigma|)}, \frac{p}{\log(|\Sigma|)}\right\} \leq c\gamma \ .$$

So we have that

$$\left\|V_v^{\text{mixed}}\right\|_p \leq \Psi_p\left(\left(L_2c^2\gamma\right)^c M_v, \left(L_2c^2\gamma\right)^c\sigma_v^2\right) \ .$$

Now by the same method as in Equation (4), we get the result. ◀

1.4.2.3 Adding a query element

In many cases, we would like to prove that these properties continue to hold even when conditioning on a query element. An example would be the case where we are interested in the weight of the elements in the bin for which the query element, q, hashes to, i.e., we would like that $\sum_{x\in S} w_x\,[h(x) = h(q)]$ is concentrated when conditioning on q. Formally, this corresponds to having the value function $v\colon \Sigma^c \times [m] \times [m]$ defined by $v(x,j,k) = w_x\,[x \in S]\,[j = k]$ and then proving concentration on $\sum_{x\in\Sigma^c\setminus\{q\}} v(x,h(x),h(q))$ when conditioning on q. We show that this holds both for simple tabulation and mixed tabulation.

▶ **Theorem 12.** *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function and let $q \in \Sigma^c$ be a designated query element. Let $v\colon \Sigma^c \times [m] \times [m] \to \mathbb{R}$ a value function, and assume that $\sum_{j\in[m]} v(x,j,k) = 0$ for all keys $x \in U$ and all $k \in [m]$. Define the random variable $V_{v,q}^{\text{simple}} = \sum_{x\in\Sigma^c\setminus\{q\}} v(x,h(x),h(q))$ and the random variables*

$$M_{v,q} = \max_{x\in\Sigma^c\setminus\{q\}, j\in[m]} |v(x,j,h(q))| \ ,$$
$$\sigma_{v,q}^2 = \frac{1}{m}\sum_{x\in\Sigma^c\setminus\{q\}}\sum_{j\in[m]} v(x,j,h(q))^2 \ ,$$

which only depend on the randomness of $h(q)$. Then for all $p \geq 2$

$$\mathrm{E}\left[\left(V_{v,q}^{\text{simple}}\right)^p \,\middle|\, h(q)\right]^{1/p} \leq \Psi_p\left(K_c\gamma_p^{c-1}M_{v,q}, K_c\gamma_p^{c-1}\sigma_{v,q}^2\right) \ ,$$

where $K_c = L_1 (L_2 c)^{c-1}$, L_1 *and* L_2 *are universal constants, and*

$$\gamma_p = \frac{\max\left\{\log(m) + \log\left(\frac{\sum_{x\in\Sigma^c}\|v[x]\|_2^2}{\max_{x\in\Sigma^c}\|v[x]\|_2^2}\right)/c, p\right\}}{\log\left(e^2 m \left(\max_{x\in\Sigma^c}\frac{\|v[x]\|_1^2}{\|v[x]\|_2^2}\right)^{-1}\right)} .$$

▶ **Theorem 13.** *Let* $h\colon \Sigma^c \to [m]$ *be a mixed tabulation hash function and let* $q \in \Sigma^c$ *be a designated query element. Let* $v\colon \Sigma^c \times [m] \times [m] \to \mathbb{R}$ *a value function, and assume that* $\sum_{j\in[m]} v(x,j,k) = 0$ *for all keys* $x \in U$ *and all* $k \in [m]$*. Define the random variable* $V_{v,q}^{\text{simple}} = \sum_{x\in\Sigma^c\setminus\{q\}} v(x,h(x),h(q))$ *and the random variables*

$$M_{v,q} = \max_{x\in\Sigma^c\setminus\{q\}, j\in[m]} |v(x,j,h(q))| ,$$

$$\sigma_{v,q}^2 = \frac{1}{m}\sum_{x\in\Sigma^c\setminus\{q\}}\sum_{j\in[m]} v(x,j,h(q))^2 ,$$

which only depend on the randomness of $h(q)$*. For all* $p \geq 2$ *then*

$$\mathrm{E}\left[\left(V_{v,q}^{\text{simple}}\right)^p \,\middle|\, h(q)\right]^{1/p} \leq \Psi_p\left(K_c\gamma_p^c M_{v,q}, K_c\gamma_p^c\sigma_{v,q}^2\right) \tag{6}$$

where $K_c = L_1 (L_2 c)^c$, L_1 *and* L_2 *are universal constants, and*

$$\gamma_p = \max\left\{1, \frac{\log(m)}{\log(|\Sigma|)}, \frac{p}{\log(|\Sigma|)}\right\} .$$

1.5 Technical Overview

1.5.1 Fully Random Hashing

1.5.1.1 Sub-Gaussian bounds

A random variable X is said to be sub-Gaussian with parameter σ if $\|X\|_p \leq \sqrt{p}\sigma$ for all $p \geq 2$. It is a well-known fact that the sum of independent bounded random variables are sub-Gaussian. In the context of fully random hashing, we have that

$$\left\|\sum_{x\in U} v(x,h(x))\right\|_p \leq \sqrt{4p}\sqrt{\sum_{x\in U}\|v[x]\|_\infty^2} . \tag{7}$$

A natural question is whether this is the best sub-Gaussian bound we can get. If we are just interested in the contribution to a single bin, i.e., $v(x,j) = w_x([j=0] - \frac{1}{m})$, then we can obtain a better sub-Gaussian bound. By using the result of Oleszkiewicz [20], we get that

$$\left\|\sum_{x\in U} v(x,h(x))\right\|_p \leq L\sqrt{\frac{p}{\log m}}\sqrt{\sum_{x\in U} w_x^2} , \tag{8}$$

where L is a universal constant. This shows that Equation (7) can be improved in certain situations. We improve on this by proving a generalization of Equation (8). We show that

$$\left\|\sum_{x\in U} v(x,h(x))\right\|_p \leq L\sqrt{\frac{p}{\log\left(\frac{e^2 m\sum_{x\in U}\|v[x]\|_\infty^2}{\sum_{x\in U}\|v[x]\|_2^2}\right)}}\sqrt{\sum_{x\in U}\|v[x]\|_\infty^2} , \tag{9}$$

where L is a universal constant. It is easy to check that if $v(x,j) = w_x([j=0] - \frac{1}{m})$ then it reduces to Equation (8) and that it is stronger than Equation (7).

1.5.1.2 Moments for general random variables

As part of our analysis we develop a couple of lemmas for general random variables which might be of independent interest. We prove a lemma that provides a simple bound for weighted sums of independent and identically distributed random variables.

▶ **Lemma 14.** *Let $(X_i)_{i\in[n]}$ and X be independent and identically distributed symmetric random variables, and let $(a_i)_{i\in[n]}$ be a sequence of reals.*[4] *If $p \geq 2$ is an even integer then*

$$\left\| \sum_{i\in[n]} a_i X_i \right\|_p \leq K \sup \left\{ \frac{p}{s} \left(\frac{\sum_{i\in[n]} a_i^s}{p} \right)^{1/s} \|X\|_s \;\middle|\; 2 \leq s \leq p \right\} ,$$

where $K \leq 4e$ is a universal constant.

If we consider Laplace distributed random variables then it is possible to show that Lemma 14 is tight up to a universal constant. Thus a natural question to ask is whether Lemma 14 is tight, i.e., can we prove a matching lower bound. But unfortunately, if you consider Gaussian distributed variables then we see that Lemma 14 is not tight. It would be nice if there existed a simple modification of Lemma 14 which had a matching lower bound.

1.5.1.3 Moments of functions of random variables

As part of the analysis of tabulation hashing, we will need to analyze random variables of the form $\Psi_p(X, Y)$ where X and Y are random variables. More precisely, we have to bound $\|\Psi_p(X, Y)\|_p$. It is not immediately clear how one would do this but we prove a general lemma that helps us in this regard.

▶ **Lemma 15.** *Let $f\colon \mathbb{R}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$ be a non-negative function which is monotonically increasing in every argument, and assume that there exist positive reals $(\alpha_i)_{i\in[n]}$ and $(t_i)_{i\in[n]}$ such that for all $\lambda \geq 0$*

$$f(\lambda^{\alpha_0} t_0, \ldots, \lambda^{\alpha_{n-1}} t_{n-1}) \leq \lambda f(t_0, \ldots, t_{n-1}) .$$

Let $(X_i)_{i\in[n]}$ be non-negative random variables. Then for all $p \geq 1$ we have that

$$\|f(X_0, \ldots, X_{n-1})\|_p \leq n^{1/p} \max_{i\in[n]} \left(\frac{\|X_i\|_{p/\alpha_i}}{t_i} \right)^{1/\alpha_i} f(t_0, \ldots, t_{n-1}) .$$

If we can choose $t_i = \|X_i\|_{p/\alpha_i}$ for all $i \in [n]$, then we get the nice expression

$$\|f(X_0, \ldots, X_{n-1})\|_p \leq n^{1/p} f(\|X_0\|_{p/\alpha_0}, \ldots, \|X_{n-1}\|_{p/\alpha_{n-1}}) .$$

Now the result is natural to compare to the triangle inequality that says that $\|X + Y\|_p \leq \|X\|_p + \|Y\|_p$, which corresponds to considering $f(x, y) = x + y$, and to Cauchy-Schwartz that says that $\|XY\|_p \leq \|X\|_{2p} \|Y\|_{2p}$, which corresponds to $f(x, y) = xy$. These two examples might point to that the $n^{1/p}$ is superfluous, but by considering $f(x_0, \ldots, x_{n-1}) = \max\{x_0, \ldots, x_{n-1}\}$ and Gaussian distributed variables, it can be shown that Lemma 15 is tight up to a constant factor.

[4] A symmetric random variable, X, is a random variable that is symmetric around zero, i.e., $\Pr[X \geq t] = \Pr[-X \geq t]$ for all $t \geq 0$.

1.5.2 Tabulation Hashing

1.5.2.1 Symmetrization

The analyses of chaoses have mainly focused on two types of chaoses: Chaoses generated by non-negative random variables and chaoses generated by symmetric random variables. It might appear strange that focus has not been on chaoses generated by mean zero random variables. The reason is that a symmetrization argument reduces the analysis of chaoses generated by mean zero random variables to the analysis of chaoses generated by symmetric random variables. More precisely, a standard symmetrization shows that

$$2^{-c}\left\|\sum_{i_0,\dots,i_{c-1}\in[n]} a_{i_0,\dots,i_{c-1}}\prod_{j\in[c]}\varepsilon_{i_j}^{(j)}X_{i_j}^{(j)}\right\|_p \leq \left\|\sum_{i_0,\dots,i_{c-1}\in[n]} a_{i_0,\dots,i_{c-1}}\prod_{j\in[c]}X_{i_j}^{(j)}\right\|_p \leq 2^c\left\|\sum_{i_0,\dots,i_{c-1}\in[n]} a_{i_0,\dots,i_{c-1}}\prod_{j\in[c]}\varepsilon_{i_j}^{(j)}X_{i_j}^{(j)}\right\|_p,$$

where $(\varepsilon_i^{(j)})_{i\in[n],j\in[n]}$ are independent Rademacher variables.[5]

In our case, we can assume that $v(x,h(x))$ is a mean zero random variable but is not necessarily symmetric. We can remedy this by using the same idea of symmetrization. We define $\varepsilon\colon \Sigma^c\to\{-1,1\}$ to be a simple tabulation sign function, more precisely, we have a fully random table, $T_\varepsilon\colon [c]\times\Sigma\to\{-1,1\}$, and ε is then defined by $\varepsilon(\alpha_0,\dots,\alpha_{c-1})=\prod_{i\in[c]}T(i,\alpha_i)$. We then prove that for all $p\geq 2$

$$2^{-c}\left\|\sum_{x\in\Sigma^c}\varepsilon(x)v(x,h(x))\right\|_p \leq \left\|\sum_{x\in\Sigma^c}v(x,h(x))\right\|_p \leq 2^c\left\|\sum_{x\in\Sigma^c}\varepsilon(x)v(x,h(x))\right\|_p. \tag{10}$$

The power of symmetrization lies in the fact that we get to assume that v is symmetric in the analysis without actually changing the value functions.

Somewhat surprisingly, we are able to improve the moment bound of Dahlgaard et al. [9] just by using symmetrization. Their result has a doubly exponential dependence on the size of the moment, p, which stems from a technical counting argument where they bound the number of terms which does not have an independent factor when expanding the expression $\left(\sum_{x\in\Sigma^c}v(x,h(x))\right)^p$. It appears difficult to directly improve their counting argument but by using Equation (10) we are able to circumvent this. Thus, just by using symmetrization and the insights of Dahlgaard et al. [9] we obtain the following result.

▶ **Lemma 16.** *Let $h\colon\Sigma^c\to[m]$ be a simple tabulation function, $\varepsilon\colon\Sigma^c\to\{-1,1\}$ be a simple tabulation sign function, and $v\colon\Sigma^c\times[m]\to\mathbb{R}$ be a value function. Then for every real number $p\geq 2$*

$$\left\|\sum_{x\in\Sigma^c}v(x,h(x))\right\|_p \leq 2^c\left\|\sum_{x\in\Sigma^c}\varepsilon(x)v(x,h(x))\right\|_p \leq \sqrt{4p}^c\sqrt{\sum_{x\in\Sigma^c}\|v[x]\|_\infty^2}.$$

[5] A Rademacher variable, ε, is a random variable chosen uniformly from the set $\{-1,1\}$, i.e., $\Pr[\varepsilon=-1]=\Pr[\varepsilon=1]=\frac{1}{2}$.

1.5.2.2 General value functions

For most applications of hashing, we are either interested in the number of balls landing in a bin or in the number of elements hashing below a threshold. But we are studying the more general setting where we have a value function. A natural question is whether it is possible to obtain a simpler proof for the simpler settings. We do not believe this to be the case since the general setting of value functions will naturally show up when proving results by induction on c. More precisely, let us consider the case where we are interested in the number of elements from a set, $S \subseteq \Sigma^c$, that hash to 0. We then want to bound $\sum_{x\in S}\left([h(x)=0]-\frac{1}{m}\right) = \sum_{x\in\Sigma^c}[x\in S]\left([h(x)=0]-\frac{1}{m}\right)$. This can be rewritten as[6]

$$\sum_{x\in\Sigma^c}[x\in S]\left([h(x)=0]-\tfrac{1}{m}\right) = \sum_{\alpha\in\Sigma}\sum_{y\in\Sigma^{c-1}}[(y,\alpha)\in S]\left([h(y)\oplus T(c-1,\alpha)=0]-\tfrac{1}{m}\right) .$$

So if we define the value function $v'\colon \Sigma\times[m]\to\mathbb{R}$ by

$$v'(\alpha,j) = \sum_{y\in\Sigma^{c-1}}[(y,\alpha)\in S]\left([h\oplus j=0]-\tfrac{1}{m}\right) ,$$

then we get that $\sum_{x\in S}\left([h(x)=0]-\frac{1}{m}\right) = \sum_{\alpha\in\Sigma} v'(\alpha, T(c-1,\alpha))$. Thus, we see that general value functions are natural to consider in the context of tabulation hashing.

Instead of shying away from general value functions, we embrace them. This force us look at the problem differently and guides us in the correct direction. Using this insight naturally leads us to use Equation (9) and we prove the following moment bound, which is strictly stronger than Lemma 16.

▶ **Lemma 17.** *Let $h\colon\Sigma^c\to[m]$ be a simple tabulation function, $\varepsilon\colon\Sigma^c\to\{-1,1\}$ be a simple tabulation sign function, and $v\colon\Sigma^c\times[m]\to\mathbb{R}$ be value function. Then for every real number $p\geq 2$*

$$\left\|\sum_{x\in\Sigma^c}\varepsilon(x)v(x,h(x))\right\|_p \leq \sqrt{K_c\frac{p\left(\max\{p,\log(m)\}\right)^{c-1}}{\log\left(1+\frac{m\sum_{x\in\Sigma^c}\|v[x]\|_\infty^2}{\sum_{x\in\Sigma^c}\|v[x]\|_2^2}\right)^c}}\sqrt{\sum_{x\in\Sigma^c}\|v[x]\|_\infty^2} ,$$

where $K_c=(Lc)^c$ for a universal constant L.

This statement is often weaker than Theorem 7 but perhaps a bit surprisingly, we will use Lemma 17 as an important step in the proof of Theorem 7.

1.5.2.3 Sum of squares of simple tabulation hashing

A key element when proving Theorem 7 is bounding the sums of squares

$$\sum_{j\in[m]}\left(\sum_{x\in\Sigma^c}v(x,h(x)\oplus j)\right)^2 . \tag{11}$$

This was also one of the main technical challenges for the analysis of Aamand et al. [2]. Instead of analyzing Equation (11), we will analyze a more general problem: Let $v_i\colon\Sigma^c\times[m]\to\mathbb{R}$ be a value function $i\in[k]$, we then want to understand the random variable.

$$\sum_{\substack{j_0,\dots,j_{k-1}\in[m]\\ \bigoplus_{i\in[k]}j_i=0}}\sum_{x_0,\dots,x_{k-1}\in\Sigma^c}\prod_{i\in[k]}v_i(x_i,j_i\oplus h(x_i)) \tag{12}$$

[6] For a partial key $y=(\beta_0,\dots,\beta_{c-2})\in\Sigma^{c-1}$, we let $h(y)=\bigoplus_{i\in[c-1]}T(i,\beta_i)$.

If we have $k = 2$ and $v_0 = v_1$ then this corresponds to Equation (11). By using a decoupling argument, it is possible to reduce the analysis of Equation (12) to the analysis of hash-based sums for simple tabulation hashing. We can then use Lemma 17 to obtain the following lemma.

▶ **Lemma 18.** *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation function, $\varepsilon\colon \Sigma^c \to \{-1, 1\}$ be a simple tabulation sign function, and $v_i\colon \Sigma^c \times [m] \to \mathbb{R}$ be a value function for $i \in [k]$. For every real number $p \geq 2$*

$$\left\| \sum_{j\in[m]} \left(\sum_{x\in\Sigma^c} \varepsilon(x) v(x, h(x)) \right)^2 \right\|_p \leq \left(\frac{Lc \max\{p, \log(m)\}}{\log\left(\frac{e^2 m \sum_{x\in\Sigma^c} \|v[x]\|_2^2}{\sum_{x\in\Sigma^c} \|v[x]\|_1^2} \right)} \right)^c \sum_{x\in\Sigma^c} \|v[x]\|_2^2 \ ,$$

where L is a universal constant.

1.5.2.4 Proving the main result

The proof of Theorem 7 is by induction on c. We will use Theorem 5 on one of the characters while fixing the other characters. This will give us an expression of the form

$$\left\| \Psi_p \left(\max_{\alpha\in\Sigma, j\in[m]} \left| \sum_{y\in\Sigma^{c-1}} v((y,\alpha), h(y) \oplus j) \right|, \frac{\sum_{\alpha\in\Sigma, j\in[m]} \left(\sum_{y\in\Sigma^{c-1}} v((y,\alpha), h(y)\oplus j) \right)^2}{m} \right) \right\|_p .$$

By applying Lemma 15, we bound this by

$$\Psi_p \left(\left\| \max_{\alpha\in\Sigma, j\in[m]} \left| \sum_{y\in\Sigma^{c-1}} v((y,\alpha), h(y)\oplus j) \right| \right\|_p, \left\| \frac{\sum_{\alpha\in\Sigma, j\in[m]} \left(\sum_{y\in\Sigma^{c-1}} v((y,\alpha), h(y)\oplus j) \right)^2}{m} \right\|_p \right) .$$

We will bound $\left\| \max_{\alpha\in\Sigma, j\in[m]} \left| \sum_{y\in\Sigma^{c-1}} v((y,\alpha), h(y)\oplus j) \right| \right\|_p$ by using the induction hypothesis, and we bound $\left\| \frac{\sum_{\alpha\in\Sigma, j\in[m]} \left(\sum_{y\in\Sigma^{c-1}} v((y,\alpha), h(y)\oplus j) \right)^2}{m} \right\|_p$ by using Lemma 18. While this sketch is simple, the actual proof is quite involved and technical since one has to be very careful with the estimates.

1.6 Mixed Tabulation Hashing in Context

Our concentration bounds for mixed tabulation hashing are similar to those Aamand et al. [2] for their tabulation-permutation hashing scheme and the schemes also have very similar efficiency, roughly a factor 2 slower than simple tabulation and orders of magnitude faster than any alternative with similar known concentration bounds. We shall make a more detailed comparison with tabulation-permutation in Section 1.6.1.

As mentioned in the beginning of the introduction, the big advantage of proving concentration bounds for mixed tabulation hashing rather than for tabulation-permutation is that mixed tabulation hashing has many other strong probabilistic properties that can now be used in tandem with strong concentration. This makes mixed tabulation an even stronger candidate to replace abstract uniform hashing in real implementations of algorithms preserving many of the asymptotic performance guarantees.

Mixed tabulation inherits all the nice probabilistic properties known for simple and twisted tabulation[7]. Dahlgaard et al. [9] introduced mixed tabulation hashing to further get good statistics over k-partitions as used in classic streaming algorithms for counting of distinct elements by Flajolet et al. [13, 14, 15], and for fast set similarity in large-scale machine learning by Li et al. [19, 23, 24].

Selective full randomness with mixed tabulation

The main result of Dahlgaard et al. [9] for mixed tabulation is that it has a certain kind of selective full randomness (they did not have a word for it). An ℓ-bit mask M with don't cares is of the form $\{0,1,?\}^\ell$. An ℓ-bit string $B \in \{0,1\}^\ell$ matches M if it is obtained from M by replacing each ? with a 0 or a 1. Given a hash function returning ℓ-bit hash values, we can use M to select the set Y of keys that match M. Consider a mixed tabulation hash function $h : \Sigma^c \to \{0,1\}^\ell$ using d derived characters. The main result of Dahlgaard et al. [9, Theorem 4] is that if the expected number of selected keys is less than $|\Sigma|/2$, then, w.h.p., the free (don't care) bits of the hash values of Y are fully random and independent. More formally,

▶ **Theorem 19** (Dahlgaard et al. [9, Theorem 4]). *Let $h : \Sigma^c \to \{0,1\}^\ell$ be a mixed tabulation hash function using d derived characters. Let M be an ℓ-bit mask with don't cares. For a given key set $X \subseteq \Sigma^c$, let Y be the set of keys from X with hash values matching M. If $\mathrm{E}[|Y|] \leq |\Sigma|/(1+\Omega(1))$, then the free bits of the hash values in Y are fully random with probability $1 - O(|\Sigma|^{1-\lfloor d/2 \rfloor})$.*

The above result is best possible in that since we only have $O(|\Sigma|)$ randomness in the tables, we cannot hope for full randomness of an asymptotically larger set Y.

In the applications from [9], we also want the size of the set Y to be concentrated around its mean and by Corollary 11, the concentration is essentially as strong as with fully random hashing and it holds for any $d \geq 1$.

In [9] they only proved weaker concentration bounds for the set Y selected in Theorem 19. Based on the concentration bounds for simple tabulation by Pătraşcu and Thorup [22], they proved that if the set Y from 19 had $\mathrm{E}[Y] \in [|\Sigma|/8, 3|\Sigma|/4]$, then within the same probability of $1 - O(|\Sigma|^{1-\lfloor d/2 \rfloor})$, it has

$$|Y| = \mathrm{E}[Y]\left(1 \pm O\left(\sqrt{\frac{\log|\Sigma|(\log\log|\Sigma|)^2}{|\Sigma|}}\right)\right). \tag{13}$$

With Corollary 11, for $\mathrm{E}[Y] = \Theta(|\Sigma|)$, we immediately tighten (13) to the cleaner

$$|Y| = \mathrm{E}[Y]\left(1 \pm O\left(\sqrt{\frac{\log|\Sigma|}{|\Sigma|}}\right)\right). \tag{14}$$

[7] This is not a black box reduction, but both twisted and mixed tabulation hashing applies simple tabulation to a some changed keys, so any statement holding for arbitrary sets of input keys is still valid. Moreover, mixed tabulation with one derived character corresponds to mixed tabulation applied to keys with an added 0-character head, and having more derived characters does not give worse results.

While the improvement is "only" a factor $(\log\log|\Sigma|)^2$, the important point here is that (14) is the asymptotic bound we would get with fully-random hashing. Also, while Dahlgaard et al. only proved (13) for the special case of $\mathrm{E}[Y] \in [|\Sigma|/8, 3|\Sigma|/4]$, our (14) is just a special case of Corollary 11 which holds for arbitrary values of $\mathrm{E}[Y]$ and arbitrary value functions.

Dahlgaard et al. presented some very nice applications of mixed tabulation to problems in counting and machine learning and machine learning. The way they use Theorem 19 is rather subtle.

1.6.1 Mixed Tabulation Hashing Versus Tabulation-Permutation Hashing

As mentioned earlier, our new concentration bounds are similar to those proved by Aamand et al. [2] for their tabulation-permutation hashing scheme. However, now we also have moment bounds covering the tail, and we have the first understanding of what happens when c is not constant. It is not clear if this new understanding applies to tabulation-permutation. As discussed above, the advantage of having the concentration bounds for mixed tabulation hashing is that we can use them in tandem with the independence result from Theorem 19, which does not hold for tabulation-permutation.

Tabulation-permutation is similar to mixed tabulation hashing in its resource consumption. Consider the mapping $\Sigma^c \to \Sigma^c$. Tabulation-permutation first uses simple tabulation $h : \Sigma^c \to \Sigma^c$. Next it applies a random permutation $\pi_i : \Sigma \overset{1-1}{\to} \Sigma$ to each output character $h(x)_i$, that is, $x \mapsto (\pi_1(h(x)_1), \ldots, \pi_c(h(x)_c)$. Aamand et al. [2] also suggest tabulation-1permutation hashing, which only permutes the most significant character. This scheme does not provide concentration for all value functions, but it does work if we select keys from intervals.

Aamand et al. [2] already made a thorough experimental and theoretical comparison between tabulation-permutation, mixed tabulation, and many other schemes. In this comparison, mixed tabulation played the role of a similar scheme with not as strong known concentration bounds. In the experiments, mixed tabulation hashing with c derived characters performed similar to tabulation-permutation in speed. Here we proved stronger concentration bounds for mixed tabulation even with a single character, where it should perform similar to tabulation-1permutation (both use $c + 1$ lookups). Both mixed tabulation hashing and tabulation-permutation hashing were orders of magnitude faster than any alternative with similar known concentration bounds. We refer to [2, 10] for more details. In particular, [10] compares mixed tabulation with popular cryptographic hash functions that are both slower and have no guarantees in these algorithmic contexts.

One interesting advantage of mixed tabulation hashing over tabulation-permutation hashing is that mixed tabulation hashing, like simple tabulation hashing, only needs randomly filled character tables. In contrast, tabulation-permutation needs tables that represent permutations. Thus, all we need to run mixed tabulation hashing is a pointer to some random bits. These could be in read-only memory shared across different applications. Read-only memory is much less demanding than standard memory since there can be no write-conflicts, so we could imagine some special large, fast, and cheap read-only memory, pre-filled with random bits, e.g., generated by a quantum-device. This would open up for larger characters, e.g., 16- or 32-bit characters, and it would free up the cache for other applications.

References

1 Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob Bæk Tejs Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. No repetition: Fast streaming with highly concentrated hashing. *CoRR*, 2020. arXiv:2004.01156.

2 Anders Aamand, Jakob Bæk Tejs Knudsen, Mathias Bæk Tejs Knudsen, Peter Michael Reichstein Rasmussen, and Mikkel Thorup. Fast hashing with strong concentration bounds. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1265–1278, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384259.

3 Anders Aamand, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.5.

4 Anders Aamand and Mikkel Thorup. Non-empty bins with simple tabulation hashing. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2498–2512. SIAM, 2019. doi:10.1137/1.9781611975482.153.

5 Radosław Adamczak and Rafał Latała. Tail and moment estimates for chaoses generated by symmetric random variables with logarithmically concave tails. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 48(4):1103–1136, 2012. doi:10.1214/11-AIHP441.

6 George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962. doi:10.1080/01621459.1962.10482149.

7 Vladimir Braverman, Kai-Min Chung, Zhenming Liu, Michael Mitzenmacher, and Rafail Ostrovsky. AMS without 4-wise independence on product domains. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPIcs*, pages 119–130. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.STACS.2010.2449.

8 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.

9 S. Dahlgaard, M. B. T. Knudsen, E. Rotenberg, and M. Thorup. Hashing for statistics over k-partitions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1292–1310, 2015. doi:10.1109/FOCS.2015.83.

10 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Practical hash functions for similarity estimation and dimensionality reduction. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6618–6628, USA, 2017. Curran Associates Inc. URL: http://dl.acm.org/citation.cfm?id=3295222.3295407.

11 Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *Proceedings of the 36th ICALP*, pages 354–365, 2009.

12 A. I. Dumey. Indexing for rapid random access memory systems. *Computers and Automation*, 5(12):6–9, 1956.

13 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985. Announced at FOCS'83.

14 Philippe Flajolet, Éric Fusy, Olivier Gandouet, and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In Analysis of Algorithms (AOFA)*, 2007.

15 Stefan Heule, Marc Nunkesser, and Alex Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the EDBT 2013 Conference*, pages 683–692, 2013.

16 Konrad Kolesko and Rafał Latała. Moment estimates for chaoses generated by symmetric random variables with logarithmically convex tails. *Statistics & Probability Letters*, 107:210–214, 2015. doi:10.1016/j.spl.2015.08.019.

17 Rafał Latała. Estimates of moments and tails of Gaussian chaoses. *The Annals of Probability*, 34(6):2315–2331, 2006. doi:10.1214/009117906000000421.

18 Joseph Lehec. *Moments of the Gaussian Chaos*, pages 327–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-15217-7_13.

19 Ping Li, Art B. Owen, and Cun-Hui Zhang. One permutation hashing. In *Proc. 26th NIPS*, pages 3122–3130, 2012.

20 Krzysztof Oleszkiewicz. On a nonsymmetric version of the Khinchine-Kahane inequality. In *Stochastic inequalities and applications*, pages 157–168. Springer, 2003.

21 Mihai Patrascu and Mikkel Thorup. Twisted tabulation hashing. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 209–228. SIAM, 2013. doi:10.1137/1.9781611973105.16.

22 Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3), June 2012. doi:10.1145/2220357.2220361.

23 Anshumali Shrivastava and Ping Li. Densifying one permutation hashing via rotation for fast near neighbor search. In *Proc. 31th International Conference on Machine Learning (ICML)*, pages 557–565, 2014.

24 Anshumali Shrivastava and Ping Li. Improved densification of one permutation hashing. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*, pages 732–741, 2014.

25 Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 90–99, 2013.

26 Mark N. Wegman and Larry Carter. New classes and applications of hash functions. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. Announced at FOCS'79.

27 Albert Lindsey Zobrist. A new hashing method with application for game playing. Technical Report 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.

In-Range Farthest Point Queries and Related Problem in High Dimensions

Ziyun Huang
Department of Computer Science and Software Engineering,
Penn State Erie, The Behrend College, USA

Jinhui Xu
Department of Computer Science and Engineering,
State University of New York at Buffalo, NY, USA

Abstract

Range-aggregate query is an important type of queries with numerous applications. It aims to obtain some structural information (defined by an *aggregate function* $F(\cdot)$) of the points (from a point set P) inside a given query range B. In this paper, we study the range-aggregate query problem in high dimensional space for two aggregate functions: (1) $F(P \cap B)$ is the farthest point in $P \cap B$ to a query point q in $\mathbb{R}^d$ and (2) $F(P \cap B)$ is the minimum enclosing ball (MEB) of $P \cap B$. For problem (1), called *In-Range Farthest Point (IFP) Query*, we develop a bi-criteria approximation scheme: For any $\epsilon > 0$ that specifies the approximation ratio of the farthest distance and any $\gamma > 0$ that measures the "fuzziness" of the query range, we show that it is possible to pre-process P into a data structure of size $\tilde{O}_{\epsilon,\gamma}(dn^{1+\rho})$ in $\tilde{O}_{\epsilon,\gamma}(dn^{1+\rho})$ time such that given any $\mathbb{R}^d$ query ball B and query point q, it outputs in $\tilde{O}_{\epsilon,\gamma}(dn^{\rho})$ time a point p that is a $(1-\epsilon)$-approximation of the farthest point to q among all points lying in a $(1+\gamma)$-expansion $B(1+\gamma)$ of B, where $0 < \rho < 1$ is a constant depending on ϵ and γ and the hidden constants in big-O notations depend only on ϵ, γ and Polylog(nd). For problem (2), we show that the IFP result can be applied to develop query scheme with similar time and space complexities to achieve a $(1+\epsilon)$-approximation for MEB. To the best of our knowledge, these are the first theoretical results on such high dimensional range-aggregate query problems. Our results are based on several new techniques, such as *multi-scale construction* and *ball difference range query*, which are interesting in their own rights and could be potentially used to solve other range-aggregate problems in high dimensional space.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Farthest Point Query, Range Aggregate Query, Minimum Enclosing Ball, Approximation, High Dimensional Space

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.75

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2206.07592`

Funding *Jinhui Xu*: The research of this author was supported in part by NSF through grant IIS-1910492 and by KAUST through grant CRG10 4663.2.

1 Introduction

Range search is a fundamental problem in computational geometry and finds applications in many fields like database systems and data mining [4, 27]. It has the following basic form: Given a set of n points P in $\mathbb{R}^d$, pre-process P into a data structure so that for any query range B from a certain range family (*e.g.,* spheres, rectangles, and halfspaces), it reports or counts the number of the points in $P \cap B$ efficiently. Range search allows us to obtain some basic information of the points that lie in a specific local region of the space.

In many applications, it is often expected to know more information than simply the number of points in the range. This leads to the study of range-aggregate query [2, 3, 6, 10, 13, 20, 21, 23, 25, 26, 32], which is a relatively new type of range search. The goal of

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 75; pp. 75:1–75:21


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

range-aggregate query is to obtain more complicated structural information (such as the diameter, the minimum enclosing ball, and the minimum spanning tree) of the points in the query range. Range-aggregate query can be generally defined as follows: Given a point set P, pre-process P into a data structure such that for any range B in a specific family, it outputs $F(P \cap B)$, where $F(\cdot)$ is a given *aggregate function* that computes a certain type of information or structure of $P \cap B$ like "diameter","minimum enclosing ball", and "minimum spanning tree". Range-aggregate queries have some interesting applications in data analytics and big data [16, 28, 29, 32], where it is often required to retrieve aggregate information of the records in a dataset with keys that lie in any given (possibly high dimensional) range.

In this paper, we study the range-aggregate query problem in high dimensions for spherical ranges. Particularly, we consider two aggregate functions for any $\mathbb{R}^d$ query ball B: (1) $F(P \cap B)$ is the farthest point in $P \cap B$ to a query point q in $\mathbb{R}^d$ and (2) $F(P \cap B)$ is the minimum enclosing ball (MEB) of $P \cap B$. We will focus in this paper on problem (1), called the *In-Range Farthest Point (IFP) Query*, and show that an efficient solution to IFP query also yields efficient solutions to the MEB problems. We start with some definitions.

▶ **Definition 1** (Approximate IFP (AIFP)). *Let P be a set of n points in $\mathbb{R}^d$, q be a point and B be a d-dimensional (closed) ball. A point $p \in P$ is a bi-criteria (ϵ, γ)-approximate in-range farthest point (or AIFP) of $q \in P$ in B, if there exists a point set P' such that the following holds, where ϵ and γ are small positive constants, and $B(1+\gamma)$ is the ball concentric with B and with radius $(1+\gamma)r$: (1) $P \cap B \subseteq P' \subseteq P \cap B(1+\gamma)$; (2) $p \in P'$; and (3) for any $p' \in P'$, $(1-\epsilon)\|p'-q\| \leq \|p-q\|$.*

Defining AIFP in this way enables us to consider all points in B and exclude all points outside of $B(1+\gamma)$. Points in the fuzzy region $B(1+\gamma) \setminus B$ may or may not be included in the farthest point query. Note that allowing fuzzy region is a commonly used strategy to deal with the challenges in many high dimensional similarity search and range query problems. For example, consider the classic near neighbor search problem, which is equivalent to spherical emptiness range search: Given a query sphere B in $\mathbb{R}^d$, report a data point p that lies in B if such a data point exists. In high dimensional space, obtaining an exact solution to such a query is very difficult. A commonly used technique for this problem is the Locality Sensitive Hashing (LSH) scheme [12]. Given a query ball B, LSH could report a data point in $B(1+\epsilon)$ for some given factor $\epsilon > 0$. In other words, a fuzzy region $B(1+\epsilon) \setminus B$ is allowed. Similarly, we can define approximate MEB for points in a given range with a fuzzy region.

▶ **Definition 2** (Minimum Enclosing Ball (MEB)). *Let P be a set of n points in $\mathbb{R}^d$. A d-dimensional (closed) ball B is an enclosing ball of P if $P \subset B$ and B is the minimum enclosing ball (MEB) of P if its radius r is the smallest among all enclosing balls. A ball B' is a $(1+\epsilon)$-approximate MEB of P for some constant $\epsilon > 0$ if it is an enclosing ball of P and its radius is no larger than $(1+\epsilon)Rad(P)$, where $Rad(P)$ is the radius of the MEB of P.*

▶ **Definition 3** (Approximate MEB (AMEB)). *Let P be a set of n points and B be any ball with radius r in $\mathbb{R}^d$. A ball B' with radius r' is a bi-criteria (ϵ, γ)-approximate MEB (or AMEB) of P in range B, if there exists a point set P' such that the following holds, where γ and ϵ are small positive constants: (1) $P \cap B \subseteq P' \subseteq P \cap B(1+\gamma)$; and (2) B' is a $(1+\epsilon)$-approximate MEB of P'.*

In this paper, we will focus on building a data structure for P so that given any query ball B and a point $q \in \mathbb{R}^d$, an AIFP of q in $P \cap B$ can be computed efficiently (*i.e.,* in sub-linear time in terms of n). Below are the main theorems of this paper. Let $\epsilon > 0, \gamma > 0$, $0 < \delta < 1$ be any real numbers.

▶ **Theorem 4.** *For any set P of n points in $\mathbb{R}^d$, it is possible to build a data structure of size $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ in $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ pre-processing time, where $0<\rho<1$ is a small constant depending on ϵ and γ. With this data structure, it is then possible to find a (ϵ,γ)-AIFP of any given query point q and query ball B in $O_{\epsilon,\gamma}(dn^{\rho}\log\delta^{-1}Polylog(nd))$ time with probability at least $1-\delta$.*

Note: In the above result, the relationship between ρ and ϵ,γ has a rather complicated dependence on several constants of p-stable distribution, which is inherited from the underlying technique of Locality Sensitive Hashing (LSH) scheme [12]. This indicates that for any ϵ,γ, we have $0<\rho<1$ and ρ approaches 1 as ϵ,γ approach 0.

We will also show how to use the AIFP data structure to answer MEB queries efficiently.

▶ **Theorem 5.** *For any set P of n points in $\mathbb{R}^d$, it is possible to build a data structure of size $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ in $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ pre-processing time, where $0<\rho<1$ is a small constant depending on ϵ and γ. With this data structure, it is then possible to find a (ϵ,γ)-AMEB for any query ball B in $O_{\epsilon,\gamma}(dn^{\rho}\log\delta^{-1}Polylog(nd))$ time with probability at least $1-\delta$.*

To our best knowledge, these are the first results on such range-aggregate problems in high dimensions. Each data structure has only a near linear dependence on d, a sub-quadratic dependence on n in space complexity, and a sub-linear dependence on n in query time.

Our Method. The main result on AIFP is based on several novel techniques, such as *multi-scale construction* and *ball difference range query*. Briefly speaking, multi-scale construction is a general technique that allow us to break the task of building an AIFP query data structure into a number of "constrained" data structures. Each such data structure is capable of correctly answering an AIFP query given that some assumption about the query holds (for example, the distance from q to its IFP is within a certain range). Multi-scale construction uses a number of "constrained" data structures of small size to cover all possible cases of a query, which leads to a data structure that can handle any arbitrary queries. Multi-scale construction is independent of the aggregate function, and thus has the potential be used as a general method for other types of range-aggregate query problems in high dimensional space. Another important technique is a data structure for the ball difference range query problem, which returns a point, if there is one, in the difference of two given query balls. The ball difference data structure is the building block for the constrained AIFP data structures, and is interesting in its own right as a new high dimensional range search problem.

Related Work. There are many results for the ordinary farthest point query problem in high dimensional space [11, 17, 19, 24]. However, to the best of our knowledge, none of them is sufficient to solve the IFP problem, and our result is the first one to consider the farthest point problem under the query setting. Our technique for the IFP problem also yields solutions to other range-aggregate queries problems, including the MEB query problem.

A number of results exist for various types of the range-aggregate query problem in fixed dimensional space. In [6], Arya, Mount, and Park proposed an elegant scheme for querying minimum spanning tree inside a query range. They showed that there exists a bi-criteria (ϵ_q,ϵ_w)-approximation with a query time of $O(\log n+(1/\epsilon_q\epsilon_w)^d)$. In [23], Nekrich and Smid introduced a data structure to compute an ϵ-coreset for the case of orthogonal query ranges and aggregate functions satisfying some special properties. Xue [30] considered the colored closest-pair problem in a (rectangular) range and obtained a couple of data structures with near linear size and polylogarithmic query time. Recently, Xue *et. al.* [31] further studied

more general versions of the closest-pair problem and achieved similar results. For the MEB problem under the range-aggregate settings, Brass *et al.* are the first to investigate the problem in 2D space, along with other types of aggregate functions (like width and the size of convex hull) [10]. They showed that it is possible to build a data structure with $O(n \cdot polylog(n))$ pre-processing space/time and $O(polylog(n))$ query time.

All the aforementioned methods were designed for fixed dimensional space, and thus are not applicable to high dimensions. Actually, range aggregation has rarely been considered in high dimensions, except for a few results that may be viewed as loosely relevant. For example, Abbar *et al.* [1] studied the problem of finding the maximum diverse set for points inside a ball with fixed radius around a query point. Their ideas are seemingly useful to our problem. However, since their ball always has the same fixed radius, their techniques are not directly applicable. In fact, a main technical challenge of our problem is how to deal with the arbitrary radius and location of the query range, which is overcome by our multi-scale construction framework. Another related work by Aumüller *et. al.* [8] has focused on random sampling in a given range. The technique is also not directly applicable to IFP.

1.1 Overviews of the Main Ideas

Below we describe the main ideas of our approaches. For simplicity, in the following we ignore the fuzziness of the query range. We approach the AIFP query problem by first looking at an easier version: given ball B and point q, find an approximate farthest point in $P \cap B$ to q, with the (strong) assumption that the radius of B is a fixed constant $r_B > 0$, and that the distance between q and its IFP in $P \cap B$ is within a range of $(d_{min}, d_{max}]$, where $d_{max} > d_{min} > 0$ are fixed constants. We call such a problem a constrained AIFP problem. We use a tuple (r_B, d_{min}, d_{max}) to denote such a constraint.

To solve the constrained AIFP problem, we develop a data structure for the *ball difference (BD) range query* problem, which is defined as follows: given two balls B_{in} and B_{out}, find a point that lies in $P \cap B_{in} \setminus B_{out}$. With such a data structure, it is possible to reduce an AIFP query with constraint (r_B, d_{min}, d_{max}) to a series of BD queries. Below we briefly describe the idea. Let $r_0 = d_{min}$, and for $i = 1, 2, 3 \ldots$, let $r_i = (1 + \epsilon) r_{i-1}$, where $\epsilon > 0$ is an approximation factor. For $i = 0, 1, \ldots$, we try to determine whether there is a point in $P \cap B$ whose distance to q is larger than r_i. Note that this can be achieved by a BD query with $B_{in} := B$ and B_{out} being the ball centered at q and with radius r_i. By iteratively doing this, eventually we will reach an index j such that it is possible to find a point $p \in B \cap P$ that satisfies the condition of $\|p - q\| > r_j$, but no point lies in $P \cap B$ whose distance to q is larger than $r_{j+1} = (1 + \epsilon) r_j$. Thus, p is a $(1 - O(\epsilon))$-approximate farthest point to q in $P \cap B$. From the definition of constrained AIFP query, it is not hard to see that this process finds the AIFP after at most $O(\log_{1+\epsilon} \frac{d_{max}}{d_{min}})$ iterations. Every BD data structure supports only B_{in} and B_{out} with fixed radii. This means that we need to build $O(\log_{1+\epsilon} \frac{d_{max}}{d_{min}})$ BD data structures for answering any AIFP query with constraint (r_B, d_{min}, d_{max}).

With the constrained AIFP data structure, we then extend it to a data structure for answering general AIFP queries. Our main idea is to use the aforementioned multi-scale construction technique to build a collection of constrained data structures, which can effectively cover (almost) all possible cases of the radius of B and the farthest distance from q to any point in $B \cap P$. More specifically, for any AIFP query, it is always possible to either answer the query easily without using any constrained data structures, or find a constrained data structure such that the AIFP query satisfies the constraint (r_B, d_{min}, d_{max}), and thus can be used to answer the AIFP query.

For AMEB query, we follow the main idea of Badoiu and Clarkson [9], and show that an AMEB query is reducible to a series of AIFP queries. More discussions are left to Section 5.

2 Constrained AIFP Query

In this section, we discuss how to construct a data structure to answer constrained AIFP queries. Particularly, given any ball B and point q satisfying the constraint (r_B, d_{min}, d_{max}),

- the radius of B is r_B,
- the distance from q to its farthest point to $P \cap B$ is within the range of $(d_{min}, d_{max}]$,

the data structure can find the AIFP to q in $P \cap B$ in sub-linear time (with high probability).

In the following, we let $\epsilon > 0$ be an approximation factor, $\gamma > 0$ be a factor that controls the region fuzziness and $0 < \delta < 1$ be a factor controlling the query success probability. The main result of this section is summarized as the following lemma.

▶ **Lemma 6.** *Let P be a set of n points in $\mathbb{R}^d$. It is possible to build a data structure for P with size $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \log(d_{max}/d_{min}))$ in $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \log(d_{max}/d_{min}))$ time, where $0 < \rho < 1$ is a real number depending on ϵ and γ, and the constants hidden in the big-O notation depend only on ϵ, γ. Given any query (B, q) that satisfies the constraint of (r_B, d_{min}, d_{max}), with probability at least $1 - \delta$, the data structure finds an (ϵ, γ)-AIFP for q in $P \cap B$ within time $O_{\epsilon,\gamma}(dn^{\rho} \log \delta^{-1} \log(d_{max}/d_{min}))$.*

In the following, we consider an AIFP query that satisfies constraint (r_B, d_{min}, d_{max}). As mentioned in last section, it is possible to reduce a constrained AIFP query to a series of *ball difference(BD) range queries*, which report a point in P that lies (approximately) in $B_{in} \setminus B_{out}$ for a given pair of $\mathbb{R}^d$ balls (B_{in} and B_{out}), or return NULL if no such point exists. Below, we describe the reduction using a ball-peeling strategy. We consider a series of balls $B_0, B_1, B_2, \ldots$ concentric at q with an exponentially increasing radius. Let $\xi > 0$ be a to-be-determined approximation factor, and $B_0 := \mathcal{B}(q, d_{min})$ which is the ball centered at q with radius d_{min}. For integer $i > 0$, let $B_{i+1} = B_i(1 + \xi)$ which is the ball obtained by enlarging the radius of B_i by a factor of $(1 + \xi)$. [1] For $i = 0, 1, 2, \ldots$, repeatedly perform a BD query with $B_{in} := B$ and $B_{out} := B_i$, until an index j is encountered such that the BD query reports a point p_j that lies in $P \cap B \setminus B_j$, but returns NULL when trying to find a point in $P \cap B \setminus B_{j+1}$. If ξ is a small enough constant, it is not hard to see that p_j is a good approximation of the IFP to q in $P \cap B$. Note that in this process, no more than $\log_{1+\xi}(d_{max}/d_{min})$ BD queries are required. This is because the distance between q and any point in B is at most d_{max}. Thus, it is not necessary to increase the radius of B_{out} to be more than d_{max} in the BD range query. The bound on the number of BD range queries then follows from the facts that the series of BD range queries starts with a B_{out} ball of radius d_{min} and each time the radius of B_{out} is increased by a factor of $1 + \xi$. This process is similar to peel a constant portion of B_{in} each time by B_{out}. See Figure 1 for an illustration.

The above discussion suggests that a constrained AIFP data structure can be built through (approximate) BD query data structures, which have the following definition. Let $\xi > 0$ be an approximation factor. A data structure is called ξ-error BD for a point set P, if given any balls B_{in} and B_{out}, it answers the following query (with high success probability):

[1] Throughout this paper we use similar notations. Let q be any point and $x > 0$ be real number. Then, $\mathcal{B}(q, x)$ denotes the ball centered at q and with radius x. Let B be any ball. For real number $y > 0$, we let $B(y)$ denote the ball obtained by enlarging (or shrinking if $y < 1$) the radius of B by a factor of y.

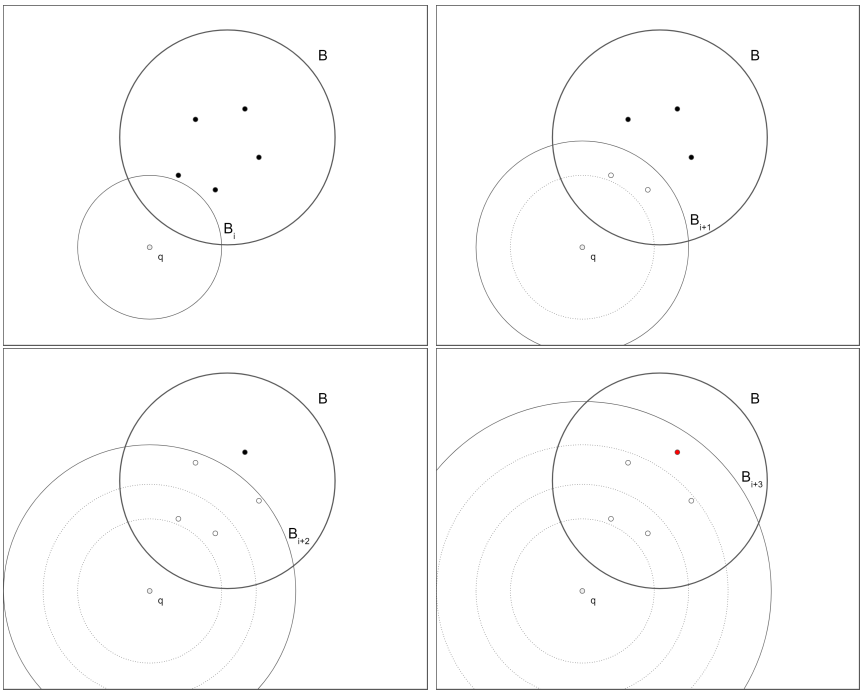


Figure 1 An illustration of answering a constrained AIFP query using BD queries.

1. If there exists a point in $P \cap (B_{in} \setminus B_{out})$, the data structure returns a point in $P \cap (B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1}))$.
2. Otherwise, it returns a point in $P \cap (B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1}))$ or NULL.

The details of how to construct a ξ-error BD data structure is left to the next subsection. Below is the main result of the BD query data structure for $\xi > 0$, fixed constant $r_{in} > 0, r_{out} > 0$ and success probability controlling factor $0 < \delta < 1$.

▶ **Lemma 7.** *It is possible to build a ξ-error BD query data structure of size $O_\xi(dn^{1+\rho}\log\delta^{-1})$ in $O_\xi(dn^{1+\rho}\log\delta^{-1})$ time, where $0 < \rho < 1$ depends only on ξ. The query time of this data structure is $O_\xi(dn^{\rho}\log\delta^{-1})$. For any pair of query balls B_{in} and B_{out} with radius r_{in} and r_{out}, respectively, the data structure answers the query with success probability at least $1-\delta$.*

Note that each BD query data structure works only for query balls B_{in} and B_{out} with fixed radii r_{in} and r_{out}, respectively. This means that the constrained AIFP data structure should consist of multiple BD data structures with different values of r_{in} and r_{out}.

From the above discussion, we know that a constrained AIFP data structure can be built by constructing a sequence of $\log(d_{max}/d_{min})$ BD data sturctures with $r_{in} := r_B$ and r_{out} being $d_{min}, (1+\xi)d_{min}, (1+\xi)^2 d_{min}, \ldots$. Such a data structure will allow us to answer constrained AIFP queries using the ball peeling strategy.

Given any constants $\epsilon > 0$, $\gamma > 0$, $0 < \delta < 1$, and constraint (r_B, d_{min}, d_{max}), the following Algorithm 1 builds a constrained AIFP data structure for a given point set P. The data structure is simply a collection of BD query data structures.

With such a collection of BD query data structures, we can answer any constrained AIFP query satisfying (r_B, d_{min}, d_{max}) by applying the ball peeling strategy mentioned before. The algorithm is formally described as the Algorithm 2 below.

Algorithm 1 Build-CAIFP($P; \epsilon, \gamma, \delta; r_B, d_{min}, d_{max}$).

Input: A $\mathbb{R}^d$ point set P with cardinality n. Constants $\epsilon > 0, \gamma > 0, 0 < \delta < 1$. Constraint tuple (r_B, d_{min}, d_{max}).
Output: A number of BD-Query data structures built with different parameters.

1: Let $\xi = \min\{(1-\epsilon)^{-1/2}-1, \gamma\}$. Construct a sequence of real numbers $r_0, r_1, r_2, \ldots, r_m$, by letting $r_0 = d_{min}$, m be the integer such that $r_0(1+\xi)^{m-1} < d_{max}$ and $r_0(1+\xi)^m \geq d_{max}$, $r_i = (1+\xi)r_{i-1}$ for $i = 1, 2, \ldots, m$, and $\delta' = \delta/m$.
2: **FOR** $i = 0, 1, 2, \ldots, m$, build a ξ-error BD query data structure for query balls with radii $r_{in} = r_B$ and $r_{out} = r_i$, with query success probability at least $1 - \delta'$.

Algorithm 2 Query-CAIFP(B, q).

Input: A constrained AIFP query (B, q) with constraint (r_B, d_{min}, d_{max}).
Output: A point p_{ans} that is an approximate farthest point in $B \cap Q$ to p, or NULL if no such point exists.

1: Initialize variable $p_{ans} \leftarrow$ NULL.
Note: In the following, we use m and r_i for $i = 0, 1, \ldots, m$ as in Algorithm 1.
2: **For** i from 0 to m: Make a query $(B, B_{out,i})$ to the BD-Query data structure BD_i, by letting $B_{out,i} := \mathcal{B}(q, r_i)$. If the query answer is NULL, **Return** p_{ans}. Otherwise update p_{ans} to be the query answer.
3: **Return** p_{ans}.

By some simple calculation, we know that the probability that all the BD queries in Algorithm 2 are successful is at least $1 - \delta$, and when this happens, the output point p_{ans} is an (ϵ, γ)-AIFP of q in $B \cap P$. This is summarized as the following lemma.

▶ **Lemma 8.** *With probability at least $1 - \delta$, Algorithm 2 outputs a point $p_{ans} \in B(1+\gamma)$ such that for any $q \in B \cap P$, $\|p_{ans} - p\| \geq (1-\epsilon)\|q - p\|$.*

Next we analyze the space/time complexity of the AIFP scheme. The query data structure is a combination of $m = O_{\epsilon,\gamma}(\log(d_{max}/d_{min}))$ BD data structures. From the discussion of BD data structures (see Lemma 7), every BD query data structure we build has space/time complexity $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1})$ where $0 < \rho < 1$ depends only on ϵ, γ. Each BD query takes $O_{\epsilon,\gamma}(dn^{\rho}\log\delta^{-1})$ time. Lemma 6 then follows.

2.1 The BD Query Scheme

In this subsection we present the BD query scheme. To our best knowledge, this is the first theoretical result to consider the BD range search problem. A very special case of BD query called the "annulus queries" where the two balls are co-centered is studied in [7]. Nonetheless, the technique is not directly applicable to general BD queries. Our BD range query scheme is based on the classic Locality Sensitive Hashing (LSH) technique [15, 5, 12] which has been a somewhat standard technique for solving the proximity problems in high dimensional space. The main idea of LSH is to utilize a family of hash functions (called an LSH family) that have some interesting properties. Given two points p and q in $\mathbb{R}^d$, if we randomly pick a function h from the LSH family, the probability that the event of $h(p) = h(q)$ happens will be high if $\|p - q\|$ is smaller than a threshold value, and the probability for the same event will be lower if $\|p - q\|$ is larger. Such a property of the LSH family allows us to develop hashing and bucketing based schemes to solve similarity search problems in high dimensional space. Below is the definition of an LSH family.

▶ **Definition 9.** *Let $0 < r_1 < r_2$ and $1 > \mathsf{P}_1 > \mathsf{P}_2 > 0$ be any real numbers. A family $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$, where U can be any set of objects, is called $(r_1, r_2, \mathsf{P}_1, \mathsf{P}_2)$-sensitive, if for any $p, q \in \mathbb{R}^d$.*

1. *if $\|p - q\| \leq r_1$, then $Pr_{\mathcal{H}}[h(p) = h(q)] \geq \mathsf{P}_1$,*
2. *if $\|p - q\| > r_2$, then $Pr_{\mathcal{H}}[h(p) = h(q)] \leq \mathsf{P}_2$.*

It was shown in [12] that for any dimension d and any $r > 0, c > 1$, an $(r, cr, \mathsf{P}_1, \mathsf{P}_2)$-sensitive family $\mathcal{H}$ exists, where $1 > \mathsf{P}_1 > \mathsf{P}_2 > 0$ depends only on c. Every hash function $h(p) : \mathbb{R}^d \to \mathbb{Z}$ maps a point p in $\mathbb{R}^d$ to an integer, and $h(p)$ has the form $h(p) = \lfloor \frac{a \cdot p + b}{r} \rfloor$ for some $\mathbb{R}^d$ vector a and integers b, r. It takes $O(d)$ time to sample a hash function h from such a family and compute $h(p)$. Our data structure will make use of two such families. Let $\mathcal{H}_{in}$ be an $(r_{in}, (1+\xi)r_{in}, \mathsf{P}_1, \mathsf{P}_2)$-sensitive family, and $\mathcal{H}_{out}$ be a $((1+\xi)^{-1} r_{out}, r_{out}, \mathsf{P}_1, \mathsf{P}_2)$-sensitive family, where **$\mathbf{0 < \mathsf{P}_1, \mathsf{P}_2 < 1}$ are constants depending only on $\boldsymbol{\xi}$**, as described in [12]. Given any BD-query (B_{in}, B_{out}) with the centers of the balls being o_{in}, o_{out} respectively, the family $\mathcal{H}_{in}$ helps us to identify points that are close enough to o_{in} (and therefore lie in B_{in}), and $\mathcal{H}_{out}$ helps us to identify points that are far away enough from o_{out} (and therefore lie outside of B_{out}).

High level idea. Our approach is based on a novel bucketing and query scheme that utilizes the properties of the LSH family. Before presenting the technical details, We first illustrate the high level idea. For convenience, we assume for now that the functions in $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$ have range $\{0, 1\}$ (this is achievable by some simple modification to these hash function families). We use a randomized process to create a hybrid random hash function $S(\cdot)$ that maps any point in $\mathbb{R}^d$ to a bit string. Such a function $S(\cdot)$ is a concatenation of a number of hash functions drawn from $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$. Given $p \in \mathbb{R}^d$, $S(\cdot)$ applies the aforementioned hash functions (drawn from $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$) on p to obtain a bit-string. With such a function $S(\cdot)$, consider comparing the bit-strings of $S(p), S(q)$ for points $p, q \in \mathbb{R}$. Intuitively, based on the properties of $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$, we know that if p, q are close enough, $S(p)$ and $S(q)$ should have many common bits in positions that are determined by functions from $\mathcal{H}_{in}$. Contrarily, if p, q are far away, $S(p)$ and $S(q)$ should have only a few common bits in positions that are determined by functions from $\mathcal{H}_{out}$.

For every point $p \in P$, we use $S(p)$ to compute a bit-string label for p, and put p into the corresponding buckets (*i.e.,* labeled with the same bit-strings). To answer a given BD query B_{in}, B_{out} with centers of the balls being o_{in}, o_{out}, respectively, we compute $S(o_{in})$ and $S(o_{out})$. Note that, based on the above discussion, we know that if a point p satisfies the condition of $p \in B_{in} \setminus B_{out}$, then $S(p)$ and $S(o_{in})$ should have many common bits in the positions determined by $\mathcal{H}_{in}$, and $S(p)$ and $S(o_{out})$ should have few common bits in the positions determined by $\mathcal{H}_{out}$. Thus, by counting the number of common bits in the labels, we can then locate buckets that are likely to contain points close to o_{in} and far away from o_{out}, *i.e.,* points are likely to be in $B_{in} \setminus B_{out}$. To achieve the desired outcome, we will create multiple set of buckets using multiple random functions $S(p)$.

Details of the Algorithms. After understanding the above general idea, we now present the data structure and the query algorithm along with the analysis. Let $\mathsf{P}'_1 = (1 + \mathsf{P}_1)/2, \mathsf{P}'_2 = (1+\mathsf{P}_2)/2, \eta = (\mathsf{P}'_1 - \mathsf{P}'_2)/3, a = \lceil (2\mathsf{P}'_1 \ln 3)/\eta^2 \rceil$. $\mathsf{P}''_1 = 2^{-2a} \cdot 4/9$, $\mathsf{P}''_2 = 2^{-2a}/3, b = \lceil \log_{1/\mathsf{P}''_2} n \rceil$, $\rho = \frac{\ln 1/\mathsf{P}''_1}{\ln 1/\mathsf{P}''_2}$, and $c = \lceil n^\rho / \mathsf{P}''_1 \rceil$. Let F_Z be a function that maps every element in $\mathbb{Z}$ randomly to 0 or 1, each with probability 1/2. The following Algorithm 3 shows how to construct a ξ-error BD range query data structure for any point set P and radii r_{in} and r_{out}. The data structure consists of c groups of buckets, each created using a random function $S(p)$ that maps a point to a bit-string of total length $2ab$.

Algorithm 3 CreateBuckets$(P, \xi, r_{in}, r_{out})$.

Input: A point set P. Parameters $\xi > 0, , r_{in} > 0, r_{out} > 0$.

Output: $\mathcal{G}_1, \mathcal{G}_2, \ldots \mathcal{G}_c$. Each $\mathcal{G}_i$ is a collection of *buckets* (*i.e.,* sets of points of P). Each bucket $G \in \mathcal{G}_i$ is labeled with *a bit-string*, which is a concatenation of sub-bit-strings $\text{LAB}_{in}(G,1)$, $\text{LAB}_{out}(G,1)$, $\text{LAB}_{in}(G,2)$, $\text{LAB}_{out}(G,2)$, ..., $\text{LAB}_{in}(G,b)$, $\text{LAB}_{out}(G,b)$. For every $p \in P$ and $i = 1, 2, \ldots, c$, p appears in one of the buckets in $\mathcal{G}_i$.

1: Initialize $\mathcal{G}_i, i = 1, 2, \ldots, c$, as empty sets. Each $\mathcal{G}_i$ will be used as a container for buckets.

2: Randomly sample abc functions from family $\mathcal{H}_{in}$, and also abc functions from family $\mathcal{H}_{out}$. Denote these functions as $h_{in,i,j,k}$ and $h_{out,i,j,k}$, for integers $1 \le i \le a, 1 \le j \le b, 1 \le k \le c$. For every $h_{in,i,j,k}$, $h_{out,i,j,k}$ and every $p \in Q$, compute $F_Z(h_{in,i,j,k}(p))$ and $F_Z(h_{out,i,j,k}(p))$.

3: **FOR** k from 1 to c:

- For every point $p \in P$, we create a bit-string $S(p)$ that concatenates $\text{LAB}_{in}(p,1), \text{LAB}_{out}(p,1), \text{LAB}_{in}(p,2), \text{LAB}_{out}(p,2), \ldots, \text{LAB}_{in}(p,b), \text{LAB}_{out}(p,b)$: For j from 1 to b, let $\text{LAB}_{in}(p,j)$, $\text{LAB}_{out}(p,j)$ be a pair of bit-strings of length a, each with the i-th bit being $F_Z(h_{in,i,j,k}(p))$, $F_Z(h_{out,i,j,k}(p))$, respectively, for $i = 1, 2, \ldots a$.
- **IF** there is already a bucket G in $\mathcal{G}_k$ with label $S(G) = S(p)$, **DO**: Put p into G.
- **ELSE, DO**: Create a new bucket G and put G into $\mathcal{G}_k$, set the label of G as $S(G) = S(p)$. Put p into G.

With the BD range query data structure created by the Algorithm 3, we can use the Algorithm 4 below to answer a BD range query for any given pair of balls (B_{in} and B_{out}). The main idea of the algorithm compute a bit-string label S for the query, then examine points in buckets with labels that satisfy certain properties (*e.g.* should have enough common bits with S). Due to the fact that we label these buckets using functions from two LSH families, it can be shown that the chance for us to find a point in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$ from one of the examined buckets will be high if there exists a point in $B_{in} \setminus B_{out}$.

In the following we show the correctness of Algorithm 4. Consider the for loop in Step 1 of Algorithm 4 when answering a query (B_{in}, B_{out}). Using the notations from Algorithm 4, for any k from 1 to c in Step 1, we have the following lemma, which shows that if a point in P lies in (or outside of) the query range, the number of common bits between its bucket label and the label computed from the query would likely (or unlikely) be high, respectively.

▶ **Lemma 10.** *Let $p \in P$ be a point that lies in $B_{in} \setminus B_{out}$, and $q \in P$ be a point that does NOT lie in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$. Let $S(p) = \text{LAB}_{in}(p,1), \text{LAB}_{out}(p,1), \text{LAB}_{in}(p,2), \text{LAB}_{out}(p,2), \ldots, \text{LAB}_{in}(p,b), \text{LAB}_{out}(p,b)$ and $S(q) = \text{LAB}_{in}(q,1), \text{LAB}_{out}(q,1), \text{LAB}_{in}(q,2), \text{LAB}_{out}(q,2), \ldots, \text{LAB}_{in}(q,b), \text{LAB}_{out}(q,b)$ be the labels of the bucket in $\mathcal{G}_k$ that contains p and q, respectively. For any $j = 1, 2, \ldots, b$, the following holds.*

- $Pr[COM(\text{LAB}_{in}(p,j), \text{LAB}_{in}(o_{in},j)) \ge t_1 \wedge COM(\text{LAB}_{out}(p,j), \text{LAB}_{out}(o_{out},j)) \ge t_2] \ge 4/9$.
- $Pr[COM(\text{LAB}_{in}(q,j), \text{LAB}_{in}(o_{in},j)) \ge t_1 \wedge COM(\text{LAB}_{out}(q,j), \text{LAB}_{out}(o_{out},j)) \ge t_2] \le 1/3$.

Proof. Since $p \in B_{in} \setminus B_{out}$, we have $\|p - o_{in}\| \le r_{in}$ and $\|p - o_{out}\| \ge r_{out}$. For any hash function $h_1 \in \mathcal{H}_{in}$ and $h_2 \in \mathcal{H}_{out}$, $\Pr[h_1(p) = h_1(o_{in})] \ge \mathsf{P}_1$ and $\Pr[h_2(p) = h_2(o_{out})] \le \mathsf{P}_2$. Thus, we have $\Pr[F_Z(h_1(p)) = F_Z(h_1(o_{in}))] \ge (\mathsf{P}_1 + 1)/2$ and $\Pr[F_Z(h_2(p)) = 1 - F_Z(h_2(o_{out}))] \le (1 - \mathsf{P}_2)/2$. This means that for any $i = 1, 2 \ldots, a$, the probability that the

Algorithm 4 BD-Query(B_{in}, B_{out}).

Input: Two $\mathbb{R}^d$ balls: B_{in} with center o_{in} and radius r_{in}, and B_{out} with center o_{out} and radius r_{out}. Assume that collections $\mathcal{G}_1, \mathcal{G}_2, \dots \mathcal{G}_c$ have already been generated by algorithm CreateBucket.

Output: A point $p \in Q$, or NULL.

Note: The algorithm probes a number of points in the buckets of $\mathcal{G}_1, \mathcal{G}_2, \dots \mathcal{G}_c$ until a suitable point is found as the output, or terminates and returns NULL when no such point can be found or the number of probes exceeds a certain limit.

1: Do the following, but terminate and return NULL when $3c$ points are examined: **FOR** k from 1 to c:
- Create a bit string S that concatenates $\text{LAB}_{in}(o_{in}, 1)$, $\text{LAB}_{out}(o_{out}, 1)$, $\text{LAB}_{in}(o_{in}, 2)$, $\text{LAB}_{out}(o_{out}, 2)$, …, $\text{LAB}_{in}(o_{in}, b)$, $\text{LAB}_{out}(o_{out}, b)$: For j from 1 to b, let $\text{LAB}_{in}(o_{in}, j)$, $\text{LAB}_{out}(o_{out}, j)$ be a pair of bit strings of length a, with the i-th bit of each string being $F_Z(h_{in,i,j,k}(o_{in}))$, $1 - F_Z(h_{out,i,j,k}(o_{out}))$, respectively, for $i = 1, 2, \dots a$.
- Create a bit string S' that concatenates $\text{LAB}'_{in}(1), \text{LAB}'_{out}(1), \text{LAB}'_{in}(2), \text{LAB}'_{out}(2), \dots,$ $\text{LAB}'_{in}(b), \text{LAB}'_{out}(b)$: Each of these sub bit strings is a random bit string of length a, drawn uniformly randomly from $\{0,1\}^a$.
- If there exists some integer j such that $\text{COM}(\text{LAB}_{in}(o_{in}, j), \text{LAB}'_{in}(j)) < t_1$ or $\text{COM}(\text{LAB}_{out}(o_{out}, j), \text{LAB}'_{out}(j)) < t_2$, where $\text{COM}(x, y)$ counts the number of common digits of 2 bit strings x, y, $t_1 = \mathsf{P}'_1 a - \eta a, t_2 = (1 - \mathsf{P}_2)a/2 - \eta a$ **CONTINUE.**
- If there is no bucket in $\mathcal{G}_k$ that is labeled with S', **CONTINUE.**
- Examine all the points in the bucket G in $\mathcal{G}_k$ that is labeled with S'. Stop when there a point $p \in G$ such that $p \in B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$. Return p.

2: Return NULL if no point is returned in the above process.

i-th bit of $\text{LAB}_{in}(p, j)$ is the same as that of $\text{LAB}_{in}(o_{in}, j)$ is at least $\mathsf{P}'_1 = (\mathsf{P}_1 + 1)/2$. Since the hash functions to determine each of the bits are drawn independently, an estimation of $X = \text{COM}(\text{LAB}_{in}(p, j), \text{LAB}_{in}(o_{in}, j))$ can be obtained by $\Pr[F_Z(h_1(p)) = F_Z(h_1(o_{in}))] \geq (\mathsf{P}_1 + 1)/2$ using the concentration inequalities for binomial distributions. Using a variant of the Chernoff inequalities from [22], we have

$$\Pr[X \leq \mathsf{P}'_1 a - \eta a] \leq e^{-(\eta a)^2/(2\mathsf{P}'_1 a)}.$$

From the definition of the parameters, we know that $\Pr[X \leq \mathsf{P}'_1 a - \eta a] \leq 1/3$ (by simple calculation). Thus, we have $\Pr[X \geq t_1] \geq 2/3$.

Let $Y = \text{COM}(\text{LAB}_{out}(p, j), \text{LAB}_{out}(o_{out}, j))$. From $\Pr[F_Z(h_2(p)) = 1 - F_Z(h_2(o_{out}))] \leq (1 - \mathsf{P}_2)/2$ and using a similar argument as above, we can also obtain $\Pr[Y \geq t_2] \geq 2/3$ (the details are omitted). Since the hash functions are drawn independently, we have $\Pr[X \geq t_1 \wedge Y \geq t_2] \geq 4/9$.

In the following we discuss the case that $q \notin B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$. This means either $\|q - o_{in}\| \geq (1+\xi) r_{in}$ or $\|q - o_{out}\| \leq (1+\xi)^{-1} r_{out}$. We first consider the case $\|q - o_{in}\| \geq (1+\xi) r_{in}$. For any hash function $h_1 \in \mathcal{H}_{in}$, $\Pr[h_1(q) = h_1(o_{in})] \leq \mathsf{P}_2$. Thus, we have $\Pr[F_Z(h_1(q)) = F_Z(h_1(o_{in}))] \leq (\mathsf{P}_2 + 1)/2$. This means that for any $i = 1, 2 \dots, a$, the probability that the i-th bit of $\text{LAB}_{in}(q, j)$ is the same as that of $\text{LAB}_{in}(o_{in}, j)$ is at most $\mathsf{P}'_2 = (\mathsf{P}_2 + 1)/2$. Again, we use a concentration inequality to obtain an estimation of $X = \text{COM}(\text{LAB}_{in}(q, j), \text{LAB}_{in}(o_{in}, j))$. Using a variant of the Chernoff inequalities from [22], we have

$$\Pr[X \geq \mathsf{P}'_2 a + \eta a] \leq e^{-(\eta a)^2/(2\mathsf{P}'_2 a + \eta a/3)}.$$

Note that $a = (2\mathsf{P}'_1 \ln 3)/\eta^2 \geq ((2\mathsf{P}'_2 + \eta/3)\ln 3)/\eta^2$, which implies that $e^{-(\eta a)^2/(2\mathsf{P}'_2 a + \eta a/3)} \leq 1/3$ (by simple calculation). Thus, we have $\Pr[X \geq \mathsf{P}'_2 a + \eta a] \leq 1/3$. Also, since $\mathsf{P}'_2 a + \eta a < \mathsf{P}'_1 a - \eta a = t_1$, we get $\Pr[X \geq t_1] \leq 1/3$. This immediately implies that $\Pr[\mathrm{COM}(\mathrm{LAB}_{in}(q,j), \mathrm{LAB}_{in}(o_{in},j)) \geq t_1 \wedge \mathrm{COM}(\mathrm{LAB}_{out}(q,j), \mathrm{LAB}_{out}(o_{out},j)) \geq t_2] \leq 1/3$.

The argument for the case $\|q - o_{out}\| \leq (1+\xi)^{-1} r_{out}$ is similar. Thus, we omit it here. This completes the proof. ◀

From the above lemma, we can conclude the following by basic calculation. For any k from 1 to c in Step 1 of Algorithm 4 (if the loop is actually executed), let $p \in P$ be a point that lies in $B_{in} \setminus B_{out}$, and $q \in P$ be a point that does NOT lie in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$, we have the following.

▶ **Lemma 11.** *Let G_p and G_q be the buckets in $\mathcal{G}_k$ that contain p and q, respectively. The probability for G_p to be examined is no smaller than $2^{-2ab}(4/9)^b = (\mathsf{P}''_1)^b$, and the probability for the event "ALL such G_p for k from 1 to c are NOT examined" is at most $(1-(\mathsf{P}''_1)^b)^c \leq 1/e$. The probability for G_q to be examined is no larger than $2^{-2ab}(1/3)^b = (\mathsf{P}''_2)^b \leq 1/n$.*

With the above lemma, we can obtain the following lemma using an argument similar to [15] for near neighbor search with LSH. This proves the correctness of the query scheme.

▶ **Lemma 12.** *If there exists a point in P that lies in $B_{in} \setminus B_{out}$, with probability at least $1/4$, Algorithm 4 reports a point in P that lies in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$.*

The complexity of the data structure is $O(dabcn)$, which is $O_\xi(dn^{1+\rho}\log n)$ from $a = O_\xi(1), b = O_\xi(\log n)$ and $c = O_\xi(n^\rho)$, and ρ and the constant hidden in the big-O notation depends only on ξ. The query time is $O(abcd)$, which is $O_\xi(dn^\rho \log n)$. To achieve $1-\delta$ success probability, it suffices to concatenate $O(\log \delta)$ such data structures together.

Due to space limit. we leave the proof of the above 2 lemmas and the full proof of Lemma 7 to the full version of the paper.

3 Multi-scale Construction

In this section, we present the *multi-scale construction* method, which is a standalone technique with potential to be used to other high dimensional range-aggregate query problems.

The multi-scale construction method is motivated by several high dimensional geometric query problems that share the following common feature: they are challenging in the general settings, but become more approachable if some key parameters are known in advance. The AIFP query problem discussed in this paper is such an example. In the previous section, we have shown how to construct an AIFP data structure if we fix the size of the query ball and know that the farthest distance lies in a given range.

The basic ideas behind multi-scale construction are the follows. Firstly, we know that if a problem is solvable when one or more key parameters are fixed, a feasible way to solve the general case of the problem is to first enumerate all possible cases of the problem defined by (the combinations of) the values of the parameters. Then, solve each case of the problem, and finally obtain the solution from that of all the enumerated cases. The multi-scale construction method follows a similar idea. More specifically, to obtain a general AIFP query data structure, the multi-scale construction method builds a set of constrained AIFP query data structures that cover all possible radii of B and farthest distance value. Secondly, since it is impossible to enumerate the infinite number of all possible values for these parameters, our idea is to sample a small set of fixed radii (based on the distribution of the points in P)

and build constrained AIFP data structures only for the set of sampled values. This will certainly introduce errors. However, good approximations are achievable by using a range cover technique.

Below we first briefly introduce two key ingredients of our method, Aggregation Tree and Range Cover, and then show how they can be used to form a multi-scale construction.

3.1 Aggregation Tree and Range Cover

In this subsection, we briefly introduce the two components of the multi-scale construction scheme: the *aggregation tree* and the *range cover* data structure. We first introduce aggregation tree, which is used in [18] as an ingredient of the range cover data structure. It is essentially a slight modification of the Hierarchical Well-Separated Tree (HST) introduced in [14]. Below is the definition of an aggregation tree: (1) Every node v (called *aggregation node*) represents a subset $P(v)$ of P, and the root represents P; (2)Every aggregation node v is associated with a representative point $re(v) \in P(v)$ and a size $s(v)$. Let $Dia(P(v))$ denotes the diameter of $P(v)$, $s(v)$ is a polynomial approximation of $Dia(P(v))$: $Dia(P(v)) \leq s(v)$, and $\frac{s(v)}{Dia(P)}$ is upper-bounded by a polynomial function $\mathcal{P}_{HST}(n, d) \geq 1$ (called *distortion polynomial*); (3) Every leaf node corresponds to one point in P with size $s(v) = 0$, and each point appears in exactly one leaf node; (4) The two children v_1 and v_2 of any internal node v form a partition of v with $\max\{s(v_1), s(v_2)\} < s(v)$; and (5) For every aggregation node v with parent v_p, $\frac{s(v_p)}{r_{out}}$ is bounded by the distortion polynomial $\mathcal{P}_{HST}(n, d) \geq 1$, where r_{out} is the minimum distance between points in $P(v)$ and points in $P \setminus P(v)$.

The above definition is equivalent to the properties of HST in [14], except that we have an additional distortion requirement (Item 5). See Figure 2 for example of an aggregate tree.

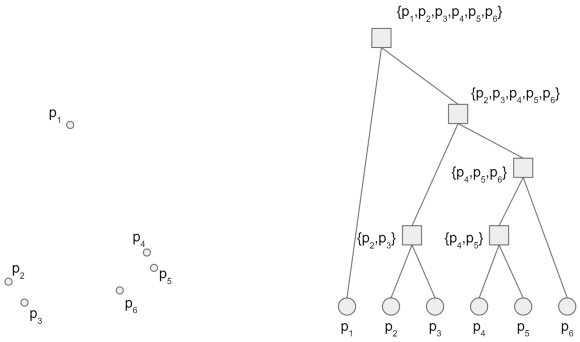


Figure 2 An illustration of an aggregation tree built for 6 points.

An aggregation tree can be constructed in $O(dn \log^2 n)$ time using the method in [14]. It is proved in [14] that the distortion polynomial is $\mathcal{P}_{HST}(n, d) = dn$. In the rest of the paper, we always assume that the distortion of an aggregation tree is $\mathcal{P}_{HST}(n, d) = dn$.

In the following, we briefly introduce range cover. Range cover is a technique proposed in [18] for solving the truth discovery problem in high dimensions. We utilize it in a completely different way to form a multi-scale construction for the AIFP query problem. Below is the algorithm (Algorithm 5). Given an aggregation tree T_p and real number parameters $\Delta \geq 8n$ and $0 < \lambda < 1$ (whose values will be determined later), the range cover algorithm creates a number of buckets for the nodes of T_P. Each bucket B_t is associated with an interval of real

Algorithm 5 RangeCover($T_P; \lambda, \Delta$).

Input: An aggregation tree T_P built over a set P of n points in $\mathbb{R}^d$; controlling factors $0 < \lambda < 1$ and an integer $\Delta \geq 4\mathcal{P}_{HST}(n, d)$.

Output: A number of buckets, where each bucket stores a number of tree nodes. Each bucket B_t is indexed by an integer t and associated with an interval $((1+\lambda)^t, (1+\lambda)^{t+1}]$.

1: For every integer t create an empty bucket B_t associated with interval $((1+\lambda)^t, (1+\lambda)^{t+1}]$. (Note that B_t will not be actually created until some tree node v is inserted into it.)

2: For every non-root node v of T_P, let v_p be its parent in T_P, r_H be $s(v_p)/\lambda$, and r_L be $\max\{s(v)/\lambda, s(v_p)/\Delta\}$. Do

- For every integer t satisfying inequality $r_L \leq (1+\lambda)^t < r_H$, insert v into bucket B_t.

number $((1+\lambda)^t, (1+\lambda)^{t+1}]$. If a value r lies in the interval of a bucket B_t, it can be shown that the diameter of every aggregation node v is small compared to r, and thus all points in $P(v)$ can be approximately viewed as one "heavy" point located at the representative point $re(v)$. Intuitively, every bucket from the range cover algorithm provides a view of P when observed from a distance r in the range of the bucket, where each node in the bucket represents a "heavy" point that is formed by the aggregation of a set of close (compared to the observing distance) clusters of points in P. Thus, the buckets of the range cover provides views of the input point set at different scales of observing distances (see Figure 3 for an illustration). The size of the output data structure is only $O(n \log n\Delta)$, as shown in [18].

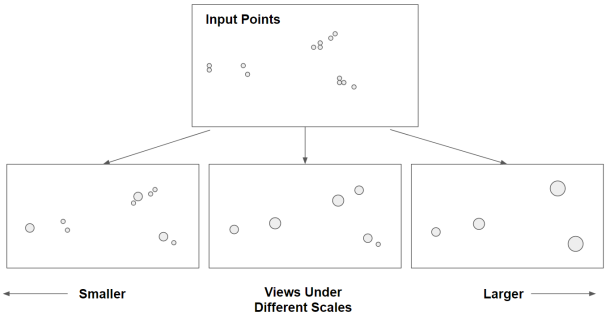


Figure 3 An illustration of range cover. The nodes in every bucket can be viewed as "heavy" points yielded by the aggregation of a set of close points. Every bucket provides a view of the input point set when observed from a certain distance. All the buckets jointly form a complete set of views of the input points at all possible scales.

Note that for many problems, fixing some key parameters also means fixing the "observing distance" of P from the perspective of solving the problem. This allows us to solve the problem based on the view of P provided by the bucket associated with the corresponding observation distance. We will show that this idea also applies to the AIFP problem.

3.2 Multi-scale Construction for AIFP

In this subsection, we use AIFP problem as an example to show how to implement multi-scale construction using the range cover data structure.

We first observe that every bucket of the range cover can be used to solve a constrained AIFP problem (with the proof given later). Given an AIFP query (B, q), if the (approximate) distance from q to a point in $P \cap B$ is known and falls in the interval of $((1+\lambda)^t, (1+\lambda)^{t+1}]$,

then the (approximate) distance from q to a point of $B_t \cap B$ (where every node v in B_t is viewed as a "heavy" point located at $re(v)$) is an AIFP of q in $P \cap B$. This means that B_t provides a good "sketch" of P that allows more efficient computation of the AIFP of q in $P \cap B$. This observation leads to the main idea of the multi-scale construction method. To obtain a general AIFP query data structure, for every bucket B_t, we construct a constrained AIFP data structure for B_t (viewed as a set of "heavy" points), exploiting the assumption that the farthest distance is in the interval of $((1+\lambda)^t, (1+\lambda)^{t+1}]$. To answer a general AIFP query, we can find the AIFP for every bucket by querying the constrained AIFP data structures associated with the bucket. In this way, we can compute AIFPs for all possible radii. When answering a general AIFP query, we first determine an approximate farthest distance of q to $P \cap B$, and then query the appropriate constrained AIFP data structures. Despite the necessity of building multiple constrained data structures, the complexity of the multi-scale construction is not high, as the total number of nodes in all buckets is only $\tilde{O}(n)$.

However, the above idea is hard to implement, because each bucket B_t is only responsible for a small range $((1+\lambda)^t, (1+\lambda)^{t+1}]$ of the possible farthest distance from q to $P \cap P$. This means that we need an accurate estimation of this distance when answering the query, which is almost as hard as the query itself. We resolve this issue by merging multiple consecutive buckets into a larger one. The resulting bucket can account for a larger range of the possible farthest distances. We then build a constrained data structure for each bucket.

This leads to the following Multi-Scale algorithm. Let $\Gamma \geq 1$ be an integer constant to be determined, and $\mathcal{A}$ be an algorithm for building a constrained data structure. In this algorithm, for each integer t, we try to merge the aggregation nodes in buckets $B_t, B_{t+1}, \ldots, B_{t+\Gamma}$ from the range cover (recall that these buckets are associated with farthest distance ranges $((1+\lambda)^t, (1+\lambda)^{t+1}], ((1+\lambda)^{t+1}, (1+\lambda)^{t+2}] \ldots, ((1+\lambda)^{t+\Gamma}, (1+\lambda)^{t+\Gamma+1}]$, respectively) into one bucket B_t^+ that could account for a larger range $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$. We then use $\mathcal{A}$ to build a data structure $\mathcal{S}_t$ for every bucket B_t^+ (by viewing every node in B_t^+ as a point).

Algorithm 6 Multi-Scale($T_P; \lambda, \Delta, \Gamma; \mathcal{A}$).

Input: An aggregation tree T_P built over a set P of n points in $\mathbb{R}^d$; controlling factors $0 < \lambda < 1$, integer $\Delta \geq 4\mathcal{P}_{HST}(n,d) = 4dn$, and integer $\Gamma \geq 1$. A routine $\mathcal{A}$ which builds a constrained data structure for any given bucket B_t and point set $re(B_t^+) := \{re(v) \mid v \in B_t^+\}$.
Output: A number of buckets, with each storing a number of tree nodes. Each bucket B_t^+ is indexed by an integer t and associated with an interval $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$. Each bucket B_t^+ is associated with data structure $\mathcal{S}_t$ built by $\mathcal{A}$.

1: Create a collection of buckets $\{B_t\}$ by calling RangeCover($T_P; \lambda, \Delta(1+\lambda)^\Gamma$).
2: For each integer t create an empty bucket B_t^+ associated with $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$.
3: For every non-root node v of T_P, enumerated in a bottom-up manner in T_P so that the children of a node is always visited earlier than the parent node, put v into B_t^+ for every t such that the following is satisfied:
   - $s(v) \leq \lambda(1+\lambda)^t$, v appears in $B_{t'} \in \{B_t\}$ for some $t \leq t' \leq t+\Gamma$, and none of v's descendants are put in B_t^+ previously.
4: For every non-empty bucket B_t^+, create a data structure $\mathcal{S}_t$ using $\mathcal{A}$ for the point set $re(B_t^+) := \{re(v) \mid v \in B_t^+\}$.

For better understanding of this scheme, we first briefly discuss the geometric properties of the buckets created by Algorithm 6. Intuitively speaking, the aggregation nodes of every bucket provide a sketch of *almost* the whole input point set P, with the exception being points that satisfying some special isolation property. This can be briefly described as follows:

(1) The diameter of each the aggregation node (viewed as a point set) should be small to the observation distances; (2) The aggregation nodes are mutually disjoint; and (3) Every point $p \in P$ is either in one of the nodes in the bucket, or it is in an aggregation node (not in the bucket) whose distance to other nodes is large. These properties are formalized as follows. (We leave the proofs of all the following claims and lemma to the full version of the paper.)

▷ Claim 13. Let v be any aggregation node v in a created bucket B_t^+. Then, $s(v) \leq \lambda(1+\lambda)^t$.

▶ **Lemma 14.** *For any $p \in P$ and and bucket B_t^+ created by Algorithm 6, one of the following holds:*

1. *There exists exactly one aggregation node $v \in B_t^+$ such that $p \in P(v)$.*
2. *Either B_t^+ is empty or there exists no aggregation node $v \in B_t^+$ such that $p \in P(v)$. There exists an aggregation node v' in T_P such that $s(v')/\lambda \leq (1+\lambda)^t$. Furthermore, let q be any point in $P \setminus P(v')$, then $\|p-q\| > (\Delta/dn)(1+\lambda)^{t+\Gamma}$.*

Although the sketch does not fully cover P, in many problems (including AIFP) these points are either negligible or easy to handle by other means due to their special properties.

From [18], we know that the running time of Algorithm 5 and the space complexity of the output data structure is $O_\lambda(n \log n\Delta)$ (where the hidden constant in the big-O notation depends only on λ). Algorithm 6 essentially merges $\Gamma + 1$ consecutive buckets B_t, B_{t+1}, …, $B_{t+\Gamma}$ created by Algorithm 5 into one bucket B_t^+. Thus, we have the following lemma.

▶ **Lemma 15.** *Excluding the time it takes for $\mathcal{A}$ to process each B_t^+ in Step 4, the running time of Algorithm 6 and the total number of nodes in all buckets is $O_\lambda(\Gamma^2 n \log n\Delta)$, where the hidden constant in big-O notation depends only on λ.*

We conclude this subsection by providing a key lemma showing that, given a constrained AIPF query (B, q) satisfying constraint (r_B, d_{min}, d_{max}) with $d_{min} \leq r_B \leq d_{max}$, if there is a bucket B_t^+ such that $(1+\lambda)^{t+1}/(1-\lambda) < d_{min} < d_{max} \leq (1+\lambda)^{t+\Gamma} - (2+2/\lambda)r_B$ (*i.e.* the range $[d_{min}, d_{max}]$ falls in interval $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$ with some gap), then, with an easy-to-handle exception, an AIFP to q in B_t^+ (by viewing every node of B_t^+ as one point) in (slightly enlarged) range B is also an AIFP of q to $P \cap B$. Formally, let $re(B_t^+) := \{re(v) \mid v \in B_t^+\}$. Let p_t be the farthest point to q in $re(B_t^+) \cap B(1+\lambda)$ if $re(B_t^+) \cap B(1+\lambda) \neq \emptyset$, and p be a $(\lambda/6, \lambda/6)$-AIFP of q in $re(B_t^+) \cap B(1+\lambda)$ [2]. Let p_N be a $(1+\lambda)$-approximate nearest neighbor of q in P.

▶ **Lemma 16.** *One of the following holds: (1) p_N is a $(2\lambda, 2\lambda)$-AIFP of q in $P \cap B$, or (2) p_t exists and $(1+\lambda)^t \leq \|q - p_t\| \leq (1+\lambda)^{t+\Gamma+1}$, and p is a $(2\lambda, 2\lambda)$-AIFP of q in $P \cap B$.*

The above lemma implies that, in Algorithm 6, if routine $\mathcal{A}$ builds a constrained AIFP data structure for farthest distance lies in interval $[(1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$, then either this data structure can be used to answer any query with constraint (r_B, d_{min}, d_{max}) (with other parameters, like r_B and the approximate factors for constrained AIFP, set properly), or the AIFP query can be solved easily using a nearest neighbor search. In the following section, we will show how to build a general AIFP query data structure through multi-scale construction by selecting appropriate parameters. With the multi-scale data structure (together with some auxiliary data structures), we can answer an AIFP query by (1) obtaining a rough estimation of the farthest distance, and (2) querying the bucket corresponding to the estimated range.

[2] Note p could be NULL here. This could happen when bucket B_t^+ is empty or $re(B_t^+) \cap B(1+\lambda) = \emptyset$.

4 General AIFP Query

In this section, we present a general (ϵ, γ)-AIFP query scheme. In the following, let B be a closed ball with radius $r_B > 0$ and q be an arbitrary point. Let $0 < \epsilon < 1$ and $0 < \gamma < 1$ be any pair of constants and $\lambda := \min(\epsilon, \gamma)/512$. We assume that r_B is λ-*aligned*, which means that $r_B = (1+\lambda)^t$ for some integer t. The alignment assumption makes it easier to implement the multi-scale construction. Note that if r_B is not λ-aligned, we can always enlarge B a little to make r_B λ-aligned and still obtain a good approximation with carefully chosen parameters.

Our main idea is to convert each query (B, q) into one or more AIFP queries (B', q) such that it is possible to find a lower bound d_{min} and an upper bound d_{max} on the farthest distance between q and a point in $B' \cap P$. With such bounds, the AIFP can then be found by querying a pre-built constrained AIFP data structure. To ensure efficiency, the gap between d_{min} and d_{max} cannot be too large, *i.e.,* d_{max}/d_{min} should be bounded by a polynomial of n and d. Since the complexity of a constrained data structure depends on d_{max}/d_{min}, a small gap will also enable us to control the size of the data structure. We start with a simple claim.

▷ Claim 17. If the distance between q and the center o_B of B is very large compared to r_B, *i.e.* $\|q - o_B\| \geq (3+\gamma)\epsilon^{-1} r_B$, then any point in $B(1+\gamma/2)$ is an (ϵ, γ)-AIFP of q in $P \cap B$.

This claim suggests that we can safely assume that the farthest distance between q and $P \cap B$ is not too large (compared to r_B), as otherwise the AIFP can be easily found with a nearest neighbor query. This helps us establish an upper bound on the farthest distance. In the following, we let $d_{max} := (4+2\gamma)\epsilon^{-1} r_B$, and assume that $\|q - o_B\| \leq (3+\gamma)\epsilon^{-1} r_B$. From simple calculation, this implies that for any $p \in P \cap B(1+\gamma)$, we have $\|p - q\| \leq d_{max}$.

Next, we try to find a lower bound for the farthest distance. Since the full argument will be rather complicated, we will thus describe only the general idea here, and leave the details to the full version of the paper. We start with a simple case.

Case 1: $\|q - o_B\| \geq (1+\gamma/64) r_B$, *i.e.* q does not lie in B and is not very close to the boundary of B. Then, clearly the farthest distance from q to any point in $P \cap B$ is at least $(\gamma/64) r_B$. Recall that we have obtained an upper bound $d_{max} := (4+2\gamma)\epsilon^{-1} r_B$. The ratio $d_{max}/(\gamma r_B/64)$ is clearly bounded by a polynomial of n, d.

In the following, we assume that $\|q - o_B\| \leq (1+\gamma/64) r_B$, which means that q lies in B or is very close to the boundary of B. Let $p_N \in P$ be a 2-nearest neighbor of q in P (*i.e.* for any $p' \in P$, $\|p_N - q\| \leq 2\|p' - q\|$) , and denote $r_N := \|p_N - q\|$. We discuss another simple case.

Case 2: $r_N > \gamma r_B/64$. Since p_N is a 2-approximate nearest neighbor, we conclude that the distance from q to any point in $P \cap B$ is at least $\gamma r_B/128$. Again, the ratio $d_{max}/(\gamma r_B/128)$ is well bounded.

In the following, we assume that $r_N \leq \gamma r_B/64$. In order to find a good lower bound on the farthest distance, our general strategy is to examine points around p_N and see whether there exists a point p' whose distance to q is sufficiently large, but still upper bounded by $(\gamma/64) r_B$. The upper bound $(\gamma/64) r_B$ ensures that $p' \in B(1+\gamma/16)$, *i.e.* p' "approximately" lies in B, and because of the fuzziness of the region, $\|p' - p\|$ can be used as a lower bound on the farthest distance. To efficiently implement this idea, we make use of an aggregation tree T_P with distortion polynomial $\mathcal{P}_{HST}(n, d) = nd$. Later, we will use this T_P to construct the query data structure. Let v_N denote the leaf node of T_P that correspond to singleton set $\{p_N\}$. We walk along the tree path from v_N to the root of T_P, and examine the nodes

(and their associated point sets) on this path. Denote by v the highest (closest to the root) node of T_P such that $p_N \in P(v)$ and $s(v) + r_N \leq \gamma r_B/64$. Note that since $r_N \leq \gamma r_B/64$ and $s(v_N) = 0$, such a node v must exist. Since $s(v)$ is an upper bound on the diameter of $P(v)$, every point $p' \in P(v)$ satisfies inequality $\|p' - q\| \leq \gamma r_B/64$, and thus $p' \in B(1+\gamma/16)$. See Figure 4 for a illustrations of the relations between q and points in $P(v)$. We consider two more cases, depending on the value of $s(v) + r_N$.

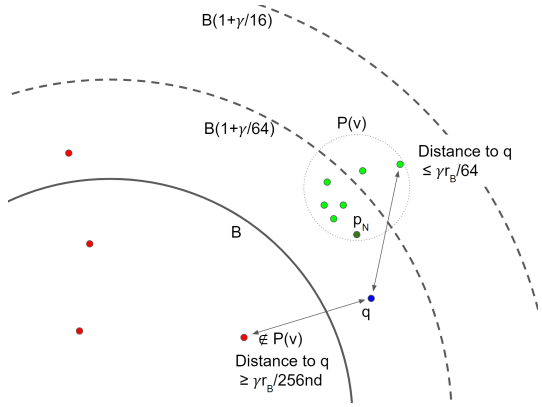


Figure 4 An illustration of points in $P(v)$, $P \setminus P(v)$ and q.

Case 3: $s(v) + r_N > \gamma r_B/512n^2d^2$. This means that $s(v) + r_N$ is at least r_B divided by a polynomial of n, d. Note that from the low distortion property of T_P, $s(v)$ is a polynomial approximation of the diameter of $P(v)$. It is possible to show by standard inequality arguments that, $s(v) + r_N$ is a polynomial approximation of q's farthest distance to any point in $P(v)$. Since $P(v) \subset B(1+\gamma/16)$, we conclude that the farthest distance from q to any point in $B(1+\gamma/16) \cap P$ is lower bounded by r_B divided by a polynomial of n, d. A careful calculation gives us an estimated lower bound $r_B/2048n^3d^3$. Since an upper bound for the farthest distance is $d_{max} = (4+2\gamma)\epsilon^{-1}r_B$, the quality of this lower bound is satisfactory because the gap between it and d_{max} is a polynomial of n, d.

Case 4: $s(v) + r_N \leq \gamma r_B/512n^2d^2$. We show that in this case, either there is a lower bound on the farthest distance, or the query can be reduced to another AIFP query where the range of farthest distances can be bounded. Let v_p be the parent of v (if v is the root of T_P, the following result still holds), then $s(v_p) + r_N \geq \gamma r_B/64$. This means that $s(v_p)$ is much larger than $s(v)$. From the property of T_P, we know that for any $p_{out} \in P \setminus P(v)$, the ratio $s(v_p)/\|p_{out} - p_N\|$ is upper bounded by the distortion polynomial $\mathcal{P}_{HST}(n, d) = nd$. In the current case, $s(v_p)/s(v)$ is indeed much larger than nd. Thus $\|p_{out} - p_N\|$ is very large compared to $s(v)$, which gives us a lower bound on $\|p_{out} - q\|$. In fact, by a careful calculation, it is possible to show that $\|q - p_{out}\| \geq \gamma r_B/256nd$ for any $p_{out} \in P \setminus P(v)$. Furthermore, since the distance between q and any point in $P(v)$ is at most $\gamma r_B/512n^2d^2$, which is much smaller than $\gamma r_B/256nd$. This implies that it is possible to use a ball centered at q to separate $P(v)$ and $P \setminus P(v)$. In fact, if let $r_{sep} := [s(v) + r_N]_\lambda$, where $[x]_\lambda$ denote the smallest real number that can be written as $(1+\lambda)^t$ for some integer t such that $(1+\lambda)^t \geq x$, then it can be shown that for every $p \in P(v)$, $p \in \mathcal{B}(q, r_{sep})$ and for every $p \in P \setminus P(v)$, $p \notin \mathcal{B}(q, (1+\gamma)r_{sep})$.

With the above argument, we divide Case 4 into 2 sub-cases. Let p_F be the actual farthest point to q in $P \cap B$. We first discuss **Case 4a**, where $p_F \in P \setminus P(v)$. From the above discussion, we have $\|q - p_F\| \geq \gamma r_B/256nd$, which gives a good lower bound on the

farthest distance. For **Case 4b** where $p_F \in P(v)$, we know that the query reduces to a problem of finding the farthest point in $p \cap \mathcal{B}(q, r_{sep})$. For this problem, it is possible to prove a $r_{sep}/8nd$ lower bound and r_{sep} upper bound on the farthest distance. The ratio of the two bounds is satisfactory.

To summarize, for Case 1,2 and 4a, let $d^{(1)}_{max} := (4+2\gamma)\epsilon^{-1}r_B$, $d^{(1)}_{min} := \gamma r_B/256nd$ and $r^{(1)}_B := r_B$, the AIFP query satisfies constraint $(r^{(1)}_B, d^{(1)}_{min}, d^{(1)}_{max})$; for Case 3, the AIFP query can be answered from constraint query $(\mathcal{B}(o_B, [(1+\gamma/16)r_B]_\lambda), q)$, which satisfies constraint $(r^{(2)}_B, d^{(2)}_{min}, d^{(2)}_{max})$, where $d^{(2)}_{min} := \gamma r_B/2048n^3d^3$, $d^{(2)}_{max} := (4+2\gamma)\epsilon^{-1}r_B$ and $r^{(2)}_B := [(1+\gamma/16)r_B]_\lambda$; for Case 4a, the AIFP query reduces to query $(\mathcal{B}(q, r_{sep}), q)$, with constraint $(r^{(3)}_B, d^{(3)}_{min}, d^{(3)}_{max})$ where $d^{(3)}_{max} := r^{(2)}_B := r_{sep}$, $d^{(3)}_{min} := d^{(3)}_{max}/8nd$. Any AIFP query can be answered from one of the three constraint queries, and we have $d^{(i)}_{max}/d^{(i)}_{min}$ no larger than $2048(4+2\gamma)n^3d^3/\epsilon$ for any $i = 1, 2, 3$.

4.1 Multi-scale Construction for General AIFP Query

In this subsection, we show how to build a Multi-Scale data structure to answer general AIFP queries. Our goal is to choose the appropriate parameters $\Gamma \geq 1$, $0 < \lambda < 1$ and $\Delta \geq 4nd$ for Algorithm 6 so that for every AIFP query with constraint $(r^{(i)}_B, d^{(i)}_{min}, d^{(i)}_{max})$, there exists a bucket B^+_t whose range $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$ wholly covers the interval $[d^{(i)}_{min}, d^{(i)}_{max}]$, and the constrained AIFP data structure built for the bucket with $d_{min} := (1+\lambda)^t$ and $d_{max} := (1+\lambda)^{t+\Gamma+1}$ can be used to answer the query. Note that we have already defined $\lambda := \min(\epsilon, \gamma)/512$ and $\Delta := 4nd$. The remaining task is to determine the value of Γ.

Observe that $d^{(i)}_{max}/d^{(i)}_{min}$ is bounded by a polynomial $\mathcal{P}_{gap}(n, d) := 2048(4+2\gamma)n^3d^3/\epsilon$. Let $\Gamma' := \lceil \log_{1+\lambda} \mathcal{P}_{gap}(n, d) \rceil$, and $\Gamma_L \geq \Gamma'$ and $\Gamma_R \geq \Gamma'$ be integer parameters to be determined later. Denote by Γ the sum of Γ_L and Γ_R, *i.e.,* $\Gamma := \Gamma_L + \Gamma_R$. For every integer t, define $r_{mid}(t) := (1+\lambda)^{t+\Gamma_L}$. Therefore, we have $r_{mid}(t)/(1+\lambda)^t \geq (1+\lambda)^{\Gamma_L} \geq \mathcal{P}_{gap}(n, d)$ and $(1+\lambda)^{t+\Gamma+1}/r_{mid}(t) \geq (1+\lambda)^{\Gamma_R} \geq \mathcal{P}_{gap}(n, d)$. For any AIFP query (B, q) with constraint $(r^{(i)}_B, d^{(i)}_{min}, d^{(i)}_{max})$, it is always possible to find a bucket B^+_t such that $r^{(i)}_B = r_{mid}(t)$. Clearly, interval $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$ wholly covers the interval $[d^{(i)}_{min}, d^{(i)}_{max}]$. If a constrained AIFP data structure is constructed for B^+_t with constraint $((1+\lambda)r_{mid}(t), (1+\lambda)^t, (1+\lambda)^{t+\Gamma+1})$, it can be used to answer the AIFP query (B, q). (See Lemma 16.)

To summarize the above discussions, we set the parameters of Algorithm 6 as the following. The algorithm then produces the data structure for (ϵ, γ)-AIFP query. Assume that a real number $0 < \delta < 1$ is given and we would like to achieve $1 - \delta$ query success probability.

- $\lambda := \min(\epsilon, \gamma)/512, \Gamma_L := \Gamma' + \lceil \log_{1+\lambda} 8 \rceil, \Gamma_R := \Gamma' + \lceil \log_{1+\lambda} 8 \rceil, \Gamma := \Gamma_L + \Gamma_R, \Delta := 4nd$.
- Routine $\mathcal{A}$: Given a non-empty bucket B^+_t, $\mathcal{A}$, it uses Algorithm 1 to creates a constrained $(\lambda/6, \lambda/6)$-AIFP data structure for point set $re(B^+_t)$ for constraint $((1+\lambda)r_{mid}(t), (1+\lambda)^t, (1+\lambda)^{t+\Gamma+1})$, with success probability at least $1 - \delta/4$.

Note that we let $\Gamma_L := \Gamma_R := \Gamma' + \lceil \log_{1+\lambda} 8 \rceil$. This allows more gap when fitting the interval $[d^{(i)}_{min}, d^{(i)}_{max}]$ in $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$, which is required by Lemma 16.

With the multi-scale data structure constructed by Algorithm 6 using the above parameters, we are able to answer any general AIFP query by reducing it to at most three constrained $(\lambda/6, \lambda/6)$-AIFP queries (where $\lambda = \min(\epsilon, \gamma)/512$) with constraints $(r^{(i)}_B, d^{(i)}_{min}, d^{(i)}_{max})$, $i = 1, 2, 3$. From Lemma 6 and the fact that the ratio d_{max}/d_{min} satisfies $d_{max}/d_{min} = (1+\lambda)^{t+\Gamma+1}/(1+\lambda)^t = (1+\lambda)^{\Gamma+1}$, which is bounded by a polynomial of n, d, we know that $\log d_{max}/d_{min}$ is $O_{\epsilon,\gamma}(\log nd)$ and the query time is $O_{\epsilon,\gamma}(dn^\rho \log \delta^{-1} \mathrm{Polylog}(nd))$

for some $\rho < 1$ depending on ϵ, γ. The multi-scale data structure consists of multiple AIFP structures built on buckets of points with total size $O_\lambda(\Gamma^2 n \log(n\Delta))$ (from Lemma 15). From Lemma 6, we know that the complexity of the data structure is $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \text{Polylog}(nd))$. We leave the detailed analysis to the full version of the paper.

5 MEB Range Aggregate Query

Given any query ball B, we find the AMEB of $P \cap B$ using an iterative algorithm by Badoiu and Clarkson [9]. Their algorithm was originally designed for finding an approximate MEB for a fixed point set P. With careful analysis we show that their approach, after some modifications, can still be used to find AMEB in any given range B. Briefly speaking, our idea is to construct a small-size coreset of $P \cap B$. The MEB of the coreset is then a (ϵ, γ)-approximate MEB of $P \cap B$. The algorithm selects the coreset in an iterative fashion. It starts with an arbitrary point p from $P \cap B$. At each iteration, it performs the following operation to add a point to the coreset: (1) Compute an (approximate) MEB of the current coreset (directly using the algorithm in [9]); (2) Identify the AIFP in $P \cap B$ to the center of the current MEB, and add it to the coreset. We can show that after $O_{\epsilon,\gamma}(\log n)$ iterations, the MEB of the coreset is then a (ϵ, γ)-AMEB of $P \cap B$.

To answer an (ϵ, γ)-AMEB query with success probability at least $1 - \delta$, we will need only one (ϵ', γ')-AIFP data structure with success probability $1 - \delta / O_{\epsilon'}(\log n)$ (where ϵ', γ' depends on ϵ, γ polynomially), whose size is $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \text{Polylog}(nd))$ for some $\rho < 1$ depending on ϵ, γ. Every AMEB query is reduced to $O_{\epsilon'}(\log n)$ AIFP queries (the time for computing the MEB every iteration is negligible compared to AIFP queries). Thus, the query time is $O_{\epsilon,\gamma}(dn^{\rho} \log \delta^{-1} \text{Polylog}(nd))$. The detailed analysis is left to the full version of the paper.

References

1. Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, Sepideh Mahabadi, and Kasturi R Varadarajan. Diverse near neighbor problem. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 207–214, 2013.
2. Mikkel Abrahamsen, Mark de Berg, Kevin Buchin, Mehran Mehr, and Ali D Mehrabi. Range-clustering queries. *arXiv preprint*, 2017. `arXiv:1705.06242`.
3. Pankaj K Agarwal, Lars Arge, Sathish Govindarajan, Jun Yang, and Ke Yi. Efficient external memory structures for range-aggregate queries. *Computational Geometry*, 46(3):358–370, 2013.
4. Pankaj K Agarwal, Jeff Erickson, et al. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.
5. Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1018–1028. SIAM, 2014.
6. Sunil Arya, David M Mount, and Eunhui Park. Approximate geometric mst range queries. In *31st International Symposium on Computational Geometry (SoCG 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
7. Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Francesco Silvestri. Distance-sensitive hashing. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 89–104, 2018.
8. Martin Aumüller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. Sampling a near neighbor in high dimensions–who is the fairest of them all? *arXiv preprint*, 2021. `arXiv:2101.10905`.
9. Mihai Badoiu and Kenneth L Clarkson. Smaller core-sets for balls. In *SODA*, volume 3, pages 801–802, 2003.

10 Peter Brass, Christian Knauer, Chan-Su Shin, Michiel Smid, and Ivo Vigan. Range-aggregate queries for geometric extent problems. In *Proceedings of the Nineteenth Computing: The Australasian Theory Symposium-Volume 141*, pages 3–10, 2013.

11 Ryan R Curtin and Andrew B Gardner. Fast approximate furthest neighbors with data-dependent candidate selection. In *International Conference on Similarity Search and Applications*, pages 221–235. Springer, 2016.

12 Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

13 Prosenjit Gupta, Ravi Janardan, Yokesh Kumar, and Michiel Smid. Data structures for range-aggregate extent queries. *Computational Geometry*, 47(2):329–347, 2014.

14 Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

15 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

16 Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in olap data cubes. *ACM SIGMOD Record*, 26(2):73–88, 1997.

17 Qiang Huang, Jianlin Feng, and Qiong Fang. Reverse query-aware locality-sensitive hashing for high-dimensional furthest neighbor search. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 167–170. IEEE, 2017.

18 Ziyun Huang, Hu Ding, and Jinhui Xu. A faster algorithm for truth discovery via range cover. *Algorithmica*, 81(10):4118–4133, 2019.

19 Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 539–545, 2003.

20 Sankalp Khare, Jatin Agarwal, Nadeem Moidu, and Kannan Srinathan. Improved bounds for smallest enclosing disk range queries. In *CCCG*. Citeseer, 2014.

21 Zhe Li, Tsz Nam Chan, Man Lung Yiu, and Christian S Jensen. Polyfit: Polynomial-based indexing approach for fast approximate range aggregate queries. *arXiv preprint*, 2020. `arXiv:2003.08031`.

22 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

23 Yakov Nekrich and Michiel HM Smid. Approximating range-aggregate queries using coresets. In *CCCG*, pages 253–256, 2010.

24 Rasmus Pagh, Francesco Silvestri, Johan Sivertsen, and Matthew Skala. Approximate furthest neighbor in high dimensions. In *International Conference on Similarity Search and Applications*, pages 3–14. Springer, 2015.

25 Saladi Rahul, Haritha Bellam, Prosenjit Gupta, and Krishnan Rajan. Range aggregate structures for colored geometric objects. In *CCCG*, pages 249–252, 2010.

26 Saladi Rahul, Ananda Swarup Das, KS Rajan, and Kannan Srinathan. Range-aggregate queries involving geometric aggregation operations. In *International Workshop on Algorithms and Computation*, pages 122–133. Springer, 2011.

27 Yufei Tao, Xiaokui Xiao, and Reynold Cheng. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems (TODS)*, 32(3):15–es, 2007.

28 Jeffrey Scott Vitter and Min Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. *Acm Sigmod Record*, 28(2):193–204, 1999.

29 Eugene Wu and Samuel Madden. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment*, 6(8), 2013.

30 Jie Xue. Colored range closest-pair problem under general distance functions. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 373–390. SIAM, 2019.

31 Jie Xue, Yuan Li, and Ravi Janardan. Approximate range closest-pair queries. *Computational Geometry*, 90:101654, 2020.

32 Xiaochun Yun, Guangjun Wu, Guangyan Zhang, Keqin Li, and Shupeng Wang. Fastraq: A fast approach to range-aggregate queries in big data environments. *IEEE Transactions on Cloud Computing*, 3(2):206–218, 2014.

Strong Approximations and Irrationality in Financial Networks with Derivatives

Stavros D. Ioannidis
Department of Informatics, King's College London, UK

Bart de Keijzer
Department of Informatics, King's College London, UK

Carmine Ventre
Department of Informatics, King's College London, UK

Abstract

Financial networks model a set of financial institutions (firms) interconnected by obligations. Recent work has introduced to this model a class of obligations called credit default swaps, a certain kind of financial derivatives. The main computational challenge for such systems is known as the clearing problem, which is to determine which firms are in default and to compute their exposure to systemic risk, technically known as their recovery rates. It is known that the recovery rates form the set of fixed points of a simple function, and that these fixed points can be irrational. Furthermore, Schuldenzucker et al. (2016) have shown that finding a weakly (or "almost") approximate (rational) fixed point is PPAD-complete.

We further study the clearing problem from the point of view of irrationality and approximation strength. Firstly, we observe that weakly approximate solutions may misrepresent the actual financial state of an institution. On this basis, we study the complexity of finding a strongly (or "near") approximate solution, and show FIXP-completeness. We then study the structural properties required for irrationality, and we give necessary conditions for irrational solutions to emerge: The presence of certain types of cycles in a financial network forces the recovery rates to take the form of roots of non-linear polynomials. In the absence of a large subclass of such cycles, we study the complexity of finding an exact fixed point, which we show to be a problem close to, albeit outside of, PPAD.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases FIXP, Financial Networks, Systemic Risk

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.76

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2109.06608 [18]

Funding *Carmine Ventre*: Partially supported by the UKRI Trustworthy Autonomous Systems Hub (EP/V00784X/1).

1 Introduction

The International Monetary Fund says that the global financial crisis (GFC) of 2007 has had long lasting consequences, including loss of growth, large public debt and even a decline of fertility rates, see [2]. Consequently, the need to assess the *systemic risk* of the financial network cannot be overstated. For example, if banks at risk of defaults could be easily identified in the complex network of financial obligations, then spread could be preemptively avoided with appropriate countermeasures such as bailouts from central banks or regulators.

In this context, the *clearing problem* introduced in [7] plays a central role. We are given a so-called financial network, that is, a graph where vertices are banks (or, more generally, financial institutions) and weighted arcs (u, v) model direct liabilities from bank u to bank v. Each bank has also some assets external to the network, that can be used to pay its liabilities.

The question is to compute a *clearing recovery rate vector*, that is, the ratio between money available (coming from assets and payments from others) over liabilities for each bank. If this ratio is bigger than 1 for a bank, then it will be able pay its dues – in this case, we simply set its rate to 1. The banks that are in default have recovery rates smaller than 1. The problem of computing clearing recovery rates (which we will also refer to as the *clearing problem*) is well understood when there are only simple debt contracts in the network, then clearing recovery rate vectors always exist, are unique, and can be computed in polynomial time [7].

However, Eisenberg and Noe's model in [7] ignores the issue of financial derivatives that may be present in the system. The deregulation allowing banks to invest in these products is considered by many as one of the triggers of the GFC. The introduction of financial derivatives to financial networks is due to [22], where the focus is on a simple and yet widely used class of conditional obligations known as Credit Default Swaps (CDSes), the idea being to "swap" or offset a bank's credit risk with that of another institution. More specifically, a CDS has three entities: a creditor v, a debtor u and a reference bank z – u agrees to pay v a certain amount whenever z defaults. Whilst CDSes were conceived in the early 1990s as a way to protect v from the insolvency of z for direct liabilities (i.e., a (v, z)-arc in the network), they quickly became a speculative tool to bet against the creditworthiness of the reference entity and have in fact been widely used both as a hedging strategy against the infamous collateralised debt obligations, whose collapse contributed to the GFC, and pure speculation during the subsequent eurozone crisis. The clearing problem in the presence of these financial derivatives is somewhat less well-characterised: it is known that the clearing recovery rate vectors can be expressed as the fixed points of a certain function, and existence of solutions is then guaranteed via a fixed-point argument [22]. On the other hand, these fixed points can be irrational, and the computational problem is PPAD-complete [23] as long as one is interested in only a *weak* approximation of a recovery rate vector.

1.1 Our Contributions

In this paper we deepen the study of the clearing problem for financial networks with CDSes from two complementary viewpoints. Firstly, we argue that weak approximations can be misleading in this domain, as the objective under the weak approximation criterion is to find an "almost" fixed point (i.e., a point which is not too far removed from its image under the function). The risk estimate provided by this concept might be very far off the actual rate, thus changing the amount of bailout needed or even whether a bank needs rescue in the first place (see, e.g., our example in Figure 1b below). A more useful (but more difficult) objective is to obtain a strong approximation, that is, a point that is geometrically close to an actual fixed point of the function. Such a risk estimate would be actionable for a regulator, as the error could be measured in terms of irrelevant decimal places. Furthermore, the banks themselves would accept the rate when the strong approximation guarantee is negligible, whereas a weak approximation could significantly misrepresent their income and are subject to be challenged, legally or otherwise.

As our first contribution, we settle the computational complexity of computing strong approximations to the clearing problem in terms of FIXP [9], by showing that the clearing problem is complete for this class. In our reduction, we provide a series of financial network gadgets that are able to compute opportune arithmetic operations over recovery rates. Interestingly, not that many FIXP-complete problems are known, although there are a few important natural such problems (three-or-more-player Nash equilibria being a notable example). Hardness reductions for this class tend to be rather technically involved. The hardness reduction that we provide here indeed has some technical obstacles as well, although

our reduction is quite natural at a high level, and could inspire further developments in the area. Our result complements the current state of the art and completes the picture about the computational complexity of the clearing problem with financial derivatives. It shows that computing strongly approximate fixed points is harder than computing weakly approximate fixed points, which holds due to PPAD being equal to the class Linear-FIXP, which is a restriction of FIXP, and this makes PPAD (indirectly) a subclass of FIXP.

▶ **Main Theorem 1** (Informal). *Computing a strong approximation to the clearing recovery rates in a financial network with CDSes is FIXP-complete.*

The FIXP-hardness of the strong approximation problem indicates that there is an additional numerical aspect contributing to the hardness of the problem, which is not present in the weak approximation problem (where the hardness is of a combinatorial nature, due to the reducibility to the end-of-the-line problem which is canonical to PPAD). For the strong approximation problem, the nature of the underlying function for which we want to find the fixed points requires, in particular, the multiplication operation, which ultimately accounts for irrationality and super-polynomial numerical precision being necessary in order to derive whether a given point is a strong approximation to a clearing vector.

We then turn our attention towards irrational solutions with the goal to determine the source of irrationality and understand when it is possible to compute the clearing recovery rate vector exactly in the form of rational numbers. We identify a structural property of cycles in an opportunely enriched network that leads to unique irrational solutions. This property exactly differentiates the CDSes that produce and propagate irrationality of the recovery rates, that we call "switched on", from those that do not, termed "switched off". We prove the following close-to-tight characterisation of irrationality:

▶ **Main Theorem 2** (Informal). *If the financial network has only "switched on" CDSes in a cycle* and *the cycle cannot be shortcut with paths of length at most three then there exist rational values for debt and asset values for which the recovery rate vector is unique and irrational. Conversely, if every cycle of the financial network does not have any "switched on" CDSes then we can compute rational recovery rates in a polynomial number of operations, provided that we have oracle access to PPAD.*

Our proof of irrationality uses a graph "algebra" (i.e., a set of network fragments and an operation on them) that is able to generate all the possible cycle structures with the property above, which uncovers a connection between the network structure of the clearing problem and the roots of non-linear equations. For the opposite direction, we provide an algorithm that exploits the acyclic structure of financial networks with solely "switched off" CDSes. This algorithm iteratively computes the recovery rates of each strongly connected component of the network. We show that even for the simpler topologies of the financial system under consideration, the problem remains PPAD-hard, hence the need for the oracle access to PPAD.

Significance of Our Results. We see our results as important analytical tools that legislators can use to regulate financial derivatives. For example, our results contribute to the ongoing debate in the US and Europe about banning speculative uses of CDSes. In particular, they support, from a computational point of view, the call to ban so-called "naked" CDSes (as already done by the EU for sovereign debt in the wake of the Eurozone crisis, see [1]). A naked CDS is purely speculative since its creditor and debtor have no direct liabilities with the reference entity. It turns out that these CDSes add arcs between potentially unconnected nodes, thus possibly adding more of the cycles that lead to irrationality and, given that strong

approximations are out of scope due to our FIXP-hardness, it is not only combinatorially but also numerically intractable to gain insight in the systemic risk of such financial networks. A mechanism to monitor the topology of a financial network might be useful to avoid the construction of cyclical structures that include CDSes.

Both of our main results introduce significant novel technical and conceptual innovations to the field. As mentioned above, our reduction for the FIXP-hardness is somewhat more direct than in previous work we are aware of. Our reduction is direct, in the sense that it starts from the algebraic circuit defined by an arbitrary problem in FIXP. The reduction employs two main steps: We firstly force the outputs of all gates in the circuit to be in the unit cube, by essentially borrowing arguments from [9], after which we produce a series of network gadgets that preserve gate-wise the computations of said circuit; this makes the reduction conceptually straightforward in its setup.

It is worth highlighting a specific technical challenge that we overcome in our proof, as we think it sheds further light on FIXP, and in particular, on the operator basis of the algebraic circuits that are used to define the class. It is known that the circuit of problems in FIXP can be restricted without loss of generality on the arithmetic basis $\{\max, +, *\}$ [9], whereas restricting the internal signals of the circuit to the unit cube (with the toolkit developed in [9]) needs some further operators, including /. For our optimisation problem to be in FIXP, we need the rather mild and realistic assumption that our instances are *non-degenerate* as defined in [23]. The function of which the fixed points define the recovery rates of non-degenerate instances is well defined, where the non-degeneracy is needed to avoid a division by 0. It turns out that non-degeneracy is incompatible with division being part of the FIXP operator basis, i.e., it seems difficult to build such a financial network that in any sense simulates a division of two signals in an algebraic circuit. To bypass this problem, our proof shows that it is possible to substitute / in the basis with the square root operator, $\sqrt{\cdot}$, whilst keeping the function well defined. This substitution can be used to simulate division with constant large powers of 2, and this turns out to be sufficient to omit the /-operator (i.e., arbitrary division). This novel observation might be useful for other problems where division is problematic to either define the fixed point function, or the reduction.

Our second result indirectly aims at characterising the "rational fragment" of FIXP: To the best of our knowledge this is the first study in this direction. A couple of observations can be drawn from our attempt. Firstly, our sufficiency conditions for irrationality suggest that any such characterisation needs to fully capture the connection between the fixed point condition and the rational root theorem; our proof currently exploits the cyclical structure of networks with "switched on" CDSes to define one particular quadratic equation with irrational roots. Whilst this captures a large class of instances, more work is needed to give a complete characterisation (see Section 6 for a discussion). Secondly, our sufficiency conditions for rational solutions highlight a potential issue with their representation. Due to the operations in the arithmetic basis, most notably multiplication, these solutions can grow exponentially large (even though each call to the PPAD oracle returns solutions of size polynomial in their input). This observation establishes a novel connection between the Blum-Shub-Smale computational model [5] (wherein the size to store any real number is assumed to be unitary and standard arithmetic operations are executed in one time unit), the rational part of FIXP, and PPAD. Our result paves way to further research on the subject.

Further Related Work. Systemic risk in financial networks has been studied extensively in the literature [3, 8, 13, 15, 16, 17, 6, 21, 19]. Game-theoretic perspectives of financial networks are studied in [4, 20]. Fixed point computations of total search problems, are studied in [9]. FIXP-complete problems are presented in [14, 10, 12, 11].

2 Model and Preliminaries

2.1 Financial Systems

Let $N = \{1, \ldots, n\}$ be a set of n banks. Each bank $i \in N$ has *external assets*, denoted by $e_i \in \mathbb{Q}_{\geq 0}$ and $e = (e_1, \ldots, e_n)$ is the external assets vector. We consider two types of liabilities among banks: *debt contracts* and *credit default swaps (CDSes)*. A debt contract requires one bank i (debtor) to pay another bank j (creditor) a certain amount $c_{i,j} \in \mathbb{Q}_{\geq 0}$. We denote by $\mathcal{DC}$ the set of all pairs of banks participating in a debt contract. A CDS requires a debtor i to pay a creditor j on condition that a third bank called the *reference bank* R is in default, meaning that R cannot fully pay its liabilities. Formally, we associate each bank i a variable $r_i \in [0, 1]$, called the *recovery rate*, that indicates the proportion of liabilities it can pay. Having $r_i = 1$, means bank i can fully pay its liabilities, while $r_i < 1$ indicates that i is in default. In case a reference bank R of a CDS is in default, the debtor i of that CDS pays the creditor j an amount of $(1 - r_R)c^R_{i,j}$, where $c^R_{i,j} \in \mathbb{Q}_{\geq 0}$ is the face value of the CDS. We denote by $\mathcal{CDS}$ the set of all triplets participating in a credit default swap. The value $c_{i,j}$ ($c^R_{i,j}$, resp.) of a debt contract (CDS, resp.) is referred to as the *notional* of the contract. Finally we let c be a three-dimensional $(n \times n \times n)$ matrix containing all contract notionals; we do not allow any bank to have a debt contract with itself, and assume that all three banks in any CDS are distinct.

▶ **Definition 1.** *A financial system is a triplet (N, e, c), where $N = \{1, .., n\}$ is a set of banks, $e = (e_1, .., e_n) \in \mathbb{Q}^n_{\geq 0}$ is the vector of external assets, and $c \in \mathbb{Q}^{n \times n \times n}_{\geq 0}$ is the three-dimensional matrix of all contract notionals.*

The *contract graph* of $I = (N, e, c)$ is defined as a directed multigraph $G_I = (V, A)$, where $V = N$ and $A = (\cup_{k \in N} A_k) \cup A_0$ where $A_0 = \{(i, j) \mid c_{i,j} \neq 0\}$ and $A_k = \{(i, j) \mid c^k_{i,j} \neq 0\}$. Each arc $(i, j) \in A_0$ is coloured blue and each $(i, j) \in A_k$ orange. For all $(i, j, R) \in \mathcal{CDS}$ we draw a dotted orange line from node R to arc $(i, j) \in A_R$, denoting that R is the reference bank of the CDS between i and j. Finally, we label each arc with the notional of the corresponding contract, and each node with the external assets of the corresponding bank.

Given a recovery rate vector $r \in [0, 1]^n$, we define the *liabilities*, *payments*, and *assets* in a financial system as follows. The liability of a bank $i \in N$ to a bank $j \in N$ under r is denoted by $l_{i,j}(r) = c_{i,j} + \sum_{k \in N}(1 - r_k)c^k_{i,j}$. That is, we sum up the liabilities from the debt contract and all CDS contracts between i and j. We denote by $l_i(r)$ the total liabilities of i: $l_i(r) = \sum_{j \in N} l_{i,j}(r)$. The payment bank i makes to bank j under r is denoted by $p_{i,j}(r)$, where $p_{i,j}(r) = r_i \cdot l_{i,j}(r)$. The assets of a bank i under r are the total amount it possesses through its external assets and incoming payments made all by other banks, i.e., $a_i(r) = e_i + \sum_{j \in N} p_{j,i}(r)$. Our research focuses on *clearing recovery rate vectors* (CRRVs).

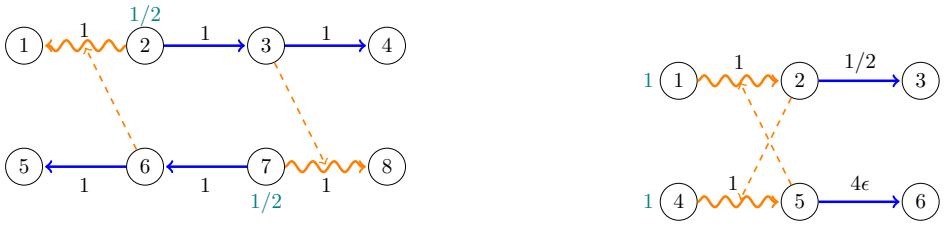
▶ **Definition 2.** *Given a financial system (N, e, c), a recovery rate vector r is called clearing if and only if for all banks $i \in N$, $r_i = \min\{1, a_i(r)/l_i(r)\}$, if $l_i(r) > 0$, and $r_i = 1$ if $l_i(r) = 0$.*

We set to 1 the recovery rate of nodes without liabilities, whereas [22] leaves these unconstrained. This is in line with the interpretation that these banks are not in default and only a cosmetic difference, as discussed in the full version of the paper [18].

We call CDS-CLEARING the problem of computing a CRRV in a given financial system with debt contracts and credit default swaps. For an instance $I \in$ CDS-CLEARING any clearing vector is a *solution* and the solution set is denoted by $\mathrm{Sol}(I)$. Let $I \in$ CDS-CLEARING and consider $f_I : [0, 1]^n \mapsto [0, 1]^n$ defined at each coordinate $i \in [n]$ by $f_I(r)_i = a_i(r)/(\max\{l_i(r), a_i(r)\})$. It is easy to see that $Sol(I)$ actually consists of the fixed point of function f_I. The existence of

at least one fixed point of f_I for every $I \in$ CDS-CLEARING was proved in [22]. Unfortunately, there exist instances of CDS-CLEARING in which all clearing vectors have irrational values, one example given in [23]. We present another example, to also illustrate our contract graphs.

▶ **Example.** *Figure 1a consists of eight banks,* $N = \{1, \ldots, 8\}$. *They all have external assets* 0 *except for banks* 2 *and* 7 ($e_j = 0$ *for* $j \neq 2, 7$, $e_2 = e_7 = 1/2$). *The set of debt contracts is* $\mathcal{DC} = \{(2,3), (3,4), (6,5), (7,6)\}$ *and the set of CDSes is* $\mathcal{CDS} = \{(2,1,6), (7,8,3)\}$. *All contract notionals are set to 1. For a recovery rate vector* r, *node 2's liability is* $l_2(r) =$



(a) CRRVs can be irrational. **(b)** Approximations of CRRVs.

Figure 1 On the left a financial system (a) with irrational solutions. On the right a financial system (b) where the weak approximate fixed point is far from the actual fixed point.

$l_{2,3}(r) + l_{2,1}(r) = c_{2,3} + (1 - r_6)c^6_{2,1} = 2 - r_6$. *For node 3, it holds* $l_3(r) = l_{3,4}(r) = c_{3,4} = 1$. *The assets of node 2 are* $a_2(r) = e_2 = 1/2$ *whereas* $a_3(r) = e_3 + p_{2,3}(r) = r_2 c_{2,3} = r_2$. *Symmetrically,* $l_7(r) = 2 - r_3, a_7(r) = 1/2$ *and* $l_6(r) = 1, a_6(r) = r_7$. *For node 1 it holds that* $a_1(r) = e_1 + p_{2,1}(r) = r_2(1 - r_6)c^6_{2,1}$ *and for node 8* $a_8(r) = r_7(1 - r_3)c^3_{7,8}$. *From the above computations and by applying the CRRV condition we get that any solution must satisfy:* $r_2 = \min\{1, 1/(2(2 - r_6))\}$, $r_6 = r_7$, $r_7 = \min\{1, 1/(2(2 - r_3))\}$, *and* $r_3 = r_2$, *implying that* $2r_2^2 - 4r_2 + 1 = 0$ *and then* $r_2 = r_3 = r_6 = r_7 = 1 - \sqrt{2}/2$ *and* $r_1 = r_5 = r_4 = r_8 = 1$.

2.2 Approximation and Complexity

Let F be a continuous function that maps a compact convex set to itself and let $\epsilon > 0$ be a small constant. A *weak ϵ-approximate fixed point* of F is a point x such that $\|x - F(x)\|_\infty < \epsilon$. A *strong ϵ-approximate fixed point* of F is a point x s.t $\exists x' : F(x') = x' \wedge \|x' - x\|_\infty < \epsilon$. Moreover, under a mild condition on the fixed point problem under consideration, known as *polynomial continuity*, a strong approximation is also a weak approximation [9], which explains the use of the terms "strong" and "weak".

Formally a *fixed point problem* Π is defined as a search problem such that for every instance $I \in \Pi$ there is an associated continuous function $F_I : D_I \to D_I$ where $D_I \subseteq \mathbb{R}^n$ (for some $n \in \mathbb{N}$) is compact and convex, such that the solutions of I are the fixed points of F_I. The problem Π is said to be *polynomially computable* if there is a polynomial q such that (i.) D_I is a convex polytope described by a set of at most $q(|I|)$ linear inequalities, each with coefficients of a size at most $q(|I|)$, and (ii.) For each x in $D_I \cap \mathbb{Q}^n$, the value $F_I(x)$ can be computed in time $q(|I| + \text{size}(x))$. Here, the "size" of a rational number means the number of bits needed to represent the numerator and the denominator in binary. Furthermore Π is said to be *polynomially continuous* if there is a polynomial q such that for each $I \in \Pi$, and rational $\epsilon > 0$, there is a rational δ of size $q(|I| + \text{size}(\epsilon))$ satisfying the following: for all $x, y \in D_I$ with $\|x - y\|_\infty < \delta$ it holds that $\|F_I(x) - F_I(y)\|_\infty < \epsilon$.

With regard to CDS-CLEARING, it is straightforward to verify that CDS-CLEARING is polynomially computable. Furthermore, [23] establishes implicitly that CDS-CLEARING is polynomially continuous under a (very) mild assumption that the authors call *non-degeneracy*.

▶ **Definition 3.** *A financial system is* non-degenerate *if and only if the following two conditions hold. Every debtor in a CDS either has positive external assets or is the debtor in at least one debt contract with a positive notional. Every bank that acts as a reference bank in some CDS is the debtor of at least one debt contract with a positive notional.*

We define CDS-CLEARING to contain only non-degenerate financial networks, both for the sake of compatibility with [23] and for the analytical convenience that non-degeneracy provides us (note that a division by 0 never occurs in $f_I(r)_i$ for these instances). In [23], it is also shown that the weak approximation version of CDS-CLEARING is PPAD-hard. The polynomial continuity of CDS-CLEARING shows that the strong approximation version of CDS-CLEARING is at least as hard as its weak approximation version. As noted above, weakly approximate fixed points may contain misleading information about whether a bank is in default or not, as shown in the next example. This motivates the study of strong approximations.

▶ **Example.** *Consider the instance in Figure 1b. It is not hard to see that* $r = (1, 1, 1, 1, 0, 1)$ *is an exact fixed point;* $r' = (1, 1 - 2\epsilon, 1, 1, 1/2 + \epsilon, 1)$ *is instead a weakly* ϵ*-approximate fixed point since* $f_2(r') = 1 - 2\epsilon$ *and* $f_5(r') = 1/2$ *implying that* $\|r' - f(r')\|_\infty \leq \epsilon$*. We observe that* r *is very far from* r' *and, in particular, as* $r'_2 = 1 - 2\epsilon < 1$*, we would conclude that 2 is in default whereas 2 can actually fully pay its liabilities since* $r_2 = 1$*.*

FIXP is the complexity class introduced to study the strong approximation and exact versions of fixed point problems [9].

▶ **Definition 4.** *The class FIXP consists of all fixed point problems* Π *that are polynomially computable, and for which for all* $I \in \Pi$ *the function* $F_I : D_I \to D_I$ *can be represented by an algebraic circuit* C_I *over the basis* $\{+, -, *, \max, \min\}$*, using rational constants, such that* C_I *computes* F_I*, and* C_I *can be constructed from* I *in time polynomial in* $|I|$*.*

The class FIXP_a *is defined as the class of search problems that are the strong approximation version of some fixed point problem that belongs to FIXP.*

The class Linear-FIXP is defined analogously to FIXP, but under the smaller arithmetic basis where only the gates $\{+, -\max, \min\}$ *and multiplication by rational constants are used.*

The classes FIXP, Linear-FIXP, and FIXP_a admit complete problems. Hardness of a search problem Π for FIXP (resp. Linear-FIXP and FIXP_a) is defined through the existence of a polynomial time computable function $\rho : \Pi' \to \Pi$, for all $\Pi \in$ FIXP (resp. FIXP_a), such that the solutions of I can be obtained from the solutions of $\rho(I)$ by applying a (polynomial-time computable) linear transformation on a subset of $\rho(I)$'s coordinates. This type of reduction is known as a *polynomial time SL-reduction*.

It is known that $\mathsf{FIXP}_a \subseteq$ PSPACE and Linear-FIXP = PPAD [9]. Consequently, the solutions of instances in Linear-FIXP are always rationals of polynomial size. An informal understanding of how the hardness of FIXP compares to PPAD (or Linear-FIXP) is as follows. PPAD captures a type of computational hardness stemming from an essentially combinatorial source. The class FIXP introduces on top of that a type of numerical hardness that emerges from the introduction of multiplication and division operations: These operations give rise to irrationality in the exact solutions to these problems, and may independently also require the computation of rational numbers of very high precision or very high magnitude.

3 FIXP-Completeness of CDS-Clearing

Our first main result shows that CDS-CLEARING and its strong approximation variant are $\mathsf{FIXP}_{(a)}$ complete. We show that we can take an arbitrary algebraic circuit and encode it in a direct way in the form of a financial system. Hence, our polynomial time hardness

reduction is implicitly defined to work from to any arbitrary fixed point problem in FIXP. The reduction is constructed by devising various financial network gadgets which enforce that certain banks in the system have recovery rates that are the result of applying one of the operators in FIXP's arithmetic base to the recovery rates of two other banks in the system: In other words, we can design our financial systems such that the interrelation between the recovery rates mimics a computation through an arbitrary algebraic circuit.

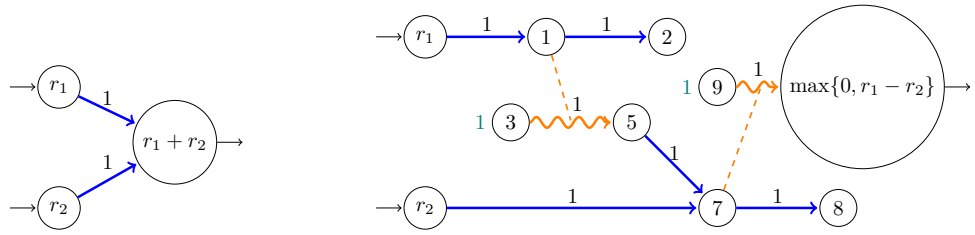
▶ **Theorem 5.** *CDS-CLEARING is FIXP-complete, and its strong approximation version is FIXP_a-complete.*

Proof Sketch. The clearing vectors for an instance $I \in$ CDS-CLEARING are the fixed points of the function f_I defined above, which can be computed using a polynomial size algebraic circuit with only $\{\max, +, *\}$, and rational constants. Note that non-degeneracy of I prevents division by 0, so that the output of the circuit is well-defined for every $x \in [0,1]^n$. This shows that CDS-CLEARING is in FIXP and that its strong approximation version is in FIXP_a.

For the FIXP-hardness of the problem, let Π be an arbitrary problem in FIXP. We describe a polynomial-time reduction from Π to CDS-CLEARING. Let $I \in \Pi$ be an instance, let $F_I : [0,1]^n \to [0,1]^n$ be I's associated fixed point function, and let C_I be the algebraic circuit corresponding to F_I. As a pre-processing step, we convert C_I to an equivalent alternative circuit C'_I that satisfies that all the signals propagated by all gates in C'_I and all the used rational constants in C'_I are contained in the interval $[0,1]$. The transformed circuit C'_I may contain two additional type of gates: Division gates and gates that computes the absolute value of the difference of two operands. We will refer to the latter type of gate as an *absolute difference gate*. The circuit C'_I will not contain any subtraction gates, and will not contain max and min gates either. The transformation procedure for C_I follows the same approach of the transformation given in Theorem 4.3 of [9] where the 3-Player Nash equilibrium problem is proved FIXP-complete, and borrows some important ideas from there. Nonetheless, there are important differences in our transformation, starting with the fact that we use a different set of types of gates in our circuit. (Details can be found in the full proof in [18].)

For notational convenience, in the remainder of the proof we may treat C'_I as the function F_I, hence we may write $C'_I(x) = y$ to denote $F_I(x) = y$. Let ρ denote the reduction to CDS-CLEARING: We construct our instance $\rho(I)$ of CDS-CLEARING (i.e., a non-degenerate financial network) from the circuit C'_I. The instance $\rho(I)$ will have the property that its clearing vectors are in one-to-one correspondence with the fixed points of C'_I, and that banks $1, \ldots, n$ in our construction correspond to the input gates of C'_I. More precisely, our construction is such that for each fixed point x of C'_I, in the corresponding clearing vector r for $\rho(I)$ it holds that $(r_1, \ldots, r_n) = x$. Our reduction works through a set of financial system *gadgets*, of which we prove that their recovery rates (under the clearing condition) must replicate the behaviour of each type of arithmetic operation that can occur in the circuit C'_I. Each of our gadgets is non-degenerate, has one or two *input banks* that correspond to the input signals of one of the types of arithmetic gate, and there is an *output bank* that corresponds to the output signal of the gate. For each of the gadgets, it holds that the output bank must have a recovery rate that equals the result of applying the respective arithmetic operation on the recovery rates of the input banks, see examples in Figure **??** ($g_{\text{pos}-}$ is a building block for the absolute difference gadget). Besides gadgets for the necessary arithmetic operations, our reduction employs an additional *duplication gadget* g_{dup} that can be used to connect the output of a particular gadget to the input of more than one subsequent gadget. A technically involved step is needed for the division gates; we replace some of the divisions in the circuit C'_I by taking successive square-roots followed by successive squaring operations, where proper

care has to be taken to ensure that the results of all these operations stay within the interval $[0, 1]$. Full definitions of our gadgets (including gadgets for squares and square roots) can be found in [18].



(a) Addition gadget g_+. **(b)** Positive subtraction gadget $g_{\text{pos}-}$ computes $\max\{0, r_1 - r_2\}$.

Figure 2 Exemplar gadgets from our reduction ρ.

In our financial system, these gadgets are then connected together according to the structure of the circuit C'_I: Output banks of (copies of) gadgets are connected to input banks of other gadgets through a single unit-cost debt contract, which mimics the propagation of a signal between two gates of the arithmetic circuit. This results in a financial system whose behaviour replicates the behaviour of the arithmetic circuit. The first layer of the financial system consists of n banks representing the n input nodes of the circuit, and the last layer of the financial system has n banks corresponding to the n output nodes of the circuit. As a final step in our reduction, the n output banks in the last layer are connected through a single unit-cost debt contract to the n input banks. This last step enforces the recovery rates of the input banks (i.e., banks $1, \ldots, n$) are equal to the recovery rates of the last layer, under the clearing requirement. Consequently, any vector of clearing recovery rates for $\rho(I)$ must then correspond to a fixed point of C'_I, where the recovery rates of the first n banks in the system equal those of the final n banks, so that $C'_I(r_1, \ldots, r_n) = (r_1, \ldots, r_n)$, i.e., $(r_1, \ldots, r_n)$ is a fixed point of C'_I. It is clear that $\rho(I)$ can be constructed in polynomial time from C'_I, and since C'_I can be constructed in polynomial time from I, the financial system $\rho(I)$ takes polynomial time to compute.

FIXP$_a$-completeness of strong approximations holds since any strong ϵ-approximation of the CRRV of $\rho(I)$ corresponds to a strong ϵ-approximate fixed point of C'_I. ◀

4 A Sufficient Structural Condition for Irrational Solutions

In this section we investigate the existence of irrational solutions in financial systems in more depth. Our starting point is the observation that irrational clearing recovery rates can only arise under certain structural conditions on the financial system (e.g., a system with no CDSes has a rational CRRV [7]). Which structural conditions must exactly hold in a financial system for irrational clearing vectors to potentially exist? In this section, we present a set of sufficient structural conditions that provides a partial answer to this question.

4.1 Switched Cycles

We define the *auxiliary graph* $G_{I,\text{aux}}$ of $I = (N, e, c)$ to be a tricoloured directed graph obtained from G_I by adding a red-coloured arc (R, i) for every $(i, j, R) \in \mathcal{CDS}$. The auxiliary graph corresponding to the instance in Figure 1b is given in Figure 3. We say that an instance I is *acyclic* if and only if its $G_{I,\text{aux}}$ contains no directed cycle. It is not hard to see that every acyclic financial system has only rational solutions. (We defer the proof to the full version [18].)

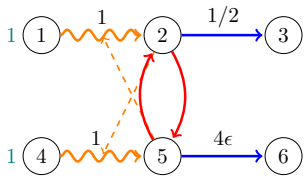


Figure 3 The auxiliary graph for the instance in Figure 1b.

We say that a node $i \in N$ is *switched off* iff it has only one incoming red arc and no outgoing blue arcs. A node $i \in N$ is *switched on* iff its incoming red arcs exceeds 1 or its incoming red arcs equals 1 and there is at least one outgoing blue arc. Note that switched on and switched off nodes are not complements of each other. A node that is not a debtor in any CDS is neither switched on nor switched off. These notions are illustrated in Figure 4.

▶ **Definition 6** (Switched Cycles). *A cycle is* red *iff it has at least one red arc. A cycle is* weakly switched *iff it is red, and for at least one red arc* (u, v) *in* C*,* v *is switched on. A cycle is* strongly switched *iff it is red, and for each red arc* (u, v) *in* C*,* v *is switched on.*

We will prove that when a non-degenerate financial system I has a strongly switched cycle (and a certain additional technical condition holds), there exist coefficients for the financial system under which all CRRVs of I are irrational. Our proof introduces a framework for formulating strongly switched cycles and consists of three main steps. Firstly, we define a set of primitive financial systems without notionals and external assets, called *fragments*. Each of these fragments has a designated start and end node. A binary concatenation operation is also introduced so to obtain financial systems that are obtainable by "stringing" together fragments. We refer to graphs obtainable through this operation as *fragment strings* or *cycles* (when the end node is linked back to the initial start node). Secondly, we equip each fragment with particular choices of rational coefficients to define *arithmetic fragments*; these allow to conveniently rewrite fragment strings, given that the objective is to preserve recovery rates at the end nodes. We prove that each of the resulting arithmetic fragment cycles has irrational CRRVs. Finally, we show that a particular class of strongly switched cycles are constructible from these fragments and for each instance I with these cycles there exist rational coefficients also for nodes and arcs not in the cycle such that all the CRRVs of I are irrational.

4.2 Fragments

We denote by $\mathcal{G}$ the set of all fragments that we will use. Few representatives of $\mathcal{G}$ are defined in Figure 5 (left), presented in our tricoloured graphical notation. Start and end nodes are indicated by short incoming and outgoing black arrows, respectively. The full description of

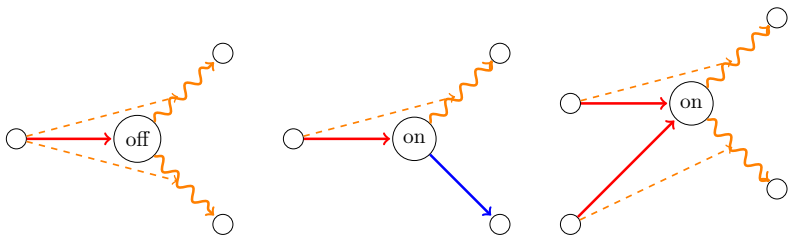


Figure 4 One switched off and two switched on nodes.

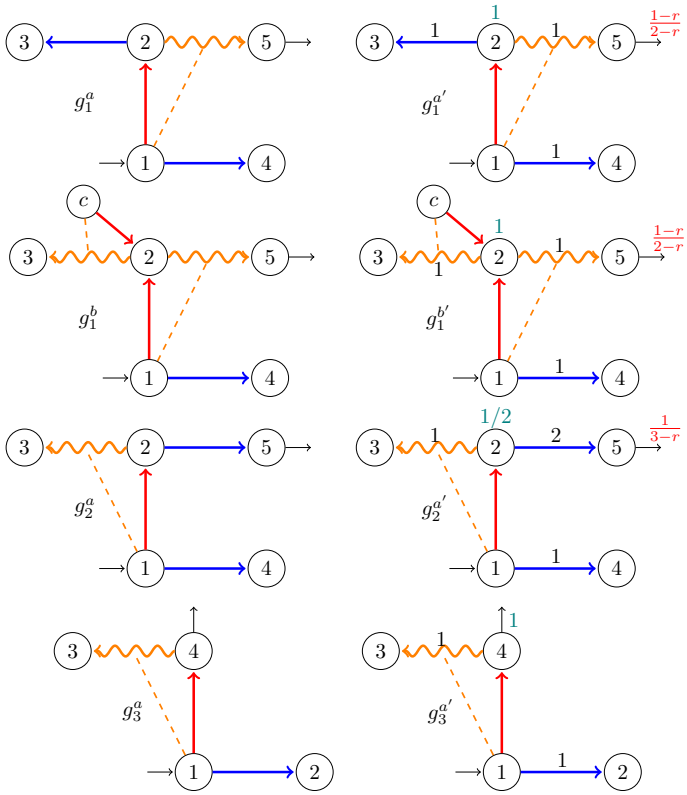


Figure 5 Some fragments in $\mathcal{G}$ (left) with their arithmetic version (right). Each fragment is labeled with its name.

$\mathcal{G}$ can be found in the full paper [18]; the additional fragments are either variants of those shown here (substituting direct liabilities with CDSes as in g_1^b vis-a-vis g_1^a) – called g_i^j, with $j \in \{a, b, c, d\}$ when $i \in [2]$ and $j \in \{a, b\}$ for $i = 3$ – or simpler configurations that just copy the recovery rate from start to end node, called d_1 and d_2.

We define a binary merging operation on ordered pairs of fragments (a, b), where every pair (a, b) is mapped to a graph obtained by taking disjoint copies of a and b, and connecting the two copies together by identifying the end node of a with the start node of b. The new start node and end node of the resulting system is the start node of the copy of a and the end node of the copy of b, respectively. We denote the result of the merge operation on fragments a and b symbolically by the notation ab. A *fragment string* is a fragment obtainable from fragments in $\mathcal{G}$ using any number of sequential applications of the merge operation. We let $\mathcal{GS}$ be the set of fragment strings (i.e., the closure of $\mathcal{G}$ under the *merge* operation). By identifying the start node with the end node of a fragment string we obtain a *fragment cycle*. The induced fragment cycle is denoted $\dot{x}g_s\dot{x}$, where $x \in \mathcal{G}$ and $g_s \in \mathcal{GS}$. Let $\mathcal{GC}$ to be the set of fragment cycles.

A fragment with fixed coefficients is called an *arithmetic fragment*, see Figure 5 (right) where we omit to show 0 external assets for some nodes. We denote by x' or x'' the arithmetic version of $x \in \mathcal{G}$. The difference between x' and x'' are minimal; for the fragments in Figure 5, the only difference between x' and x'' is that the notional for the bottom right liability (e.g., arc from node 1 to node 4 in g_1^a) is valued 2 rather than 1. The red labels at the end of an arithmetic fragment indicate the assets of the end node under any clearing vector as a

function of the recovery rate r of the start node. Importantly, both x' and x'' have the same recovery rate at the end node. The *merge* operation and notation used for fragments apply to arithmetic fragments as well. The following observation can be derived by inspection.

▶ **Observation 7.** *Let x'_1 and x'_2 be any two consecutive arithmetic fragments in a string or cycle C of arithmetic fragments. Let r be the recovery rate of the start node of x'_1 under a clearing vector of C. It holds that:*

- *If $x'_1 \in \{g_i^{j'}, g_i^{j''} : i \in [2], j \in \{a,b,c,d\}\}$ and $x'_2 \in \{g_i^{j'} : i \in [2], j \in \{a,b,c,d\}\} \cup \{d'_1, d'_2\}$, then the recovery rate of the end node of x'_1, is $(1-r)/(2-r)$ or $1/(3-r)$.*
- *If $x'_1 \in \{g_3^{j'}, g_3^{j''} : j \in \{a,b\}\}$ and $x'_2 \in \{g_i^{j''} : i \in [3], j \in \{a,b,c,d\}\}$, then the recovery rate of the end node of x'_1 is $1/(3-r)$.*

We now give a notion of equivalence between arithmetic fragment strings.

▶ **Definition 8.** *Let x_s^1, x_s^2 be two arithmetic fragment strings. We say that x_s^1 and x_s^2 are* equivalent *iff the recovery rate of the end node of x_s^1 equals the recovery rate of the end node of x_s^2 for all possible choices $r \in [0,1]$ of the recovery rate of the input nodes of x_1^s and x_2^s.*

Equivalence enables us to simplify big fragment string and cycles to simpler ones while preserving the recovery rate of the end node. This is achieved by a set of rewriting rules.

Rule 0: Replace an occurrence of a fragment $g_i^{j'}$ ($g_i^{j''}$, respectively), where $i \in [3]$ and $j \in \{a,b,c,d\}$, with the fragment $g_i^{a'}$ ($g_i^{a''}$, respectively).

Rule 1: Replace an occurrence of a fragment $g_2^{a'}$ (respectively $g_2^{a''}$) by $g_1^{a'} g_1^{a'}$ respectively $g_1^{a''} g_1^{a'}$ if the fragment $g_2^{a'}$ (or respectively $g_2^{a''}$) is followed by one of the fragments in $\{g_1^{a'}, g_2^{a'}, g_3^{a'}, d'_1, d'_2\}$.

Rule 2: Replace an occurrence of a consecutive pair of fragments $g'_3 g_i^{a''}$, where $g'_3 \in \{g_3^{a'}, g_3^{a''}\}$, and $i \in [3]$, by the fragments $g_2^{a'} g_i^{a'}$. By Observation 7, the recovery rates of the end nodes of g'_3 and $g_2^{a'}$ are identical under this substitution, under any clearing vector, so that the two fragment strings are equivalent.

Rule 3: Remove an occurrence of d'_1 or d'_2. This substitution is straightforward from the fact that both d'_1 and d'_2 just transfer the recovery rate from the start to the end node.

4.3 Irrationality of Strongly Switched Cycles

Consider any instance $I = (N, e, c)$ with auxiliary graph $G_{I,\text{aux}}$. If I has a strongly switched cycle, then this cycle is composed entirely of the fragments in $\mathcal{G}$. This is formalised as follows.

▶ **Definition 9.** *Let G' be a fragment cycle, and let C' be the unique directed cycle in G'. The fragment cycle G' is said to* agree with a cycle C of $G_{I,\text{aux}}$ *iff there is a mapping $\xi : V(G') \to V(G_{I,aux})$ with the following properties:*

- *For all $(v,w) \in E(G')$, $(\xi(v), \xi(w))$ is in $E(G_{I,aux})$ and has the same color as (v,w).*
- *ξ restricted to the domain $V(C')$ defines a bijection between $V(C')$ and $V(C)$.*
- *For each CDS (i,j,R) in G', $(\xi(i), \xi(j), \xi(R))$ is a CDS of G.*

Note that the above points imply that ξ restricted to $V(C')$ defines an arc-color-preserving isomorphism between C' and C. However, this isomorphism does not necessarily extend to node sets larger than C': nodes in $V(G') \setminus V(C')$ may be mapped by ξ to the same vertex of $G_{I,aux}$. We then define the fragment cycle $G' = G'_n \cup G'_l$ to simply *agree with a cycle C of $G_{I,aux}$, if G' agrees with C of $G_{I,aux}$ through a mapping ξ for which it additionally holds that*

- *all nodes outside C' are mapped to vertices outside C,*
- *For every pair of nodes $\{u,v\} \subseteq V(G')$, where $v \in G'_n$ and $u \in G'_l$, $\xi(u) \neq \xi(v)$, and*
- *for every node $u \in G'_l$, $\xi(u)$ has an outgoing arc pointing towards a node not in C',*

where G'_n is the set of nodes in the fragment cycle G' labelled with a number (as $1, \ldots, 5$ in Figure 5) and G'_l is the set of fragment nodes labelled with a letter (as c in Figure 5).

The notion of *simple* agreement informally requires that the neighbouring nodes of C' are sufficiently "independent" from each other and from the cycle C, under the mapping ξ. This brings us to the definition of a *simple* strongly switched cycle (which makes precise our condition on off-cycle paths between the nodes in the cycles in the informal statement of our second main theorem in the introduction).

▶ **Definition 10.** *A cycle C of $G_{I,aux}$ is a* simple *strongly switched cycle iff C is strongly switched, and for each red arc (u, v) of C there are non-red arcs (u, u') and (v, v') such that $u', v' \notin C$. Furthermore, if (u, u') or (v, v') is orange, then the reference bank R of the corresponding CDS is not in C and R has an outgoing non-red arc pointing to a node not in C.*

The fragments in $\mathcal{G}$, can represent any strongly switched cycle: If $G_{I,\text{aux}}$ has a strongly switched cycle C, then there is a fragment cycle G' consisting of fragments in $\mathcal{G}$ s.t G' agrees with C of $G_{I,\text{aux}}$. Similarly, if $G_{I,\text{aux}}$ has a simple strongly switched cycle C, then there is a fragment cycle G' consisting of fragments in $\mathcal{G}$ s.t G' simply agrees with C of $G_{I,\text{aux}}$. All switched on nodes of C correspond to the 2-labeled nodes of a g_2^j or g_1^j fragment, for some $j \in \{a, b, c, d\}$. The next lemmas show that we can set the coefficients in any strongly switched fragment cycle s.t the fragment cycle admits only irrational clearing recovery rates.

▶ **Lemma 11.** *For all fragment cycles $C \in \mathcal{GC}$ consisting of only fragments in $\{g_1^j : j \in \{a, b, c, d\}\}$, there exist coefficients s.t. the clearing recovery rate vector of C is irrational.*

Proof Sketch. Consider a fragment cycle consisting exclusively of only fragments in $\{g_1^j : j \in \{a, b, c, d\}\}$. For all $j \in \{a, b, c, d\}$, fix the coefficients of all g_1^j fragments in the cycle to obtain the arithmetic version $g_1^{j'}$. Use rewriting Rule 0 to replace all $g_1^{j'}$ occurrences by $g_1^{a'}$. The resulting arithmetic fragment cycle consists of a number of consecutive copies of $g_1^{a'}$, say k of them. Consider now any clearing vector r for the fragment cycle. We can prove by induction (details in the full paper [18]) that the end node of the ith fragment has recovery rate equal to $(f_i - rf_{i-2})/(f_{i+2} - rf_i)$, where f_i is the ith Fibonacci number, with $f_0 = 0$.

We know that the end node of the last fragment in the fragment cycle has a recovery rate that coincides with the recovery rate r of the start node of the first fragment. Therefore, in a clearing vector of recovery rates, it holds that $r = (f_n - rf_{k-2})/(f_{k+2} - rf_k)$ which is equivalent to solving the equation $r^2 f_k - (f_{k+2} + f_{k-2})r + f_k = 0$. Since $f_{k+2} + f_{k-2} = f_{k+1} + f_k + f_{k-2} = 2f_k + f_{k-1} + f_{k-2} = 3f_k$, computing the recovery rate of the initial node 1 comes down to solving the quadratic equation $r^2 - 3r + 1 = 0$. Solving this equation we obtain that the only solution in $[0, 1]$ is $r = (3 - \sqrt{5})/2$ which is irrational, thus the CRRV of the strongly switched arithmetic fragment cycle is irrational and is unique. ◀

The next lemma (proof omitted) extends the above to a larger class of arithmetic fragments.

▶ **Lemma 12.** *For all fragment cycles composed of fragments $\mathcal{G}$ in which every occurrence of a fragment in $\{g_3^j : j \in \{a, b\}\}$ is followed by a fragment in $\{g_i^j : i \in [2], j \in \{a, b, c, d\}\}$, there exist coefficients s.t the clearing recovery rate vector of C is irrational.*

▶ **Theorem 13.** *Let I be a non-degenerate financial system such that $G_{I,aux}$ has a simple strongly switched cycle. Then there exist rational coefficients for I such that all clearing vectors of I are irrational.*

Proof Sketch. Let C be a strongly switched cycle of $G_{I,\text{aux}}$ and let G' be a fragment cycle that simply agrees with C through a mapping ξ satisfying the conditions stated in Definition 9. By Lemma 12, there are coefficients for G' such that all clearing vectors of G' are irrational.

In $G_{I,\text{aux}}$, we can now set the notionals and external assets on the vertices and arcs through the mapping ξ. This assignment of coefficients is well-defined by the properties of ξ stated in Definition 9 (i.e., there are no two arcs or vertices that get assigned multiple conflicting coefficients this way). We set the remaining coefficients of $G_{I,\text{aux}}$ (i.e., the coefficients on the arcs and vertices outside the image of ξ) as follows: external assets to 0; notionals of (v, w) to 1, if $\xi^{-1}(v)$ is a node labeled with a letter, and w is not in the image of ξ, or 0 viceversa.

Let G'' denote the subgraph of G formed by the image of ξ. Note that no payments flow from G'' to any node outside G'' under any clearing vector. It then follows by Lemma 12 and the simple agreement properties, that under this setting of the coefficient of $G_{I,\text{aux}}$, every clearing vector is irrational (and in particular these irrational recovery rates emerge in the nodes of G''). This establishes our claim. ◀

5 Financial Systems with Guaranteed Rational Solutions

In the previous section, we identified a sufficient structural condition for the ability of a financial system to have irrational clearing vectors. In this section we investigate how close these conditions are to a characterisation, by attempting to answer the opposite question: Under which structural conditions are rational clearing vectors guaranteed to exist in a financial system? The answer to this relates again to the notion of switched cycles: We will show that if a given non-degenerate financial system does not possess any weakly switched cycle, then there must exist clearing vectors of the system that are rational. We investigate furthermore the computational complexity of finding a clearing vector in this case: Solutions can, informally stated, be computed by solving a linear number of PPAD-complete problems. This latter result is achieved through identifying a natural class of financial systems for which the problem of computing an exact fixed point is PPAD-complete.

The results in this section indicate that the structural conditions for irrationality formulated in the previous section do close in on a characterisation, although there is still a "gray area" left: For those instances of financial systems that do have weakly switched cycles, but do not have any simple strongly switched cycles, we are not yet able to determine by the structural interrelationships of the financial contracts whether these systems are likely to possess rational or irrational solutions. This forms an interesting remaining problem that we leave open. The main result we will prove in this section is thus the following.

▶ **Theorem 14.** *Let I be a non-degenerate financial system. If $G_{I,aux}$ does not have any weakly switched cycles, then all clearing vectors of I are rational.*

We start by showing that for a particular subclass of financial systems without weakly switched cycles, the clearing vector computation problem lies in Linear-FIXP, which is equal to PPAD, and thus the clearing vectors of such financial system must have polynomial size rational coefficients.

▶ **Definition 15.** *An instance $I = (N, e, c)$ of a financial system is said to have the* dedicated CDS debtor property *iff for every node $i \in N$ that is a debtor of at least one CDS of I, the following holds: There are no debt contracts (with a non-zero notional) in which i is the debtor, and all CDSes (with a non-zero notional) for which i is the debtor share the same reference bank.*

▶ **Lemma 16.** *(The exact computation version of)* CDS-CLEARING *restricted to non-degenerate financial systems with the dedicated CDS debtor property is PPAD-complete.*

The proof, which is omitted, works by showing that for this special case of the problem, one can rewrite function f into a function f' where multiplication is omitted. This is done by disregarding the recovery rates of those nodes that are debtors of CDSes and instead expressing their individual CDS payments in a way that does not require multiplication. Furthermore, the remaining nodes do not need multiplication under our original formulation of f. Secondly, PPAD-hardness is established from minor modifications of the proof of the main theorem in [23].

The above PPAD-completeness result (and more precisely the PPAD-membership part of the result), shows that non-degenerate instances with the dedicated CDS debtor property must have polynomial size rational solutions. We use this fact to prove Theorem 14.

Proof sketch of Theorem 14. Consider the graph D that has as its nodes the strongly connected components (SCCs) of $G_{I,\text{aux}}$, and has an arc from a node S to a node T if and only if there exists an arc in $G_{I,\text{aux}}$ that runs from a node in S to a node in T. It is clear that D is a directed acyclic graph.

We may show that we can find a rational clearing vector for $G_{I,\text{aux}}$ by finding rational clearing vectors of the separate SCCs of the system. However, both the assets and the liabilities of the nodes in a given SCC might depend on the contracts from outside the SCC that point into the SCC. Similarly, the liabilities of the nodes in the SCC might depend on arcs pointing from the SCC to external nodes. We may overcome this problem by including the outward-pointing arcs of an SCC into the subinstances that we aim to solve for, and by iterating over the SCCs according to the topological order of D: That is, we first find clearing vectors to the set $\mathcal{S}_1$ of SCCs that have no incoming arc in D. For such SCCs, the assets and liabilities of the nodes are not influenced by external arcs pointing into the SCC. We subsequently find clearing rates for the set of SCCs $\mathcal{S}_2$ that succeed $\mathcal{S}_1$ in the topological order defined by D. In general, we define $\mathcal{S}_j$ inductively as the set of SCCs that directly succeed $\mathcal{S}_{j-1}$ in the topological order defined by D, and we iteratively find clearing rates to the set of SCCs $\mathcal{S}_j$, given the clearing rates computed for $\mathcal{S}_1, \ldots, \mathcal{S}_{j-1}$, until we have obtained a clearing vector covering all nodes in the system. A crucial observation that motivates this approach is that the absence of any weakly switched cycle of $G_{I,\text{aux}}$ causes all SCCs to satisfy the dedicated CDS debtor property, and that therefore the clearing vector computation problem considered in each iteration lies in PPAD. At each iteration, we are thus guaranteed that there are rational recovery rates, and finding them requires solving a PPAD-complete problem. However, there are quite a few details required to turn the above ideas into a rigorous proof, and we defer these to the full version of this paper [18]. ◀

The procedure outlined in the proof of Theorem 14 requires solving a PPAD-complete problem in each iteration, and the number of such iterations is at most linear in the instance size. Since solving each of these problems in PPAD yields a rational solution of size polynomial in the input, one might be tempted to think that the procedure in its entirety is capable of finding a polynomial size rational solution for any financial system that has no weakly switched cycles. Unfortunately, the latter is not true: Observe that in each iteration of the procedure, the PPAD-complete problem instance that is solved, is actually constructed using the rational recovery rate vectors that are computed in the preceding iterations. The coefficients in the PPAD-complete problem instance that is to be solved in any given iteration, are thus polynomially sized in the output recovery rates of the previous iteration. Altogether, this means that the coefficient sizes potentially grow by a polynomial factor in each iteration, and that the final recovery rates output by the procedure are potentially of exponential size.

Indeed, there are examples of financial systems without weakly switched cycles for which the rational clearing vector has recovery rates that require an exponential number of bits to write down. A simple example is obtained by taking some of the gadgets in the reduction used in our FIXP-completeness result (Theorem 5). By taking a duplication gadget followed by a multiplication gadget that is connected to the two output nodes of the duplication gadget. We may then take multiple copies of these, and chain them together to form an acyclic financial system. If we now give the first node in the chain (i.e., the input node of the first duplication gadget) some small amount of positive external assets $c < 1$, this acyclic financial system essentially performs a sequence of successive squaring operations on the number c, under the unique clearing vector. The resulting recovery rates on the output nodes of the multiplication gadgets are then doubly exponentially small in magnitude, with respect to the number of squaring repetitions. Thus, the resulting clearing recovery rates require a number of bits that is exponential in the size of the financial system.

If one is willing to discard the complexity issues that arise from working with large-size rational numbers, it is possible to study the procedure in the proof of Theorem 14 in the Blum-Shub-Smale model of computation. Under this computational model, any real number takes one unit of space to store, regardless of its size. Moreover, standard arithmetic operations are assumed to take unit time.[1] The proof of Theorem 14 then implies that when one has oracle access to PPAD, it is possible to find rational clearing vectors in polynomial time under this model of computation. The class of problems polynomial time solvable under the Blum-Shub-Smale model is commonly denoted by $\mathsf{P}_{\mathbb{R}}$. Hence, we obtain the following corollary.

▶ **Corollary 17.** *The exact computation version of* CDS-CLEARING*, restricted to instances without weakly switched cycles, is in the complexity class* $P_{\mathbb{R}}^{PPAD}$.

6 Conclusions

In this paper we study two questions of significance related to the systemic risk in financial networks with CDSes, a widely used and potentially disruptive class of financial derivatives. Firstly, we settle the computational complexity of computing strong approximations of each bank's exposure to systemic risk, arguably the right notion of approximation of interest to industry – a conceptual point so far overlooked in the literature. We show that this problem is FIXP-complete. Secondly, we initiate the study of the rational fragment of FIXP by studying the conditions under which rational solutions for CDS-CLEARING exist. Our results here are not conclusive in that there is a gap between our necessary and sufficient conditions, the cycles which involve both switched on and switched off nodes being not fully understood. [2] We conjecture that for any network with a weakly switched cycle there exist rational values for assets and liabilities that lead to irrational solutions; however, our arguments and scheme cannot be easily generalised to those instances. We leave providing a full characterisation as an open problem.

Further research directions are suggested by our work. It would be interesting to study whether Corollary 17's connection between CDS-CLEARING, PPAD, and $\mathsf{P}_{\mathbb{R}}$ (i.e. polynomial time under the Blum-Shub-Smale model of computation [5]) holds more generally for the entire rational subset of problems in FIXP. Furthermore, it is interesting to pursue finding

[1] For a formal and more accurate definition of the Blum-Shub-Smale model, see the book [5].
[2] We regard the *simplicity* condition we made (i.e., about off-cycle paths between cycle nodes) as a technicality, which is less interesting and likely somewhat easier to deal with.

polynomial-time *constant* approximation algorithms of clearing recovery rate vectors: Also from an applied point of view, achieving a good approximation factor here (say with an additive approximation term of 1/100) might yield solutions that are useful in most practical circumstances and could be considered acceptable by financial institutions. We note here that a 1/2-strong approximation is easy to compute (a recovery rate vector of only 1/2s would indeed suffice).

References

1 Euro-parliament bans "naked" credit default swaps. EUBusiness. `https://www.eubusiness.com/news-eu/finance-economy-cds.dij`.

2 Lasting effects: The global economic recovery 10 years after the crisis. IMFBlogs. URL: `https://blogs.imf.org/2018/10/03/lasting-effects-the-global-economic-recovery-10-years-after-the-crisis/`.

3 Daron Acemoglu, Asuman Ozdaglar, and Alireza Tahbaz-Salehi. Systemic risk and stability in financial networks. *American Economic Review*, 105(2):564–608, 2015.

4 Nils Bertschinger, Martin Hoefer, and Daniel Schmand. Strategic payments in financial networks. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 46:1–46:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ITCS.2020.46`.

5 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer Science & Business Media, 1998.

6 Rodrigo Cifuentes, Gianluigi Ferrucci, and Hyun Song Shin. Liquidity risk and contagion. *Journal of the European Economic association*, 3(2-3):556–566, 2005.

7 Larry Eisenberg and Thomas H Noe. Systemic risk in financial systems. *Management Science*, 47(2):236–249, 2001.

8 Matthew Elliott, Benjamin Golub, and Matthew O Jackson. Financial networks and contagion. *American Economic Review*, 104(10):3115–53, 2014.

9 Kousha Etessami and Mihalis Yannakakis. On the complexity of nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

10 Aris Filos-Ratsikas, Yiannis Giannakopoulos, Alexandros Hollender, Philip Lazos, and Diogo Poças. On the complexity of equilibrium computation in first-price auctions. *arXiv preprint arXiv:2103.03238*, 2021.

11 Aris Filos-Ratsikas, Kristoffer Arnsfelt Hansen, Kasper Høgh, and Alexandros Hollender. Fixp-membership via convex optimization: Games, cakes, and markets. *arXiv preprint*, 2021. `arXiv:2111.06878`.

12 Jugal Garg, Ruta Mehta, Vijay V. Vazirani, and Sadra Yazdanbod. Settling the complexity of leontief and PLC exchange markets under exact and approximate equilibria. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 890–901. ACM, 2017. `doi:10.1145/3055399.3055474`.

13 Paul Glasserman and H Peyton Young. How likely is contagion in financial networks? *Journal of Banking & Finance*, 50:383–399, 2015.

14 Paul W Goldberg and Alexandros Hollender. The hairy ball problem is ppad-complete. *Journal of Computer and System Sciences*, 2021.

15 Sebastian Heise and Reimer Kühn. Derivatives and credit contagion in interconnected networks. *The European Physical Journal B*, 85(4):1–19, 2012.

16 Brett Hemenway and Sanjeev Khanna. Sensitivity and computational complexity in financial networks. *Algorithmic Finance*, 5(3-4):95–110, 2016.

17 Daning Hu, J Leon Zhao, Zhimin Hua, and Michael CS Wong. Network-based modeling and analysis of systemic risk in banking systems. *MIS quarterly*, pages 1269–1291, 2012.

18 Stavros D. Ioannidis, Bart de Keijzer, and Carmine Ventre. Strong approximations and irrationality in financial networks with financial derivatives. *CoRR*, abs/2109.06608, 2021. arXiv:2109.06608.

19 Pál András Papp and Roger Wattenhofer. Default ambiguity: finding the best solution to the clearing problem. *arXiv preprint*, 2020. arXiv:2002.07741.

20 Pál András Papp and Roger Wattenhofer. Network-aware strategies in financial systems. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 91:1–91:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.91.

21 Leonard CG Rogers and Luitgard AM Veraart. Failure and rescue in an interbank network. *Management Science*, 59(4):882–898, 2013.

22 Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Clearing payments in financial networks with credit default swaps. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 759–759, 2016.

23 Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Finding clearing payments in financial networks with credit default swaps is PPAD-complete. *LIPIcs: Leibniz International Proceedings in Informatics*, 67, 2017.

Regularized Box-Simplex Games and Dynamic Decremental Bipartite Matching

Arun Jambulapati
Stanford University, CA, USA

Yujia Jin
Stanford University, CA, USA

Aaron Sidford
Stanford University, CA, USA

Kevin Tian
Stanford University, CA, USA

Abstract

Box-simplex games are a family of bilinear minimax objectives which encapsulate graph-structured problems such as maximum flow [41], optimal transport [29], and bipartite matching [5]. We develop efficient near-linear time, high-accuracy solvers for regularized variants of these games. Beyond the immediate applications of such solvers for computing Sinkhorn distances, a prominent tool in machine learning, we show that these solvers can be used to obtain improved running times for maintaining a (fractional) ϵ-approximate maximum matching in a dynamic decremental bipartite graph against an adaptive adversary. We give a generic framework which reduces this dynamic matching problem to solving regularized graph-structured optimization problems to high accuracy. Through our reduction framework, our regularized box-simplex game solver implies a new algorithm for dynamic decremental bipartite matching in total time $\widetilde{O}(m \cdot \epsilon^{-3})$, from an initial graph with m edges and n nodes. We further show how to use recent advances in flow optimization [11] to improve our runtime to $m^{1+o(1)} \cdot \epsilon^{-2}$, thereby demonstrating the versatility of our reduction-based approach. These results improve upon the previous best runtime of $\widetilde{O}(m \cdot \epsilon^{-4})$ [6] and illustrate the utility of using regularized optimization problem solvers for designing dynamic algorithms.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases bipartite matching, decremental matching, dynamic algorithms, continuous optimization, box-simplex games, primal-dual method

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.77

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2204.12721

Funding *Yujia Jin*: supported by a Stanford Graduate Fellowship and the Dantzig-Lieberman Operations Research Fellowship.
Aaron Sidford: supported in part by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship.
Kevin Tian: supported in part by a Google Ph.D. Fellowship, a Simons-Berkeley VMware Research Fellowship, a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, and a PayPal research award.

Acknowledgements We thank anonymous reviewers for their feedback, Amin Saberi and David Wajc for helpful conversations, Jason Altschuler for providing a reference for the unaccelerated convergence rate of Sinkhorn's algorithm (in the original submission, we claimed no such rate had been stated previously), and Monika Henzinger and Thatchaphol Saranurak for helpful information regarding prior work.

1 Introduction

Efficient approximate solvers for graph-structured convex programming problems have led to a variety of recent advances in combinatorial optimization. Motivated by problems related to maximum flow and optimal transportation, a recent line of work [40, 30, 41, 42, 29, 13] developed near-linear time, accelerated solvers for a particular family of convex programming objectives we refer to in this paper as *box-simplex games*:

$$\min_{x\in\Delta^m}\max_{y\in[-1,1]^n} y^\top \mathbf{A}x + c^\top x - b^\top y \text{ where } \Delta^m := \{x\in\mathbb{R}^m_{\geq 0} |\, \|x\|_1 = 1\}\,. \tag{1}$$

Box-simplex games, (1), are bilinear problems where a maximization player is constrained to the box (the ℓ_∞ ball) and a minimization player is constrained to the simplex (the nonnegative ℓ_1 shell). The problem provides a convenient encapsulation of linear programming problems with ℓ_1 or ℓ_∞ structure; (1) can be used to solve box-constrained ℓ_∞ regression problems [41, 42] and maximizing over the box player yields the following ℓ_1 regression problem

$$\min_{x\in\Delta^m} c^\top x + \|\mathbf{A}x - b\|_1\,. \tag{2}$$

Furthermore, solvers for (1) and (2) are used in state-of-the-art algorithms for approximate maximum flow [41], optimal transport (OT) [29], (width-dependent) positive linear programming [7], and semi-streaming bipartite matching [5].

One of the main goals of our work is to develop efficient algorithms for solving *regularized* variants of the problems (1) and (2). An example of particular interest is the following

$$\min_{x\in\Delta^m | \mathbf{B}^\top x = d} c^\top x + \mu H(x), \text{ where } \mu \geq 0 \text{ and } H(x) := \sum_{i\in[m]} x_i \log x_i. \tag{3}$$

The case of (3) when $\mathbf{B}\in\mathbb{R}^{m\times n}$ is the (unsigned) edge-vertex incidence matrix of a complete bipartite graph, and d is a pair of discrete distributions supported on the sides of the bipartition, is known as the *Sinkhorn distance* objective. This is used in machine learning [14] as an efficiently-computable approximation to optimal transport distances: c corresponds to movement costs, and d encodes the prescribed marginals. This objective has favorable properties, e.g. differentiability [44], and there has been extensive work by both theorists and practitioners to solve (3) and analyze its properties (see e.g. [14, 4] and references therein). Choosing $\mathbf{A}$ and b to be sufficiently large multiples of $\mathbf{B}^\top$ and d, it can be shown that solutions to the following regularized variant of (2) yield approximate solutions to (3),

$$\min_{x\in\Delta^m} c^\top x + \|\mathbf{A}x - b\|_1 + \mu H(x)\,. \tag{4}$$

Beyond connections to Sinkhorn distances, there are additional reasons why it may be desirable to solve regularized box-simplex games. For example, regularization could speed up algorithms and allow high-precision solutions to be computed more efficiently. Further, obtaining a high-precision solution to a regularized version of the problem yields a more canonical and predictable approximate solution than an arbitrary low-precision approximation to the unregularized problem. Moreover, regularization potentially makes optimal solutions more stable to input changes. For box-simplex games stemming from bipartite matching we quantify this stability and show all of these properties allow regularized solvers to yield faster algorithms for a particular dynamic matching problem.

Altogether, the main contributions of this paper are the following.

1. We give improved running times for the problem of *dynamic decremental bipartite matching* (DDBM) with an *adaptive adversary*, a fundamental problem in dynamic graph algorithms. Our algorithm follows from a general black-box reduction we develop from DDBM to solving (variants of) regularized box-simplex games to high precision.
2. We give efficient solvers for (variants of) the regularized box-simplex problems (3), (4).
3. As a byproduct, we also show how to apply our new solvers (and additional tools from the literature) to obtain state-of-the-art methods for computing Sinkhorn distances.

Formally, the *DDBM* problem we consider is the following: given a bipartite graph undergoing edge deletions, maintain an *ϵ-approximate (maximum) matching*,[1] that is a matching which has size at least a $(1-\epsilon)$-fraction of the maximum (for a pre-specified) value of ϵ. Unless specified otherwise, we consider the *adaptive adversary model* where edge deletions can be specified adaptively to the matching returned. Further, we allow the matching output by the algorithm to be *fractional*, rather than integral.

We show how to reduce solving the DDBM problem to solving a sequence of regularized box-simplex games. This reduction yields a new approach to dynamic matching; this approach is inspired by prior work, e.g. [6], but conceptually distinct in that it decouples the solving of optimization subproblems from characterizing their solutions. For our specific DDBM problem, the only prior algorithm achieving an amortized polylogarithmic update time (for constant ϵ) is in the recent work of [6], which derives their dynamic algorithm as an application of the *congestion balancing* technique. Our reduction eschews this combinatorial tool and directly argues, via techniques from convex analysis, that solutions to appropriate regularized matching problems can be used dynamically as approximate matchings while requiring few recomputations. We emphasize our use of fast *high-accuracy solvers*[2] in the context of our reduction to obtain our improved runtimes, as our approach leverages structural characteristics of the exact solutions which we only show are inherited by approximate solutions when solved to sufficient accuracy.

Our work both serves as a proof-of-concept of the utility of regularized linear programming solvers as a subroutine in dynamic graph algorithms, and provides the tools necessary to solve said problems in various structured cases. This approach to dynamic algorithm design effectively separates a "stability analysis" of the solution to a suitable optimization problem from the computational burden of solving that problem to high accuracy: any improved solver would then have implications for faster dynamic algorithms as well. As a demonstration of this flexibility, we give three uses of our reduction framework (which proceed via different solvers) in obtaining our improved DDBM update time. We hope our work opens the door to exploring the use of the powerful continuous optimization toolkit, especially techniques originally designed for non-dynamic problems, for their dynamic counterparts.

Paper organization. We overview our contributions in Section 1.1, and related prior work in Section 1.2. We state preliminaries in Section 2. In Section 3.1, we describe our framework for reducing DDBM to a sequence of regularized optimization problems satisfying certain properties, and in Section 3.2 we give three different instantiations of the framework, obtaining a variety of DDBM solvers. Finally in Section 4 we provide our main algorithm for regularized box-simplex games. In the full version we provide additional discussions on a recent advancement for faster DDBM solvers, proofs for Section 3 and Section 4, and additional results for approximating Sinkhorn distances efficiently.

[1] This is sometimes also referred to as a $(1+\epsilon)$-multiplicatively approximate matching in the literature.
[2] Throughout, we typically use the term "high-accuracy" to refer to an algorithm whose runtime scales polylogarithmically in the inverse accuracy (as opposed to e.g. polynomially).

1.1 Our results

A framework for faster DDBM. We develop a new framework for solving the DDBM problem of computing an ϵ-approximate maximum matching in a dynamic graph undergoing edge deletions from an adaptive adversary. Our framework provides a reduction from this DDBM problem to solving various regularized formulations of box-simplex games.

To illustrate the reduction, suppose we have bipartite $G = (V, E)$ and, for simplicity, that we know M^*, the size of the (maximum cardinality) matching. As demonstrated in [5], solving the ℓ_1 regression problem $\min_{x \in M^* \Delta^m} -c^\top x + \|\mathbf{A}x - b\|_1$, to ϵM^* additive accuracy for appropriate choices of $\mathbf{A}$, b, and c yields an ϵ-approximate maximum cardinality matching. Intuitively, $\mathbf{A}$ and b penalize violations of the matching constraints, and c is a multiple of the all-ones vector capturing the objective of maximizing the matching size. However, ℓ_1 regression objectives do not necessarily have unique minimizers: as such the output of directly minimizing these objectives is difficult to characterize beyond (approximate) optimality. This induces difficulty in using solutions to such problems directly in dynamic graph algorithms.

Our first key observation (building upon intuition from congestion balancing [6]) is that, beyond enabling faster runtime guarantees, regularization provides more robust solutions which are resilient to edge deletions in dynamic applications. We show that if

$$x_\epsilon^* := \min_{x \in M^* \cdot \Delta^m} -c^\top x + \|\mathbf{A}x - b\|_1 + \epsilon H(x) \tag{5}$$

is the solution to the *regularized* box-simplex formulation of bipartite matching, then x_ϵ^* enjoys favorable stability properties allowing us to argue about its size under deletions.

The stability of solutions to (5) is fairly intuitive: the entropy regularizer encourages the objective to spread the matching uniformly, when all else is held equal. For example, when G is a complete bipartite graph on $2n$ vertices, standard linear programming relaxations of matching do not favor either of (i) an integral perfect matching, and (ii) a fractional matching spreading mass evenly across many edges, over the other. However, using (i) as our approximate matching on a graph undergoing deletions is substantially more unstable; an adaptive adversary can remove edges corresponding to our matching, forcing $\Omega(n)$ recomputations. On the other hand, no deletions can cause this type of instability for strategy (ii): as each edge receives weight $\frac{1}{n}$ in the fractional matching, the only way to reduce the fractional matching size by ϵn is to remove $O(\epsilon n^2)$ edges: thus $O(\epsilon^{-1})$ recomputations intuitively suffice for maintaining an ϵ-approximate matching. This distinction underlies the use of *high-accuracy* solvers in our reduction; indeed, while they obtain large matching values in an original graph, approximate solutions may not carry the same types of dynamic matching value stability. We note similar intuition motivated the approach in [6].

To make this argument more rigorous, consider using x_ϵ^* as our approximate matching for a number of iterations corresponding to edge deletions, until its size restricted to the smaller graph has decreased by a factor of $1 - O(\epsilon)$. By using strong convexity of (5) in the ℓ_1 norm, we argue that whenever the objective value of x_ϵ^* has worsened, the maximum matching size itself must have gone down by a (potentially much smaller) amount. A tighter characterization of this strong convexity argument shows that we only need to recompute a solution to slight variants of (5) roughly $\widetilde{O}(\epsilon^{-2})$ times throughout the life of the algorithm. Combined with accelerated $\widetilde{O}(\frac{m}{\epsilon})$-time solvers for regularized box-simplex games (which are slight modifications of (5)), this strategy yields an overall runtime of $\widetilde{O}(\frac{m}{\epsilon^3})$, improving upon the recent state-of-the-art decremental result of [6].

We formalize these ideas in Section 3, where we demonstrate that a range of regularization strategies (see Definition 5) such as (5) are amenable to this reduction. Roughly, as long as our regularized objective is "at least as strongly convex" as the entropic regularizer, and

closely approximates the matching value in the static setting, then it can be used in our DDBM algorithm. Combining this framework with solvers for regularized matching problems, we give three different results. The first two obtain amortized update times of roughly $\widetilde{O}(\epsilon^{-3})$, in Theorems 9 and 10 via box-simplex games and matrix scaling, respectively (though the latter holds only for dense graphs). We give an informal statement of the former here.

▶ **Theorem 1** (informal, see Theorem 9). *Let $G = (V, E)$ be bipartite, $|V| = n$, $|E| = m$, and $\epsilon \geq \text{poly}(m^{-1})$. There is a deterministic algorithm maintaining an ϵ-approximate matching in a dynamic bipartite graph with adversarial edge deletions running in time $O(m \log^5 m \cdot \epsilon^{-3})$.*

Notably, our algorithm (deterministically) returns a *fractional matching*. There is a black-box reduction from dynamic integral matching maintenance to dynamic fractional matching maintenance contained in [45], but this reduction is bottlenecked at an amortized $\widetilde{O}(\epsilon^{-4})$ runtime (see e.g. Appendix A.2, [6]). Improving this reduction is a key open problem.

High-accuracy solvers for regularized box-simplex games. To use our DDBM framework, we give a new algorithm for solving regularized box-simplex games of the form:

$$\min_{x \in \Delta^m} \max_{y \in [0,1]^n} f_{\mu,\epsilon}(x, y) := y^\top \mathbf{A}^\top x + c^\top x - b^\top y + \mu H(x) - \frac{\epsilon}{2} \left(y^2\right)^\top |\mathbf{A}|^\top x, \tag{6}$$

where ϵ and $\mu = \Omega(\epsilon)$ are regularization parameters and y^2, $|\mathbf{A}|$ are entrywise. The terms $H(x)$ and $(y^2)^\top |\mathbf{A}|^\top x$ in (6) are parts of a primal-dual regularizer proposed in [29] (and a variation of a similar regularizer of [41]) used in state-of-the-art algorithms for approximately solving (unregularized) box-simplex games. This choice of regularization enjoys favorable properties over the joint box-simplex domain, and sidesteps the infamous ℓ_∞-strong convexity barrier that has limited previous attempts at acceleration for this problem. Under relatively mild restrictions on problem parameters (see discussion at the start of Section 4), we develop a *high accuracy solver* for (6), stated informally here.

▶ **Theorem 2** (informal, see Theorem 25). *Given an instance of (6), with $\mu = \Omega(\epsilon)$, $\|\mathbf{A}\|_\infty \leq 1$, and $\sigma \geq \text{poly}(m^{-1})$ Algorithm 4 returns x with $\max_{y \in [0,1]^n} f_{\mu,\epsilon}(x, y) - f_{\mu,\epsilon}(x^\star, y^\star) \leq \sigma$ in time $\widetilde{O}(\text{nnz}(\mathbf{A}) \cdot \frac{1}{\sqrt{\mu\epsilon}})$ where $(x^\star, y^\star)$ is the optimizer of (6).*

Our solver follows recent developments in solving unregularized box-simplex games. We analyze an approximate extragradient algorithm based on the mirror prox method of [37], and prove that iterates of the regularized problem (6) enjoy multiplicative stability properties previously shown for the iterates of mirror prox on the unregularized problem [13]. Leveraging these tools, we also show the regularizer-operator pair satisfies technical conditions known as *relative Lipschitzness* and *strong monotonicity*, thus enabling a similar convergence analysis as in [13]. This yields an efficient algorithm for solving (6).

Roughly, when the scale of the problem (defined in terms of the matrix operator norm $\|\mathbf{A}\|_\infty$ and appropriate norms of b and c) is bounded,[3] our algorithm for computing a high-precision optimizer to (6) runs in $\widetilde{O}(\frac{1}{\sqrt{\mu\epsilon}})$ iterations, each bottlenecked by a matrix-vector product through $\mathbf{A}$. When $\mu \approx \epsilon$, the optimizer of the regularized variant is an $O(\epsilon)$-approximate solution to the unregularized problem (1), and hence Theorem 25 recovers state-of-the-art runtimes (scaling as $\widetilde{O}(\epsilon^{-1})$) for box-simplex games up to logarithmic factors. We achieve our improved dependence on μ in Theorem 25 by trading off the scales of the

[3] Our runtimes straightforwardly extend to depend appropriately on these norms in a scale-invariant way.

primal and dual domains. This type of argument is well-known for *separable regularizers* [9], but a key technical novelty of our paper is demonstrating a similar analysis holds for non-separable regularizers compatible with box-simplex games e.g. the one from [29], which has not previously been done. To our knowledge, Theorem 25 is the first result for solving general regularized box-simplex games to high accuracy in nearly-linear time. We develop our box-simplex algorithm and prove Theorem 25 in Section 4.

Improved rates for the Sinkhorn distance objective. We apply our accelerated solver for (6) in computing approximations to the Sinkhorn distance objective (3), a fundamental algorithmic problem in the practice of machine learning, at a faster rate. It is well-known that solving the *regularized* Sinkhorn problem (3) with μ scaling much larger than the target accuracy ϵ enjoys favorable properties in practice [14] (compared to its unregularized counterpart, the standard OT distance). In [3], the authors show that Sinkhorn iteration studied in prior work solves (3) to additive accuracy ϵ at an unaccelerated rate of $\widetilde{O}(\frac{1}{\mu\epsilon})$. For completeness we provide a proof of this result (up to logarithmic factors) in Appendix C.3 in the full version of this paper.

As a straightforward application of the solver we develop for (6), we demonstrate that we can attain an accelerated rate of $\widetilde{O}(\frac{1}{\sqrt{\mu\epsilon}})$ for approximating (3) to additive accuracy ϵ via a first-order method. More specifically, the following result is based on reducing the "explicitly constrained" Sinkhorn objective (3) to a "soft constrained" regression variant of the form (4), where our box-simplex game solver is applicable. We now state our first result on improved rates for approximating Sinkhorn distance objectives.

▶ **Theorem 3** (informal, see Theorem 11 in full version). *Let $\mu \in [\Omega(\epsilon), O(\frac{\|c\|_\infty}{\log m})]$ in (3) corresponding to a complete bipartite graph with m edges. There is an algorithm based on the regularized box-simplex game solver of Theorem 25 which obtains an ϵ-approximate minimizer to (3) in time $\widetilde{O}(m \cdot \frac{\|c\|_\infty}{\sqrt{\mu\epsilon}})$.*

By leveraging the particular structure of the Sinkhorn distance and its connection to a primitive in scientific computing and theoretical computer science known as *matrix scaling* [35, 12, 2], we give a further-improved solver for (3) in Theorem 4. This solver has a nearly-linear runtime scaling as $\widetilde{O}(\frac{1}{\mu})$, which is a high-precision solver for the original Sinkhorn objective. Our high-precision Sinkhorn solver applies powerful second-order optimization tools from [12] based on the *box-constrained Newton's method* for matrix scaling, yielding our second result on improved Sinkhorn distance approximation rates.

▶ **Theorem 4** (informal, see Theorem 12 in full version). *Let $\mu, \epsilon = O(\|c\|_\infty)$ in (3) corresponding to a complete bipartite graph with m edges. There is an algorithm based on the matrix scaling solver of [12] which obtains an ϵ-approximate minimizer to (3) in time $\widetilde{O}(m \cdot \frac{\|c\|_\infty}{\mu})$.*

We present both Theorems 3 and 4 because they follow from somewhat incomparable solver frameworks. While the runtime of Theorem 3 is dominated by that of Theorem 4, it is a direct application of a more general solver (Theorem 25), which also applies to regularized regression or box-simplex objectives where the optimum does not have a characterization as a matrix scaling. Moreover, the algorithm of Theorem 4 is a second-order method which leverages recent advances in solving Laplacian systems, and hence may be less practical than its counterpart in Theorem 3. Finally, we note that due to subtle parameterization differences for our DDBM applications, the DDBM runtime attained by using our box-simplex solver within our reduction framework is more favorable on sparse graphs ($m \ll n^2$), compared to that obtained by the matrix scaling solver.

1.2 Prior work

Dynamic matching. Dynamic graph algorithms are an active area of research in the theoretical computer science, see e.g. [27, 15, 6, 31, 17, 21, 1, 26, 19, 22, 36, 20] and references therein. These algorithms have been developed under various dynamic graph models, including the additions and deletions on *vertices* or *edges*, and *oblivious adversary* model where the updates to the graph are fixed in advance (i.e. do not depend on randomness used by the algorithm), and the *adaptive adversary* model in which updates are allowed to respond to the algorithm, potentially adversarially. We focus on surveying deterministic dynamic matching algorithms with edge streams, which perform equally well under oblivious and adaptive updates; we remark the dynamic matching algorithms have also been studied under vertex addition and deletion model in [8]. For a more in-depth discussion and corresponding developments in other settings, see [45].

Many variants of the particular dynamic problem of maintaining matchings in bipartite graphs have been studied, such as the *fully dynamic* [25], *incremental* [24, 23], and *decremental* [6] cases. However, known conditional hardness results [28, 32] suggest that attaining a polylogarithmic update time for maintaining an exact fully dynamic matching may be unattainable, prompting the study of restricted variants which require maintaining an approximate matching. The works most relevant to our paper are those of [24], which provides a $\widetilde{O}(\epsilon^{-4})$ amortized update time algorithm for computing an ϵ-approximate matching for incremental bipartite matching, and [6], which achieves a similar $\widetilde{O}(\epsilon^{-4})$ update time for decremental bipartite matching. Our main DDBM results, stated in Theorems 9 and 10, improve upon [6] by roughly a factor of ϵ^{-1} in the decremental setting.

Box-simplex games. Box-simplex games, as well as ℓ_1 and ℓ_∞ regression, are equivalent to linear programs in full generality [33], have widespread utility, and hence have been studied extensively by the continuous optimization community. Here we focus on discussing *near-linear time* approximation algorithms, i.e. algorithms which run in time near-linear in the sparsity of the constraint matrix, potentially depending inverse polynomially on the desired accuracy. Interior point methods solve these problems with polylogarithmic dependence on accuracy, but are second-order and often encounter polynomial runtime overhead in the dimension (though there are exceptions, e.g. [43] and references therein).

A sequence of early works e.g. [37, 38, 39] on primal-dual optimization developed first-order methods for solving games of the form (1). These works either directly operated on the objective (1) as a minimax problem, or optimized a smooth approximation to the objective recast as a convex optimization problem. While these techniques obtained iteration complexities near-linear in the sparsity of the constraint matrix $\mathbf{A}$, they either incurred an (unaccelerated) ϵ^{-2} dependence on the accuracy ϵ, or achieved an ϵ^{-1} rate of convergence at the cost of additional dimension-dependent factors. This was due to the notorious "ℓ_∞ strong convexity barrier" (see Appendix A, [42]), which bottlenecked classical acceleration analyses over an ℓ_∞-constrained domain. [41] overcame this barrier by utilizing the primal-dual structure of (1) through a technique called "area convexity", obtaining a $\widetilde{O}(\epsilon^{-1})$-iteration algorithm. Since then, [13] demonstrated that fine-grained analyses of the classical algorithms of [37, 39] also obtain comparable rates for solving (1). Finally, we mention that area convexity has found applications in optimal transport and positive linear programming [29, 7].

Sinkhorn distances. Since [14] proposed Sinkhorn distances for machine learning applications, a flurry of work has aimed at developing algorithms with faster runtimes for (3). A line of work by [4, 16, 34] has analyzed the theoretical guarantees of the classical Sinkhorn

matrix scaling algorithm for this problem, due to the characterization of its solution as a diagonal rescaling of a fixed matrix. These algorithms obtain rates scaling roughly as $\widetilde{O}(\|c\|_\infty^2 \epsilon^{-2})$ for solving (3) to additive accuracy ϵ. Perhaps surprisingly, to our knowledge no guarantees for solving (3) which improve as the regularization parameter μ grows are currently stated in the literature, a shortcoming addressed by this work. Finally, we remark that Sinkhorn iteration has also received extensive treatment from the theoretical computer science community, e.g. [35], due to connections with algebraic complexity; see [18] for a recent overview of these connections.

2 Preliminaries

General notation. We denote $[n] := \{1, 2, \ldots, n\}$ and let $\mathbf{0}$ and $\mathbf{1}$ denote the all-0 and all-1 vectors. Given $v \in \mathbb{R}^d$, v_i or $[v]_i$ denotes the i^{th} entry of v, and for any subset $E \subseteq [d]$ we use v_E or $[v]_E$ to denote the vector in $\mathbb{R}^d$ zeroing out v on entries outside of E. We use $([v]_i)_+ = \max([v]_i, 0)$ to denote the operation truncating negative entries. We use $v \circ w$ to denote elementwise multiplication between any $v, w \in \mathbb{R}^d$. Given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we use $\mathbf{A}_{ij}$ to denote its $(i,j)^{\text{th}}$ entry, and denote its i^{th} row and j^{th} column by $\mathbf{A}_{i:}$ and $\mathbf{A}_{:j}$ respectively; its nonzero entry count is $\text{nnz}(\mathbf{A})$. We use $\mathbf{diag}\,(v)$ to denote the diagonal matrix where $[\mathbf{diag}\,(v)]_{ii} = v_i$, for each i. Given two quantities M and M', for any $c > 1$ we say M is a c-approximation to M' if it satisfies $\frac{1}{c}M' \le M \le cM'$. For $\epsilon \ll 1$, we say M is an ϵ(-multiplicative)-approximation of M' if $(1-\epsilon)M' \le M \le (1+\epsilon)M'$. Throughout the paper, we use $|\mathbf{A}|$ to denote taking the elementwise absolute value of a matrix $\mathbf{A}$, and v^2 to denote the elementwise squaring of a vector v when clear from context.

Norms. $\|\cdot\|_p$ denotes the ℓ_p norm of a vector or corresponding operator norm of a matrix. In particular, $\|\mathbf{A}\|_\infty = \max_i \|\mathbf{A}_{i:}\|_1$. We use $\|\cdot\|$ interchangeably with $\|\cdot\|_2$. We use Δ^m to denote an m-dimensional simplex, i.e. $x \in \Delta^m \iff x \in \mathbb{R}^d_{\ge 0}, \|x\|_1 = 1$.

Graphs. A graph $G = (V, E)$ has vertices V and edges E; we abbreviate $n := |V|$ and $m := |E|$ whenever the graph is clear from context. For bipartite graphs, $V = L \cup R$ denotes the bipartition. We let $\mathbf{B} \in \{0,1\}^{E \times V}$ be the (unsigned edge-vertex) incidence matrix with $\mathbf{B}_{ev} = 1$ if v is an endpoint of e and $\mathbf{B}_{ev} = 0$ otherwise.

Bregman divergence. Given any convex distance generating function (DGF) $q(x)$, we use $V^q_{x'}(x) = q(x) - q(x') - \langle \nabla q(x'), x - x' \rangle \ge 0$ as its induced Bregman divergence. When the DGF is clear from context, we abbreviate $V := V^q$. By definition, V satisfies

$$\langle -\nabla V_{x'}(x), x - u \rangle = V_{x'}(u) - V_x(u) - V_{x'}(x) \text{ for any } x, x', u. \tag{7}$$

Computational model. We use the standard word RAM model, where one can perform each basic arithmetic operations on $O(\log n)$-bit words in constant time.

3 Dynamic decremental bipartite matching

Here we provide a reduction from maintaining an approximately maximum matching in a decremental bipartite graph to solving regularized matching problems to sufficiently high precision. In Section 3.1 we give this framework and then, in Section 3.2, we provide various instantiations of our framework based on different solvers, to demonstrate its versatility.

3.1 DDBM framework

Here we provide our general framework for solving DDBM, which assumes that for bipartite $G = (V, E)$ and approximate matching value M there is a canonical regularized matching problem with properties given in Definition 5; we later provide multiple such examples. Throughout this section, $\text{MCM}(E)$ denotes the size of the maximum cardinality matching on edge set E; the vertex set V is fixed throughout, so we omit it in definitions.

▶ **Definition 5** (Canonical regularized objective). *Let $G = (V, E_0)$ be a bipartite graph and $M \geq 0$ be an 8-approximation of $\text{MCM}(E_0)$. For all $E \subseteq E_0$ with $\text{MCM}(E) \geq \frac{M}{8}$, let $f_{M,E} : \mathbb{R}^E_{\geq 0} \to \mathbb{R}$,*

$$\nu^E := \min_{x \in \mathbb{R}^E_{\geq 0}} f_{M,E}(x) \text{ , and } x^E := \text{argmin}_{x \in \mathbb{R}^E_{\geq 0}} f_{M,E}(x). \tag{8}$$

*We call the set of $\{f_{M,E}\}_{\text{MCM}(E) \geq \frac{M}{8}}$ a family of (ϵ, β)-*canonical regularized objectives *(CROs) for $G(E_0)$ and M if the following four properties hold.*

1. *For all $E \subseteq E_0$ with $\text{MCM}(E) \geq \frac{M}{8}$, $-\nu^E$ is an $\frac{\epsilon}{8}$-approximation of $\text{MCM}(E)$.*
2. *For all $E \subseteq E_0$ with $\text{MCM}(E) \geq \frac{M}{8}$, $f_{M,E}$ is equivalent to f_{M,E_0} with the extra constraint that $x_{E_0 \setminus E}$ is fixed to 0 entrywise.*
3. *For any $E' \subseteq E \subseteq E_0$ with $\text{MCM}(E) \geq \frac{M}{8}$ and $\text{MCM}(E') \geq \frac{M}{8}$,*

$$f_{M,E'}(x^{E'}) - f_{M,E}(x^E) \geq \beta V^H_{x^E}(x^{E'}) \text{ where } H(x) := \sum_e x_e \log x_e \tag{9}$$

4. *For any $x \in \mathbb{R}^E_{\geq 0}$ such that $8Mx$ is a feasible matching on (V, E),*

$$8M \|x_E\|_1 - \frac{\epsilon}{128} M \leq -f_{M,E}(x) \leq 8M \|x_E\|_1 + \frac{\epsilon}{128} M. \tag{10}$$

We further define the following notion of a canonical solver for a given CRO, which solves the CRO to sufficiently high accuracy, and rounds the approximate solution to feasibility.

▶ **Definition 6** (Canonical solver). *For (ϵ, β)-CROs $\{f_{M,E}\}_{\text{MCM}(E) \geq \frac{M}{8}}$, we call $\mathcal{A}$ an $(\epsilon, \mathcal{T})$-*canonical solver *if it has subroutines* Solve *and* Round *running in $O(\mathcal{T})$ time, satisfying:*

1. Solve *finds an approximate solution $\hat{x}^E$ of $f_{M,E}$ satisfying*

$$\left(1 + \frac{\epsilon}{8}\right) \nu^E \leq f_{M,E}(\hat{x}^E) \leq \left(1 - \frac{\epsilon}{8}\right) \nu^E. \tag{11a}$$

$$\left\|\hat{x}^E - x^E\right\|_1 \leq \frac{\epsilon}{1100}. \tag{11b}$$

2. Round *takes $\hat{x}^E$ and returns $\tilde{x}^E$ where $8M\tilde{x}^E$ is a feasible matching for $G(E)$, and:*

$$\left(1 + \frac{\epsilon}{8}\right) \nu^E \leq f_{M,E}(\tilde{x}^E) \leq \left(1 - \frac{\epsilon}{8}\right) \nu^E. \tag{12a}$$

$$\tilde{x}^E \leq \hat{x}^E \text{ monotonically.} \tag{12b}$$

Our DDBM framework, Algorithm 1, uses CRO solvers satisfying the approximation guarantees of Definition 6 to dynamically maintain an approximately maximum matching. We state its correctness and runtime in Proposition 7, and defer a proof to Appendix A.1 in the full version.

In the following, we let E_0 be the original graph's edge set, and $E_1, E_2, \ldots, E_K$ be the sequence of edge sets recomputed in Line 8, before termination for E_{K+1} on Line 4.

Algorithm 1 DecMatching$(\epsilon, G = (V, E))$.

Input: $\epsilon \in (0, \frac{1}{8})$, graph $G = (V, E)$
Parameters: Family of CROs $\{f_{M,E}\}_{E \text{ is MP}}$, $(\epsilon, \mathcal{T})$-canonical solver (Solve, Round)
1 Compute M with $\frac{1}{2}\text{MCM}(E) \le M \le \text{MCM}(E)$, via the greedy algorithm
2 $\hat{x}^E \leftarrow \mathsf{Solve}(f_{M,E})$
3 $\tilde{x}^E \leftarrow \mathsf{Round}(\hat{x}^E)$, $M_{\text{est}} \leftarrow M$
4 **while** $M_{\text{est}} > \frac{1}{4}M$ **do**
5 $E_{\text{del}} \leftarrow \emptyset$
6 **while** *edge e is deleted and* $\|\tilde{x}^E_{E_{\text{del}}}\|_1 \le \frac{\epsilon}{8}\left\|\tilde{x}^E\right\|_1$ **do**
 ▷ recompute whenever the deleted approximate matching size reaches a factor $\Theta(\epsilon)$
7 $E_{\text{del}} \leftarrow E_{\text{del}} \cup \{e\}$
8 $E \leftarrow E \setminus E_{\text{del}}$, $E_{\text{del}} \leftarrow \emptyset$
9 $\hat{x}^E \leftarrow \mathsf{Solve}(f_{M,E})$ ▷ find high-accuracy minimizer of $F_{M,E}$ satisfying (11a) and (11b)
10 $\tilde{x}^E \leftarrow \mathsf{Round}(\hat{x}^E)$ ▷ round to feasible matching satisfying (12a) and (12b)
11 Compute M_{est} with $\frac{1}{2}\text{MCM}(E) \le M_{\text{est}} \le \text{MCM}(E)$, via the greedy algorithm

▶ **Proposition 7.** *Let $\epsilon \in (0, 1)$ and $M \ge 0$. Given a family of (ϵ, β)-CROs $\{f_{M,E}\}$ for $G = (V, E_0)$, and an $(\epsilon, \mathcal{T})$-canonical solver for the family, Algorithm 1 satisfies the following.*
1. *When $M_{\text{est}} > \frac{1}{4}M$ on Line 4, where M_{est} estimates $\text{MCM}(E_k)$: at any point in the loop of Lines 6 to 7, $8M\tilde{x}^{E_k}_{E_k \setminus E_{\text{del}}}$ is an ϵ-approximate matching of $G(V, E_k \setminus E_{\text{del}})$.*
2. *When $M_{\text{est}} \le \frac{1}{4}M$ on Line 4, where M_{est} estimates $\text{MCM}(E)$: $\text{MCM}(E) \le \frac{1}{2}\text{MCM}(E_0)$.*

The runtime of the algorithm is $O(m + (\mathcal{T} + m) \cdot \frac{M}{\beta\epsilon})$.

Proof sketch. We summarize proofs of the two properties, and our overall runtime bound.

Matching approximation properties. By the greedy matching guarantee in Line 4, it holds that for any E_k (the edge set recomputed in the k^{th} iteration of Line 8 before termination), its true matching size $\text{MCM}(E_k)$ must be no smaller than $\frac{M}{4}$. Consequently, we can use the CRO family to approximate the true matching size up to $O(\epsilon)$ multiplicative factors, and by the guarantee (12a), this implies $8M\tilde{x}^{E_k}_{E_k \setminus E_{\text{del}}}$ is an $O(\epsilon)$ approximation of the true matching size. Also, our algorithm's termination condition and the guarantee on M_{est} immediately imply $\text{MCM}(E_{K+1}) \le \frac{1}{2}\text{MCM}(E_0)$.

Iteration bound. We use a potential argument. Given $E_{k+1} \subset E_k$, corresponding to consecutive edge sets requiring recomputation, we use the following inequalities:

$$
\begin{aligned}
&f_{M,E_{k+1}}\left(x^{E_{k+1}}\right) - f_{M,E_k}\left(x^{E_k}\right) \overset{(i)}{\ge} \beta V^H_{x^{E_k}}\left(x^{E_{k+1}}\right) \qquad (13)\\
&\overset{(ii)}{\ge} \beta \sum_{i \in E_{\text{del}}} \left([x^{E_{k+1}}]_i \log[x^{E_{k+1}}]_i - [x^{E_k}]_i \log[x^{E_k}]_i - \left(1 + \log[x^{E_k}]_i\right) \cdot \left([x^{E_{k+1}}]_i - [x^{E_k}]_i\right)\right)\\
&\overset{(iii)}{=} \beta \sum_{i \in E_{\text{del}}} [x^{E_k}]_i \overset{(iv)}{\ge} \beta\left(\left\|\hat{x}^{E_k}_{E_{\text{del}}}\right\|_1 - \left\|\hat{x}^{E_k} - x^{E_k}\right\|_1\right) \overset{(v)}{\ge} \beta\left(\left\|\tilde{x}^{E_k}_{E_{\text{del}}}\right\|_1 - \left\|\hat{x}^{E_k} - x^{E_k}\right\|_1\right),
\end{aligned}
$$

where (i) uses the third property in (9), (ii) uses convexity of the scalar function $c \log c$, (iii) uses that $x^{E_{k+1}}_{E_{\text{del}}}$ is 0 entrywise, (iv) uses the triangle inequality, and (v) uses the monotonicity property (12b). Moreover, between recomputations we have that the ℓ_1-norm of deleted edges satisfies $\left\|\tilde{x}^{E_k}_{E_{\text{del}}}\right\|_1 = \Omega(\epsilon)$, and our solver guarantees $\left\|\hat{x}^{E_k} - x^{E_k}\right\|_1 = O(\epsilon)$. Since the overall function decrease before termination is $O(M)$ given the stopping criterion in Line 4, the algorithm terminates after $O(\frac{M}{\beta\epsilon})$ recomputations. ◀

Using this generic DDBM framework, we obtain improved decremental matching algorithms by defining families of CROs $f_{M,E(G)}$ with associated $(\epsilon, \mathcal{T})$-canonical solvers satisfying Definition 6. In Appendix A.2, we give a regularized primal-dual construction of $f_{M,E}$, and adapt the solver of Section 4 to develop a canonical solver for the family (specifically, as the subroutine Solve). Similarly, in Appendix A.3, we show how to construct an appropriate family of $f_{M,E}$ using Sinkhorn distances, and apply the matrix scaling method presented in Appendix C.2 (based on work of [12]) to appropriately instantiate Solve.

While both algorithms, as stated, only maintain an approximate fractional matchings, this fractional matching can be rounded at any point to an explicit integral matching via e.g. the cycle-canceling procedure of Proposition 3 in [5] in time $O(m \log m)$, or dynamically (albeit at amoritized cost $\widetilde{O}(\epsilon^{-4})$ using [45]). Moreover, our algorithm based on the regularized box-simplex solver (Theorem 9) is deterministic, and both work against an adaptive adversary. Repeatedly applying Proposition 7, we obtain the following overall claim.

▶ **Corollary 8.** *Let $G = (V, E(G))$ be bipartite, and suppose for any subgraph $(V, E_0 \subseteq E(G))$, we are given a family of (ϵ, β)-CROs and an $(\epsilon, \mathcal{T})$-canonical solver for the family. There is a deterministic algorithm maintaining a fractional ϵ-approximate matching in a dynamic bipartite graph with adversarial edge deletions, running in time $O\left(m \log^3 n + (\mathcal{T} + m) \cdot \frac{M}{\beta\epsilon} \cdot \log n\right)$.*

Proof. It suffices to repeatedly apply Proposition 7 until we can safely conclude $\mathrm{MCM}(E) = 0$, which by the second property can only happen $O(\log n)$ times. ◀

3.2 DDBM solvers

In this section, we demonstrate the versatility of the DDBM framework in Section 3.1 by instantiating it with different classes of CRO families, and applying different canonical solvers on these families. By using regularized box-simplex game solvers developed in this paper (see Section 4), we give an $\widetilde{O}(m\epsilon^{-3})$ time algorithm for maintaining a ϵ-multiplicatively approximate fractional maximum matching in a m-edge bipartite graph undergoing a sequence of edge deletions, improving upon the previous best running time of $\widetilde{O}(m\epsilon^{-4})$ [6]. We also use our framework to obtain different decremental matching algorithms with runtime $\widetilde{O}(n^2\epsilon^{-3})$ and $O(m^{1+o(1)}\epsilon^{-2})$, building on recent algorithmic developments in the literature on matrix scaling. The former method uses box-constrained Newton's method solvers for matrix scaling problems in [12] (these ideas are also used in Appendix C.2), and the latter uses a recent almost-linear time high-accuracy Sinkhorn-objective solver in [11], a byproduct of their breakthrough maximum flow solver. We defer readers to corresponding sections in Appendix A for omitted proofs.

Given a bipartite graph initialized at $G = (V, E_0)$ with unsigned incidence matrix $\mathbf{B} \in \{0,1\}^{E \times V}$; we denote $n := |V|$ and $m := |E_0|$. The first family of CROs one can consider is the regularized box-simplex game objective in form:

$$\min_{(x,\xi)\in\Delta^{E+1}} \max_{y\in[0,1]^V} f_{M,E}(x,\xi,y) := -\mathbf{1}_E^\top(8Mx) - y^\top\left(8M\mathbf{B}^\top x - \mathbf{1}\right) + \gamma^{\mathsf{x}} H(x,\xi) + \gamma^{\mathsf{y}}\left(y^2\right)^\top \mathbf{B}^\top x,$$

$$\text{where } \gamma^{\mathsf{x}} = \widetilde{\Theta}\left(\epsilon M\right),\ \gamma^{\mathsf{y}} = \Theta\left(\epsilon M\right),\ \text{ and}$$

$$f_{M,E}(x) := \min_{\xi \mid (x,\xi)\in\Delta^{E+1}} \max_{y\in[0,1]^V} f_{M,E}(x,\xi,y). \tag{14}$$

We prove this is a family of $(\epsilon, \gamma^{\mathsf{x}})$-CROs (see Lemma 8 in full version). The canonical solver for this family uses RemoveOverflow (Algorithm 4, [5]) as Round and uses the regularized box-simplex games developed later in this paper (see Section 4) as Solve, which finds an ϵ-approximate solution of (14) in time $\widetilde{O}(\frac{m}{\epsilon})$. Combining all these components with the DDBM framework in Corollary 8 leads to the following DDBM solver based on regularized box-simplex games.

▶ **Theorem 9.** *Let $G = (V, E)$ be bipartite and let $\epsilon \in [\Omega(m^{-3}), 1)$. There is a deterministic algorithm for the DDBM problem which maintains an ϵ-approximate matching, based on solving regularized box-simplex games, running in time $O\left(m\epsilon^{-3}\log^5 n\right)$.*

Our second CRO family is the following regularized Sinkhorn distance objective:

$$\min_{\binom{x}{x^{\mathsf{dum}}}\in\mathbb{R}^{\widetilde{E}}_{\geq 0} \mid 2|R|\widetilde{\mathbf{B}}^\top\binom{x}{x^{\mathsf{dum}}}=d} f^{\mathsf{sink}}_{M,E}(x^{\mathsf{tot}}) := 2|R|\mathbf{1}_E^\top x + \gamma H\left(x, x^{\mathsf{dum}}\right) \text{ where } \gamma = \widetilde{\Theta}\left(\epsilon M\right),$$
$$f^{\mathsf{sink}}_{M,E}(x) := \min_{x^{\mathsf{dum}}\in\mathbb{R}^{E\setminus E_0}_{\geq 0}} f^{\mathsf{sink}}_{M,E}(x, x^{\mathsf{dum}}), \tag{15}$$

where we extend graph $G = (V, E)$ to a balanced bipartite graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ by introducing dummy vertices and edges. The extended graph allows us to write the inequality constraint $\mathbf{B}^\top x = \mathbf{1}_V$ equivalently as the linear constraint $2|R|\widetilde{\mathbf{B}}^\top\binom{x}{x^{\mathsf{dum}}} = d$ for some defined $d \in \mathbb{R}^{\widetilde{V}}$ as some properly-extended vector of $\mathbf{1}_V$. This allows us to apply known matrix scaling solver to such approximating Sinkhorn distance objective in literature.

We prove this is a family of (ϵ, γ)-CROs (see Lemma 10 in full version). The canonical solver for this family uses truncation to E as Round and uses the matrix scaling solver from [12] based on box-constrained Newton's method as Solve, which finds an ϵ-approximate solution of (15) in time $\widetilde{O}(n^2/\epsilon)$. Combining all these components with the DDBM framework in Corollary 8 leads to the following DDBM solver based on approximating Sinkhorn distances.

▶ **Theorem 10.** *Let $G = (V, E)$ be bipartite and $\epsilon \in [\Omega(m^{-3}), 1)$. There is a randomized algorithm for the DDBM problem which maintains an ϵ-approximate matching with probability $1 - n^{-\Omega(1)}$, based on matrix scaling solver of [12], running in time $\widetilde{O}\left(n^2\epsilon^{-3}\right)$.*

Alternatively, for the same (ϵ, γ)-CRO family as in (15), one can use the same Round procedure and the recent high-accuracy almost-linear time graph flow problems solver of [11] for Solve as a canonical solver. Since this new solver can find high-accuracy solutions of entropic-regularized problems of the form (15) within a runtime of $(|E_0| + O(|V|))^{1+o(1)} = m^{1+o(1)}$, this gives a third DDBM solver, which yields and improved an dependence on ϵ^{-1}.

▶ **Theorem 11.** *Let $G = (V, E)$ be bipartite and $\epsilon \in [\Omega(m^{-3}), 1)$. There is a randomized algorithm for the DDBM problem which maintains an ϵ-approximate matching with probability $1 - n^{-\Omega(1)}$, based on the Sinkhorn objective solver of [11], running in time $m^{1+o(1)}\epsilon^{-2}$.*

4 Regularized box-simplex games

In this section, we develop a high-accuracy solver for regularized box-simplex games:

$$\min_{x\in\Delta^m}\max_{y\in[0,1]^n} f_{\mu,\epsilon}(x, y) := y^\top\mathbf{A}^\top x + c^\top x - b^\top y + \mu H(x) - \frac{\epsilon}{2}\left(y^2\right)^\top|\mathbf{A}|^\top x,$$
$$\text{where } H(x) := \sum_{i\in[m]} x_i \log x_i \text{ is the standard entropic regularizer,} \tag{16}$$

where we recall absolute values and squaring act entrywise.

For ease of presentation, we make the following assumptions for some $\delta > 0$.

1. Upper bounds on entries: $\|\mathbf{A}\|_\infty \leq 1$, $\|b\|_\infty \leq B_{\max}$, $\|c\|_\infty \leq C_{\max}$. For simplicity, we assume $B_{\max} \geq C_{\max} \geq 1$; else, $C_{\max} \leftarrow \max(1, C_{\max})$ and $B_{\max} \leftarrow \max(C_{\max}, B_{\max})$.
2. Lower bounds on matrix column entries: $\max_i |\mathbf{A}_{ij}| \geq \delta$ for every $j \in [n]$.

We defer the detailed arguments of why these assumptions are without loss of generality to Appendix B in the full version. Our algorithm acts on the induced (monotone) gradient operator of the regularized box-simplex objective (16), namely $(\nabla_x f_{\mu,\epsilon}(x,y), -\nabla_y f_{\mu,\epsilon}(x,y))$, defined as

$$g_{\mu,\epsilon}(x,y) := \left(\mathbf{A}y + c + \mu(\mathbf{1} + \log(x)) - \frac{\epsilon}{2}\left|\mathbf{A}\right|(y^2), -\mathbf{A}^\top x + b + \epsilon \cdot \mathbf{diag}\,(y)\left|\mathbf{A}\right|^\top x\right). \quad (17)$$

Further, it uses the following joint (non-separable) regularizer of

$$r_{\mu,\epsilon}(x,y) := \rho \sum_{i\in[m]} x_i \log x_i + \frac{1}{\rho} x^\top \left|\mathbf{A}\right|(y^2) \quad \text{where} \quad \rho = \sqrt{\frac{2\mu}{\epsilon}}, \quad (18)$$

variants of which have been used in [41, 29, 5, 13]. When clear from context, we drop subscripts and refer to these as operator g and regularizer r. Our method is the first high-accuracy near-linear time solver for the regularized problem (16), yielding an $O(\sigma)$-approximate solution with a runtime scaling polylogarithmically in problem parameters and σ. We utilize a variant of the mirror prox [37] method for strongly monotone objectives, which appeared in [9, 13] for regularized saddle point problems with separable regularizers.

In Section 4.1, we present high-level ideas of our algorithm, which uses a mirror prox outer loop (Algorithm 2) and an alternating minimization inner loop (Algorithm 3) to implement outer loop steps; we also provide convergence guarantees. In Section 4.2, we state useful properties of the regularizer (18), and discuss a technical detail ensuring iterate stability in our method. In Section 4.3, we provide our full algorithm for regularized box-simplex games, Algorithm 4 and give guarantees in Theorem 25. Omitted proofs are in Appendix B.

4.1 Algorithmic framework

In this section, we give the algorithmic framework we use to develop our high-precision solver, which combines an outer loop inspired by mirror prox [37] with a custom inner loop for implementing each iteration. We first define an approximate solution for a proximal oracle.

▶ **Definition 12** (Approximate proximal oracle solution). *Given a convex function f over domain $\mathcal{Z}$ and $\sigma \geq 0$, we say $z' \in \mathcal{Z}$ is a* σ-approximate solution *for a proximal oracle if z' satisfies $\langle \nabla f(z'), z' - z\rangle \leq \sigma$. We denote this approximation property by $z' \leftarrow_\sigma \operatorname{argmin}_{z\in\mathcal{Z}} f$.*

We employ such approximate solutions as the proximal oracle within our "outer loop" method. Our outer loop is a variant of mirror prox (Algorithm 2) which builds upon both the mirror prox type method in [41] for solving unregularized box-simplex games and the high-accuracy mirror prox solver developed in [9, 13] for bilinear saddle-point problems on geometries admitting separable regularizers. We first give a high-level overview of the analysis, which requires bounds on two properties. First, suppose g is ν-strongly monotone with respect to regularizer r, i.e.

$$\text{for any } w, z \in \mathcal{Z},\ \langle g(w) - g(z), w - z\rangle \geq \nu \langle \nabla r(w) - \nabla r(z), w - z\rangle. \quad (19)$$

Further, suppose it is α-*relatively Lipschitz* with respect to r and Algorithm 4 (see Definition 1 of [13]), i.e. for any consecutive iterates z_{k-1}, $z_{k-1/2}$, z_k of our algorithm,[4]

$$\left\langle g(z_{k-1/2}) - g(z_{k-1}), z_{k-1/2} - z_k\right\rangle \leq \alpha\Big(V_{z_{k-1/2}}(z_k) + V_{z_{k-1}}(z_{k-1/2})\Big). \quad (20)$$

[4] This property (i.e. relative Lipschitzness restricted to iterates of the algorithm) was referred to as "local relative Lipschitzness" in [13], but we drop the term "local" for simplicity.

With these assumptions, we show that the strongly monotone mirror prox step makes progress by decreasing the divergence to optimal solution $V_{z_k}(z^\star)$ by a factor of $\frac{\alpha}{\alpha+\nu}$: this implies $\widetilde{O}(\frac{\alpha}{\nu})$ iterations suffice for finding a high-accuracy solution. We provide the formal convergence guarantee of MirrorProx in Proposition 13, which also accommodates the error of each approximate proximal step used in the algorithm. This convergence guarantee is generic and does not rely on the concrete structure of g, r in our box-simplex problem.

Algorithm 2 MirrorProx().

Input: $\frac{\sigma}{2}$-approximate proximal oracle, operator and regularizer pair (g, r) such that g is ν-strongly monotone and α-relatively Lipschitz with respect to r
Parameters: Number of iterations K

$z_0 \leftarrow \operatorname{argmin}_{z\in\mathcal{Z}} r(z)$
for $k = 1, \ldots, K$ **do**
  $z_{k-1/2} \leftarrow_{\sigma/2} \operatorname{argmin}_{z\in\mathcal{Z}} \left\{\langle g\left(z_{k-1}\right), z\rangle + \alpha V_{z_{k-1}}(z)\right\}$
  $z_k \leftarrow_{\sigma/2} \operatorname{argmin}_{z\in\mathcal{Z}} \left\{\left\langle g\left(z_{k-1/2}\right), z\right\rangle + \alpha V_{z_{k-1}}(z) + \nu V_{z_{k-1/2}}(z)\right\}$
return z_K

▶ **Proposition 13** (Convergence of Algorithm 2). *Given regularizer r with range at most Θ, suppose g is ν-strongly-monotone with respect to r (see (19)), and is α-relatively-Lipschitz with respect to r (see (20)). Let z_K be the output of Algorithm 2. Then, $V^r_{z_K}(z^\star) \le \left(\frac{\alpha}{\nu+\alpha}\right)^K \Theta + \frac{\sigma}{\nu}$.*

Given the somewhat complicated nature of our joint regularizer, we cannot solve the proximal problems required by Algorithm 2 in closed form. Instead, we implement each proximal step to the desired accuracy by using an alternating minimization scheme, similarly to the implementation of approximate proximal steps in [41, 29].

To analyze our algorithm, we use a generic progress guarantee for alternating minimization from [29] to solve each subproblem, stated below.

Algorithm 3 AltMin($\gamma^{\mathsf{x}}, \gamma^{\mathsf{y}}, \mathbf{A}, \theta, T, x_{\text{init}}, y_{\text{init}}$).

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}_{\ge 0}$, $\gamma^{\mathsf{x}} \in \mathbb{R}^m$, $\gamma^{\mathsf{y}} \in \mathbb{R}^n$, $T \in \mathbb{N}$, $\theta > 0$, $x_{\text{init}} \in \Delta^m$, $y_{\text{init}} \in [0,1]^n$
Output: Approximate minimizer to $\langle(\gamma^{\mathsf{x}}, \gamma^{\mathsf{y}}), z\rangle + \theta r(z)$ for $r(z)$ in (18)

$x^{(0)} \leftarrow x_{\text{init}}$, $y^{(0)} \leftarrow y_{\text{init}}$;
for $0 \le t \le T$ **do**
  $x^{(t+1)} \leftarrow \operatorname{argmin}_{x\in\Delta^m} \left\{\langle\gamma^{\mathsf{x}}, x\rangle + \theta r(x, y^{(t)})\right\}$;
  $y^{(t+1)} \leftarrow \operatorname{argmin}_{y\in[0,1]^n} \left\{\langle\gamma^{\mathsf{y}}, y\rangle + \theta r(x^{(t+1)}, y)\right\}$;
return $(x^{(T+1)}, y^{(T)})$

▶ **Lemma 14** (Alternating minimization progress, Lemma 5 and Lemma 7, [29]). *Let $r : \mathcal{X}\times\mathcal{Y} \to \mathbb{R}$ be jointly convex, $\theta > 0$, and γ^x and γ^y be linear operators on $\mathcal{X}, \mathcal{Y}$. Define*

$$x_{\mathsf{OPT}}, y_{\mathsf{OPT}} = \operatorname{argmin}_{x\in\mathcal{X}} \operatorname{argmin}_{y\in\mathcal{Y}} f(x,y) := \langle\gamma^x, x\rangle + \langle\gamma^y, y\rangle + \theta r(x,y). \tag{21}$$

Suppose $f(x,y)$ is twice-differentiable and satisfies: for all $x' \ge \frac{1}{2}x$ entrywise, $x', x \in \mathcal{X}$ and $y', y \in \mathcal{Y}$, $\nabla^2 f(x', y') \succeq \frac{1}{\kappa} f(x,y)$. Then the iterates of Algorithm 3 satisfy

$$f(x^{(t+2)}, y^{(t+1)}) - f(x_{\mathsf{OPT}}, y_{\mathsf{OPT}}) \le \left(1 - \frac{1}{2\kappa}\right)\left(f(x^{(t+1)}, y^{(t)}) - f(x_{\mathsf{OPT}}, y_{\mathsf{OPT}})\right).$$

Combining this lemma with the structure of our regularizer (18), we obtain the following guarantees, showing Algorithm 3 finds a $\frac{\sigma}{2}$-approximate solution to the proximal oracle.

▶ **Corollary 15** (Convergence of Algorithm 3). *Let $\delta, \sigma \in (0,1)$, $\rho \geq 1$. Suppose we are given $\gamma \in \mathcal{Z}_* = \mathcal{X}_* \times \mathcal{Y}_*$ with $\max(\|\gamma^x\|_\infty, \|\gamma^y\|_1) \leq B$, and define the proximal subproblem solution $x_{\mathsf{OPT}}, y_{\mathsf{OPT}} = \operatorname{argmin}_{x \in \Delta^m} \operatorname{argmin}_{y \in [0,1]^n} f(x,y) := \langle \gamma^x, x \rangle + \langle \gamma^y, y \rangle + \theta r(x,y)$ for some $\theta > 0$. If the Hessian condition in Lemma 14 holds with a constant $\kappa > 0$, and all simplex iterates x of Algorithm 3 satisfy $x \geq \delta$ elementwise, then the algorithm finds a $\frac{\sigma}{2}$-approximate solution to the proximal oracle within $T = O\left(\log\left(\frac{\rho(B+mn\theta)^2}{\delta\sigma\theta}\right)\right)$ iterations.*

4.2 Helper lemmas

Before providing our full method and analysis, here we list a few helper lemmas, which we rely on heavily in our later development. The first characterizes a useful property of $r_{\mu,\epsilon}$, showing that its Hessian is locally well-approximated by a diagonal matrix, which induces appropriate local norms for the blocks $x \in \mathcal{X}$, $y \in \mathcal{Y}$. We use this to prove the "strong monotonicity" (Lemma 20) and "relative Lipschitzness" (Lemma 22) bounds required in Section 4.3.

▶ **Lemma 16** (Bounds on regularizer). *Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ has $\|\mathbf{A}\|_\infty \leq 1$. For any $z = (x,y) \in \Delta^m \times [0,1]^n$, $r = r_{\mu,\epsilon}$ defined as in (18), and $\bar{x} \in \mathbb{R}^m_{>0}$, $\langle x, \mathbf{A}y \rangle \leq \|x\|_{\mathbf{diag}\left(\frac{1}{\bar{x}}\right)} \|y\|_{\mathbf{diag}\left(|\mathbf{A}|^\top \bar{x}\right)}$. Further, if $\rho \geq 3$, the matrix*

$$\mathbf{D}(x) := \begin{pmatrix} \frac{\rho}{2}\mathbf{diag}\left(\frac{1}{x}\right) & \mathbf{0} \\ \mathbf{0} & \frac{1}{\rho}\mathbf{diag}\left(|\mathbf{A}|^\top x\right) \end{pmatrix} \tag{22}$$

satisfies the following relationship with the Hessian matrix of $r(z)$:

$$\mathbf{D}(x) \preceq \nabla^2 r(z) = \begin{pmatrix} \rho \cdot \mathbf{diag}\left(\frac{1}{x}\right) & \frac{2}{\rho}\mathbf{A}\mathbf{diag}\left(y\right) \\ \frac{2}{\rho}\mathbf{diag}\left(y\right)\mathbf{A}^\top & \frac{2}{\rho}\mathbf{diag}\left(|\mathbf{A}|^\top x\right) \end{pmatrix} \preceq 4\mathbf{D}(x). \tag{23}$$

We also introduce the following notion of a padding oracle (cf. Definition 2 of [10]), which helps us control the multiplicative stability of iterates when running our algorithm.

▶ **Definition 17.** *Given $\delta > 0$, and any $\bar{z} = (\bar{x}, y) \in \Delta^m \times [0,1]^n$, a padding oracle $\mathcal{O}_\delta$ returns $z = (x,y)$ by setting $\hat{x}_i = \max(\bar{x}_i, \delta)$ coordinate-wise and letting $x = \frac{\hat{x}}{\|\hat{x}\|_1}$.*

This padding oracle has two merits which we exploit. First, the error incurred due to padding is small proportional to the padding size δ, which finds usage in proving the correctness of our main algorithm, Algorithm 4 (see Proposition 24).

▶ **Lemma 18** (Error of padding, cf. Lemma 6, [10]). *For $\delta > 0$ and $\bar{z} = (\bar{x}, y) \in \Delta^m \times [0,1]^n$ let $z = (x,y) \in \Delta^m \times [0,1]^n$ where $x = \mathcal{O}_\delta(\bar{x})$ (Definition 17), then for r in (18), and any $w \in \mathcal{Z} = \Delta^m \times [0,1]^n$, $V_z^r(w) - V_{\bar{z}}^r(w) \leq \left(\rho + \frac{8}{\rho}\right) m\delta$.*

Second, padding ensures that the iterates of our algorithm satisfy $x = \Omega(\delta)$ entrywise, i.e. no entries of our simplex iterates x are too small. This helps ensure the stability of iterates throughout one call of Algorithm 3, formally through the next lemma.

▶ **Lemma 19** (Iterate stability in Algorithm 3). *Suppose $\epsilon \leq 1$, $\rho \geq 6$, and $\alpha \geq \frac{36}{\rho}(\mu \log \frac{4}{\delta} + 3C_{\max})$. Let (x_k, y_k) denote blocks of z_k, the k^{th} iterate of Algorithm 2. In any iteration k of Algorithm 2, calling Algorithm 3 to implement Line 3, if $x_{k-1} \geq \frac{\delta}{2}$ entrywise, $x^{(t+1)} \in x_{k-1} \cdot \left[\exp\left(-\frac{1}{9}\right), \exp\left(\frac{1}{9}\right)\right]$, for all $t \in [T]$. Calling Algorithm 3 to implement Line 4, if $x_{k-1/2} \geq \frac{\delta}{4}$ entrywise, $x^{(t+1)} \in x_{k-1}^{\frac{\alpha}{\alpha+\nu}} \circ x_{k-1/2}^{\frac{\nu}{\alpha+\nu}} \cdot \left[\exp\left(-\frac{1}{9}\right), \exp\left(\frac{1}{9}\right)\right]$ for all $t \in [T]$.*

4.3 Regularized box-simplex solver and its guarantees

We give our full high-accuracy regularized box-simplex game solver as Algorithm 4, which combines Algorithm 2, Algorithm 3, and a padding step to ensure stability. For space considerations, we defer the complete statement of algorithm to the full version of the paper.

Algorithm 4 RegularizedBS$(\mathbf{A}, b, c, \epsilon, \mu, \sigma)$.

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^m$, $b \in \mathbb{R}^n$, accuracy $\sigma \in (m^{-10}, 1)$, $72\epsilon \leq \mu \leq 1$
Output: Approximate solution pair (x, y) to (16)

1 **Global:** $\delta \leftarrow \frac{\epsilon\sigma^2}{m^2}$, $\rho \leftarrow \sqrt{\frac{2\mu}{\epsilon}}$, $\nu \leftarrow \frac{1}{2}\sqrt{\frac{\mu\epsilon}{2}}$, $\alpha \leftarrow 18C_{\max} + 32\sqrt{\frac{\mu\epsilon}{2}}\log\frac{4}{\delta}$
2 **Global:** $T \leftarrow O\left(\log \frac{mnB_{\max}\alpha\rho}{\delta\sigma}\right)$, $K \leftarrow O\left(\frac{\alpha}{\nu}\log\left(\frac{\nu \log m}{\sigma}\right)\right)$ for appropriate constants
3 $(x_0, y_0) \leftarrow (\frac{1}{m} \cdot \mathbf{1}_m, \mathbf{0}_n)$
4 **for** $k = 1$ *to* K **do**
5 $\quad(\gamma^{\mathsf{x}}, \gamma^{\mathsf{y}}) \leftarrow \mathsf{GradBS}(x_{k-1}, y_{k-1}, x_{k-1}, y_{k-1}, 0)$
6 $\quad(x_{k-\frac{1}{2}}, y_{k-\frac{1}{2}}) \leftarrow \mathsf{AltminBS}(\gamma^{\mathsf{x}}, \gamma^{\mathsf{y}}, \alpha, x_{k-1}, y_{k-1})$
7 $\quad(\gamma^{\mathsf{x}}, \gamma^{\mathsf{y}}) \leftarrow \mathsf{GradBS}(x_{k-\frac{1}{2}}, y_{k-\frac{1}{2}}, x_{k-1}, y_{k-1}, \nu)$
8 $\quad(x^{(T+1)}, y^{(T)}) \leftarrow \mathsf{AltminBS}(\gamma^{\mathsf{x}}, \gamma^{\mathsf{y}}, \alpha + \nu, x_{k-\frac{1}{2}}, y_{k-\frac{1}{2}})$
9 $\quad x_k \leftarrow \frac{1}{\left\|\max\left(x^{(T+1)}, \delta\right)\right\|_1} \cdot \max\left(x^{(T+1)}, \delta\right)$, $y_k \leftarrow y^{(T)}$ ▷ Implement padding $\mathcal{O}_\delta(x^{(T+1)})$
10 **function** $\mathsf{GradBS}(x, y, x_0, y_0, \Theta)$
11 $\quad g^{\mathsf{x}} \leftarrow \mathbf{A}y + c + \mu(\mathbf{1} + \log(x)) - \frac{\epsilon}{2}|\mathbf{A}|(y^2)$, $g^{\mathsf{y}} \leftarrow -\mathbf{A}^\top x + b + \epsilon\mathbf{diag}(y)|\mathbf{A}|^\top x$
12 $\quad g_r^{\mathsf{x}} \leftarrow -\alpha\rho(1 + \log x_0) - \frac{\alpha}{\rho}|\mathbf{A}|y_0^2 - \Theta\rho(1 + \log x) - \frac{\Theta}{\rho}|\mathbf{A}|y^2$
13 $\quad g_r^{\mathsf{y}} \leftarrow -\frac{2\alpha}{\rho}\mathbf{diag}(y_0)|\mathbf{A}|^\top x_0 - \frac{2\Theta}{\rho}\mathbf{diag}(y)|\mathbf{A}|^\top x)$
14 $\quad$ **return** $(g^{\mathsf{x}} + g_r^{\mathsf{x}}, g^{\mathsf{y}} + g_r^{\mathsf{y}})$
15 **function** $\mathsf{AltminBS}(\gamma^{\mathsf{x}}, \gamma^{\mathsf{y}}, \theta, x^{(0)}, y^{(0)})$ ▷ Implement approximate proximal oracle via AltMin
16 $\quad$ **for** $0 \leq t \leq T$ **do**
17 $\quad\quad x^{(t+1)} \leftarrow \frac{1}{\left\|\exp\left(-\frac{1}{\theta\rho}\gamma^{\mathsf{x}} - \frac{1}{\rho^2}|\mathbf{A}|\left(y^{(t)}\right)^2\right)\right\|_1} \cdot \exp\left(-\frac{1}{\theta\rho}\gamma^{\mathsf{x}} - \frac{1}{\rho^2}|\mathbf{A}|\left(y^{(t)}\right)^2\right)$
18 $\quad\quad y^{(t+1)} \leftarrow \mathrm{med}\left(0, 1, -\frac{\rho}{2\theta} \cdot \frac{\gamma^{\mathsf{y}}}{|\mathbf{A}|^\top x^{(t+1)}}\right)$
19 $\quad$ **return** $(x^{(T+1)}, y^{(T)})$

In order to analyze the convergence of Algorithm 4, we begin by observing that the operator in (17) satisfies strong monotonicity with respect to our regularizer (18).

▶ **Lemma 20** (Strong monotonicity). *Let $\mu \geq \frac{\epsilon}{2}$ and $\rho := \sqrt{\frac{2\mu}{\epsilon}}$. The gradient operator $g_{\mu,\epsilon}$ (17) is $\nu := \frac{1}{2}\sqrt{\frac{\mu\epsilon}{2}}$-strongly monotone (see (19)) with respect to $r_{\mu,\epsilon}$ defined in (18).*

Next, we show iterate stability through each loop of alternating minimization (i.e. from Line 5 to Line 6, and Line 7 to Line 8 respectively), via Lemma 19.

▶ **Corollary 21** (Iterate stability in Algorithm 4). *Assume the same parameter bounds as Lemma 19, and that $\delta \in (0, m^{-1})$. In the k^{th} outer loop of Algorithm 4, $x_{k-1} \geq \frac{\delta}{2}$ entrywise. Further, for all iterates $x^{(t+1)}$ computed in Line 5 to Line 6 and x_{OPT} as defined in (21) with $\theta = \alpha$, $\frac{1}{2}x_{k-1} \leq x^{(t+1)}, x_{\mathsf{OPT}} \leq 2x_{k-1}$, and $x^{(t+1)}, x_{\mathsf{OPT}} \geq \frac{\delta}{4}$, entrywise. Similarly, for all iterates $x^{(t+1)}$ computed in Line 7 to Line 8 and x_{OPT} as defined in (21) with $\theta = \alpha + \nu$, $\frac{1}{2}x_{k-1/2} \leq x^{(t+1)}, x_{\mathsf{OPT}} \leq 2x_{k-1/2}$ and $x^{(t+1)}, x_{\mathsf{OPT}} \geq \frac{\delta}{4}$, entrywise.*

Under iterate stability, our next step is to prove that our operator $g_{\mu,\epsilon}$ is relatively Lipschitz with respect to our regularizer $r_{\mu,\epsilon}$ (as defined in (20)).

▶ **Lemma 22** (Relative Lipschitzness). *Assume the same parameter bounds as in Lemma 19. In the k^{th} outer loop of Algorithm 4, let $\bar{z}_k \leftarrow (x^{(T+1)}, y^{(T)})$ from Line 8 be z_k before the padding operation. Then, $x_{k-1/2}, \bar{x}_k \in [\frac{1}{2}x_{k-1}, 2x_{k-1}]$ elementwise and*

$$\left\langle g(z_{k-1/2}) - g(z_{k-1}), z_{k-1/2} - \bar{z}_k \right\rangle \leq \alpha \left(V_{z_{k-1}}(z_{k-1/2}) + V_{z_{k-1/2}}(\bar{z}_k) \right) \quad \text{for } \alpha = 4 + 32\sqrt{\frac{\mu\epsilon}{2}}.$$

Next, we give a convergence guarantee on the inner loops (from Line 5 to Line 6, and Line 7 to Line 8) in Algorithm 4, as an immediate consequence of Corollary 15.

▶ **Corollary 23** (Inner loop convergence in Algorithm 4). *Assume the same parameter bounds as in Lemma 19. For γ defined in Line 5, suppose for an appropriate constant $T = \Omega\left(\log \frac{mnB_{\max}\alpha\rho}{\delta\sigma}\right)$. Then, for all k iterate $z_{k-1/2} = (x_{k-1/2}, y_{k-1/2})$ of Line 6 satisfies*

$$\left\langle \nabla g(z_{k-1}) + \alpha \nabla V_{z_{k-1}}(z_{k-1/2}), z_{k-1/2} - w \right\rangle \leq \frac{\nu\sigma}{4}, \text{ for all } w \in \mathcal{Z}.$$

Similarly, for γ defined in Line 7, iterate $\bar{z}_k = (x_{(T+1)}, y_{(T)})$ of Line 8 satisfies

$$\left\langle \nabla g(z_{k-1}) + \alpha \nabla V_{z_{k-1}}(\bar{z}_k) + \nu \nabla V_{z_{k-1/2}}(\bar{z}_k), \bar{z}_k - w \right\rangle \leq \frac{\nu\sigma}{4}, \text{ for all } w \in \mathcal{Z}.$$

We now analyze the progress made by each outer loop of Algorithm 4. The proof is very similar to that of Proposition 13; the only difference is controlling the extra error incurred in the padding step of Line 9, which we bound via Lemma 18.

▶ **Proposition 24** (Convergence of Algorithm 4). *Assume the same parameter bounds as in Lemma 19, and that $\delta \leq \frac{\sigma}{4\rho\alpha m}$. Algorithm 4 returns z_K satisfying $V^r_{z_K}(z^\star) \leq \frac{3\sigma}{\nu}$, letting (for an appropriate constant) $K = \Omega(\frac{\alpha}{\nu}\log(\frac{\nu \log m}{\sigma}))$.*

We are now ready to prove the main theorem of this section, which gives a complete convergence guarantee of Algorithm 4 by combining our previous claims.

▶ **Theorem 25** (Regularized box-simplex solver). *Given regularized box-simplex game (16) with $72\epsilon \leq \mu \leq 1$ and optimizer $(x^\star, y^\star)$, and letting $\sigma \in (m^{-10}, 1)$,* RegularizedBS *(Algorithm 4) returns x^K satisfying $\left\|x^K - x^\star\right\|_1 \leq \frac{\sigma}{C_{\max}\log^2 m}$ and $\max_{y\in[0,1]^n} f_{\mu,\epsilon}(x^K, y) - f_{\mu,\epsilon}(x^\star, y^\star) \leq \sigma$. The total runtime of the algorithm is $O(\text{nnz}(\mathbf{A}) \cdot (\frac{C_{\max}}{\sqrt{\mu\epsilon}} + \log(\frac{m}{\sigma\epsilon})) \cdot \log(\frac{C_{\max}\log m}{\sigma}) \log(\frac{mnB_{\max}}{\sigma}))$.*

As a corollary, we obtain an approximate solver for regularized box-simplex games in the following form (which in particular does not include a quadratic regularizer):

$$\min_{x\in\Delta^m} \max_{y\in[0,1]^n} f_\mu(x,y) = y^\top \mathbf{A}^\top x + c^\top x - b^\top y + \mu H(x), \text{ where } H(x) := \sum_{i\in[m]} x_i \log x_i. \quad (24)$$

▶ **Corollary 26** (Half-regularized approximate solver). *Given regularized box-simplex game (24) with regularization parameters $72\epsilon \leq \mu \leq 1$ and optimizer $(x^\star, y^\star)$, and letting $\epsilon \in (m^{-10}, 1)$, Algorithm 4 with $\sigma \leftarrow \frac{\epsilon}{2}$ returns x^K satisfying $\max_{y\in[0,1]^n} f_\mu(x^K, y) - f_\mu(x^\star, y^\star) \leq \epsilon$. The total runtime of the algorithm is $O\left(\text{nnz}(\mathbf{A}) \cdot (\frac{C_{\max}}{\sqrt{\mu\epsilon}} + \log\left(\frac{m}{\epsilon}\right)) \cdot \log\left(\frac{C_{\max}\log m}{\epsilon}\right) \log\left(\frac{mnB_{\max}}{\epsilon}\right)\right)$.*

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 335–344. IEEE Computer Society, 2016.

2 Zeyuan Allen-Zhu, Yuanzhi Li, Rafael Mendes de Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 890–901, 2017.

3 Jason Altschuler, Francis Bach, Alessandro Rudi, and Jonathan Niles-Weed. Massively scalable sinkhorn distances via the nyström method. *Advances in neural information processing systems*, 32, 2019.

4 Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1964–1974, 2017.

5 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Alexandria, VA, USA, January 9 - 12, 2022*, 2022.

6 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1123–1134, 2020.

7 Digvijay Boob, Saurabh Sawlani, and Di Wang. Faster width-dependent algorithm for mixed packing and covering lps. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15253–15262, 2019.

8 Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 384–393. IEEE, 2014.

9 Yair Carmon, Yujia Jin, Aaron Sidford, and Kevin Tian. Variance reduction for matrix games. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11377–11388, 2019.

10 Yair Carmon, Yujia Jin, Aaron Sidford, and Kevin Tian. Coordinate methods for matrix games. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 283–293, 2020. doi:10.1109/FOCS46700.2020.00035.

11 Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *arXiv preprint*, 2022. arXiv:2203.00671.

12 Michael B Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix scaling and balancing via box constrained newton's method and interior point methods. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 902–913. IEEE, 2017.

13 Michael B. Cohen, Aaron Sidford, and Kevin Tian. Relative lipschitzness in extragradient methods and a direct recipe for acceleration. In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, pages 62:1–62:18, 2021.

14 Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2292–2300, 2013.

15 David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 914–925. ACM, 2019.

16 Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1366–1375, 2018.

17 Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. *CoRR*, abs/2101.07233, 2021. `arXiv:2101.07233`.

18 Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. Operator scaling: Theory and applications. *Found. Comput. Math.*, 20(2):223–290, 2020.

19 Gramoz Goranci. Dynamic graph algorithms and graph sparsification: New techniques and connections. *CoRR*, abs/1909.06413, 2019. `arXiv:1909.06413`.

20 Gramoz Goranci, Monika Henzinger, and Pan Peng. The power of vertex sparsifiers in dynamic graph algorithms. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

21 Gramoz Goranci, Monika Henzinger, and Pan Peng. Dynamic Effective Resistances and Approximate Schur Complement on Separable Graphs. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

22 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2212–2228. SIAM, 2021.

23 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1886–1898, 2019.

24 Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 227–239, 2014.

25 Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557, 2013.

26 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2562–2574. SIAM, 2020.

27 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms. *CoRR*, abs/2102.11169, 2021. `arXiv:2102.11169`.

28 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30, 2015.

29 Arun Jambulapati, Aaron Sidford, and Kevin Tian. A direct $\tilde{O}(1/\epsilon)$ iteration parallel algorithm for optimal transport. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11355–11366, 2019.

30 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226, 2014.

31 Peter Kiss. Improving update times of dynamic matching algorithms from amortized to worst case. *CoRR*, abs/2108.10461, 2021. `arXiv:2108.10461`.

32 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287, 2016.

33 Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 230–249, 2015.

34 Tianyi Lin, Nhat Ho, and Michael I. Jordan. On efficient optimal transport: An analysis of greedy and accelerated mirror descent algorithms. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 3982–3991, 2019.

35 Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 644–652, 1998.

36 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961. IEEE Computer Society, 2017.

37 Arkadi Nemirovski. Prox-method with rate of convergence O(1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

38 Yurii Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005.

39 Yurii Nesterov. Dual extrapolation and its applications to solving variational inequalities and related problems. *Math. Program.*, 109(2-3):319–344, 2007.

40 Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 263–269, 2013.

41 Jonah Sherman. Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460. ACM, 2017.

42 Aaron Sidford and Kevin Tian. Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science*, pages 922–933. IEEE, 2018.

43 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 859–869, 2021.

44 François-Xavier Vialard. An elementary introduction to entropic regularization and proximal methods for numerical optimal transport, 2019.

45 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 194–207, 2020.

A PTAS for Packing Hypercubes into a Knapsack

Klaus Jansen
Universität Kiel, Germany

Arindam Khan
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Marvin Lira
Universität Kiel, Germany

K. V. N. Sreenivas
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Abstract

We study the d-dimensional hypercube knapsack problem (d-D Hc-Knapsack) where we are given a set of d-dimensional hypercubes with associated profits, and a knapsack which is a unit d-dimensional hypercube. The goal is to find an axis-aligned non-overlapping packing of a subset of hypercubes such that the profit of the packed hypercubes is maximized. For this problem, Harren (ICALP'06) gave an algorithm with an approximation ratio of $(1 + 1/2^d + \varepsilon)$. For $d = 2$, Jansen and Solis-Oba (IPCO'08) showed that the problem admits a polynomial-time approximation scheme (PTAS); Heydrich and Wiese (SODA'17) further improved the running time and gave an efficient polynomial-time approximation scheme (EPTAS). Both the results use structural properties of 2-D packing, which do not generalize to higher dimensions. For $d > 2$, it remains open to obtain a PTAS, and in fact, there has been no improvement since Harren's result.

We settle the problem by providing a PTAS. Our main technical contribution is a structural lemma which shows that any packing of hypercubes can be converted into another *structured* packing such that a high profitable subset of hypercubes is packed into a constant number of *special* hypercuboids, called $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes. As a side result, we give an almost optimal algorithm for a variant of the strip packing problem in higher dimensions. This might have applications for other multidimensional geometric packing problems.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Multidimensional knapsack, geometric packing, cube packing, strip packing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.78

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2202.11902

Funding *Klaus Jansen*: Research supported by German Research Foundation (DFG), project JA 612/25-1.
Arindam Khan: Research partly supported by Pratiksha Trust Young Investigator Award, Google India Research Award, and Google ExploreCS Award.
Marvin Lira: Research supported by German Research Foundation (DFG), project JA 612/25-1.

Acknowledgements We thank Roberto Solis-Oba and three anonymous reviewers for their comments and suggestions on this paper.

1 Introduction

Multidimensional geometric packing problems are well-studied natural generalizations of the classical knapsack and bin packing problems. In the d-dimensional geometric knapsack problem (d-D Gen-Knapsack), where d is a fixed constant parameter, we are given a set of n items $\mathcal{I} := \{1, 2, \ldots, n\}$. Each item $i \in [n]$ is a d-dimensional (d-D) hypercuboid with

side length $s_k(i) \in (0, 1]$ along the k^{th} dimension and profit $p(i) \in \mathbb{Q}_{>0}$. The goal is to pack a subset of hypercuboids into a d-D unit knapsack (i.e., $[0, 1]^d$) such that the profit is maximized. We need the packing of the hypercuboids to be axis-aligned and non-overlapping. In this paper, we study d-dimensional hypercube knapsack (d-D Hc-Knapsack), a special case of d-D Gen-Knapsack, where all the items are hypercubes, i.e., for all $i \in [n]$, the i^{th} hypercuboid is a hypercube of side length $s(i)$.

d-D Gen-Knapsack generalizes the classical (1-D) knapsack problem [35] and thus is NP-hard. It finds numerous applications in scheduling, ad placement, and cutting stock [18]. For 2-D Gen-Knapsack, Jansen and Zhang [27] gave a $(2 + \varepsilon)$-approximation algorithm. Gálvez et al. [18] gave a 1.89-approximation and $(3/2 + \varepsilon)$-approximation for the cardinality version (i.e., all items have the same profit). Adamaszek and Wiese [3] gave a QPTAS when the input size is quasi-polynomially bounded. Gálvez et al. [19] later gave a pseudo-polynomial time $(4/3 + \varepsilon)$-approximation. For 3-D Gen-Knapsack, the present best approximation ratio is $7 + \varepsilon$ [14]. For $d \geq 4$, Sharma [41] has given a $(1 + \varepsilon)3^d$-approximation algorithm. Interestingly for $d \geq 2$, unlike d-D Gen-Knapsack, d-D Hc-Knapsack is not a generalization of 1-D knapsack. Leung et al. [36] showed 2-D Hc-Knapsack is strongly NP-hard, using a reduction from the 3-partition problem. The NP-hardness status of d-D Hc-Knapsack for $d > 2$, was open for a long time. Recently, Lu et al. [37, 38] settled the status by showing that d-D Hc-Knapsack is also NP-hard for $d > 2$.

A related problem is d-D Hc-Binpacking where we are given d-D hypercubes and the goal is to pack them into the minimum number of unit hypercubes. Back in 2006, Bansal et al. [5] gave an APTAS for this problem. Their algorithm starts by classifying the input set into small, medium, and large items. They first pack the large items ($O(1)$ in number) near-optimally by brute force and then pack the small items using Next Fit Decreasing Height (NFDH) [11] in the gaps left in between the large items. The remaining unpacked (small and medium) items are packed into additional bins using NFDH. However, this constructive approach can not be used to devise a PTAS for d-D Hc-Knapsack. Specifically, after packing the large items, the total space left to pack the small items can be very small and this space can be fragmented among several voids in between the large items. This renders packing the small items difficult, especially if the small items occupy a small volume in the optimal solution and carry a lot of profit. This turns out to be a bottleneck case. Otherwise, if the volume of small items in an optimal solution is significant, or if their profit is not significant, or if the empty space in the optimal solution is significant, we can easily devise a PTAS. In fact, the algorithm in [5] can be adapted to devise PTASes for the special cases of d-D Hc-Knapsack: (i) cardinality case, and (ii) when each item has the profit:volume ratio in the range $[1, r]$ for a fixed constant r. However, the case of arbitrary profits remains very difficult to handle.

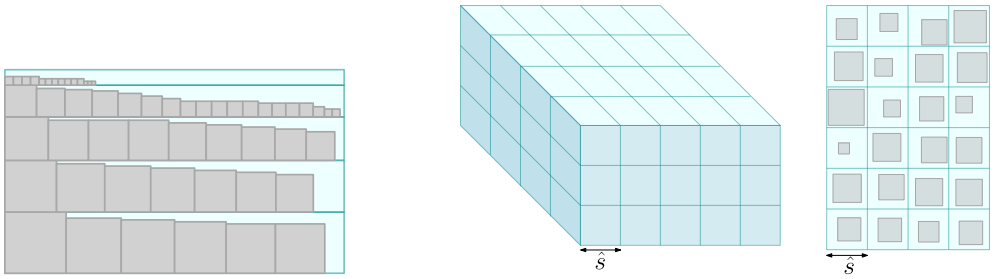
For d-D Hc-Knapsack, Harren [21] gave a $(1 + 1/2^d + \varepsilon)$-approximation algorithm, by removing a least profitable large item and using the empty space to pack the small items. Interestingly, the approximation ratio gets smaller as d grows. He also studied d-D Hc-StripPacking, where, given a set of hypercubes and a strip with a $(d - 1)$-dimensional base and unbounded height, the goal is to pack all the hypercubes while minimizing the height. He gave an APTAS for the special case of d-D Hc-StripPacking when the ratio between the shortest and longest sides of the base is bounded by a constant.

Jansen and Solis-Oba [26] gave a PTAS for 2-D Hc-Knapsack, by overcoming the above-mentioned bottleneck. They consider an optimal packing and categorize the packed items into *small, medium,* and *large* items. Then, by extending the edges of the large items, the remaining space is divided into $O(1)$ number of rectilinear regions, where each region

is classified as either *large* or *elongated* or *small*, based on the largest item intersecting or completely contained in it. Then, to make sure that a region has no items partially intersecting it, they repack all the items as follows. The *large* region is rectangular and its dimensions are much larger compared to the largest item intersecting it. Here NFDH can repack a high profitable subset of the items. An *elongated* region is again rectangular and it has one dimension much longer than the other. Here an algorithm for strip packing [28] is used to repack the items. Finally, the *small* regions are handled by applying the above transformations recursively as they can have complicated shapes (they may not be rectangular). Recently, Heydrich and Wiese [23] gave an EPTAS, effectively achieving the best-possible approximation. However, a PTAS for d-D Hc-Knapsack for $d > 2$ remains elusive as it is difficult to find such transformations for higher dimensions. Even for 3-D, the best-known approximation ratio remains $9/8$ [21]. In 2-D, many structural theorems [4, 18] show that a near-optimal solution exists where all items are packed into $O(1)$-number of rectangular regions where items are either packed as a stack or packed using NFDH. But these results do not generalize to higher dimensions. E.g., while extending the approach of *large* and *elongated* blocks of [26] to $d > 2$, we may not obtain a near-optimal structure packed in d-D hypercuboids; rather, we might obtain complicated rectilinear regions. However, prior to our work, there exist no algorithms which considered packing in such rectilinear regions.

1.1 Our Contributions

For the d-D Hc-Knapsack problem, we design a PTAS, thus settling the problem. Our main structural result intuitively says that any packing, by incurring a loss of only ε-fraction of the profit, can be transformed into $O(1)$ number of special hypercuboidal regions such that each region either contains a single large item, or contains items very small compared to its dimensions ($\mathcal{V}$-Box, see Figure 1a), or contains items placed along a multidimensional grid ($\mathcal{N}$-Box, see Figure 1b). We then provide an algorithm that guesses the arrangement of these hypercuboids and uses results of [26, 35] to find a near-optimal packing of items in these special boxes.



(a) An $\mathcal{V}$-Box is just a huge cuboid when compared to its size parameter $\hat{s}$. Each item has side length at most $\hat{s}$ and the item set assigned to it can be packed by NFDH.

(b) An $\mathcal{N}$-Box is a multidimensional grid. An item set assigned to an $\mathcal{N}$-Box can be packed by placing each item in a cell.

Figure 1 $\mathcal{V}$-Box and $\mathcal{N}$-Box.

For our structural theorem, we start with an optimal solution. First, we divide the items in the optimal packing into large, medium, and small items such that the medium items have a tiny profit and can be discarded. Then, by extending the facets of the large items, we create a non-uniform grid with $O(1)$ hypercuboidal cells. However, some items may intersect the facets of these cells. We consider the facets of all the cells and identify (based on the intersecting items) each of the facets as either *good* or *bad*. Intuitively, if we only have good facets then we can repack items so that no items intersect the facets. To get rid of the bad

facets, we merge some cells so that they form composite cells, which we call *collections* (See Figure 2). Now we need to repack the items in the *collections* which may have complex rectilinear shapes. If the collection has $k \in \{0, 1, \ldots, d\}$ long dimensions then we call it a k-elongated collection. Note that we can have $d + 1$ types of collections.

Our structural theorem can be viewed as a generalization of [26, 21]; however, due to the complicated shape of collections, we had to overcome several technical obstacles in the process. For simplicity, we explain the intuition for $d = 3$, where we obtain a packing of items into four types of rectilinear regions (collections) as in Figure 2.

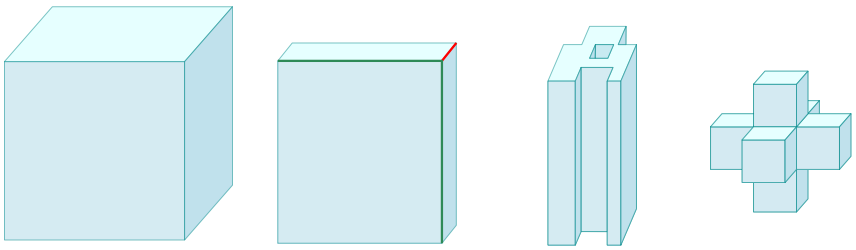


Figure 2 Repacking items inside the left-most collection is easy (using NFDH). The right-most collections are handled recursively. However, repacking the items in the other two collections is challenging (we needed a novel strip packing algorithm).

The first collection (3-elongated) is a $\mathcal{V}$-Box, i.e., all its three dimensions are long compared to the largest item packed in it. The simple NFDH algorithm suffices to repack the items inside it while losing only a few items with a small profit. Although repacking in large collections can be easily handled by NFDH, there exist no previous algorithms to repack items in the remaining types of collections. Now for $k \in [d-1]$, collections are long in at least one dimension and thus can be viewed as a type of strip (though the base may not be rectangular). Harren [21] showed how to repack items almost optimally in a d-dimensional strip with a rectangular base with a bounded aspect ratio. However, their approach doesn't trivially extend to the 1- and 2-elongated collections in Figure 2, as the base can be non-rectangular or it may not satisfy the property of bounded aspect ratio even though it is rectangular. In this paper, we circumvent this bottleneck by designing a PTAS (under resource augmentation) for d-dimensional strip packing to pack items on such complicated bases. Now we explain how we use the strip packing algorithm to pack these collections.

In the second collection (2-elongated), one dimension (marked in red) is reasonably short compared to the largest item intersecting it and the other dimensions (in green) are long. We first divide the items in the collection into large, medium, and small items and ensure that the total volume of medium items is marginal. We then consider the large items and round their sizes to up to $O(1)$ types using linear grouping [28]. The number of (rounded) large items that can fit on along the short dimension is $O(1)$. So, similar to [28], we solve an LP that represents the fractional strip packing with (rounded) large items where the items are allowed to be sliced along the long dimensions. We then convert this into an integral packing of large items using resource augmentation along the long dimensions. Then we pack as many small items as possible in the gaps left. The rest of the small items, together with the medium items, are packed on the top.

The third collection (1-elongated), though complicated, is ensured (by our construction) to have a base with a bounded aspect ratio[1]. Moreover, it can be viewed as a union of $O(1)$ number of smaller disjoint rectangles (which we call *base cells*). We again classify each item

[1] More accurately, the minimal cuboid, which completely contains the base, has bounded-aspect ratio.

in this collection as large, medium, or small. As above, we round the large items to $O(1)$ types. Note that the number of large items that can fit on the base of the strip is only a constant (due to the bounded aspect ratio). We pack them integrally by first solving the fractional LP where items are allowed to be sliced along the long dimension. Our main novelty, in this case, is the way in which we pack the small items on the top of the strip. Unlike [21], we can't use NFDH directly to pack the small items as the base isn't rectangular. Instead, we first merge a few base cells to form a larger base cell which is rectangular (we show that this is indeed possible) and is large enough to accommodate the largest item (and thus any item). Now, having at least one large base cell, we distribute the small and medium items among all the base cells so that when they are packed using NFDH in strips on these base cells, the maximum height is minimized. Observe that if we pack the items on a single base cell, we may not efficiently use the entire area of the base and this can lead to a very tall strip. Thus this process of distributing the items among all the base cells is important.

We use our near-optimal strip packing algorithm on these 1- or 2-elongated collections to repack the items while losing only a small profit. Interestingly, the strip packing algorithm for both kinds of bases (in general, for any k-elongated collection where $k \in [d-1]$) is the same; the only change is the number of short dimensions of the base that is given as a parameter to the algorithm. We provide the algorithm for d-dimensional strip packing in the full version [24].

Finally, for 0-elongated collections, we apply the above process recursively. Using a shifting argumentation, we show that $O_\varepsilon(1)$ steps of recursion is sufficient. We also show that the resulting packing is a packing into $O(1)$ number of $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes. With more technical ingenuity, these ideas can be extended to the case of $d > 3$.

We believe that our techniques will be helpful for other multidimensional geometric packing problems involving rectilinear regions that are not hypercuboids.

1.2 Related Work

For geometric bin packing, Caprara gave a 1.691^{d-1} asymptotic approximation [9]. For $d = 2$, it admits no APTAS [5] and there is a 1.406-approximation algorithm [7]. There are several other variants of bin packing such as vector packing [6, 40], strip packing [22, 16, 25], sliced packing [13, 17], mixed packing [34], etc. The knapsack problem is also studied under several generalizations such as vector knapsack [15], fair knapsack [39], and mixed knapsack [33]. Packing problems are also well-studied under guillotine cut constraints [8, 30, 31, 1, 32]. Guillotine cut also has interesting connections with the maximum independent set of rectangles problem [2, 20]. We refer the readers to [10, 29] for a survey on multidimensional packing.

1.3 Overview of the Paper

In Section 2, we present some preliminaries. Section 3 contains the result for d-D Hc-Knapsack. The result for strip packing, which will be used as a subroutine to obtain our main structural result is given in Section 3.1.2. Due to space limitations, many proofs had to be moved to the full version [24].

2 Notations and Preliminaries

Let $[n] := \{1, 2, \ldots, n\}$. For any set of items $\mathcal{I}$, we define $\hat{s}(\mathcal{I}) := \max_{i \in \mathcal{I}} s(i)$. For any k-dimensional region R, we denote its volume by $\mathrm{VOL}_k(R)$. We extend this notation to an item i or a set of items $\mathcal{I}$, i.e., $\mathrm{VOL}_d(i) = (s(i))^d$, and $\mathrm{VOL}_d(\mathcal{I}) = \sum_{i \in \mathcal{I}} \mathrm{VOL}_d(i)$. Also, the profit of a packing is the sum of the profits of the packed items $\mathcal{Q}$: $p(\mathcal{Q}) := \sum_{i \in \mathcal{Q}} p(i)$.

Consider a set of points X in $[0,1]^d$ where each point $x \in X$ can be represented as $(x_1, x_2, \ldots, x_d)$. For any set of dimensions $\mathcal{D} \subseteq [d]$, we define the projection of X onto $\mathcal{D}$ as the set $\left\{ \left(x_{d_1}, \ldots, x_{d_{|\mathcal{D}|}}\right) \mid (x_1, \ldots, x_k) \in X \right\}$, where $\left\{ d_1, \ldots, d_{|\mathcal{D}|} \right\} = \mathcal{D}$ and $d_i < d_{i+1}$ for all $i \in [|\mathcal{D}| - 1]$.

We only consider axis-aligned packing of hypercuboids and hypercubes. Thus, a d-D hypercuboid C is given by the position of its lower corner $(x_1, \ldots, x_d) \in \mathbb{R}^d$ and side lengths $\ell_1, \ldots, \ell_d \in \mathbb{R}_+$. Then, C is the cartesian product of the intervals $\prod_{i=1}^{d} [x_i, x_i + \ell_i]$, and $\mathrm{VOL}_d(C) := \prod_{i=1}^{d} \ell_i$ and its surface area is defined as the sum of the volumes of its $(d-1)$-dimensional facets, i.e., $\mathrm{SURF}_d(C) := 2\sum_{i=1}^{d} \prod_{j=1, j\neq i}^{d} \ell_j = 2\sum_{i=1}^{d} \mathrm{VOL}_d(C)/\ell_i$. A packing is a subset $\mathcal{Q} \subseteq \mathcal{I}$ of items with positions $\mathrm{pos} : \mathcal{Q} \longrightarrow [0,1]^d$. It is valid, if each item $i \in \mathcal{Q}$ with the lower corner positioned at $\mathrm{pos}(i)$ is completely included in the unit hypercube $[0,1]^d$ and each pair of items $i, j \in \mathcal{Q}$ is positioned non-overlapping.

Consider a set of items lying in d dimensional space. For any region in the d dimensional space, we define the profit of that region as the profit of the set of items *completely* contained in that region.

We will use the higher dimensional variant of the well-known Next Fit Decreasing Height (NFDH) algorithm extensively. For the sake of completeness, we will provide a brief description of the algorithm here.

NFDH is a two dimensional packing algorithm introduced in [11] for packing a given set of rectangles in a bigger rectangular region $\mathcal{R}$. Informally, it works as follows. First, it sorts the given set of rectangles in decreasing order of heights (the vertical dimension). Then, it starts off by packing the rectangles on the base of $\mathcal{R}$ side-by-side as much as possible. Then, this level is closed, i.e., the base of $\mathcal{R}$ is shifted vertically to the top of the first rectangle packed. In the next step, the remaining rectangles are packed on the new base to whatever extent is possible. This process continues.

One can easily extend the NFDH algorithm to pack hypercuboids into a bigger hypercuboidal region. Let $k \in \mathbb{N}_{\geq 2}$. For any k-dimensional hypercuboid x, let $x^{(i)}$ denote the i-dimensional hypercuboid obtained by considering only the first i dimensions of x. We can extend this notation to sets: Let J be a set of k-dimensional hypercuboids. Then $J^{(i)} = \{x^{(i)} | x \in J\}$.

We define the NFDH algorithm in k dimensions (k-NFDH) recursively as follows. If $k = 2$, then the algorithm is simply NFDH as described above. Suppose $k > 2$. Let S be a set of k-dimensional hypercuboids to be packed into a k-dimensional hypercuboidal region $\mathcal{R}$. We first sort the rectangles in decreasing order of their lengths in the k^{th} dimension. Then we pick the largest possible prefix P of S such that $P^{(k-1)}$ can be completely packed on the base of $\mathcal{R}$ using $(k-1)$-NFDH. We pack P on the base of $\mathcal{R}$, and shift the base to the top of the tallest (longest in the k^{th} dimension) item in P. We repeat this process with $S \backslash P$.

The pseudocode of k-NFDH and some illustrations have been provided in the full version [24]. We will abbreviate k-NFDH by just NFDH. The value of k will be clear from the context. Harren [21] gave a surface area based efficiency guarantee of NFDH algorithm for hypercubes.

▶ **Lemma 1** ([21]). *Consider a set S of hypercubes with side lengths at most δ and a hypercuboid C. The NFDH algorithm either packs all items from S into C or the total volume left free inside of C is at most $\delta\, \mathrm{SURF}_d(C)/2$.*

Now we define $\mathcal{V}$-Box and $\mathcal{N}$-Box.

▶ **Definition 2** ($\mathcal{V}$-Box and $\mathcal{N}$-Box). *Let B be a d-dimensional hypercuboid, $\mathcal{Q}$ be a set of items packed in it, and let $\hat{s}$ be an upper bound on the side lengths of the items in $\mathcal{Q}$. We say that B is a $\mathcal{V}$-Box if its side lengths can be written as $n_1\hat{s}, n_2\hat{s}, \ldots, n_d\hat{s}$ where $n_1, n_2, \ldots, n_d \in \mathbb{N}_+$*

and the volume of the packed items is at most $\mathrm{VOL}_d(B) - \hat{s}\frac{\mathrm{SURF}_d(B)}{2}$. *We say that* B *is an* $\mathcal{N}$*-Box if its side lengths can be written as* $n_1\hat{s}, n_2\hat{s}, \ldots, n_d\hat{s}$ *where* $n_1, n_2, \ldots, n_d \in \mathbb{N}_+$ *and the number of packed items is at most* $\prod_{i=1}^{d} n_i$.

As the side lengths are integral multiples of $\hat{s}$, the number of distinct $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes in the near-optimal structure will be polynomially bounded. The following corollary follows from Lemma 1.

▶ **Corollary 3.** *Let* B *be a* $\mathcal{V}$*-Box with the item set* $\mathcal{Q}$ *packed in it. Then these items can be repacked into* B *using NFDH.*

Proof. We can either pack all the items in $\mathcal{I}$ into B using NFDH, or, by Lemma 1, the free volume inside of B is at most $\hat{s}\,\mathrm{SURF}_d(B)/2$. The latter possibility however implies that we packed items with a volume of at least $\mathrm{VOL}_d(B) - \hat{s}\,\mathrm{SURF}_d(B)/2$, which is an upper bound on $\mathrm{VOL}_d(\mathcal{I})$ by the definition of a $\mathcal{V}$-Box. ◀

▶ **Observation 4.** *Let* B *be an* $\mathcal{N}$*-Box with the item set* $\mathcal{Q}$ *packed in it. As* $|\mathcal{Q}| \leq \prod_{i=1}^{d} n_i$, *we can divide* B *into* $\prod_{i=1}^{d} n_i$ *cells such that each cell has length* $\hat{s}$ *in each of the* d *dimensions and we can place each item in* $\mathcal{Q}$ *in one cell.*

In this paper, we will often require that a set of hypercuboidal regions forms a grid.

▶ **Definition 5.** *A* d-dimensional grid *(see Figure 3(a)) C is a subset of the set of hypercuboids* $\left\{ \prod_{i=1}^{d} [g_{i,j_i-1}\ ,\ g_{i,j_i}] \;\middle|\; 1 \leq j_i \leq n_i \text{ for all } i \in [d] \right\}$, *where each* $n_i \in \mathbb{N}_+$ *denotes the number of grid layers in the* i^{th} *dimension, and* $g_{i,0} < \cdots < g_{i,n_i}$ *for all* $i \in [d]$ *are the grid boundaries. Each hypercuboid in C is called a* cell.

Each grid C has $|C| \leq \prod_{i=1}^{d} n_i$ cells. A grid C can also be refined by splitting some cells.

▶ **Lemma 6.** *Let* C *be a* d*-D grid with* $n_1, \ldots, n_d \in \mathbb{N}_+$ *layers and boundaries* $g_{i,j} \in \mathbb{R}$ *for* $i \in [d]$ *and* $j \in [n_i]$. *Let* R *be a set of* d*-D hypercuboids. Then the region of the grid which is not intersected by* R *is a new grid with* $n'_i \leq n_i + 2|R|$ *layers in each dimension* $i \in [d]$.

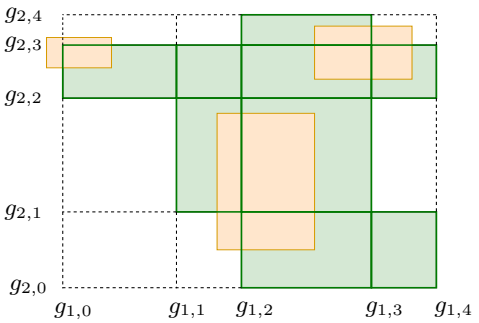
Proof. Each hypercuboid in R is the cartesian product of d intervals $\prod_{i=1}^{d} [a_i, b_i]$. The grid C can now be refined by adding the boundaries a_i and b_i in each dimension i. This increases the number of layers in each dimension by at most $2|R|$. After this refinement, each cell is either completely contained in a hypercuboid in R, and hence discarded from C, or does not intersect R. This can be seen in Figure 3(b). ◀

3 Knapsack

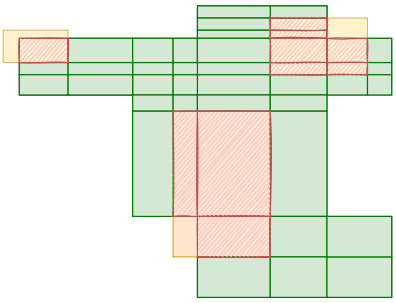
In this section, we will devise a PTAS for the d-D Hc-Knapsack problem.

3.1 Structure of a Nearly Optimal Solution

First, we prove that given an optimal packing $\mathrm{OPT}_{\text{knapsack}}(\mathcal{I})$ of the input set $\mathcal{I}$, there exists another packing containing a subset of the items packed in a simple structure which can be searched for, in polynomial time. For this, we modify the optimal packing to obtain a near-optimal packing in which all the items are packed into either $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes except for a constant number of large items. The total number of these boxes is $\mathcal{O}(1)$ and their sizes come from a set whose cardinality is polynomial in $|\mathcal{I}|$. Hence, this near-optimal packing can be searched for, in polynomial time.



(a) 3 orange rectangles partially covering a grid with 4 layers in each dimension.The dotted lines indicate the grid boundaries and the grid cells are indicated by the green rectangles.



(b) Refined grid with ≤ 10 layers in each dimension. The new grid cells are indicated in green. The shaded cells are those discarded from the refined grid as they overlap with the items.

Figure 3 Splitting the empty space inside of a grid.

▶ **Theorem 7.** *For each $0 < \varepsilon < 1/2^{d+2}$, there is a packing with the following properties:*

(i) *It consists of $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes whose total number is bounded by a constant $C_{\text{boxes}}(d, \varepsilon)$, which depends only on ε and d.*

(ii) *The number of items in the packing that are not packed in these boxes is bounded by a constant $C_{\text{large}}(d, \varepsilon)$, which depends only on ε and d.*

(iii) *The profit of the packing is at least $\left(1 - 2^{d+2}\varepsilon\right) \text{OPT}_{\text{knapsack}}(\mathcal{I})$.*

To prove the theorem, first, in Section 3.1.1, we consider a packing in an arbitrary grid (note that the knapsack can be viewed as a grid containing a single cell) and merge some of its cells into collections based on the packing of the items into the grid. The reason for choosing an arbitrary grid is that, as we will see, we recursively divide the knapsack into grids (which can have arbitrary shapes) and consider each of them separately. Then in Section 3.1.3, we show how to repack the items in a collection using NFDH or by a strip packing algorithm described in Section 3.1.2.

3.1.1 Partitioning a Grid into Collections

Consider a grid and a set of items $\mathcal{J}$ packed into it. Assume that the grid has at most N_{layer} layers in each dimension. Thus, it consists of at most $N := N_{\text{layer}}^d$ cells. Our goal is to repack (a subset of) $\mathcal{J}$ to obtain a simpler structure, by losing only a small profit. The overall grid can have a complex structure, so we consider each cell (a hypercuboid) and repack the items in the cell. However, it might be problematic to repack the items that intersect a cell only partially. Consider one such item and one of the cells it intersects. There must be a facet of this cell that cuts into this item. If this facet is orthogonal to a long dimension of the cell, then we can remove a strip of small profit and pack these intersecting items in that strip. However, an issue arises when this facet is orthogonal to a short dimension. Therefore, we merge the two cells that share this facet. Doing this iteratively, we merge some cells into a *collection of cells* or simply *collection*. See left of Figure 4.

For a cell a, we will denote the side length of the largest item of $\mathcal{J}$ partially or fully packed in it by $\hat{s}(a)$ and we denote the side lengths of the cell by $a_1, \ldots, a_d$. Furthermore, we sort the dimensions using a stable sorting algorithm [2], such that the side lengths of a

[2] A stable sorting makes the order unambiguous. This simplifies some proofs that compare the orders for different collections.

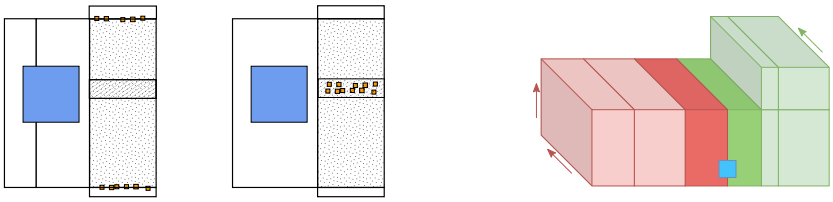


Figure 4 *(i)* The left figure shows five cells of a grid in the two-dimensional case. The orange items are small compared to the height of one of the cells (shaded with dots) they intersect. So, we transfer them to the interior of that cell by removing a least profitable strip (shaded in stripes). However, the blue item cannot fit in any of the cells it intersects, so we merge both these cells. *(ii)* The right figure demonstrates merging. An item (blue) intersects a bad facet between a 2-elongated collection (red) and a 1-elongated collection (green). After merging, we obtain a single 1-elongated collection.

in those dimensions are in non-increasing order. We will refer to that order as σ_a and thus, $a_{\sigma_a(i)}$ is the i^{th} largest side length of a. We start by defining the values which we will use to distinguish between long and short sides of a cell as

$$\alpha_d := \frac{2d}{\varepsilon}, \text{ and } \alpha_k := \frac{2}{\varepsilon}\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \text{ for all } k \in [d-1] \tag{1}$$

The parameter $C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right)$ is a constant depending on $d, k, \alpha_{k+1}, N, \varepsilon$. Its exact value is derived in the full version [24], but for our purposes, it suffices to note that for all $k \in [d-1]$, $C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) \geq 1$ and its value gets larger when k gets smaller. Hence, $1 < 2d/\varepsilon = \alpha_d < \ldots < \alpha_1$.

Now we can define a cell a to be k-elongated for $0 \leq k \leq d$ iff k of its side lengths are long compared to some size parameter s.

▶ **Definition 8.** *A cell a is k-elongated for $k \in [0, d]$ and size parameter $s > 0$ iff*

(i) $a_{\sigma_a(i)} > \alpha_k s$ *for all* $1 \leq i \leq k$ *and*

(ii) $a_{\sigma_a(i)} \leq \alpha_i s$ *for all* $k < i \leq d$.

If a is a k-elongated cell, then for any $i \in [k]$ we call $\sigma_a(i)$ to be a *long* dimension of a and the other dimensions to be *short*. Note that for each size parameter, each cell is k-elongated for exactly one value of k. A facet of a k-elongated cell is called *good* if it is orthogonal to a long dimension and *bad* otherwise. We will see that the items that intersect only the good facets of a cell don't cause much of an issue; the complicated machinery that we devise is to deal with the items intersecting the bad facets.

For a group of cells to be merged into a collection, they should satisfy some additional properties. We define a *k-elongated collection* C as a set of cells that are k-elongated using the size of the largest item completely or partially packed in that collection as a common size parameter and are aligned in their long dimensions. More formally,

▶ **Definition 9.** *A set of cells C is a k-elongated collection if*

(i) *each cell $a \in C$ is k-elongated for the size parameter $\hat{s}(C) := \max_{c \in C}(\hat{s}(c))$,*

(ii) $\sigma_a(i) = \sigma_b(i)$ *for all* $1 \leq i \leq k$ *and all* $a, b \in C$ *and*

(iii) *the projection of a and b onto the dimension $\sigma_a(i)$ is the same for all $1 \leq i \leq k$ and all $a, b \in C$.*

(iv) *The set of cells form a path-connected region.*

By properties (i) and (ii) of a k-elongated collection, we can define the k dimensions that are long for any cell of the collection to be the long dimensions of the collection itself. Property (iii) strictly limits the arrangement of the cells in long dimensions. Thus, a larger k makes

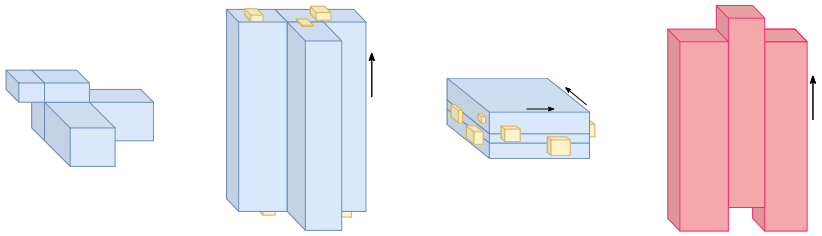


Figure 5 Different collections in 3-D with some items intersecting good facets. The arrow marks indicate the long dimensions. From left to right: 0-, 1-, 2-elongated collections, and a non-example of a collection (violates property (iii).)

the shape of the whole collection less complex (See Figure 5). A common facet between two cells $a \in C$ and $b \in D$ of two different collections C and D is called an outer facet for each of those collections. For each dimension, each cell has two facets that are orthogonal to this dimension. The facet with the lower coordinate in this dimension is called the bottom facet and the other one is called the top facet. For each long dimension of a collection the *bottom facet of the collection* is formed by the union of the bottom facets of its cells; the *top facet of the collection* is formed by the top facets of its cells.

Initially, we consider each cell of our grid as a collection containing only itself. Whenever there is an item intersecting an outer facet between two collections C and D, we will merge those collections if that facet is bad (i.e. orthogonal to a short dimension) for both of them. Such a case is visualized on the right of Figure 4. The next lemma proves that merging collections in this case will create a new collection.

▶ **Lemma 10.** *Let C be a k_C-elongated collection and let D be a k_D-elongated collection. Let $a \in C$ and $b \in D$ be two cells that have a common facet which is bad for both of them. Then $C \cup D$ is a $\min(k_C, k_D)$-elongated collection.*

Proof. Note that both k_C, k_D are strictly less than d since a, b share a common facet f which is bad for both. W.l.o.g., we will assume that $\hat{s}(C) \geq \hat{s}(D)$. Let f has side lengths $f_1, \ldots, f_d$ where f_i denotes the length in the i^{th} dimension (for simplicity, we assume f is a d-dimensional hypercuboid with a side of zero length). Note that the facet f has the same side lengths as a and b in any dimension except the dimension in which it has zero length. This implies that a and b are of almost the same shape. In the dimension orthogonal to f, both a and b have a rather short length, as f is a bad facet for both collections. Using these arguments, we can derive the following properties:

- **(i)** $a_{\sigma_f(i)} = f_{\sigma_f(i)} = b_{\sigma_f(i)}$ for all $1 \leq i \leq d-1$ (Since f is common to both a, b)
- **(ii)** $\sigma_f(i) = \sigma_a(i)$ for all $1 \leq i \leq k_C$ (Since f is bad for a which is a k_C-elongated cell)
- **(iii)** $\sigma_f(i) \in \{\sigma_a(i), \sigma_a(i+1)\}$ for all $k_C < i < d$ (explained below)
- **(iv)** $\sigma_f(i) = \sigma_b(i)$ for all $1 \leq i \leq k_D$
- **(v)** $\sigma_f(i) \in \{\sigma_b(i), \sigma_b(i+1)\}$ for all $k_D < i < d$

The property (iii) above is due to the fact that a and f have the same lengths in all dimensions except one, say $d_\perp$. So, the order of dimensions $\sigma_a(1), \sigma_a(2), \ldots, \sigma_a(d)$ can be obtained by inserting $d_\perp$ in the list $\sigma_f(1), \sigma_f(2), \ldots, \sigma_f(d-1)$ at the appropriate index but we know that this index lies after k_C since $d_\perp$ is short for a.

If we assume $k_C > k_D$, we obtain the inequality $f_{\sigma_f(k_C)} = a_{\sigma_a(k_C)} > \alpha_{k_C} \hat{s}(C)$ by (i), (ii) and Definition 8 of the k_C-elongated cell a. We also obtain the contradictory inequality $f_{\sigma_f(k_C)} = b_{\sigma_f(k_C)} \leq b_{\sigma_b(k_C)} \leq \alpha_{k_C} \hat{s}(D) \leq \alpha_{k_C} \hat{s}(C)$ by (i), (iv) and (v) and Definition 8 of the k_D-elongated cell b. Thus, our choice $\hat{s}(C) \geq \hat{s}(D)$ determines that $k_C \leq k_D$.

As $\hat{s}(C \cup D) = \hat{s}(C)$ and C is a k_C-elongated collection, we know that $C \cup D$ fulfills property (i) of Definition 9 for a k_C-elongated collection for every $a' \in C$. Let $b' \in D$. We have to show that b' is a k_C-elongated cell using the size parameter $\hat{s}(C)$ as well. For every $1 \leq i \leq k_C$ we can use Definition 9 of the k_D-elongated collection D and (i), (iv) and (ii) to prove $b'_{\sigma_{b'}(i)} = b_{\sigma_b(i)} = f_{\sigma_f(i)} = a_{\sigma_a(i)} > \alpha_{k_C}\hat{s}(C)$. For every $k_C < i \leq k_D$ we can again use Definition 9 of the collection D and properties (i), (iv) and (iii) to prove $b'_{\sigma_{b'}(i)} = b_{\sigma_b(i)} = f_{\sigma_f(i)} \leq a_{\sigma_a(i)} \leq \alpha_i\hat{s}(C)$. For every $k_D < i \leq d$ we already know that $b'_{\sigma_{b'}(i)} \leq \alpha_i\hat{s}(D) \leq \alpha_i\hat{s}(C)$ because of Definition 8 for the k_D-elongated cell b'. Thus, b' is a k_C-elongated cell using the size parameter $\hat{s}(C)$.

Now we only need to prove properties (ii) and (iii) of Definition 9 for every pair of cells in $C \cup D$ to show that it is a collection. For a pair $a', a'' \in C$ or $b', b'' \in D$ this is trivial, because C and D are k_C- or k_D-elongated collections and $k_C \leq k_D$. For the pair a and b, those properties are again trivial, because both have all their long sides on the common facet f. For any $a' \in C$ and $b' \in D$ the properties follow as the pairs a' and a, a and b, b and b' satisfy these properties. ◀

After this merging procedure, we will assign each item $x \in \mathcal{J}$ to a collection C as follows: If x is completely packed in a collection, then we assign it to that collection itself. If it is partially contained, then we assign it to one of the collections (breaking ties arbitrarily) with which it intersects only via the good outer facets. The next lemma shows that this assignment is indeed possible.

▶ **Lemma 11.** *Let G be the set of all collections that were derived after the merging procedure. Let $x \in \mathcal{J}$ be an item that is packed in the grid but isn't completely packed in any collection in G. Then there exists a collection $C \in G$ in which x is partially packed but x does not intersect any of its bad outer facets.*

Proof. First, we define an order on the set of the collections G. We associate each k_C-elongated collection $C \in G$ with the tuple of the long sides $\left(a_{\sigma_a(1)}, \ldots, a_{\sigma_a(k_C)}\right)$ for some cell $a \in C$ in a descending order. By property (iii) in the definition of a k_C-elongated collection, the tuple is independent of the choice of the cell. We now define the strict lexicographic order $\prec$ on these tuples. In other words, for a k_C-elongated collection C, a k_D-elongated collection D and cells $a \in C$ and $b \in D$, we have $C \prec D$ iff

- **(i)** $k_C < k_D$ and $a_{\sigma_a(i)} = b_{\sigma_b(i)}$ for all $1 \leq i \leq k_C$, or
- **(ii)** there is some $1 \leq k \leq \min(k_C, k_D)$ such that $a_{\sigma_a(i)} = b_{\sigma_b(i)}$ for all $i \in [k-1]$ and $a_{\sigma_a(k)} < b_{\sigma_b(k)}$.

Let $H \subseteq G$ be the set of collections where x is partially contained in. Let $C \in H$ be a maximal collection in H according to $\prec$ and let it be k_C-elongated. Now we want to prove that x does not intersect any bad outer facet of C. Assume there is a bad outer facet f of C which is intersected by x. This facet f belongs to some cells $a \in C$ and $b \in D$ where D is a different collection than C. The facet f has to be a good facet of D, because it is bad for C and the collections C and D were not merged. Let $d_\perp$ be the dimension which is orthogonal to f. Like in the proof of Lemma 10, we know that $a_i = b_i$ for all $i \in \{1, \ldots, d\} \setminus \{d_\perp\}$ because of their common facet f. As f is bad for C this contains all the sides of a that are long using the size parameter $\hat{s}(C)$. If a side of a is long using the size parameter $\hat{s}(C)$ and larger than $b_{d_\perp}$, then this side is also long using the size parameter $\hat{s}(D)$, because $b_{d_\perp}$ is already considered as long using this size parameter. Thus, we have two cases: In the first case, each side of a which is long using the size parameter $\hat{s}(C)$ is larger than $b_{d_\perp}$. Then each long side of a is also a long side of b while b has at least one additional long side $b_{d_\perp}$. Thus, we have $C \prec D$ by (i). In the second case there are only $k < k_C$ sides of a long using

the size parameter $\hat{s}(C)$ and larger than $b_{d_\perp}$. Then the k longest sides of a and b have the same size and for the $(k+1)^{\text{th}}$ longest sides we have $a_{\sigma_a(k+1)} < b_{d_\perp} = b_{\sigma_b(k+1)}$. Thus, we have $C \prec D$ by (ii). In both cases, the maximality of C is violated, and therefore no bad outer facet of C can be intersected by x. ◀

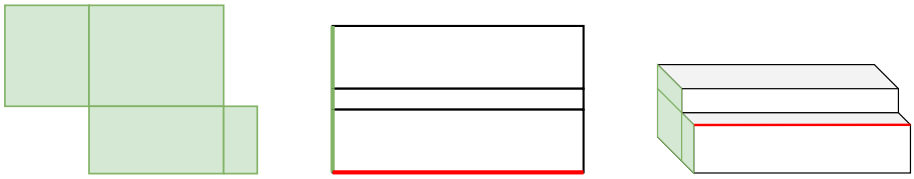
3.1.2 Strip Packing with Resource Augmentation

Before we proceed to repack the items inside of those collections, we will have a closer look at an arbitrary k-elongated collection for some $k \in [d-1]$. This constraint on k ensures that the collection is large in at least one dimension and short in at least one dimension. Thus our collection is some kind of a strip (may not be hypercuboidal). Previously, we reordered the dimensions for a collection to have the k long dimensions first; however, for the strip packing problem we assume the opposite, i.e., we suppose that the collection is short in the first $(d-k)$ dimensions and long in the last k dimensions. Each cell of this collection is the cartesian product of some intervals and property (iii) of Definition 9 ensures that the last k intervals are the same for any cell in this collection. By splitting the cartesian product for a cell after the first $(d-k)$ (short) intervals and before the last interval, we can represent the cell as $a \times B \times [h_1, h_2]$ where a is a $(d-k)$-dimensional hypercuboid that depends on the selected cell and both the $(k-1)$-dimensional hypercuboid B and the values h_1 and h_2 are common to all cells of the collection. This set of hypercuboids $\{a\}$ forms a $(d-k)$-dimensional grid A that is the projection of our collection onto the first $(d-k)$ dimensions. With these definitions, our collection can be viewed as a d-dimensional strip with base $(\cup_{a\in A} a) \times B$ and height $(h_2 - h_1)$. We would like to repack this collection using a strip packing algorithm. For this purpose, we devise an algorithm to pack hypercubes on a base (which can be extended by a slight amount in the long dimensions) with the properties described above. With this motivation, we formally define the strip packing problem and state the main result in the next few paragraphs.

Let $k \in [d-1]$. Let $\varepsilon > 0$ be some accuracy parameter and $N \in \mathbb{N}_+, \alpha \geq 1$ be some constants depending on d, ε. The input consists of a set of items $\mathcal{I}$. In the entire subsection, we abbreviate $\hat{s}(\mathcal{I})$ by $\hat{s}$.

The $(d-1)$-dimensional base on which we need to pack the input set $\mathcal{I}$ is the cartesian product $A \times B$ where A is some $(d-k)$ dimensional grid of at most N cells and B is some $(k-1)$ dimensional hypercuboid. The side length of B in the i^{th} dimension ($i \in [k-1]$) is denoted by b_i. Each side length of each cell in A has to be at most $\alpha\hat{s}$ and each b_i has to be at least $N\alpha^{d-k}\hat{s}$. As it can be seen, compared to $\hat{s}$, all the edge lengths in A are short and all the edge lengths in B are long. Hence, we refer to the dimensions of the coordinate system parallel to the edges of A as *short dimensions* and to the dimensions parallel to the edges of B as *long dimensions*. In the 3-dimensional setting, we have two possible values for k. If $k = 1$, then B vanishes and our base is just a grid of rectangles A. Such a base can be seen in Figure 6a. If $k = 2$, then B is an interval and A is just a set of non-overlapping intervals because it is 1-dimensional. In this case, the base can be visualized like in Figure 6b as a set of flat but wide rectangles, that are positioned on top of each other. So for the 3-dimensional case, we either have a complex structure through A or long sides through B but never both. In higher dimensions however, we can have both. An example for this is the base of a 4-dimensional strip in Figure 6c. By scaling every dimension equally, it is assumed that the volume of the grid A is normalized: $\text{VOL}_{d-k}(A) := \sum_{a\in A} \text{VOL}_{d-k}(a) = 1$. We also assume that any item in $\mathcal{I}$ can be packed on the base $A \times B$. The goal is to find a non-overlapping packing of the items in $\mathcal{I}$ into the region given by the set product of

A, B and h, that is $\mathrm{CONT}(A, B, h) := (\bigcup_{a \in A} a) \times B \times [0, h]$ where h is called the height of the packing. The optimal (i.e. minimal) height h for which it is possible to pack $\mathcal{I}$ in $\mathrm{CONT}(A, B, h)$ is denoted by $\mathrm{OPT}_{\mathrm{strip}}(\mathcal{I}, A, B)$.



(a) $d = 3, k = 1$. **(b)** $d = 3, k = 2$. **(c)** $d = 4, k = 2$.

Figure 6 The first two figures show two possible bases for a 3-dimensional strip. In the first figure, A is shown in green and we have no B. In the second figure, the grid A (in green) is 1-dimensional and thus simple, and B is shown in red (the long side). The last figure shows a possible 3-dimensional base with $k = 2$ of a 4-dimensional strip with A shown in green (the short sides) and B shown in red (the long side).

We only consider instances where a solution exists. Since we assume that $\mathrm{VOL}_{d-k}(A) = 1$, we have that $\hat{s} \leq 1$ as a larger item would not fit into the strip. Note that for each $i \in [k-1]$, $b_i \geq N\alpha^{d-k}\hat{s} \geq N(\alpha\hat{s})^{d-k} \geq \sum_{a \in A} \mathrm{VOL}_{d-k}(a) = \mathrm{VOL}_{d-k}(A) = 1$. In this section, we describe an algorithm based on Harren's [21] multidimensional generalization of the algorithm by Kenyon and Rémila [28] for 2-dimensional strip packing.

The differences between our algorithm and Harren's algorithm[21] can be summed up as follows: Due to the complex nature of A, we have multiple hypercuboids, each having a bounded aspect ratio. This changes the analysis of the algorithm. It also changes the details like packing the medium and small items on the top, because we have to split one strip into multiple smaller strips (having hypercuboidal bases) and distribute the items among them. The addition of very long sides in form of B, like in Figure 6b, breaks the bounded aspect ratio property in a more crucial way. Harren's algorithm first packs the large items on the base and then extends this packing along the height of the strip. This relies on the fact that, due to bounded aspect ratio, not many large items can be packed on the base, and thus we have to consider only a constant number of so-called *configurations*. If we have additional long sides in the base in form of B, we don't have such a bound on the number of large items that can fit in the base. For example, in Figure 6b, an item can be large with respect to A (shown in green), but the number of such items that can fit on the base $A \times B$ can't be bounded by a constant since the side shown in red can be arbitrarily long.

We work around this problem as follows: First, we consider $(d-k)$ dimensional packings, i.e., *configurations* of the items, on A. Then we not only extend these configurations along the height of the strip but also along each of the $(k-1)$ dimensions of B. For example, in Figure 6b, we first create one-dimensional packings on A and then extend each of these packings both in the rightward direction and along the height of the strip.

▶ **Theorem 12.** *Let $\mathcal{I}$, A and B be the input for the d-dimensional strip packing problem defined above and let $\varepsilon > 0$ be some additional accuracy parameter. Then there is an algorithm which packs all items of $\mathcal{I}$ into the region $\mathrm{CONT}(A, B + \hat{s}, h)$ where $B + \hat{s}$ is the hypercuboid B after increasing each side length by $\hat{s}$ and the height h is given by*

$$h \leq (1 + \mathcal{O}(\varepsilon)) \, \mathrm{OPT}_{\mathrm{strip}}(\mathcal{I}, A, B) + \mathcal{O}(\hat{s})$$

The constant omitted in the expression $\mathcal{O}(\varepsilon)$ is a function of k. The constant omitted in the expression $\mathcal{O}(\hat{s})$ is a function of $d, k, \alpha, N, \varepsilon$.

Due to space limitations, the detailed proof of Theorem 12 is given in the full version [24]. Our algorithm first classifies the input items into large, medium, and small items. The side lengths of large items are rounded using linear grouping [12] such that there is only a constant number of different sizes. When we try to pack a subset of large items by arranging them in the first $(d-k)$ dimensions while giving them the same position in the last k dimensions, we only have to consider a constant number of those subsets. This is because the volume of the strip in the first $(d-k)$ dimensions is small and thus only a small subset of large items is packable this way and items of the same size can be considered to be identical. For each configuration, we take a packing that only uses the first $(d-k)$ dimensions, and extend it to use also the last k dimensions by placing multiple items of the same size next to each other in those long dimensions. This creates an $\mathcal{N}$-Box from each item in each configuration. An LP is used to determine how large a configuration is extended in those k dimensions. The packings for the different configurations are treated as layers stacked on top of each other.

After the large items are packed, we use the gaps between the large items to create $\mathcal{V}$-Boxes to hold some of the small items. The remaining small items and medium items are placed in additional $\mathcal{V}$-Boxes on top of this packing. Again this needed several technical adaptations, as unlike [21], we have multiple different bases and we need to distribute the remaining items in a balanced way such that the total height is minimized. See the full version [24] for the details.

As we mentioned earlier, we use this theorem to repack a k-elongated collection Q ($k \in [d-1]$). So, we use α_{k+1} as α since we know that for every cell in $\mathcal{Q}$, the first k dimensions have length at least $\alpha_k \hat{s}(\mathcal{Q}) \geq N\alpha_{k+1}^{d-k}\hat{s}(\mathcal{Q})$ (see Remark 15 and definition of α_k) and each of the last $(d-k)$ dimensions have length at most $\alpha_{k+1}\hat{s}(\mathcal{Q})$. Now let us note a few useful remarks about the above theorem applied to our collection Q. The proofs of these remarks follow from a few lemmas in the full version [24].

▶ Remark 13. A more exact bound for the height of the packing obtained in Theorem 12 is

$$\left(1+\left(2+\max\left(3,2^{k-1}\right)\right)\varepsilon\right)\mathrm{OPT}_{\mathrm{strip}}\left(Q,A,B\right)+2\left(C_{\mathrm{configs}}\left(d,k,\alpha_{k+1},N,\varepsilon\right)+3\right)\hat{s}\left(\mathcal{Q}\right)$$

Here C_{configs} is only dependent on its parameters.

▶ Remark 14. The algorithm of Theorem 12 packs all items in $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes. The number of $\mathcal{N}$-Boxes is bounded by $C_{\mathrm{configs}}\left(d,k,\alpha_{k+1},N,\varepsilon\right)/C_{\rho}\left(d,k,\alpha_{k+1},N,\varepsilon\right)^{d-k}$ and the number of $\mathcal{V}$-Boxes by $C_{\mathrm{configs}}\left(d,k,\alpha_{k+1},N,\varepsilon\right)N2^{d-k}/C_{\rho}\left(d,k,\alpha_{k+1},N,\varepsilon\right)^{(d-k)^2}+N$. Here C_{configs} and C_{ρ} are only dependent on their parameters.

▶ Remark 15. The value $C_{\mathrm{configs}}\left(d,k,\alpha_{k+1},N,\varepsilon\right)$ from Remarks 13 and 14 has a lower bound of $C_{\mathrm{configs}}\left(d,k,\alpha_{k+1},N,\varepsilon\right) > N\alpha_{k+1}^{d-k} \geq \alpha_{k+1}$.

3.1.3 Repacking a Collection

Depending on the type of the collection, we can simplify the packing of the items assigned to each collection. Let $\mathcal{C}$ be a k-elongated collection and let $\mathcal{Q}$ be the set of items assigned to $\mathcal{C}$. Let $\hat{s} := \hat{s}\left(\mathcal{Q}\right)$. We consider the cases $k = d, 1 \leq k < d$, and $k = 0$ separately.

First of all, if $\mathcal{C}$ is a d-elongated collection, then it consists of only a single cell as it has no bad facets and hence can not be merged with any other cell. Thus we can repack (after removing only a small profit subset) it efficiently using NFDH because it is a hypercuboidal region that is large in all dimensions. The following lemma states this. The proof can be found in the full version [24].

▶ **Lemma 16.** *If $\mathcal{C}$ is a d-elongated collection, then it can be transformed into a single $\mathcal{V}$-Box with a loss of profit at most $6d\varepsilon p\left(\mathcal{Q}\right)$.*

For a k-elongated collection with $1 \leq k < d$, we can apply the strip packing algorithm of Theorem 12 as shown in the next lemma.

▶ **Lemma 17.** *A k-elongated collection with $1 \leq k < d$ can be transformed into a constant number of $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes by losing profit at most $\left(k + 7 + \max\left(6, 2^k\right)\right) \varepsilon p\left(\mathcal{Q}\right)$ where $\mathcal{Q}$ is the set of items assigned to it.*

Proof. We begin by shrinking the collection to compensate for the enlargement that the strip packing algorithm will cause. Define one of the long dimensions of the collection to be its height and let h be the length in this dimension. Let $c := 4 + \max\left(3, 2^{k-1}\right)$. We assumed $\varepsilon < 1/2^{(d+2)} < 1/(2c)$. Divide the strip into $\lfloor 1/\left(c\varepsilon\right) \rfloor$ slices along the height. After that each slice has a height of at least

$$\begin{aligned}
c\varepsilon h &= \left(4 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h \\
&\geq \left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h + 2\varepsilon\alpha_k \hat{s} && \text{(since } h \geq \alpha_k \hat{s}\text{)} \\
&= \left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h + 4\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s} && \text{(by definition of } \alpha_k\text{)} \\
&\geq \left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h + 2\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s} + 6\hat{s}.
\end{aligned}$$

By clearing the slice with the lowest profit of items completely contained in it, we leave a gap with a height of $\left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h + 2\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s} + 4\hat{s}$. This causes a loss of profit at most $\frac{1}{\lfloor 1/(c\varepsilon) \rfloor} p\left(\mathcal{Q}\right) \leq \frac{1}{1/(c\varepsilon) - 1} p\left(\mathcal{Q}\right) \leq \frac{1}{1/(c\varepsilon) - 1/(2c\varepsilon)} p\left(\mathcal{Q}\right) = 2c\varepsilon p\left(\mathcal{Q}\right)$. The items intersecting the bottom or top facet of the collection fit in a gap of height $2\hat{s}$. We reduce the height of the collection to

$$h' := h - \left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h - 2\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s}$$

by shifting the items intersecting the (removed) region at the top into a gap of height $\left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h + 2\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s} + \hat{s}$.

For each other large dimension we create $\lceil 1/\varepsilon \rceil$ slices and clear the slice with the lowest profit, losing at most $\varepsilon p\left(\mathcal{Q}\right)$ and create a gap of width at least $\frac{1}{\lceil 1/\varepsilon \rceil} \alpha_k \hat{s} - 2\hat{s} \geq \frac{1}{1/\varepsilon + 1} \alpha_k \hat{s} - 2\hat{s} \geq \frac{\varepsilon}{2} \alpha_k \hat{s} - 2\hat{s} \geq 4\hat{s}$. Now we can shift the items intersecting the orthogonal facets into that gap and reduce the length of the collection in this direction by $\hat{s}$. After this shifting process, we will be left with a subset of items $\mathcal{Q}' \subseteq \mathcal{Q}$.

To be able to use the strip packing algorithm from Theorem 12, the base of our strip has to be in a special representation. For this, we create a $(d-k)$-dimensional grid A by projecting the cells of our collection to their $(d-k)$ short dimensions. We define B as a tuple holding the lengths in the $(k-1)$ long dimensions of the collection that are not the height. Now our base can be represented as the cartesian product $A \times B$. As a last preparation step, we scale the whole collection and the items in it by $f := 1/\operatorname{VOL}_{d-k}\left(A\right)^{1/(d-k)}$ in each dimension to normalize the volume of A to 1. Note that each value in the tuple B is at least $\alpha_k \hat{s} f \geq N \alpha_{k+1}^{d-k} \hat{s} f$ by the definition of α_k and Remark 15. Thus, we can use the strip packing algorithm Theorem 12 using the bound on the height by Remark 13 to compute a packing of height at most

$$\begin{aligned}
&\left(1 + \left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon\right) \operatorname{OPT}_{\text{strip}}\left(Q', A, B\right) + 2\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s} f \\
&\leq \left(1 + \left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon\right) h' f + 2\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s} f \\
&\leq h' f + \left(2 + \max\left(3, 2^{k-1}\right)\right) \varepsilon h f + 2\left(C_{\text{configs}}\left(d, k, \alpha_{k+1}, N, \varepsilon\right) + 3\right) \hat{s} f \\
&= h f.
\end{aligned}$$

After scaling back, the packing fits into our collection. ◀

Repacking 0-elongated Collections. For a 0-elongated collection, we first distinguish the items into large, medium, and small items such that the profit of medium items is very small so that they can be discarded. We then further distinguish between the cases when (i) there is a large item of very small profit or (ii) every large item has a significant profit. In the first case, we remove the large item to make enough space to repack the small items. In the second case, we use the fact that the number of large items can only be $O(1)$. So, we partition the grid into smaller grids and solve these recursively. We finally prove that we only require $O(1)$ number of recursive steps that we require are only a constant in number.

From now on, lets assume that $\mathcal{C}$ is a 0-elongated collection. To partition $\mathcal{Q}$ into large, medium, and small items, we define the following values. $\rho'_0 := 1, \rho'_{i+1} := \left(\frac{(\rho'_i)^d}{4dN\alpha_1^d}\right)^{d+1}$ and $\rho_i := \rho'_i \hat{s}$ for all $i \geq 0$. Let $\eta \in \mathbb{N}_+$ be minimal such that $p(\{ x \in \mathcal{Q} \mid \rho_\eta < s(x) < \rho_{\eta-1} \}) \leq \varepsilon p(\mathcal{Q})$. Note that $\eta \leq 1/\varepsilon$. We now partition $\mathcal{Q}$ into sets $\mathcal{S} := \{ x \in \mathcal{Q} \mid s(x) \leq \rho_\eta \}$, $\mathcal{M} := \{ x \in \mathcal{Q} \mid \rho_\eta < s(x) < \rho_{\eta-1} \}$ and $\mathcal{L} := \{ x \in \mathcal{Q} \mid \rho_{\eta-1} \leq s(x) \}$. We call $\mathcal{S}$ (resp. $\mathcal{M}, \mathcal{L}$) to be the set of small (resp. medium, large) items. Note that since $\eta \geq 1$ and $\rho_0 = \hat{s}$, the set of large items can't be empty. This simple observation will be useful later.

If there is a large item with a small profit, then after discarding that large item, the entire empty space can be divided into a constant number of $\mathcal{V}$-Boxes. It can then be shown that these $\mathcal{V}$-Boxes have enough volume to pack all the small items. This is formalized in the following lemma. The proof can be found in the full version [24].

▶ **Lemma 18.** *Suppose $\mathcal{C}$ is a 0-elongated collection. If an item in $\mathcal{L}$ has profit at most $\varepsilon p(\mathcal{Q})$, then the collection $\mathcal{C}$ can be transformed into a packing of profit at least $(1-2\varepsilon)\, p(\mathcal{Q})$ containing a constant number of large items and a constant number of $\mathcal{V}$-Boxes.*

If all the large items have a good profit (i.e., a profit of more than $\varepsilon p(\mathcal{Q})$), we cannot discard any of them to create space. Therefore, it is hard to repack the items directly so instead we will take a recursive approach using a shifting argumentation.

▶ **Lemma 19.** *Suppose $\mathcal{C}$ is a 0-elongated collection. If every item in $\mathcal{L}$ has a profit of at least $\varepsilon p(\mathcal{Q})$, then $\mathcal{C}$ can be transformed into a constant number of large items, $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes while losing profit at most $\max\left(6d+1, d+13, d+7+2^{d-1}\right) \varepsilon p(\mathcal{Q})$.*

Proof. In this case, we have at most $1/\varepsilon$ large items. We can split the area of the collection, which is not covered by the large items into a grid of at most $2/\varepsilon$ additional layers in each dimension and recursively start from Section 3.1.1 with the classification of cells and merging into collections. Each of these collections is a k-elongated collection ($k > 0$) or a 0-elongated collection. The former collections can be repacked using Lemma 16 or Lemma 17. Let's look at the 0-elongated collections. Note that the total profit of these collections is at most $(1-\varepsilon)p(\mathcal{Q})$ since we recurse only if there is a large item in $\mathcal{Q}$ of profit at least $\varepsilon p(\mathcal{Q})$. For a 0-elongated collection in this recursive step, we either use Lemma 18 if there exists a large item of small profit or, we continue the recursion. Note, that if we reach a depth of $\lceil \log_{1-\varepsilon}(\varepsilon) \rceil$ in this recursion, then we can just discard the remaining items and stop. The reason for this is that we packed in each recursive step at least one large item with a non-negligible profit, so at this maximal recursion depth, all the items over all collections have a profit at most $(1-\varepsilon)^{\lceil \log_{1-\varepsilon}(\varepsilon) \rceil}\, p(\mathcal{Q}) \leq \varepsilon p(\mathcal{Q})$.

During this process, the initial region was split into different collections, where each of those was assigned a partition of the items $\mathcal{Q}' \subseteq \mathcal{Q}$. At this point, no profit was lost except for the items that are lost at the maximal recursion depth with profit at most $\varepsilon p(\mathcal{Q})$. When repacking each collection by transforming them into large items, $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes, we

lose again some profit, which depends on the method we used for repacking. We lose at most $6d\varepsilon p(\mathcal{Q}')$ through Lemma 16, at most $\left(d+6+\max\left(6,2^{d-1}\right)\right)\varepsilon p(\mathcal{Q}')$ through Lemma 17, at most $2\varepsilon p(\mathcal{Q}')$ through Lemma 18. As those subsets for the collections are distinct the loss in this repacking step is bounded by $\max\left(6d, d+12, d+6+2^{d-1}, 2\right)\varepsilon p(\mathcal{Q})$. Noting that $2 < \max\left(6d, d+12, d+6+2^{d-1}\right)$ and adding the factor of $\varepsilon p(\mathcal{Q})$ that we may lose during the recursion proves the claim. ◀

The last thing to do is to prove, that the number of $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes that are created is constant. As those boxes are created out of the cells of the grid in Section 3.1.1, we start by bounding the size of this grid during the recursive algorithm. The following lemma follows from the fact that in each of the recursive step (having depth at most $\lceil \log_{1-\varepsilon} \varepsilon \rceil$) the number of layers increases by at most $2/\varepsilon$ in each of the dimensions. The proof can be found in the full version [24].

▶ **Lemma 20.** *Consider any grid $\mathcal{G}$ obtained during the recursive algorithm starting from the knapsack with an optimal packing as a grid with a single cell. Then the number of layers in each dimension in $\mathcal{G}$ is upper bounded by $C_{\text{layer}}(\varepsilon) := 2\left\lceil \log_{1-\varepsilon}(\varepsilon)\right\rceil/\varepsilon+1$.*

▶ Remark 21. Due to the above lemma, the bound on the number of layers also gives an upper bound $C_{\text{N}}(d,\varepsilon) := (C_{\text{layer}}(\varepsilon))^d$ on the number of cells in any grid that we consider in Section 3.1.1.

Using these results, we obtain the following lemma which bounds the number of $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes. The proof can be found in the full version [24].

▶ **Lemma 22.** *The numbers of large items, and the total number of $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes generated by the transformation of an optimal packing can be bounded by C_{large} and C_{boxes}, respectively. C_{large} and C_{boxes} are constants that depend only on d, ε.*

Now we can prove Theorem 7.

Proof of Theorem 7. Consider an optimal packing and interpret the initial knapsack as a cell of a grid with one layer in each dimension. Then we start with the procedure explained in Section 3.1.1 to classify the unit hypercube either as d-elongated or 0-elongated. By Lemmas 16, 18, and 19, we can simplify the structure of the packing and lose a profit of at most $\max(6d+1, d+13, d+8+2^{d-1})\varepsilon p(\mathcal{I}) \leq 2^{d+2}\varepsilon p(\mathcal{I})$ as we have $d \geq 2$. Lemma 22 bounds the number of $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes and the large items packed outside them. ◀

3.2 Algorithm

Using the results of Section 3.1, we can construct a PTAS for d-D Hc-Knapsack.

▶ **Theorem 23.** *Let $d \geq 2$ and $\varepsilon > 0$. There is an algorithm which returns for each instance of the d-dimensional hypercube knapsack packing problem given by a set of items a packing with profit at least $(1-\varepsilon)\,\text{OPT}_{\text{knapsack}}(\mathcal{I})$ with a running time which is polynomial in $|\mathcal{I}|$.*

Proof. Using Theorem 7 with an accuracy of $2^{-d-3}\varepsilon$ we know that there is a packing with a simple structure and profit at least $(1-\varepsilon/2)\,\text{OPT}_{\text{knapsack}}(\mathcal{I})$. Let $\mathcal{B}$ be the set of $\mathcal{N}$-Boxes and $\mathcal{V}$-Boxes in this structure, $\mathcal{L} \subseteq \mathcal{I}$ the set of items packed outside of those boxes and $\mathcal{S} \subseteq \mathcal{I}\backslash\mathcal{L}$ the set of items packed inside of them.

As the number of items in $\mathcal{L}$ is constant, there are at most $|\mathcal{I}|^{C_{\text{large}}(d,\varepsilon)}$ choices for this subset $\mathcal{L} \subseteq \mathcal{I}$. There are at most $C_{\text{boxes}}(d,\varepsilon)+1$ choices for the number of boxes in $\mathcal{B}$ and for each box there are at most:

- 2 choices whether it is a $\mathcal{V}$-Box or an $\mathcal{N}$-Box,
- $|\mathcal{I}|$ choices for the size parameter $\hat{s}$ of the box,
- $|\mathcal{I}|^d$ choices for the side lengths of the box.

All of these choices are polynomial in the number of items $|\mathcal{I}|$. Thus, by iterating over all possible choices, we can assume at this point that we know the set of items $\mathcal{L}$ and the set $\mathcal{B}$ of $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes of that nearly optimal solution. As both $|\mathcal{L}|$ and $|\mathcal{B}|$ are bounded by a constant, we can find a packing of those items and boxes in the unit hypercube in constant time. The last step left to do is to pack items from $\mathcal{I}\backslash\mathcal{L}$ in the boxes with a nearly optimal profit. We can do this by solving a special variant of the Generalized Assignment Problem (GAP) [42]. In GAP, we are given a set of one-dimensional knapsacks with a capacity each and a set of items that may have a possibly different size and profit for each knapsack. The goal in this problem is to find a feasible packing with maximal profit. We will consider our $O(1)$ number of $\mathcal{V}$-Boxes and $\mathcal{N}$-Boxes to be knapsacks. Consider a $\mathcal{V}$-Box B_V with size parameter $\hat{s}$. We set its capacity to be $\mathrm{VOL}_d(B) - \hat{s}(\mathrm{SURF}_d(B)/2)$. For any item i, we set its size with respect to B_V as $\mathrm{VOL}_d(i)$ if $s(i) \leq \hat{s}$ and ∞ otherwise. On the other hand, consider an $\mathcal{N}$-Box B_N with size parameter $\hat{s}$ and $\{n_i\}_{i\in d}$ denoting the number of cells in each dimension. We set its capacity to be $n_1 n_2 \dots n_d$. For any item i, we set its size with respect to B_N as 1 if $s(i) \leq \hat{s}$ and ∞ otherwise. The profit of any item i with respect to any box is just set as $p(i)$. This boils down to a variant of GAP with $O(1)$ number of knapsacks, which admits a PTAS [18]. Thus by solving this instance, we get a subset $\mathcal{S}' \subseteq \mathcal{I}\backslash\mathcal{L}$ with profit at least $p(\mathcal{S}') \geq (1-\varepsilon/2)\,p(\mathcal{S})$. Using Observation 4 and Corollary 3 we can ensure to pack those items inside the boxes. The total profit packed is $p(\mathcal{S}') + p(\mathcal{L}) \geq \left(1-\frac{\varepsilon}{2}\right)p(\mathcal{S}) + p(\mathcal{L}) > \left(1-\frac{\varepsilon}{2}\right)(p(\mathcal{S}) + p(\mathcal{L})) \geq \left(1-\frac{\varepsilon}{2}\right)^2 \mathrm{OPT}_{\text{knapsack}}(\mathcal{I}) \geq (1-\varepsilon)\,\mathrm{OPT}_{\text{knapsack}}(\mathcal{I})$. ◀

4 Conclusion

We have designed a PTAS for a variant of the knapsack problem (d-D Hc-Knapsack), where the items are d-dimensional hypercubes with arbitrary profits. En route, we have also developed a near-optimal algorithm for a variant of the d-dimensional hypercube strip packing problem, where the base need not be hypercuboidal, but we are allowed to extend some of the sides of the base (the long dimensions) by a small amount. Extending the techniques of [23], we believe that it might be possible to design an EPTAS for d-D Hc-Knapsack.

The knapsack problem for squares, cubes, and hypercubes is thus almost settled. Whether there exists a PTAS for the knapsack variant where items are rectangles is still open. It is also interesting to improve the current best approximation ratios ($(5+\varepsilon)$ with rotations and $(7+\varepsilon)$ without rotations [14]) for the knapsack problem where items are cuboids (3D) because of its practical relevance. Another interesting task of theoretical importance would be to improve the current best approximation ratio (which is $(3^d+\varepsilon)$ due to [41]) for the knapsack problem where items are d-dimensional hypercuboids.

References

1 Fidaa Abed, Parinya Chalermsook, José R. Correa, Andreas Karrenbauer, Pablo Pérez-Lantero, José A. Soto, and Andreas Wiese. On guillotine cutting sequences. In *APPROX*, pages 1–19, 2015.

2 Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *Journal of the ACM*, 66(4):29:1–29:40, 2019.

3 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *SODA*, pages 1491–1505, 2015.

4 Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *ISAAC*, pages 77–86, 2009.

5 Nikhil Bansal, José R Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of operations research*, 31(1):31–49, 2006.

6 Nikhil Bansal, Marek Eliáš, and Arindam Khan. Improved approximation for vector bin packing. In *SODA*, pages 1561–1579. SIAM, 2016.

7 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *SODA*, pages 13–25, 2014.

8 Nikhil Bansal, Andrea Lodi, and Maxim Sviridenko. A tale of two dimensional bin packing. In *FOCS*, pages 657–666, 2005.

9 Alberto Caprara. Packing d-dimensional bins in d stages. *Mathematics of Operations Research*, 33(1):203–215, 2008.

10 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

11 Edward G Coffman, Jr, Michael R Garey, David S Johnson, and Robert Endre Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

12 W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

13 Max A Deppert, Klaus Jansen, Arindam Khan, Malin Rau, and Malte Tutas. Peak demand minimization via sliced strip packing. In *APPROX/RANDOM*, volume 207, pages 21:1–21:24, 2021.

14 Florian Diedrich, Rolf Harren, Klaus Jansen, Ralf Thöle, and Henning Thomas. Approximation algorithms for 3d orthogonal knapsack. *Journal of Computer Science and Technology*, 23(5):749, 2008.

15 Alan M Frieze, Michael RB Clarke, et al. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, 15(1):100–109, 1984.

16 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, Klaus Jansen, Arindam Khan, and Malin Rau. A tight $(3/2+\varepsilon)$ approximation for skewed strip packing. In *APPROX/RANDOM*, volume 176, pages 44:1–44:18, 2020.

17 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, and Kamyar Khodamoradi. Approximation algorithms for demand strip packing. In Mary Wootters and Laura Sanità, editors, *APPROX/RANDOM*, volume 207 of *LIPIcs*, pages 20:1–20:24, 2021.

18 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *FOCS*, pages 260–271, 2017.

19 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved approximation algorithms for 2-dimensional knapsack: Packing into multiple l-shapes, spirals, and more. In *SoCG*, volume 189 of *LIPIcs*, pages 39:1–39:17, 2021.

20 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A $(2+\varepsilon)$-approximation algorithm for maximum independent set of rectangles. *CoRR*, abs/2106.00623, 2021. `arXiv:2106.00623`.

21 Rolf Harren. Approximation algorithms for orthogonal packing problems for hypercubes. *Theoretical Computer Science*, 410(44):4504–4532, 2009.

22 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob Van Stee. A $(5/3+\varepsilon)$-approximation for strip packing. *Computational Geometry*, 47(2):248–267, 2014.

23 Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. In *SODA*, pages 79–98, 2017.

24 Klaus Jansen, Arindam Khan, Marvin Lira, and K. V. N. Sreenivas. A ptas for packing hypercubes into a knapsack, 2022. doi:10.48550/ARXIV.2202.11902.

25 Klaus Jansen and Malin Rau. Closing the gap for pseudo-polynomial strip packing. In *ESA*, volume 144 of *LIPIcs*, pages 62:1–62:14, 2019.

26 Klaus Jansen and Roberto Solis-Oba. Packing squares with profits. *SIAM Journal on Discrete Mathematics*, 26:263–279, January 2012. doi:10.1137/080717110.

27 Klaus Jansen and Guochuan Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 47(3):323–342, 2007.

28 Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000.

29 Arindam Khan. *Approximation algorithms for multidimensional bin packing*. PhD thesis, Georgia Institute of Technology, 2015.

30 Arindam Khan, Arnab Maiti, Amatya Sharma, and Andreas Wiese. On guillotine separable packings for the two-dimensional geometric knapsack problem. In *SoCG*, volume 189, pages 48:1–48:17, 2021.

31 Arindam Khan and Madhusudhan Reddy Pittu. On guillotine separability of squares and rectangles. In *APPROX/RANDOM*, pages 47:1–47:22, 2020.

32 Arindam Khan and Eklavya Sharma. Tight Approximation Algorithms For Geometric Bin Packing with Skewed Items. In *APPROX/RANDOM*, pages 22:1–22:23, 2021.

33 Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas. Approximation algorithms for generalized multidimensional knapsack. *CoRR*, abs/2102.05854, 2021.

34 Arindam Khan, Eklavya Sharma, and K. V. N. Sreenivas. Geometry meets vectors: Approximation algorithms for multidimensional packing. *CoRR*, abs/2106.13951, 2021. arXiv:2106.13951.

35 E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

36 Joseph Y. T. Leung, Tommy W. Tam, C. S. Wong, Gilbert H. Young, and Francis Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

37 Yiping Lu, Danny Z Chen, and Jianzhong Cha. Packing cubes into a cube in ($d > 3$)-dimensions. In *COCOON*, pages 264–276. Springer, 2015.

38 Yiping Lu, Danny Z Chen, and Jianzhong Cha. Packing cubes into a cube is np-complete in the strong sense. *Journal of Combinatorial Optimization*, 29(1):197–215, 2015.

39 Deval Patel, Arindam Khan, and Anand Louis. Group fairness for knapsack problems. In *AAMAS*, pages 1001–1009, 2021.

40 Sai Sandeep. Almost optimal inapproximability of multidimensional packing problems. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 245–256, 2022. doi:10.1109/FOCS52979.2021.00033.

41 Eklavya Sharma. Harmonic Algorithms for Packing d-Dimensional Cuboids into Bins. In *FSTTCS*, pages 32:1–32:22, 2021.

42 David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1):461–474, 1993.

A Faster Interior-Point Method for Sum-Of-Squares Optimization

Shunhua Jiang
Columbia University, New York, NY, USA

Bento Natura
London School of Economics, UK

Omri Weinstein
The Hebrew University, Jerusalem, Israel
Columbia University, New York, NY, USA

Abstract

We present a faster interior-point method for optimizing sum-of-squares (SOS) polynomials, which are a central tool in polynomial optimization and capture convex programming in the Lasserre hierarchy. Let $p = \sum_i q_i^2$ be an n-variate SOS polynomial of degree $2d$. Denoting by $L := \binom{n+d}{d}$ and $U := \binom{n+2d}{2d}$ the dimensions of the vector spaces in which q_i's and p live respectively, our algorithm runs in time $\tilde{O}(LU^{1.87})$. This is polynomially faster than state-of-art SOS and semidefinite programming solvers [16, 15, 27], which achieve runtime $\tilde{O}(L^{0.5}\min\{U^{2.37}, L^{4.24}\})$.

The centerpiece of our algorithm is a dynamic data structure for maintaining the inverse of the Hessian of the SOS barrier function under the *polynomial interpolant basis* [27], which efficiently extends to multivariate SOS optimization, and requires maintaining spectral approximations to low-rank perturbations of *elementwise (Hadamard) products*. This is the main challenge and departure from recent IPM breakthroughs using inverse-maintenance, where low-rank updates to the slack matrix readily imply the same for the Hessian matrix.

2012 ACM Subject Classification Mathematics of computing → Continuous functions; Mathematics of computing → Convex optimization; Mathematics of computing → Semidefinite programming; Mathematics of computing → Stochastic control and optimization

Keywords and phrases Interior Point Methods, Sum-of-squares Optimization, Dynamic Matrix Inverse

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.79

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2202.08489`

Funding *Shunhua Jiang*: Supported by NSF CAREER award CCF-1844887.
Bento Natura: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 757481–ScaleOpt).
Omri Weinstein: Supported by NSF CAREER award CCF-1844887 and ISF grant #3011005535.

Acknowledgements The second author would like to thank Vissarion Fisikopoulos and Elias Tsigaridas for introducing him from a practical perspective to Sum-of-Squares Optimization under the interpolant basis.

1 Introduction

Polynomial optimization is a fundamental problem in many areas of applied mathematics, operations research, and theoretical computer science, including combinatorial optimization [5, 36, 4], statistical estimation [13, 14], experimental design [26], control theory [12], signal

processing [31], power systems engineering [11], discrete geometry [2, 3] and computational algebraic geometry [20]. In the most basic formulation, we are given a collection of k real n-variate polynomials $f_1, \cdots, f_k$ and an objective function $f : \mathbb{R}^n \to \mathbb{R}$, and the goal is to minimize f over the set $\mathcal{S} := \{t \in \mathbb{R}^n \mid \forall i \in \{1, \cdots, k\} : f_i(t) \geq 0\}$, that is, to find

$$\inf_{t \in \mathbb{R}^n} \{f(t) \mid t \in \mathcal{S}\}, \tag{1}$$

which is equivalent to checking polynomial nonnegativity $\sup_{c \in \mathbb{R}}\{c \mid f(t) - c \geq 0,\ \forall t \in \mathcal{S}\}$. This is then equivalent to computing $\sup_{c \in \mathbb{R}}\{c \mid f - c \in \mathcal{K}(\mathcal{S})\}$, where $\mathcal{K}(\mathcal{S})$ denotes the convex cone of all polynomials of degree at most $\deg(f)$ that are non-negative on the set $\mathcal{S}$. This is an instance of the more general conic programming:

$$\min_{x \in \mathbb{R}^N} c^\top x \text{ s.t. } Ax = b, x \in \mathcal{K}, \tag{2}$$

where $\mathcal{K} \subset \mathbb{R}^N$ is some convex cone[1]. The conic optimization problem over the cone $\mathcal{K}(\mathcal{S})$ is intractable in general because there is no simple characterization of $\mathcal{K}(\mathcal{S})$. Nevertheless, there always exists an increasing family of convex cones of *weighted sum-of-squares* polynomials that converges to any such cone $\mathcal{K}(\mathcal{S})$.

We first introduce the notion of *sum-of-squares* (SOS) polynomials: Denoting by $\mathcal{V}_{n,d}$ the vector space of all n-variate polynomials of (total) degree at most d, a polynomial $p \in \mathcal{V}_{n,2d}$ is said to be *sum-of-squares* (SOS) if it can be written as a finite sum of square polynomials, i.e., there exist $q_1, \cdots, q_\ell$ such that $p = \sum_{i=1}^{\ell} q_i^2$. The set $\Sigma_{n,2d}$ of SOS polynomials of degree at most $2d$ is a (proper) cone contained in $\mathcal{V}_{n,2d}$, of dimension $U := \dim(\mathcal{V}_{n,2d}) = \binom{n+2d}{2d}$, as the vector space $\mathcal{V}_{n,2d}$ is isomorphic to $\mathbb{R}^U$. If p can be written as $p = \sum_{i=1}^{k} f_i s_i$ for $s_1 \in \Sigma_{n,2d_1}, \cdots, \Sigma_{n,2d_k}$ and k nonzero polynomials $\mathbf{f} := (f_1, \cdots, f_k)$, then it is said to be *weighted sum-of-squares* (WSOS).

Putinar's Positivstellensatz [29] states that under mild conditions, any polynomial p that is non-negative on $\mathcal{S}$ can be written as a WSOS polynomial $\sum_{i=1}^{k} f_i s_i$, albeit with (potentially) *unbounded degree* s_i's. In WSOS optimization we consider sum-of-squares polynomials s_i with bounded degree, so the hierarchy of WSOS optimization with increasing degree (known as the Lasserre hierarchy) can be viewed as a tool for approximating general polynomial optimization. For more details of this approximation scheme for polynomial optimization, we refer the readers to the textbooks [19, 7].

This paper concerns algorithms for *(W)SOS optimization*, which is the conic optimization program (2) where the underlying cone $\mathcal{K}$ is the (W)SOS cone:

$$\min_{x \in \mathbb{R}^U} c^\top x \text{ s.t. } Ax = b, x \in \Sigma_{n,2d}, \tag{3}$$

where $x \in \mathcal{V}_{n,2d}$ is the vector of coefficients which encodes the polynomial. Henceforth, we focus on the case where $\mathcal{K} = \Sigma_{n,2d}$ is the SOS cone. See our full version for how to extend our algorithm for SOS optimization to WSOS.

The computational complexity of solving Problem 3 naturally depends on the dimensions

$$L := \dim(\mathcal{V}_{n,d}) = \binom{n+d}{d} \quad , \quad U := \dim(\mathcal{V}_{n,2d}) = \binom{n+2d}{2d} \tag{4}$$

of the underlying vector spaces (Note that $L \leq U \leq L^2$). We now turn to explain the previous approaches for SOS optimization solvers.

[1] A subset $\mathcal{K} \subset \mathbb{R}^N$ is a convex cone if $\forall\, x, y \in \mathcal{K}$ and $\alpha, \beta \in \mathbb{R}_+$, $\alpha x + \beta y \in \mathcal{K}$.

SOS Optimization as SDPs

A fundamental fact is that the *dual* SOS cone is a *slice of the SDP cone* [24]. More formally, for any *fixed bases* $\mathbf{p} = (p_1, p_2, \cdots, p_L)$ and $\mathbf{q} = (q_1, q_2, \cdots, q_U)$ to $\mathcal{V}_{n,d}$ and $\mathcal{V}_{n,2d}$ respectively, there exists a unique linear mapping $\Lambda : \mathbb{R}^U \to \mathbb{R}^{L\times L}$ satisfying

$$\Lambda(\mathbf{q}(t)) = \mathbf{p}(t)\mathbf{p}(t)^\top, \quad \forall t \in \mathbb{R}^n. \tag{5}$$

Here we define $\mathbf{p}(t) = (p_1(t), p_2(t), \cdots, p_L(t))^\top$ and $\mathbf{q}(t) = (q_1(t), q_2(t), \cdots, q_U(t))^\top$. An equivalent way to view the definition of Λ in (5) is as follows: For polynomials $p_i, p_j \in \mathbf{p}$ there are unique coefficients λ_{iju} such that $p_i p_j = \sum_{u\in U} \lambda_{iju} q_u$. These λ_{iju} define the mapping Λ unambiguously.

This in turn implies that a polynomial $s \in \mathcal{V}_{n,2d}$ (we view s as a vector in $\mathbb{R}^U$ that corresponds to its coefficients over the basis $\mathbf{q}$) is in the dual SOS cone $\Sigma^*_{n,2d}$ if and only if $\Lambda(s)$ is a *positive semidefinite* (PSD) matrix (proved by [24], see Theorem 10 for details). As [27] recently observed, the choice of the bases $\mathbf{p}, \mathbf{q}$ crucially affects the complexity of the optimization problem, more on this below.

Equation (5) implies the well-known fact that optimization over SOS polynomials (3) can be reduced to semidefinite programming

$$\min_{X\succeq 0}\{\langle C, X\rangle \mid \mathrm{tr}(A_i X) = b_i, \ \forall i \in [m]\}, \tag{SDP}$$

and can thus be solved using off-the-shelf SDP solvers. However, despite recent breakthroughs on the runtime of general SDP solvers via *interior-point methods* (IPMs) [16, 15], this SDP reformulation of (3) does not scale well for moderately large degrees, i.e., whenever $U \ll L^2$ in (4). This is because the SDP reformulation always incurs a factor of at least L^2, even when $U \ll L^2$, as this is the SDP variable size (the PSD matrix X has size $L \times L$). Indeed, for the current fast-matrix-multiplication (FMM) exponent $\omega \approx 2.37$ [21, 1], the running time of state-of-the-art SDP solvers [16, 15] for SOS optimization (Problem 3) is[2]

$$\widetilde{O}\left(L^{0.5} \cdot \min\{UL^2 + U^{2.37},\ L^{4.24}\}\right). \tag{6}$$

An alternative approach is to solve Problem 3 directly by designing an *ad-hoc* IPM for the dual SOS cone, avoiding the blowup in the SDP reformulation. This was exactly the motivation of [27]. Like all aforementioned SDP solvers, [27]'s SOS solver is based on IPMs [25], which iteratively minimize the original objective function plus a barrier function via Newton steps. When applied to the SOS Problem (3), *the choice of the specific bases* $\mathbf{p}, \mathbf{q}$ *crucially affects the structure of the (Hessian of the) barrier function* $F(s) = F(\Lambda(s))$, and hence the cost-per-iteration of the IPM. As such, choosing a "good" and efficient basis is key to a fast algorithm for (3). One of the main contribution of [27] is an efficient basis for the SOS cone, which efficiently scales to multivariate SOS, yielding an IPM whose total runtime is

$$\widetilde{O}\left(L^{0.5}U^{\omega}\right) \approx \widetilde{O}\left(L^{0.5}U^{2.37}\right). \tag{7}$$

Our main result is a polynomially faster IPM for Problem 3:

▶ **Theorem 1** (Main Result, Informal version of Theorem 16). *With current FMM exponent, there is an algorithm for solving Problem* (3)*, whose total running time is* $\widetilde{O}\left(LU^{1.87}\right)$.

Indeed, this runtime is polynomially faster than (7) and (6), as shown in Figure 1. We now turn to elaborate on the technical approach for proving Theorem 1.

[2] We use $\widetilde{O}(\cdot)$ to hide $U^{o(1)}$ and $\log(1/\delta)$ factors.

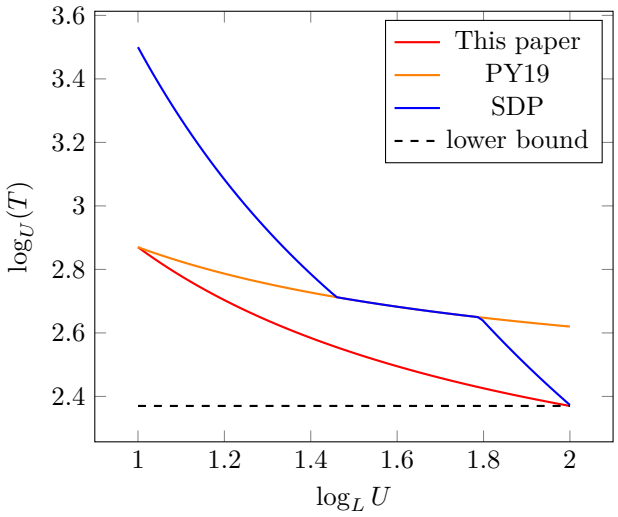


Figure 1 Overview of current running times of recent solvers for SOS. The lower bound bound stems from solving a linear system in U variables, i.e., $T = \Omega(U^\omega)$ where $\omega \approx 2.37$.

Faster IPMs via Inverse-Maintenance

Interior-Point Methods (IPMs [18, 30]) are a powerful class of second-order optimization algorithms for convex optimization, which essentially reduce a conic optimization problem (2) to solving *a sequence of slowly-changing linear systems* (via Newton steps). Since their discovery in the mid 80's, IPMs have emerged as the "gold-standard" of convex optimization, as they are known to converge fast in *both theory and practice* [35]. The main computational cost of IPMs is computing, in each iteration, the inverse of the Hessian of the underlying *barrier function* $F(s) = F(\Lambda(s))$, which naively costs at least U^ω time per iteration for the SOS optimization problem [27]. A recent influential line of work [9, 16], inspired by [37]'s seminal work, has demonstrated that *dynamically maintaining* the inverse of the Hessian matrix under *low-rank* updates using clever *data structures*, can lead to much cheaper *cost-per-iteration*. All of these results rely on a careful combination of dynamic data structures with the geometry (e.g., spectral approximation) of the underlying optimization method and barrier function. Similar techniques have been extended to other optimization problems as well [23, 38, 17, 33, 34]. This paper extends this line of work to SOS optimization.

Our Techniques

Following the framework of [27], we also choose the *polynomial interpolant basis* representation, which corresponds to a linear operator $\Lambda : \mathbb{R}^U \to \mathbb{R}^{L \times L}$ is $\Lambda(s) = P^\top \operatorname{diag}(s) P$, where $P \in \mathbb{R}^{U \times L}$ is the matrix whose entries are the evaluation of the Lagrange interpolation polynomials, through some unisolvent[3] set of points in $\mathcal{V}_{n,d}$ (see Section 3 for a formal definition). This basis induces the aforementioned convenient form of Λ, and generalizes to the multivariate case. The Hessian of the barrier function $F(s) = -\log\det(\Lambda(s))$ is given by

$$H(s) = \left(P(P^\top \operatorname{diag}(s) P)^{-1} P^\top\right)^{\circ 2} \in \mathbb{R}^{U \times U},$$

[3] Any set of points in $\mathbb{R}^n$ for which the evaluation of a polynomial in $\mathcal{V}_{n,d}$ on these points uniquely defines the polynomial.

where $A \circ B$ denotes the element-wise (Hadamard) product of two matrices. The main bottleneck of each iteration of IPMs is to compute the Hessian inverse $H(x)^{-1}$ of the Newton step, which naïvely takes $O(U^\omega)$ time.

In IPM theory, it has long been known that it suffices to compute a spectral approximation of the Hessian. We follow the "lazy update" framework in recent developments of LP and SDP solvers [9, 16], which batches together low-rank updates to $M := P(P^\top \operatorname{diag}(s)P)^{-1}P^\top$, where $\operatorname{rk}(M) = L$. In each iteration, we can compute a spectral approximation $M^{\text{new}} = M + UV^\top$, where U, V are low rank matrices with size $U \times r$ where $r \ll U$ is chosen to optimize the runtime. Since $\widetilde{M} \approx M$ implies that $\widetilde{M}^{\circ 2} \approx M^{\circ 2}$, this also gives a spectral approximation of the Hessian.

The main challenge here, compared to previous LP and SDP solvers [37, 22, 9, 16, 15], is that low-rank updates to M do not readily translate to a low-rank update to $(M^{\circ 2})^{-1}$, since Hadamard-products can *increase* the rank $\operatorname{rk}(A \circ B) \leq \operatorname{rk}(A) \cdot \operatorname{rk}(B)$, in contrast to standard matrix multiplication which does not increase the rank $\operatorname{rk}(AB) \leq \max\{\operatorname{rk}(A), \operatorname{rk}(B)\}$. This means that we cannot directly apply Woodbury's identity to efficiently update the inverse of the Hessian, which is the common approach in all aforementioned works. Instead, we employ the following property which relates rank-one Hadamard-product perturbations to standard matrix products

$$M \circ (u \cdot v^\top) = \operatorname{diag}(u) \cdot M \cdot \operatorname{diag}(v),$$

which means that we can translate the rank-r update of M into a rank-Lr update of $M^{\circ 2}$ for $r \leq L$. With some further calculations, applying Woodbury's identity on the resulting matrix, implies that we can compute $((M^{\text{new}})^{\circ 2})^{-1}$ in time

$$O\big(\mathcal{T}_{\text{mat}}(U, U, Lr)\big),$$

which is never worse than $\mathcal{T}_{\text{mat}}(U, U, U) = U^\omega$ as long as $r \leq U/L$. Modifying the amortization tools of [16] and [15], combined with basic spectral theory for Hadamard products, we show that our amortized cost per iteration is bounded by

$$O\big(U^2 + U^{\omega - 1/2} \cdot L^{1/2}\big),$$

which becomes $O\big(U^2 + U^{1.87} L^{0.5}\big)$ if we plug in the current matrix multiplication exponent.

2 Preliminaries

In this section we provide the definitions and the tools that we will use. For any integer $n > 0$, we define $[n] = \{1, 2, \cdots, n\}$. We use $\mathbb{R}_+$ and $\mathbb{R}_{\geq 0}$ to denote the set of positive and non-negative real numbers respectively. We use $0_n, 1_n \in \mathbb{R}^n$ to denote the all-zero and all-one vectors of size n.

Given a vector $v \in \mathbb{R}^n$, for any $m \leq n$, we use $v_{[:m]} \in \mathbb{R}^m$ to denote the first m entries of v. For a vector $v \in \mathbb{R}^n$, we use $\operatorname{diag}(v) \in \mathbb{R}^{n \times n}$ to denote the diagonal matrix whose diagonal entries are v. For a square matrix $A \in \mathbb{R}^{n \times n}$, we use $\operatorname{diag}(A) \in \mathbb{R}^n$ to denote the vector of the diagonal entries of A. We use $\operatorname{rk}(A)$ to denote the rank of a matrix A. We use $\ker(A)$ and $\operatorname{Im}(A)$ to denote the kernel space and the column space of A.

We say a matrix $A \in \mathbb{R}^{n \times n}$ is PSD (denoted as $A \succeq 0$) if A is symmetric and $x^\top A x \geq 0$ for all $x \in \mathbb{R}^n$. We use $\mathbb{S}^{n \times n}$ to denote the set of PSD matrices of size $n \times n$. The spectral norm of a matrix $A \in \mathbb{R}^{n \times d}$ is defined as $\|A\|_2 = \max_{x \in \mathbb{R}^d, \|x\|_2 = 1} \|Ax\|_2$. The Frobenius norm of A is defined as $\|A\|_F = \sqrt{\sum_{i \in [n]} \sum_{j \in [d]} A_{i,j}^2}$. For any PSD matrix $M \in \mathbb{S}^{n \times n}$, we define the M-norm as $\|x\|_M = \sqrt{x^\top M x}$, $\forall x \in \mathbb{R}^n$.

We use $\mathcal{T}_{\mathrm{mat}}(a,b,c)$ to denote the time to multiply two matrices of sizes $a \times b$ and $b \times c$. A basic fact of fast matrix multiplication is that $\mathcal{T}_{\mathrm{mat}}(a,b,c) = \mathcal{T}_{\mathrm{mat}}(b,c,a) = \mathcal{T}_{\mathrm{mat}}(c,a,b)$ (see e.g. [6]), and we will use these three terms interchangeably.

▶ **Fact 2** (Woodbury identity). *Let $A \in \mathbb{R}^{n\times n}, C \in \mathbb{R}^{k\times k}, U \in \mathbb{R}^{n\times k}, V \in \mathbb{R}^{k\times n}$ where A and C are invertible, then*

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}.$$

▶ **Definition 3** (Hadamard product). *For any two matrices $A, B \in \mathbb{R}^{m\times n}$, the Hadamard product $A \circ B$ is defined as*

$$(A\circ B)_{i,j} = A_{i,j}\cdot B_{i,j}, \quad \forall i \in [m], j \in [n].$$

We also use $A^{\circ 2}$ to denote $A \circ A$.

The Hadamard product has the following properties (the proofs are straightforward).

▶ **Fact 4** (Properties of Hadamard product). *For matrices $A, B \in \mathbb{R}^{m\times n}$, and vectors $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, we have the following properties.*
1. $x^\top (A\circ B)y = \mathrm{tr}[\mathrm{diag}(x)A\,\mathrm{diag}(y)B^\top]$,
2. $A \circ (x\cdot y^\top) = \mathrm{diag}(x)\cdot A\cdot \mathrm{diag}(y)$.

▶ **Definition 5** (Spectral approximation). *For any two symmetric matrices $A, \widetilde{A} \in \mathbb{R}^{n\times n}$, any parameter $\epsilon \in (0,1)$, we say $\widetilde{A}$ and A are ϵ-spectral approximation of each other, denoted as $\widetilde{A} \approx_\epsilon A$, if we have*

$$e^{-\epsilon}\cdot x^\top Ax \le x^\top \widetilde{A}x \le e^{\epsilon}\cdot x^\top Ax, \quad \forall x \in \mathbb{R}^n.$$

Spectral approximation has the following properties.

▶ **Fact 6** (Properties of spectral approximation). *For any two PSD matrices $A, \widetilde{A} \in \mathbb{R}^{n\times n}$, any parameter $\epsilon \in (0,1)$, if $\widetilde{A} \approx_\epsilon A$, then we have*
1. *$B^\top AB \approx_\epsilon B^\top \widetilde{A}B$, for any matrix $B \in \mathbb{R}^{n\times n}$.*
2. *If both A and $\widetilde{A}$ are invertible, then $A^{-1} \approx_\epsilon \widetilde{A}^{-1}$.*
3. *$e^{-\epsilon}\,\mathrm{tr}[A] \le \mathrm{tr}[\widetilde{A}] \le e^{\epsilon}\,\mathrm{tr}[A]$.*
4. *$\widetilde{A}^{\circ 2} \approx_{2\epsilon} A^{\circ 2}$.*

Proof. The proofs of the first three claims are straightforward. We only prove the last claim.

For any vector $x \in \mathbb{R}^n$, we have

$$\begin{aligned}
x^\top A^{\circ 2}x =\ & \mathrm{tr}[\mathrm{diag}(x)A\,\mathrm{diag}(x)A] \\
=\ & \mathrm{tr}[A^{1/2}\,\mathrm{diag}(x)A\,\mathrm{diag}(x)A^{1/2}] \\
\le\ & e^{\epsilon}\cdot \mathrm{tr}[A^{1/2}\,\mathrm{diag}(x)\widetilde{A}\,\mathrm{diag}(x)A^{1/2}] \\
=\ & e^{\epsilon}\cdot \mathrm{tr}[\widetilde{A}^{1/2}\,\mathrm{diag}(x)A\,\mathrm{diag}(x)\widetilde{A}^{1/2}] \\
\le\ & e^{2\epsilon}\cdot \mathrm{tr}[\widetilde{A}^{1/2}\,\mathrm{diag}(x)\widetilde{A}\,\mathrm{diag}(x)\widetilde{A}^{1/2}] = e^{2\epsilon}\cdot x^\top \widetilde{A}^{\circ 2}x
\end{aligned}$$

where the first step follows from Fact 4, the second and the fourth steps follow from the trace invariance under cyclic permutations and the fact that $A^{1/2}$ exists when A is PSD, the third and the fifth steps follow from Part 3 of this fact.

Similarly we can prove $x^\top A^{\circ 2}x \ge e^{-2\epsilon}\cdot x^\top \widetilde{A}^{\circ 2}x$. Thus we have $A^{\circ 2} \approx_{2\epsilon} \widetilde{A}^{\circ 2}$. ◀

3 Background of sum-of-squares optimization

In this section we provide the background of sum-of-squares optimization. We refer the readers to [28, 27] for more details.

▶ **Definition 7** (Polynomial space). *We use $\mathcal{V}_{n,d}$ to denote the set of n-variate polynomials over the reals of degree at most d, where the degree means the total degree, i.e., the degree of $x_1^{d_1}\cdots x_n^{d_n}$ is $\sum_{i=1}^n d_i$.*

▶ **Definition 8** (Degree of polynomial space). *We define $L := \dim(\mathcal{V}_{n,d}) = \binom{n+d}{n}$ and $U := \dim(\mathcal{V}_{n,2d}) = \binom{n+2d}{n}$.*

After fixing a basis $(p_1, p_2, \cdots, p_L)$ of $\mathcal{V}_{n,d}$, there exists a one-to-one correspondence between any polynomial $p = \sum_{i=1}^L x_i \cdot p_i \in \mathcal{V}_{n,d}$ and the vector $[x_1, x_2, \cdots, x_L] \in \mathbb{R}^L$. From now on when the basis is clear from context, we will use $\mathcal{V}_{n,d}$ and $\mathbb{R}^L$ interchangeably, and similarly $\mathcal{V}_{n,2d}$ and $\mathbb{R}^U$ interchangeably.

▶ **Definition 9** (SOS polynomials). *A polynomial $p \in \mathcal{V}_{n,2d}$ is said to be a sum-of-squares (SOS) polynomial if p can be written as a sum of squares of polynomials, i.e. $p = \sum_{i=1}^M q_i^2$ for some $M \in \mathbb{N}$ and polynomials $q_1, q_2, \cdots, q_M \in \mathcal{V}_{n,d}$.*

We use $\Sigma_{n,2d}$ to denote the set of n-variate SOS polynomials of degree at most $2d$.

The set $\Sigma_{n,2d}$ is a closed convex and pointed cone in $\mathcal{V}_{n,2d}$ with non-empty interior (Theorem 17.1 of [24]). The SOS optimization problem requires the variable $x \in \mathbb{R}^U$ to be in the SOS cone, and it is a special case of conic programming. Given a constraint matrix $A \in \mathbb{R}^{m\times U}$ where $m \le U$, and $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^U$, the SOS optimization can be written in the following primal-dual formulation:

$$\begin{aligned} \text{Primal:}\quad \min\ & \langle c, x\rangle & \text{Dual:}\quad \max\ & \langle y, b\rangle \\ \text{s.t.}\ & Ax = b & \text{s.t.}\ & A^\top y + s = c \\ & x \in \Sigma_{n,2d}\,, & & s \in \Sigma_{n,2d}^*\,. \end{aligned} \tag{SOS}$$

Here $\Sigma_{n,2d}^* := \{s \in \mathbb{R}^U \mid s^\top x \ge 0,\ \forall x \in \Sigma_{n,2d}\}$ denotes the dual cone of $\Sigma_{n,2d}$.

Nesterov in [24] noted that the dual SOS cone allows the following characterization.

▶ **Theorem 10** (Dual cone characterization, Theorem 17.1 of [24]). *For any ordered bases $\mathbf{p} = (p_1, \ldots, p_L)$ and $\mathbf{q} = (q_1, \ldots, q_U)$ of $\mathcal{V}_{n,d}$ and $\mathcal{V}_{n,2d}$, let $\Lambda : \mathbb{R}^U \to \mathbb{R}^{L\times L}$ be the unique linear mapping satisfying $\Lambda(\mathbf{q}) = \mathbf{p}\mathbf{p}^\top$.*[4] *Then the dual cone $\Sigma_{n,2d}^*$ admits the characterization under the bases $\mathbf{p}$ and $\mathbf{q}$:*

$$\Sigma_{n,2d}^* = \big\{s \in \mathbb{R}^U \mid \Lambda(s) \succeq 0\big\}. \tag{8}$$

As barrier functions for the cone of positive semidefinite matrices are well-known, this also gives rise to a barrier function for the dual SOS cone. With the standard log-det barrier for the semidefinite cone, the following function $F : \Sigma_{n,2d}^* \to \mathbb{R}$ is a barrier function for $\Sigma_{n,2d}^*$:

$$F(s) = -\log\det(\Lambda(s)).$$

Furthermore, the barrier parameter ν_F of $F(s)$ is bounded by the barrier parameter L of the original log-det barrier function ([24]).

[4] This equation means $\forall t \in \mathbb{R}^n$, $\Lambda([q_1(t), \cdots, q_U(t)]^\top) = [p_1(t), \cdots, p_L(t)]^\top \cdot [p_1(t), \cdots, p_L(t)]$.

Interpolant basis

The barrier function depends on the choice of the basis for both $\mathcal{V}_{n,d}$ and $\mathcal{V}_{n,2d}$, as the linear map Λ depends on these two bases. We follow the approach of [27] and focus on the so-called *interpolant* bases, which generalises well to multivariate polynomials and is numerically stable.

For the vector space $\mathcal{V}_{n,2d}$, consider a set of *unisolvent* points $\mathcal{T} = \{t_1, t_2, \cdots, t_U\} \subseteq \mathbb{R}^n$, which is a set points such that every polynomial in $\mathcal{V}_{n,2d}$ is uniquely determined by its values on the points in $\mathcal{T}$. For univariate polynomials any set of U points suffices, but this does not hold anymore for the multivariate case. To also ensure numerical stability, the so called (approximate) Fekete points can be used as unisolvent points [32, 8].

The interpolant basis is defined as follows. Let us fix a set of unisolvent points $\mathcal{T} = \{t_1, t_2, \cdots, t_U\} \subseteq \mathbb{R}^n$. Now every $t_u \in \mathcal{T}$ implies a Lagrange polynomial q_u which is the unique polynomial that satisfies $q_u(t_u) = 1$ and $q_u(t_v) = 0$ for all $t_v \neq t_u \in \mathcal{T}$. The Lagrange polynomials form a basis $\mathbf{q} = (q_1, \cdots, q_U)$ of $\mathcal{V}_{n,2d}$. Choose any basis $\mathbf{p} = (p_1, \ldots, p_L)$ of $\mathcal{V}_{n,d}$. Define the matrix $P \in \mathbb{R}^{U \times L}$ as

$$P_{u,\ell} = p_\ell(t_u), \quad \forall u \in [U], \ell \in [L].$$

By the definition of the Lagrange polynomials, $p_i p_j = \sum_{u=1}^{U} p_i(t_u) p_j(t_u) q_u$, so we have $\mathbf{p}\mathbf{p}^\top = P^\top \operatorname{diag}(\mathbf{q}) P$. Thus under the bases $\mathbf{p}$ and $\mathbf{q}$, the linear map $\Lambda : \mathbb{R}^U \to \mathbb{R}^{L \times L}$ takes on the following convenient form:

$$\Lambda(s) = P^\top \operatorname{diag}(s) P. \tag{9}$$

For more details on how to pick the unisolvent points and how to construct P, we refer the readers to [27] and the references therein.

4 Algorithm

Since in this paper we focus on the theoretical running time of the algorithm, for simplicity we use the barrier method (see e.g. [30, Chapter 2]) instead of the more sophisticated Skajaa–Ye Algorithm used by [27].

The dual formulation of (SOS) is equivalent to the following optimization problem

$$\min -b^\top y \quad s.t. \;\; y \in \overline{D}_F,$$

where with an abuse of the notation we define $F : \mathbb{R}^m \to \mathbb{R}_+$ to be the barrier function

$$F(y) = -\log\det(\Lambda(c - A^\top y)) \tag{10}$$

for $c \in \mathbb{R}^U$, and $A \in \mathbb{R}^{m \times U}$, and $\Lambda(s) = P^\top \operatorname{diag}(s) P$ is the linear operator defined in Eq (9). $D_F \subseteq \mathbb{R}^m$ is the domain of F, and $\overline{D}_F$ is the closure of D_F.

The barrier parameter of the barrier function F is $\nu_F = L$. The gradient and the Hessian of the barrier function F are (define $s := c - A^\top y$):

$$g(y) = A \cdot \operatorname{diag}\left(P\left(P^\top \operatorname{diag}(s) P\right)^{-1} P^\top\right),$$
$$H(y) = A \cdot \left(P\left(P^\top \operatorname{diag}(s) P\right)^{-1} P^\top\right)^{\circ 2} \cdot A^\top.$$

For any $\eta > 0$, define a function $F_\eta : \mathbb{R}^m \to \mathbb{R}$:

$$F_\eta(y) = -\eta \cdot b^\top y + F(y).$$

The gradient and the Hessian of $F_\eta(y)$ are:

$$g_\eta(y) = -\eta \cdot b + A \cdot \operatorname{diag}\left(P\left(P^\top \operatorname{diag}(s) P\right)^{-1} P^\top\right),$$
$$H_\eta(y) = A \cdot \left(P\left(P^\top \operatorname{diag}(s) P\right)^{-1} P^\top\right)^{\circ 2} \cdot A^\top.$$

Note that $H_\eta(y) = H(y)$ for any η.

In each iteration the barrier method increases η by a factor of $1 + \Theta(\frac{1}{\sqrt{L}})$, and it performs a Newton step

$$y \leftarrow y - H_\eta(y)^{-1} \cdot g_\eta(y).$$

By standard IPM theory it suffices to use a spectral approximation of the Hessian matrix in the Newton step. For more details see e.g. [30].

The main technical part of our algorithm is to efficiently maintain a matrix N that is the spectral approximation of the inverse of the Hessian matrix. To do this, we maintain another matrix $\widetilde{S}$ that is a spectral approximation of $S := P^\top \operatorname{diag}(s) P$, and we use the subroutine LowRankUpdate(Algorithm 3, Lemma 14) to update $\widetilde{S}$. After $\widetilde{S}$ is updated, we use another subroutine UpdateHessianInv (Algorithm 2, Lemma 11) to update N. A complete description of our algorithm can be found in Algorithm 1.

5 Updating Hessian inverse efficiently

In this section we prove how to update the Hessian inverse efficiently. We present the algorithm UpdateHessianInv in Algorithm 2.

▶ **Lemma 11** (Hessian inverse update). *In the algorithm UpdateHessianInv (Algorithm 2), the inputs are the maintained matrices T, N and the updates $V_1, V_2 \in \mathbb{R}^{L \times r}$ where r satisfies $Lr \le U$. The inputs satisfy that for some $\widetilde{S} \in \mathbb{S}^{L \times L}$,*

$$T = \widetilde{S}^{-1} \in \mathbb{R}^{L \times L},$$
$$N = \left(A \cdot (P \widetilde{S}^{-1} P^\top)^{\circ 2} \cdot A^\top\right)^{-1} \in \mathbb{R}^{m \times m},$$

Let $\widetilde{S}^{\text{new}} = \widetilde{S} + V_1 V_2^\top$. The algorithm outputs two matrices T^{new}, N^{new} such that

$$T^{\text{new}} = (\widetilde{S}^{\text{new}})^{-1} \in \mathbb{R}^{L \times L},$$
$$N^{\text{new}} = \left(A \cdot (P (\widetilde{S}^{\text{new}})^{-1} P^\top)^{\circ 2} \cdot A^\top\right)^{-1} \in \mathbb{R}^{m \times m}.$$

Furthermore, the algorithm takes $O(\mathcal{T}_{\text{mat}}(U, U, Lr))$ time.

Proof. We first prove the correctness by analyzing each step of the algorithm.

Step 1. Compute $\overline{V_1}, \overline{V_2} \in \mathbb{R}^{L \times r}$ and $T^{\text{new}} \in \mathbb{R}^{L \times L}$.

$$\begin{aligned} T^{\text{new}} &= T + \overline{V_1} \cdot \overline{V_2}^\top \\ &= T - T V_1 \cdot (I + V_2^\top T V_1)^{-1} \cdot V_2^\top T^\top \\ &= (\widetilde{S} + V_1 V_2^\top)^{-1} = (\widetilde{S}^{\text{new}})^{-1}, \end{aligned}$$

where the first two steps follow from algorithm description, the third step follows from the Woodbury identity (Fact 2) and $T = \widetilde{S}^{-1}$.

Thus T^{new} satisfies the requirement of the output.

Algorithm 1 Main SOS algorithm.

Parameters: $\delta \in (0,1)$, $\epsilon_N \in (0, 0.05)$, $\alpha = \frac{\epsilon_N}{20\sqrt{L}}$, $t = 40\epsilon_N^{-1}\sqrt{L}\log(L/\delta)$.
Input : $A \in \mathbb{R}^{m \times U}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^U$
Output : A near feasible and optimal solution.

Construct $P \in \mathbb{R}^{U \times L}$ of the interpolant basis. Convert A, b, c to the interpolant basis.
Use Lemma 22 to obtain a modified dual SOS optimization problem which has an initial solution $(y, s) \in \mathbb{R}^m \times \mathbb{R}^U$ that is optimal for F_η, where $\eta = 1$.
$\widetilde{S} \leftarrow S \leftarrow P^\top \operatorname{diag}(s) P$; // $\widetilde{S}, S \in \mathbb{R}^{L \times L}$
$T \leftarrow S^{-1}$; // $T \in \mathbb{R}^{L \times L}$
$N \leftarrow \left(A(PTP^\top)^{\circ 2} A^\top\right)^{-1}$; // $N \in \mathbb{R}^{m \times m}$
$g \leftarrow -\eta \cdot b + A \cdot \operatorname{diag}\left(P(P^\top \operatorname{diag}(s) P)^{-1} P^\top\right)$; // $g \in \mathbb{R}^m$
for $i = 1, 2, \cdots, t$ **do**
  $\delta_y \leftarrow -N \cdot g$; // $\delta_y \in \mathbb{R}^m$
  $y^{\mathrm{new}} \leftarrow y + \delta_y$; // $y^{\mathrm{new}} \in \mathbb{R}^m$
  $s^{\mathrm{new}} \leftarrow c - A^\top y^{\mathrm{new}}$; // $s^{\mathrm{new}} \in \mathbb{R}^U$
  $\eta^{\mathrm{new}} \leftarrow \eta \cdot (1 + \alpha)$;
  $S^{\mathrm{new}} \leftarrow P^\top \operatorname{diag}(s^{\mathrm{new}}) P$; // $S^{\mathrm{new}} \in \mathbb{R}^{L \times L}$
  $\widetilde{S}^{\mathrm{new}}, V_1, V_2 \leftarrow \textsc{LowRankUpdate}(S^{\mathrm{new}}, \widetilde{S})$;
    // Lemma 14, $\widetilde{S}^{\mathrm{new}} \in \mathbb{R}^{L \times L}$, $V_1, V_2 \in \mathbb{R}^{L \times r_i}$ or $V_1 = V_2 =$ null
  if $V_1 = V_2 =$ null **then**
    $T^{\mathrm{new}} \leftarrow (\widetilde{S}^{\mathrm{new}})^{-1}$; // $T^{\mathrm{new}} \in \mathbb{R}^{L \times L}$
    $N^{\mathrm{new}} \leftarrow \left(A \cdot (PT^{\mathrm{new}} P^\top)^{\circ 2} \cdot A^\top\right)^{-1}$; // $N^{\mathrm{new}} \in \mathbb{R}^{m \times m}$
  else
    $T^{\mathrm{new}}, N^{\mathrm{new}} \leftarrow \textsc{UpdateHessianInv}(T, N, V_1, V_2)$;
      // Lemma 11, $T^{\mathrm{new}} \in \mathbb{R}^{L \times L}, N^{\mathrm{new}} \in \mathbb{R}^{m \times m}$
  $g^{\mathrm{new}} \leftarrow -\eta^{\mathrm{new}} \cdot b + A \cdot \operatorname{diag}\left(P\left(P^\top \operatorname{diag}(s^{\mathrm{new}}) P\right)^{-1} P^\top\right)$; // $g^{\mathrm{new}} \in \mathbb{R}^m$
  $(\eta, y, s, \widetilde{S}, T, N, g) \leftarrow (\eta^{\mathrm{new}}, y^{\mathrm{new}}, s^{\mathrm{new}}, \widetilde{S}^{\mathrm{new}}, T^{\mathrm{new}}, N^{\mathrm{new}}, g^{\mathrm{new}})$;
return (y, s)

Step 2. Compute $Y', Z' \in \mathbb{R}^{U \times (L+r)}$ and $Y, Z \in \mathbb{R}^{U \times (L+r)r}$. We prove that Y and Z satisfy $(PTP^\top)^{\circ 2} + Y \cdot Z^\top = (PT^{\mathrm{new}} P^\top)^{\circ 2}$:

$$\begin{aligned}
(PTP^\top)^{\circ 2} + Y \cdot Z^\top &= (PTP^\top)^{\circ 2} + \sum_{i=1}^r \operatorname{diag}(u_i) \cdot \left(Y' \cdot (Z')^\top\right) \cdot \operatorname{diag}(v_i) \\
&= (PTP^\top)^{\circ 2} + \left(Y' \cdot (Z')^\top\right) \circ \left(\sum_{i=1}^r u_i \cdot v_i^\top\right) \\
&= (PTP^\top)^{\circ 2} + \left(2PTP^\top + (P\overline{V_1}) \cdot (P\overline{V_2})^\top\right) \circ \left((P\overline{V_1}) \cdot (P\overline{V_2})^\top\right) \\
&= \left(PTP^\top + (P\overline{V_1}) \cdot (P\overline{V_2})^\top\right)^{\circ 2} \\
&= (PT^{\mathrm{new}} P^\top)^{\circ 2},
\end{aligned}$$

where the first step follows from the algorithm description of Y and Z, the second step follows from Part 2 of Fact 4 that $\operatorname{diag}(x) \cdot A \cdot \operatorname{diag}(y) = A \circ (x \cdot y^\top)$, the third step follows from $Y' \cdot (Z')^\top = 2PTP^\top + (P\overline{V_1}) \cdot (P\overline{V_2})$ and $(P\overline{V_1}) \cdot (P\overline{V_2}) = \sum_{i=1}^r u_i \cdot v_i^\top$ (see algorithm description of Y' and Z'), the last step follows from $T^{\mathrm{new}} = T + \overline{V_1} \cdot \overline{V_2}^\top$.

Algorithm 2 UpdateHessianInv.

Input : $T \in \mathbb{R}^{L\times L}$, $N \in \mathbb{R}^{m\times m}$, $V_1, V_2 \in \mathbb{R}^{L\times r}$
Output: $T^{\text{new}} \in \mathbb{R}^{L\times L}$, $N^{\text{new}} \in \mathbb{R}^{m\times m}$

1 // Step 1
2 $\overline{V_1} \leftarrow -TV_1 \cdot (I + V_2^\top T V_1)^{-1}$; // $\overline{V_1} \in \mathbb{R}^{L\times r}$
3 $\overline{V_2} \leftarrow TV_2$; // $\overline{V_2} \in \mathbb{R}^{L\times r}$
4 $T^{\text{new}} \leftarrow T + \overline{V_1} \cdot \overline{V_2}^\top$; // $T^{\text{new}} \in \mathbb{R}^{L\times L}$
5 // Step 2
6 $Y' \leftarrow [2PT, P\overline{V_1}]$; // $Y' \in \mathbb{R}^{U\times(L+r)}$
7 $Z' \leftarrow [P, P\overline{V_2}]$; // $Z' \in \mathbb{R}^{U\times(L+r)}$
8 $Y \leftarrow [\text{diag}(u_1)Y', \cdots, \text{diag}(u_r)Y']$, u_i is the i-th column of $P\overline{V_1}$; // $Y \in \mathbb{R}^{U\times(L+r)r}$
9 $Z \leftarrow [\text{diag}(v_1)Z', \cdots, \text{diag}(v_r)Z']$, v_i is the i-th column of $P\overline{V_2}$; // $Z \in \mathbb{R}^{U\times(L+r)r}$
10 // Step 3
11 $N^{\text{new}} \leftarrow N - N\cdot(AY)\cdot\left(I + (AZ)^\top N(AY)\right)^{-1}\cdot(AZ)^\top\cdot N$; // $N^{\text{new}} \in \mathbb{R}^{m\times m}$
12 **return** $T^{\text{new}}, N^{\text{new}}$

Step 3. **Compute $N^{\text{new}} \in \mathbb{R}^{m\times m}$.**

$$\begin{aligned} N^{\text{new}} &= N - N\cdot(AY)\cdot\left(I + (AZ)^\top N(AY)\right)^{-1}\cdot(AZ)^\top\cdot N \\ &= \left(A\cdot(PTP^\top)^{\circ 2}\cdot A^\top + (AY)\cdot(AZ)^\top\right)^{-1} \\ &= \left(A\cdot(PT^{\text{new}}P^\top)^{\circ 2}\cdot A^\top\right)^{-1}, \end{aligned}$$

where the first step follows from the algorithm description of N^{new}, the second step follows from $N = \left(A\cdot(PTP^\top)^{\circ 2}\cdot A^\top\right)^{-1}$ and the Woodbury identity (Fact 2), and the last step follows from $(PT^{\text{new}}P^\top)^{\circ 2} = (PTP^\top)^{\circ 2} + Y\cdot Z^\top$.

Thus N^{new} satisfies the requirement of the output.

Time complexity. It is easy to see that the most time-consuming step is to compute N^{new} on Line 11, and in total this step takes $O(\mathcal{T}_{\text{mat}}(m,U,Lr) + \mathcal{T}_{\text{mat}}(m,m,Lr) + (Lr)^\omega)$ time.

Since $Lr \le U$ and $m \le U$, overall this algorithm takes at most $O(\mathcal{T}_{\text{mat}}(U,U,Lr))$ time. ◀

6 Correctness

6.1 Standard results from IPM theory

We use the following two results of the barrier method that hold for any cone with a barrier function. The proofs are standard, (see e.g., [30, Section 2.4]). For completeness we include a proof in the full version.

▶ **Lemma 12** (Invariance of Newton step, [30]). *Consider the following optimization problem:* $\min -b^\top y$ *s.t.* $y \in \overline{D}_F$, *where* $F : \mathbb{R}^m \to \mathbb{R}_+$ *is a barrier function with barrier parameter* ν_F, $D_F \subseteq \mathbb{R}^m$ *is the domain of* F, *and* $\overline{D}_F$ *is the closure of* D_F. *For any* $\eta \ge 1$, *define* $F_\eta(y) = -\eta b^\top y + F(y)$. *Let* $g_\eta(y) \in \mathbb{R}^m$ *and* $H(y) \in \mathbb{R}^{m\times m}$ *denote the gradient and the Hessian of* F_η *at* y.

Let $0 < \epsilon_N \le 0.05$ *be a parameter. If a feasible solution* $y \in D_F$, *a parameter* $\eta > 0$, *and a positive definite matrix* $\widetilde{H} \in \mathbb{S}^{n\times n}$ *satisfy the following:*

$$\|g_\eta(y)\|_{H(y)^{-1}} \le \epsilon_N, \quad \widetilde{H} \approx_{0.02} H(y).$$

Then $\eta^{\text{new}} = \eta\cdot(1 + \frac{\epsilon_N}{20\sqrt{\nu_F}})$, $y^{\text{new}} = y + \delta_y$ *where* $\delta_y = -\widetilde{H}^{-1}g_{\eta^{\text{new}}}(y)$ *satisfy* $y^{\text{new}} \in D_F$ *and*

$$\|\delta_y\|_{H(y)} \le 2\epsilon_N, \quad \|g_{\eta^{\text{new}}}(y^{\text{new}})\|_{H(y^{\text{new}})^{-1}} \le \epsilon_N.$$

▶ **Lemma 13** (Approximate optimality, [30]). *Consider the following optimization problem:* $\min -b^\top y$ *s.t.* $y \in \overline{D}_F$, *where* $F : \mathbb{R}^m \to \mathbb{R}_+$ *is a barrier function with barrier parameter* ν_F, $D_F \subseteq \mathbb{R}^m$ *is the domain of* F, *and* $\overline{D}_F$ *is the closure of* D_F. *Let* OPT *be the optimal objective value of this optimization problem. For any* $\eta \geq 1$, *define* $F_\eta(y) = -\eta b^\top y + F(y)$. *Let* $g_\eta(y) \in \mathbb{R}^m$ *and* $H(y) \in \mathbb{R}^{m \times m}$ *denote the gradient and the Hessian of* F_η *at* y.

Let $0 < \epsilon_N \leq 0.05$. *If a feasible solution* $y \in D_F$ *satisfies* $\|g_\eta(y)\|_{H(y)^{-1}} \leq \epsilon_N$, *then we have* $-b^\top y \leq \mathrm{OPT} + \frac{\nu_F}{\eta} \cdot (1 + 2\epsilon_N)$.

6.2 Low rank update

We use the following low rank update procedure of [16] and [15], which we modify by using a cutoff when $r \geq U/L$. The proof of the following lemma can be found in [15, Theorem 10.8].

Algorithm 3 LowRankUpdate of [16].

Parameters: A real number $\epsilon_S < 0.01$.
Input : New exact matrix $S^{\mathrm{new}} \in \mathbb{R}^{L \times L}$, old approximate matrix $\widetilde{S} \in \mathbb{R}^{L \times L}$,
Output : New approximate matrix $\widetilde{S}^{\mathrm{new}} \in \mathbb{R}^{L \times L}$, update matrices $V_1, V_2 \in \mathbb{R}^{L \times r}$.

$Z_{\mathrm{mid}} \leftarrow (S^{\mathrm{new}})^{-1/2} \widetilde{S} (S^{\mathrm{new}})^{-1/2} - I$
Compute spectral decomposition $Z_{\mathrm{mid}} = X \operatorname{diag}(\lambda) X^\top$
Let $\pi : [L] \to [L]$ be a sorting permutation such that $|\lambda_{\pi(i)}| \geq |\lambda_{\pi(i+1)}|$.
if $|\lambda_{\pi(1)}| \leq \epsilon_S$ **then**
  $\widetilde{S}_{\mathrm{new}} \leftarrow \widetilde{S}$;
  return $(\widetilde{S}^{\mathrm{new}}, 0, 0)$
else
  $r \leftarrow 1$;
  while $2r \leq U/L$ and $(|\lambda_{\pi(2r)}| > \epsilon_S$ or $|\lambda_{\pi(2r)}| > (1 - 1/\log L)|\lambda_{\pi(r)}|)$ **do**
    $r \leftarrow r + 1$;
  $r \leftarrow 2r$;
  if $r \geq U/L$ **then**
    $\widetilde{S}^{\mathrm{new}} \leftarrow S^{\mathrm{new}}$; `// Here we deviate from [16]`
    return $(\widetilde{S}^{\mathrm{new}}, \mathtt{null}, \mathtt{null})$
  else
    $\lambda^{\mathrm{new}}_{\pi(i)} \leftarrow \begin{cases} 0 & \text{if } i = 1, 2, \ldots, r \\ \lambda_{\pi(i)} & \text{else} \end{cases}$
    $\Omega \leftarrow \operatorname{supp}(\lambda^{\mathrm{new}} - \lambda)$; // $|\Omega| = r$
    $V_1 \leftarrow ((S^{\mathrm{new}})^{1/2} \cdot X \cdot \operatorname{diag}(\lambda^{\mathrm{new}} - \lambda))_{:,\Omega}$; // $V_1 \in \mathbb{R}^{L \times r}$
    $V_2 \leftarrow ((S^{\mathrm{new}})^{1/2} \cdot X)_{:,\Omega}$; // $V_2 \in \mathbb{R}^{L \times r}$
    $\widetilde{S}^{\mathrm{new}} \leftarrow \widetilde{S} + (S^{\mathrm{new}})^{1/2} X \operatorname{diag}(\lambda^{\mathrm{new}} - \lambda) X^\top (S^{\mathrm{new}})^{1/2}$;
    // $\widetilde{S}^{\mathrm{new}} = \widetilde{S} + V_1 V_2^\top \in \mathbb{R}^{L \times L}$
    return $(\widetilde{S}^{\mathrm{new}}, V_1, V_2)$;

▶ **Lemma 14** (Low rank update). *The algorithm* LowRankUpdate *(Algorithm 3) has the following properties:*

(i) *The output matrix* $\widetilde{S}^{\mathrm{new}} = \widetilde{S} + V_1 V_2^\top$ *is a spectral approximation of the input matrix:* $\widetilde{S}^{\mathrm{new}} \approx_{\epsilon_S} S^{\mathrm{new}}$.

(ii) *Consider a total of t iterations of* LowRankUpdate. *Initially $\widetilde{S}^{(0)} = S^{(0)}$, and we use $(S^{(i)}, \widetilde{S}^{(i-1)})$ and $(\widetilde{S}^{(i)}, V_1^{(i)}, V_2^{(i)})$ to denote the input and the output of the i-th iteration. We define the rank r_i to be the rank of $V_1^{(i)}$ if $V_1^{(i)} \neq \texttt{null}$, and otherwise we define $r_i = U/L$.*
If the input exact matrices $S^{(0)}, S^{(1)}, \cdots, S^{(t)} \in \mathbb{R}^{L\times L}$ satisfy

$$\|(S^{(i-1)})^{-1/2} S^{(i)} (S^{(i-1)})^{-1/2} - I\|_F \leq 0.02, \ \forall i \in [t]. \tag{11}$$

Then for any non-increasing sequence $g \in \mathbb{R}_+^L$, the ranks r_i satisfy

$$\sum_{i=1}^{t} r_i \cdot g_{r_i} \leq O(t \cdot \|g\|_2 \cdot \log L).$$

Furthermore, the algorithm LowRankUpdate *takes $O(L^\omega)$ time.*

6.3 Slowly moving guarantee

In SOS optimization, the matrix $S = P^\top \operatorname{diag}(s) P$ corresponds to the slack matrix of the SDP. The following lemma proves similar to SDP, in SOS the matrix S is changing slowly. Using this lemma we will prove that the requirement Eq. (11) of Lemma 14 is satisfied, which means we can approximate the change to the slack by a low-rank matrix.

▶ **Lemma 15** (Slowly moving guarantee). *Let $c \in \mathbb{R}^U$ and $A \in \mathbb{R}^{m\times U}$ be the input to the optimization problem. Let $P \in \mathbb{R}^{U\times L}$ be the matrix of the interpolant basis.*

For any $y \in \mathbb{R}^m$ and $y^{\text{new}} = y + \delta_y \in \mathbb{R}^m$, let $s = c - A^\top y \in \mathbb{R}^U$ and $S = P^\top \operatorname{diag}(s) P \in \mathbb{R}^{L\times L}$. Similarly define s^{new} and S^{new} from y^{new}. Let $H(y) = A \cdot \big(P\big(P^\top \operatorname{diag}(s) P\big)^{-1} P^\top\big)^{\circ 2} \cdot A^\top \in \mathbb{R}^{m\times m}$. If $s, s^{\text{new}} \in \Sigma_{n,2d}^*$, then S and S^{new} are both PSD, and we have

$$\|S^{-1/2} S^{\text{new}} S^{-1/2} - I\|_F = \|\delta_y\|_{H(y)}.$$

Proof. Note that if $s, s^{\text{new}} \in \Sigma_{n,2d}^*$, then by the dual cone characterization (Theorem 10) S and S^{new} are both PSD.

For convenience we define $M = PS^{-1}P^\top \in \mathbb{R}^{U\times U}$. Note that $H(y) = A \cdot M^{\circ 2} \cdot A^\top$. We also define $\delta_s = s^{\text{new}} - s = -A^\top \delta_y$. $\forall u \in [U]$, we use $p_u \in \mathbb{R}^L$ to denote the u-th row of P.

$$\begin{aligned}
\|S^{-1/2} S^{\text{new}} S^{-1/2} - I\|_F^2 &= \|S^{-1/2}\big(S^{\text{new}} - S\big) S^{-1/2}\|_F^2 \\
&= \|S^{-1/2}\big(P \operatorname{diag}(\delta_s) P^\top\big) S^{-1/2}\|_F^2 \\
&= \operatorname{tr}\big(S^{-1}(P^\top \operatorname{diag}(\delta_s) P) S^{-1}(P^\top \operatorname{diag}(\delta_s) P)\big) \\
&= \operatorname{tr}\big(S^{-1}(\sum_{u\in U} (\delta_s)_u \cdot p_u p_u^\top) S^{-1}(\sum_{v\in U} (\delta_s)_v \cdot p_v p_v^\top)\big) \\
&= \sum_{u,v\in U} (\delta_s)_u (\delta_s)_v \cdot \operatorname{tr}\big(S^{-1} p_u p_u^\top S^{-1} p_v p_v^\top\big) \\
&= \sum_{u,v\in U} (\delta_s)_u (\delta_s)_v \cdot (p_v^\top S^{-1} p_u)^2 \\
&= \sum_{u,v\in U} (\delta_s)_u (\delta_s)_v \cdot M_{uv}^2 \ = \ \|\delta_s\|_{M^{\circ 2}}^2,
\end{aligned} \tag{12}$$

where the third step follows from $\|A\|_F^2 = \operatorname{tr}(A^\top A)$ and the cyclic property of trace, and the sixth step again follows from the cyclic property of trace.

Since $\delta_s = -A^\top \delta_y$, we have

$$\|\delta_s\|_{M^{\circ 2}}^2 = \delta_s^\top M^{\circ 2} \delta_s = \delta_y^\top A M^{\circ 2} A^\top \delta_y = \delta_y^\top H(y) \delta_y = \|\delta_y\|_{H(y)}^2. \tag{13}$$

Combining Eq. (12) and (13) we get the bound in the lemma statement. ◀

6.4 Proof of correctness

Finally we are ready to prove the correctness of Algorithm 1.

▶ **Theorem 16** (Correctness of Algorithm 1). *Consider the following optimization problem with $A \in \mathbb{R}^{m\times U}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^U$:*

$$\begin{aligned} \textit{Primal:}\quad & \min \langle c, x\rangle && \textit{Dual:}\quad \max \langle y, b\rangle \\ \text{s.t.}\quad & Ax = b && \text{s.t.}\quad A^\top y + s = c \\ & x \in \Sigma_{n,2d}\,, && s \in \Sigma^*_{n,2d}\,. \end{aligned}$$

Let OPT *denote the optimal objective value of this optimization problem. Assume Slater's condition and that any primal feasible $x \in \Sigma_{n,2d}$ satisfies $\|x\|_1 \le R$.*

Then for any error parameters $\delta \in (0,1)$, $\epsilon_S \le 0.01$, and $\epsilon_N \le 0.05$, Algorithm 1 outputs $x \in \Sigma_{n,2d}$ that satisfies

$$\langle c, x\rangle \le \mathrm{OPT} + \delta \cdot R\|c\|_\infty \quad \textit{and} \quad \|Ax - b\|_1 \le 8\delta L \cdot (LR\|A\|_\infty + \|b\|_1).$$

Proof. We consider the optimization problem $\min -b^\top y$ s.t. $y \in \overline{D}_F$, where $F : \mathbb{R}^m \to \mathbb{R}_+$ is the barrier function defined in Eq. (10), and $\overline{D}_F$ is the closure of the domain of F. The barrier parameter of F is $\nu_F = L$. This optimization problem is equivalent to the dual formulation and its optimal value is $-$OPT. For any η, let $F_\eta(y) = -\eta b^\top y + F(y)$.

In the beginning Algorithm 1 first uses Lemma 22 to convert the optimization problem to another form which has an initial feasible solution y that is close to the optimal solution of F_η with $\eta = 1$. The initial y satisfies $\|g_\eta(y)\|_{H(y)^{-1}} \le \epsilon_N$ by Lemma 22. Initially we also have $\widetilde{S} = S = P^\top \operatorname{diag}(s) P$ (Line 3 in Algorithm 1).

Next we prove the correctness of Algorithm 1 inductively. At each iteration, we assume the following induction hypothesis is satisfied: (1) $\|g_\eta(y)\|_{H(y)^{-1}} \le \epsilon_N$, (2) $\widetilde{S} \approx_{\epsilon_S} S$. We aim to prove that the updated y^{new}, η^{new}, S^{new}, and $\widetilde{S}^{\mathrm{new}}$ still satisfy these two conditions.

In Lemma 11 we have proved that in Algorithm 1 we always maintain $N = \big(A \cdot (P\widetilde{S}^{-1}P^\top)^{\circ 2} \cdot A^\top\big)^{-1}$. Let $\widetilde{H} = N^{-1}$, we have

$$\widetilde{H} = A \cdot \big(P\widetilde{S}^{-1}P^\top\big)^{\circ 2} \cdot A^\top \approx_{2\epsilon_S} A \cdot \big(PS^{-1}P^\top\big)^{\circ 2} \cdot A^\top = H(y),$$

where in the second step we use the induction hypothesis that $\widetilde{S} \approx_{\epsilon_S} S$, and by Fact 6 we have $\widetilde{S}^{-1} \approx_{\epsilon_S} S^{-1}$, and hence $P\widetilde{S}^{-1}P^\top \approx_{\epsilon_S} PS^{-1}P^\top$, and hence $(P\widetilde{S}^{-1}P^\top)^{\circ 2} \approx_{2\epsilon_S} (PS^{-1}P^\top)^{\circ 2}$.

The new vector y^{new} is computed as $y^{\mathrm{new}} = y + \delta_y$ where $\delta_y = -\widetilde{H}^{-1} g_\eta(y)$ (Line 8 and 9 of Algorithm 1). And η is updated to $\eta^{\mathrm{new}} = \eta \cdot (1 + \frac{\epsilon_N}{20\sqrt{L}})$ (Line 11 of Algorithm 1). Since $\|g_\eta(y)\|_{H(y)^{-1}} \le \epsilon_N$, and $\widetilde{H} \approx_{2\epsilon_S} H(y)$ where $2\epsilon_S \le 0.02$ by its definition in Algorithm 3, the requirements of Lemma 12 are satisfied, so we have

$$\|g_{\eta^{\mathrm{new}}}(y^{\mathrm{new}})\|_{H(y^{\mathrm{new}})^{-1}} \le \epsilon_N, \quad \text{and} \quad \|\delta_y\|_{H(y)} \le 2\epsilon_N.$$

This proves the first induction hypothesis.

Then using Lemma 15 and since $\epsilon_N \le 0.01$ by its definition in Algorithm 1, we have

$$\|S^{-1/2}(S^{\mathrm{new}})S^{-1/2} - I\|_F \le \|\delta_y\|_{H(y)} \le 2\epsilon_N \le 0.02.$$

Thus the input matrix S^{new} to LowRankUpdate satisfies the requirement of Eq. (11) of Lemma 14, and we have that $\widetilde{S}^{\mathrm{new}} \approx_{\epsilon_S} S^{\mathrm{new}}$. This proves the second induction hypothesis.

Finally, we know that after $t = 40\epsilon_N^{-1}\sqrt{L}\log(L/\delta)$ iterations, η becomes $(1+\frac{\epsilon_N}{20\sqrt{L}})^t \geq 2L/\delta^2$, so using Lemma 13, we have

$$b^\top y \geq \mathrm{OPT} - \frac{\nu_F}{\eta}\cdot(1+2\epsilon_N) \geq \mathrm{OPT} - \delta^2.$$

Thus the initialization lemma (Lemma 22) ensures that we have a solution $x \in \Sigma_{n,2d}$ to the original primal optimization problem which satisfies

$$\langle c, x\rangle \leq \mathrm{OPT} + \delta\cdot R\|c\|_\infty, \qquad \|Ax-b\|_1 \leq 8\delta L\cdot(LR\|A\|_\infty + \|b\|_1). \qquad \blacktriangleleft$$

7 Time complexity

7.1 Worst case time

We first bound the worst case running time of Algorithm 1. The running time of the i-th iteration depends on the updated rank r_i of LowRankUpdate, which is defined to be the size of $V_1^{(i)}$ if $V_1^{(i)} \neq \texttt{null}$, and U/L otherwise (see Lemma 14).

▶ **Lemma 17** (Worst case time of Algorithm 1). *In Algorithm 1, the initialization time is $O(U^\omega)$, and the running time in the i-th iteration is $O(\mathcal{T}_{\mathrm{mat}}(U,U,\min\{Lr_i,U\}))$.*

Proof.

Initialization time. The most time-consuming step of initialization is Line 5, where computing $N = \left(A(PTP^\top)^{\circ 2}A^\top\right)^{-1}$ takes $O(\mathcal{T}_{\mathrm{mat}}(U,U,L)+\mathcal{T}_{\mathrm{mat}}(U,U,m))$ time. This is bounded by $O(U^\omega)$ since $L, m \leq U$.

Time per iteration. In each iteration the most time-consuming steps are (1) computing S^{new} and calling LowRankUpdate on Line 12-14, (2) executing the if-clause on Line 15-20, and (3) computing g^{new} on Line 21.

1. Computing S^{new} on Line 12 takes $\mathcal{T}_{\mathrm{mat}}(U,L,L)$ time. Calling LowRankUpdate on Line 14 takes $O(L^\omega)$ time by Lemma 14.
2. In the if-clause on Line 15-20, if $V_1 = V_2 = \texttt{null}$, then $Lr_i \geq U$, and we compute $N^{\mathrm{new}} = \left(A\cdot(PT^{\mathrm{new}}P^\top)^{\circ 2}\cdot A^\top\right)^{-1}$, which takes $O(\mathcal{T}_{\mathrm{mat}}(U,U,U))$ time. Otherwise we call UpdateHessianInv on Line 19, which takes $O(\mathcal{T}_{\mathrm{mat}}(U,U,Lr_i))$ time by Lemma 11. In total the if-clause has running time $O(\mathcal{T}_{\mathrm{mat}}(U,U,\min\{Lr_i,U\}))$.
3. Computing the gradient $g = -\eta^{\mathrm{new}}\cdot b + A\cdot\mathrm{diag}\left(P\left(P^\top\mathrm{diag}(s^{\mathrm{new}})P\right)^{-1}P^\top\right)$ on Line 21 takes $O(\mathcal{T}_{\mathrm{mat}}(U,U,L))$ time since $m \leq U$.

Thus the total time per iteration is $O(\mathcal{T}_{\mathrm{mat}}(U,U,\min\{Lr_i,U\}))$. $\blacktriangleleft$

7.2 Amortized time

In this section we bound the amortized running time of Algorithm 1.

Let ω be the matrix multiplication exponent, let α be the dual matrix multiplication exponent. The current best values are $\omega \approx 2.373$ and $\alpha \approx 0.314$ [21, 10, 1]. Note that the current best values of ω and α satisfies that $\alpha \geq 5-2\omega$. We use the following modified lemma from [15]:

▶ **Lemma 18** (Helpful lemma for amortization, modified version of Lemma 10.13 of [15]). *Let t denote the total number of iterations. Let $r_i \in [L]$ be the rank for the i-th iteration for $i \in [t]$. Assume r_i satisfies the following condition: for any vector $g \in \mathbb{R}_+^L$ which is non-increasing, we have $\sum_{i=1}^t r_i \cdot g_{r_i} \leq O(t \cdot \|g\|_2)$.*

If the cost in the i-th iteration is $O(\mathcal{T}_{\mathrm{mat}}(U, U, \min\{Lr_i, U\}))$, when $\alpha \geq 5 - 2\omega$, the amortized cost per iteration is $U^{2+o(1)} + U^{\omega - 1/2 + o(1)} \cdot L^{1/2}$.

We include the proof of Lemma 18 for completeness. The main difference between this proof and that of [15] is that we cut off at U/L instead of L. Our proof makes use of the following two facts about ω and α (Lemma A.4 and Lemma A.5 of [9]).

▶ **Fact 19** (Relation of ω and α). $\omega \leq 3 - \alpha$.

▶ **Fact 20** (Upper bound of $\mathcal{T}_{\mathrm{mat}}(n, n, r)$). *For any $r \leq n$, we have that $\mathcal{T}_{\mathrm{mat}}(n, n, r) \leq n^{2+o(1)} + r^{\frac{\omega-2}{1-\alpha}} \cdot n^{2 - \frac{\alpha(\omega-2)}{(1-\alpha)} + o(1)}$.*

Proof of Lemma 18. For r_i that satisfies $r_i \leq U/L$, we have

$$\begin{aligned} \mathcal{T}_{\mathrm{mat}}(U, U, Lr_i) &\leq U^{2+o(1)} + (Lr_i)^{\frac{\omega-2}{1-\alpha}} \cdot U^{2 - \frac{\alpha(\omega-2)}{1-\alpha} + o(1)} \\ &= U^{2+o(1)} + U^{2 - \frac{\alpha(\omega-2)}{1-\alpha} + o(1)} \cdot L^{\frac{\omega-2}{1-\alpha}} \cdot r_i^{\frac{\omega-2}{1-\alpha}}, \end{aligned} \tag{14}$$

where the first step follows from Fact 20.

Define a sequence $g \in \mathbb{R}_+^L$ such that for $r \in [L]$,

$$g_r = \begin{cases} r^{\frac{\omega-2}{1-\alpha} - 1} & \text{if } r \leq U/L, \\ (U/L)^{\frac{\omega-2}{1-\alpha}} \cdot r^{-1} & \text{if } r > U/L. \end{cases}$$

Note that g is non-increasing because $\frac{\omega-2}{1-\alpha} \leq 1$ (Fact 19). Then using the condition in the lemma statement, we have

$$\begin{aligned} \sum_{i=1}^t \min\{r_i^{\frac{\omega-2}{1-\alpha}}, (U/L)^{\frac{\omega-2}{1-\alpha}}\} &= \sum_{i=1}^t r_i \cdot g_{r_i} \\ &\leq t \cdot \|g\|_2 \\ &\leq t \cdot \Big(\int_{x=1}^{U/L} x^{\frac{2(\omega-2)}{1-\alpha} - 2} \mathrm{d}x + (U/L)^{\frac{2(\omega-2)}{1-\alpha}} \cdot \int_{x=U/L}^{L} x^{-2} \mathrm{d}x \Big)^{1/2} \\ &\leq t \cdot \Big(c \cdot (U/L)^{\frac{2(\omega-2)}{1-\alpha} - 1} + (U/L)^{2(\frac{\omega-2}{1-\alpha})} \cdot (U/L)^{-1} \Big)^{1/2} \\ &= t \cdot O((U/L)^{\frac{(\omega-2)}{1-\alpha} - 1/2}), \end{aligned} \tag{15}$$

where the first step follows from the definition of $g \in \mathbb{R}^L$, the second step follows from the assumption $\sum_{t=1}^t r_i \cdot g_{r_i} \leq t \cdot \|g\|_2$ in the lemma statement, the third step follows from upper bounding the ℓ_2 norm $\|g\|_2^2 = \sum_{r=1}^L g_r^2$, and the fourth step follows $\frac{2(\omega-2)}{1-\alpha} \geq 1$ when $\alpha \geq 5 - 2\omega$, so the integral $\int_{x=1}^{U/L} x^{\frac{2(\omega-2)}{1-\alpha} - 2} \mathrm{d}x = c \cdot x^{\frac{2(\omega-2)}{1-\alpha} - 1} \big|_1^{U/L} = O\big((U/L)^{\frac{2(\omega-2)}{1-\alpha} - 1}\big)$ where $c := 1/(\frac{2(\omega-2)}{1-\alpha} - 1)$.

Thus we have

$$\begin{aligned}
&\sum_{t=1}^{t} \mathcal{T}_{\mathrm{mat}}(U,U,\min\{Lr_i,U\}) \\
&\le \sum_{t=1}^{t} \left(U^{2+o(1)} + U^{2-\frac{\alpha(\omega-2)}{1-\alpha}+o(1)} \cdot L^{\frac{\omega-2}{1-\alpha}} \cdot \min\{r_i^{\frac{\omega-2}{1-\alpha}}, (U/L)^{\frac{\omega-2}{1-\alpha}}\}\right) \\
&= t \cdot U^{2+o(1)} + U^{2-\frac{\alpha(\omega-2)}{1-\alpha}+o(1)} \cdot L^{\frac{\omega-2}{1-\alpha}} \cdot \sum_{t=1}^{t} \min\{r_i^{\frac{\omega-2}{1-\alpha}}, (U/L)^{\frac{\omega-2}{1-\alpha}}\} \\
&\le t \cdot U^{2+o(1)} + U^{2-\frac{\alpha(\omega-2)}{1-\alpha}+o(1)} \cdot L^{\frac{\omega-2}{1-\alpha}} \cdot t \cdot (U/L)^{\frac{(\omega-2)}{1-\alpha}-1/2} \\
&= t \cdot (U^{2+o(1)} + U^{\omega-1/2+o(1)} \cdot L^{1/2}),
\end{aligned}$$

where the first step follows from Eq. (14) and $\mathcal{T}_{\mathrm{mat}}(U,U,U) = U^{\omega} = U^{2-\frac{\alpha(\omega-2)}{1-\alpha}} \cdot L^{\frac{\omega-2}{1-\alpha}} \cdot (U/L)^{\frac{\omega-2}{1-\alpha}}$, the second step follows from moving summation inside, the third step follows from Eq. (15), and the last step follows from adding the terms together. ◀

Now we are ready to prove our main theorem for the amortized time of Algorithm 1.

▶ **Theorem 21** (Time of Algorithm 1). *When $\alpha \ge 5 - 2\omega$, the running time of Algorithm 1 is* $(U^2 \cdot L^{1/2} + U^{\omega-1/2} \cdot L) \cdot (\log(1/\delta) + U^{o(1)})$.

Proof. Using Lemma 17 the initialization time is $O(U^{\omega}) \le O(U^{\omega-1/2} \cdot L)$ since $U \le L^2$.

Using Lemma 14 we know that the ranks r_i indeed satisfy the requirement of Lemma 18, and since the worst case time per iteration is $O(\mathcal{T}_{\mathrm{mat}}(U,U,\min\{Lr_i,U\}))$ (Lemma 17), using Lemma 18 the time per iteration is $U^{2+o(1)} + U^{\omega-1/2+o(1)} \cdot L^{1/2}$. Since there are in total $t = 40\epsilon_N^{-1}\sqrt{L}\log(L/\delta)$ iterations, we get the total running time as claimed. ◀

7.3 Comparison with previous results

In this section we compare the running time of [27], [16, 15], and our result. We assume that $m = \Theta(U)$ when making the comparisons.

Ignoring $\log(1/\delta)$ and $U^{o(1)}$ factors, and since $L \le U \le L^2$, the running times are

$$\begin{aligned}
\text{[27] (SOS)} &: L^{0.5} \cdot U^{\omega}, \\
\text{[16, 15] (SDP)}^5 &: L^{0.5} \cdot \min\{UL^2 + U^{\omega},\ L^4 + L^{2\omega-0.5}\}, \\
\text{Ours (SOS)} &: L^{0.5} \cdot (U^2 + U^{\omega-0.5} \cdot L^{0.5}).
\end{aligned}$$

Current ω and α

Plugging in the current best values $\omega \approx 2.373$ and $\alpha \approx 0.314$, we have

$$\begin{aligned}
\text{[27] (SOS)} &: L^{0.5} \cdot U^{2.373}, \\
\text{[16, 15] (SDP)} &: L^{0.5} \cdot \min\{UL^2 + U^{2.373},\ L^{4.246}\} \\
&= L^{0.5} \cdot \begin{cases} UL^2 & \text{when } U \in (L, L^{1.457}], \\ U^{2.373} & \text{when } U \in (L^{1.457}, L^{1.789}], \\ L^{4.246} & \text{when } U \in (L^{1.789}, L^2), \end{cases} \\
\text{Ours (SOS)} &: L^{0.5} \cdot (U^2 + U^{1.873}L^{0.5}).
\end{aligned}$$

[5] When solving SOS, [16] has running time $O(L^{0.5} \cdot (UL^2 + U^{\omega} + L^{\omega})) \le O(L^{0.5} \cdot (UL^2 + U^{\omega}))$, and [15] has running time $O(L^{0.5} \cdot (U^2 + L^4) + U^{\omega} + L^{2\omega}) \le O(L^{4.5} + L^{2\omega})$ since $L \le U \le L^2$.

Note that our running time is always better than the previous results, and for several values of L and U we improve by a polynomial factor. See Figure 1 for an illustration.

8 Initialization

There exist standard techniques to transform a convex program to a form that has an easily obtainable strictly feasible point, see e.g. [39]. We follow the initialization procedure presented by [9] and [16] and adapt to SOS optimization. Similar initialization lemma exists for WSOS optimization. The proof of this lemma can be found in our full version.

Let the matrix $P \in \mathbb{R}^{U \times L}$ and the operator $\Lambda : \mathbb{R}^U \to \mathbb{R}^{L \times L}$ that $\Lambda(s) = P^\top \operatorname{diag}(s) P$ be defined as in the interpolant basis paragraph of Section 3.

▶ **Lemma 22** (Initialization). *Given an instance of* (SOS) *that fulfills Slater's condition, and let R be an upper bound on the ℓ_1-norm of the primal feasible solutions, i.e. all primal feasible x of* (SOS) *fulfill $\|x\|_1 \le R$, and let $\delta \in (0,1)$. We define $\overline{A} \in \mathbb{R}^{(m+1)\times(U+2)}$, $\overline{b} \in \mathbb{R}^{m+1}$, and $\overline{c} \in \mathbb{R}^{U+2}$ as*

$$\overline{A} = \begin{bmatrix} A & 0 & \frac{1}{R}b - A\overline{g}^0 \\ 1_U^\top & 1 & 0 \end{bmatrix}, \quad \overline{b} = \begin{bmatrix} \frac{1}{R}b \\ 1 + \langle 1_U, \overline{g}^0 \rangle \end{bmatrix}, \quad \overline{c} = \begin{bmatrix} \frac{\delta}{\|c\|_\infty} c \\ 0 \\ 1 \end{bmatrix},$$

and let

$$\overline{x}^0 = \begin{bmatrix} \overline{g}^0 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{U+2}, \quad \overline{y}^0 = \begin{bmatrix} 0_m \\ -1 \end{bmatrix} \in \mathbb{R}^{m+1}, \text{ and } \overline{s}^0 = \begin{bmatrix} 1_U + \frac{\delta}{\|c\|_\infty} c \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{U+2},$$

where $\overline{g}^0 = g_{\Sigma^*}(\overline{s}^0_{[:U]}) \in \mathbb{R}^U$ for the gradient function $g_{\Sigma^*}(s) := \operatorname{diag}(P(P^\top \operatorname{diag}(s) P)^{-1} P^\top)$ that maps from $\mathbb{R}^U$ to $\mathbb{R}^U$. This defines the auxiliary primal-dual system

$$\begin{aligned} \min \langle \overline{c}, \overline{x} \rangle \qquad & \qquad \max \langle \overline{y}, \overline{b} \rangle \\ \overline{A}\overline{x} = \overline{b} \qquad & \qquad \overline{A}^\top \overline{y} + \overline{s} = \overline{c} \\ \overline{x} \in \Sigma_{n,2d} \times \mathbb{R}^2_{\ge 0}\,, \qquad & \qquad \overline{s} \in \Sigma^*_{n,2d} \times \mathbb{R}^2_{\ge 0}\,. \end{aligned} \tag{Aux-SOS}$$

Then $(\overline{x}^0, \overline{y}^0, \overline{s}^0)$ are feasible to the auxiliary system (Aux-SOS).

Further, under the canonical barrier (we use $\overline{a}_i$ to denote the i-th column of $\overline{A}$):

$$\overline{F}_\eta(\overline{y}) = -\eta\langle \overline{y}, \overline{b} \rangle - \log\det\Big(\Lambda\big((\overline{c} - \overline{A}^\top \overline{y})_{[:U]}\big)\Big) - \log(\overline{c}_{U+1} - \langle \overline{a}_{U+1}, \overline{y} \rangle) - \log(\overline{c}_{U+2} - \langle \overline{a}_{U+2}, \overline{y} \rangle),$$

we have that $\|\overline{g}_{\eta^0}(\overline{y}^0)\|_{\overline{H}(\overline{y}^0)^{-1}} = 0$ for $\eta^0 = 1$.

Further, for any solution $(\overline{x}, \overline{y}, \overline{s})$ to (Aux-SOS) *with duality gap $\le \delta^2$, its restriction $\widehat{x} := \overline{x}_{[:U]}$ fulfills*

$$\begin{aligned} \langle c, \widehat{x} \rangle &\le \min_{Ax = b, x \in \Sigma_{n,2d}} \langle c, x \rangle + \delta \cdot R\|c\|_\infty, \\ \|A\widehat{x} - b\|_1 &\le 8\delta L \cdot (LR\|A\|_\infty + \|b\|_1), \\ \widehat{x} &\in \Sigma_{n,2d}. \end{aligned}$$

References

1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

2 Christine Bachoc and Frank Vallentin. New upper bounds for kissing numbers from semidefinite programming. Technical report, Journal of the American Mathematical Society, 2006.

3 Brandon Ballinger, Grigoriy Blekherman, Henry Cohn, Noah Giansiracusa, Elizabeth Kelly, and Achill Schürmann. Experimental study of energy-minimizing point configurations on spheres. *Experimental Mathematics*, 18(3):257–283, 2009. doi:10.1080/10586458.2009.10129052.

4 Boaz Barak, Samuel B. Hopkins, Jonathan A. Kelner, Pravesh K. Kothari, Ankur Moitra, and Aaron Potechin. A nearly tight sum-of-squares lower bound for the planted clique problem. *SIAM J. Comput.*, 48(2):687–735, 2019. doi:10.1137/17M1138236.

5 Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *2011 IEEE 52nd annual symposium on foundations of computer science (FOCS)*, pages 472–481. IEEE, 2011.

6 Markus Bläser. Fast matrix multiplication. *Theory of Computing*, pages 1–60, 2013.

7 Grigoriy Blekherman, Pablo A Parrilo, and Rekha R Thomas. *Semidefinite optimization and convex algebraic geometry*. SIAM, 2012.

8 L. Bos, S. De Marchi, A. Sommariva, and M. Vianello. Computing multivariate fekete and leja points by numerical linear algebra. *SIAM Journal on Numerical Analysis*, 48(5):1984–1999, 2010. doi:10.1137/090779024.

9 Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.

10 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the 2018 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046. SIAM, 2018.

11 Bissan Ghaddar, Jakub Marecek, and M. Mevissen. Optimal power flow as a polynomial optimization problem. *IEEE Transactions on Power Systems*, 31:539–546, 2016.

12 Roxana Heß, Didier Henrion, Jean-Bernard Lasserre, and Tien Son Pham. Semidefinite approximations of the polynomial abscissa. *SIAM J. Control. Optim.*, 54(3):1633–1656, 2016. doi:10.1137/15M1033198.

13 Samuel B. Hopkins, Pravesh K. Kothari, Aaron Potechin, Prasad Raghavendra, Tselil Schramm, and David Steurer. The power of sum-of-squares for detecting hidden structures. In *58th IEEE Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 720–731. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.72.

14 Samuel B. Hopkins and Jerry Li. Mixture models, robustness, and sum of squares proofs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, (STOC)*, pages 1021–1034. ACM, 2018. doi:10.1145/3188745.3188748.

15 Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving sdp faster: A robust ipm framework and efficient implementation, 2021. arXiv:2101.08208.

16 Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In *2020 IEEE 61st annual symposium on foundations of computer science (FOCS)*, pages 910–918. IEEE, 2020.

17 Shunhua Jiang, Yunze Man, Zhao Song, Zheng Yu, and Danyang Zhuo. Fast graph neural tangent kernel via kronecker sketching. *arXiv preprint AAAI'22*, 2021. arXiv:2112.02446.

18 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th annual ACM symposium on Theory of computing (STOC)*, pages 302–311, 1984.

19 Jean Bernard Lasserre. *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2015. doi:10.1017/CBO9781107447226.

20 M. Laurent. *Sums of squares, moment matrices and optimization over polynomials*, pages 155–270. Number 149 in The IMA Volumes in Mathematics and its Applications Series. Springer Verlag, Germany, 2009.

21 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.

22 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 424–433. IEEE, 2014.

23 Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory (COLT)*, pages 2140–2157. PMLR, 2019.

24 Yurii Nesterov. Squared functional systems and optimization problems. In *High performance optimization*, pages 405–440. Springer, 2000.

25 Yurii Nesterov and Arkadi Nemirovski. Interior-point polynomial algorithms in convex programming. In *Siam Studies in Applied Mathematics*, 1987. doi:10.1137/1.9781611970791.

26 Dávid Papp. Optimal designs for rational function regression. *Journal of the American Statistical Association*, 107(497):400–411, 2012. doi:10.1080/01621459.2012.656035.

27 Dávid Papp and Sercan Yildiz. Sum-of-squares optimization without semidefinite programming. *SIAM Journal on Optimization*, 29(1):822–851, 2019.

28 Pablo Parrilo. *Sum of squares : theory and applications : AMS short course, sum of squares : theory and applications, January 14-15, 2019, Baltimore, Maryland.* American Mathematical Society, Providence, Rhode Island, 2020.

29 Mihai Putinar and Florian-Horia Vasilescu. Positive polynomials on semi-algebraic sets. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 328(7):585–589, 1999. doi:10.1016/S0764-4442(99)80251-1.

30 James Renegar. *A Mathematical View of Interior-Point Methods in Convex Optimization.* Society for Industrial and Applied Mathematics, January 2001. doi:10.1137/1.9780898718812.

31 Tae Roh, Bogdan Dumitrescu, and Lieven Vandenberghe. Multidimensional FIR filter design via trigonometric sum-of-squares optimization. *J. Sel. Topics Signal Processing*, 1(4):641–650, 2007. doi:10.1109/JSTSP.2007.910261.

32 Alvise Sommariva and Marco Vianello. Computing approximate fekete points by qr factorizations of vandermonde matrices. *Computers & Mathematics with Applications*, 57(8):1324–1336, 2009.

33 Zhao Song, Shuo Yang, and Ruizhe Zhang. Does preprocessing help training over-parameterized neural networks? *Advances in Neural Information Processing Systems*, 34, 2021.

34 Zhao Song, Lichen Zhang, and Ruizhe Zhang. Training multi-layer over-parametrized neural network in subquadratic time. *arXiv preprint*, 2021. arXiv:2112.07628.

35 Gilbert Strang. Karmarkar's algorithm and its place in applied mathematics. *The Mathematical Intelligencer*, 9(2):4–10, 1987.

36 Ning Tan. *On the Power of Lasserre SDP Hierarchy.* PhD thesis, EECS Department, University of California, Berkeley, December 2015. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-236.html.

37 Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 332–337. IEEE, 1989.

38 Jan van den Brand, Binghui Peng, Zhao Song, and Omri Weinstein. Training (overparametrized) neural networks in near-linear time. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185, pages 63:1–63:15, 2021. doi:10.4230/LIPIcs.ITCS.2021.63.

39 Yinyu Ye, Michael J Todd, and Shinji Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of operations research*, 19(1):53–67, 1994.

Tight Approximation Algorithms for Two-Dimensional Guillotine Strip Packing

Arindam Khan
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Aditya Lonkar
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Arnab Maiti
Indian Institute of Technology, Kharagpur, India

Amatya Sharma
Indian Institute of Technology, Kharagpur, India

Andreas Wiese
Technische Universität München, Germany

Abstract

In the Strip Packing problem (SP), we are given a vertical half-strip $[0, W] \times [0, \infty)$ and a set of n axis-aligned rectangles of width at most W. The goal is to find a non-overlapping packing of all rectangles into the strip such that the height of the packing is minimized. A well-studied and frequently used practical constraint is to allow only those packings that are guillotine separable, i.e., every rectangle in the packing can be obtained by recursively applying a sequence of edge-to-edge axis-parallel cuts (guillotine cuts) that do not intersect any item of the solution. In this paper, we study approximation algorithms for the Guillotine Strip Packing problem (GSP), i.e., the Strip Packing problem where we require additionally that the packing needs to be guillotine separable. This problem generalizes the classical Bin Packing problem and also makespan minimization on identical machines, and thus it is already strongly NP-hard. Moreover, due to a reduction from the Partition problem, it is NP-hard to obtain a polynomial-time $(3/2 - \varepsilon)$-approximation algorithm for GSP for any $\varepsilon > 0$ (exactly as Strip Packing). We provide a matching polynomial time $(3/2 + \varepsilon)$-approximation algorithm for GSP. Furthermore, we present a pseudo-polynomial time $(1 + \varepsilon)$-approximation algorithm for GSP. This is surprising as it is NP-hard to obtain a $(5/4 - \varepsilon)$-approximation algorithm for (general) Strip Packing in pseudo-polynomial time. Thus, our results essentially settle the approximability of GSP for both the polynomial and the pseudo-polynomial settings.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Approximation Algorithms, Two-Dimensional Packing, Rectangle Packing, Guillotine Cuts, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.80

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/pdf/2202.05989.pdf

Funding *Arindam Khan*: Arindam Khan was supported in part by Pratiksha Trust Young Investigator Award, Google CSExplore Award, and Google India Research Award.
Andreas Wiese: Andreas Wiese was partially supported by the Fondecyt Regular grant 1200173.

Acknowledgements A part of this work was done when Arnab Maiti and Amatya Sharma were undergraduate interns at Indian Institute of Science.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 80; pp. 80:1–80:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Two-dimensional packing problems form a fundamental research area in combinatorial optimization, computational geometry, and approximation algorithms. They find numerous practical applications in logistics [9], cutting stock [23], VLSI design [26], smart-grids [20], etc. The Strip Packing problem (SP), a generalization of the classical Bin Packing problem and also the makespan minimization problem on identical machines, is one of the central problems in this area. We are given an axis-aligned vertical half-strip $[0, W] \times [0, \infty)$ and a set of n axis-aligned rectangles (also called *items*) $I := \{1, 2, \ldots, n\}$, where for each rectangle i we are given an integral width $w_i \leq W$, and an integral height h_i; we assume the rectangles to be open sets. The goal is to pack all items such that the maximum height of the top edge of a packed item is minimized. The packing needs to be *non-overlapping*, i.e., such a packing into a strip of height H maps each rectangle $i \in I$ to a new translated open rectangle $R(i) := (left(i), right(i)) \times (bottom(i), top(i))$ where $right(i) = left(i) + w_i$, $top(i) = bottom(i) + h_i$, $left(i) \geq 0$, $bottom(i) \geq 0$, $right(i) \leq W$, $top(i) \leq H$ and for any $i, j \in I$, we must have $R(i) \cap R(j) = \emptyset$. We assume that items are not allowed to be rotated.

The best known polynomial time approximation algorithm for SP has an approximation ratio of $(5/3+\varepsilon)$ for any constant $\varepsilon > 0$ [24] and a straight-forward reduction from Partition shows that it is NP-hard to approximate the problem with a ratio of $(3/2 - \varepsilon)$ for any $\varepsilon > 0$. Maybe surprisingly, one can approximate SP better in pseudo-polynomial time: there is a pseudo-polynomial time $(5/4 + \varepsilon)$-approximation algorithm [27] and it is NP-hard to obtain a $(5/4 - \varepsilon)$-approximation algorithm with this running time [25]. Hence, it remains open to close the gap between $(5/3 + \varepsilon)$ and $(3/2 - \varepsilon)$ for polynomial time algorithms, and even in pseudo-polynomial time, there can be no $(1+\varepsilon)$-approximation for the problem for arbitrarily small $\varepsilon > 0$.

SP is particularly motivated from applications in which we want to cut out rectangular pieces of a sheet or stock unit of raw material, i.e., metal, glass, wood, or, cloth, and we want to minimize the amount of wasted material. For cutting out these pieces in practice, axis-parallel end-to-end cuts, called *guillotine cuts*, are popular due to their simplicity of operation [46]. In this context, we look for solutions to cut out the individual objects by a recursive application of guillotine cuts that do not intersect any item of the solution. Applications of guillotine cutting are found in crepe-rubber mills [42], glass industry [40], paper cutting [35], etc. In particular, this motivates studying geometric packing problems with the additional constraint that the placed objects need to be separable by a sequence of guillotine cuts (see Figure 1). Starting from the classical work by Christofides et al. [10] in 1970s, settings with such guillotine cuts are widely studied in the literature [16, 47, 6, 34, 15, 11, 17, 12]. In fact, many heuristics for guillotine packing have been developed to efficiently solve benchmark instances, based on tree-search, branch-and-bound, dynamic optimization, tabu search, genetic algorithms, etc. Khan et al. [32] mentions "a staggering number of recent experimental papers" on guillotine packing and lists several such recent experimental papers.

A related notion is k-stage packing, originally introduced by Gilmore and Gomory [23]. Here, each stage consists of either vertical or horizontal guillotine cuts (but not both). In each stage, each of the pieces obtained in the previous stage is considered separately and can be cut again by using either horizontal or vertical guillotine cuts. In k-stage packing, the number of cuts to obtain each rectangle from the initial packing is at most k, plus an additional cut to trim (i.e., separate the rectangles itself from a waste area). Intuitively, this means that in the cutting process we change the orientation of the cuts $k - 1$ times.

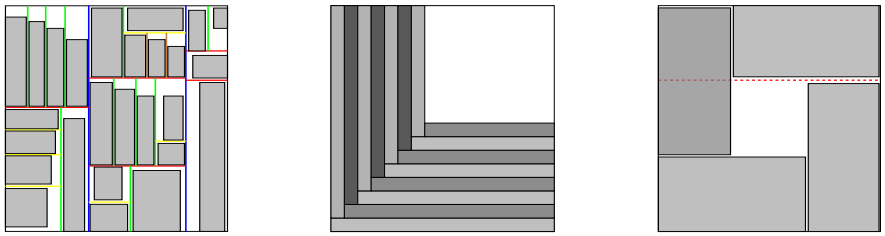


Figure 1 Packing (a) is a 5-stage guillotine separable packing, packing (b) is a $(n-1)$-stage guillotine separable packing, packing (c) is not guillotine separable as any end-to-end cut in the strip intersects a rectangle.

Therefore, in this paper, we study the Guillotine Strip Packing problem (GSP). The input is the same as for SP, but we require additionally that the items in the solution can be separated by a sequence of guillotine cuts, and we say then that they are *guillotine separable*. Like general SP without requiring the items to be guillotine separable, GSP generalizes Bin Packing (when all items have the same height) and makespan minimization on identical machines (when all items have the same width). Thus, it is strongly NP-hard, and the same reduction from Partition mentioned above yields a lower bound of $(3/2-\varepsilon)$ for polynomial time algorithms (see the full version [31] for more details). For asymptotic approximation, GSP is well understood. Kenyon and Rémila [29] gave an asymptotic polynomial time approximation scheme (APTAS) for (general) SP. Their algorithm produces a 5-stage packing (hence, guillotine separable), and thus yields an APTAS for GSP as well. Later, Seiden et al. [43] settled the asymptotic approximation status of GSP under k-stage packing. They gave an APTAS for GSP using 4-stage guillotine cuts, and showed $k=2$ stages cannot guarantee any bounded asymptotic performance ratio, and $k=3$ stages lead to asymptotic performance ratios close to 1.691. However, in the non-asymptotic setting, approximation ratio of GSP is not yet settled. Steinberg's algorithm [45] yields a 2-approximation algorithm for GSP and this is the best known polynomial time approximation algorithm for the problem.

In this paper we present approximation algorithms for GSP which have strictly better approximation ratios than the best known algorithms for SP, and in the setting of pseudo-polynomial time algorithms we even beat the lower bound that holds for SP. Moreover, we show that all our approximation ratios are essentially the best possible.

1.1 Our Contribution

We present a polynomial time $(3/2+\varepsilon)$-approximation algorithm for GSP. Due to the mentioned lower bound of $(3/2-\varepsilon)$, our approximation ratio is essentially tight. Also, we present a pseudo-polynomial time $(1+\varepsilon)$-approximation algorithm, which is also essentially tight since GSP is strongly NP-hard.

For the pseudo-polynomial time $(1+\varepsilon)$-approximation, we first prove that there exists a structured solution with height at most $(1+\varepsilon)\mathrm{OPT}$ (OPT denotes the height of the optimal solution) in which the strip is divided into $O(1)$ rectangular boxes inside which the items are *nicely packed*, e.g., horizontal items are stacked on top of each other, vertical items are placed side by side, and small items are packed greedily with the Next-Fit-Decreasing-Height algorithm [13] (see Figure 2(a) and also Figure 4. Also, refer to Section 2 for item classification). This result starkly contrasts SP (i.e., where we do not require the items to be guillotine separable): for that problem, it is already unlikely that we can prove that there

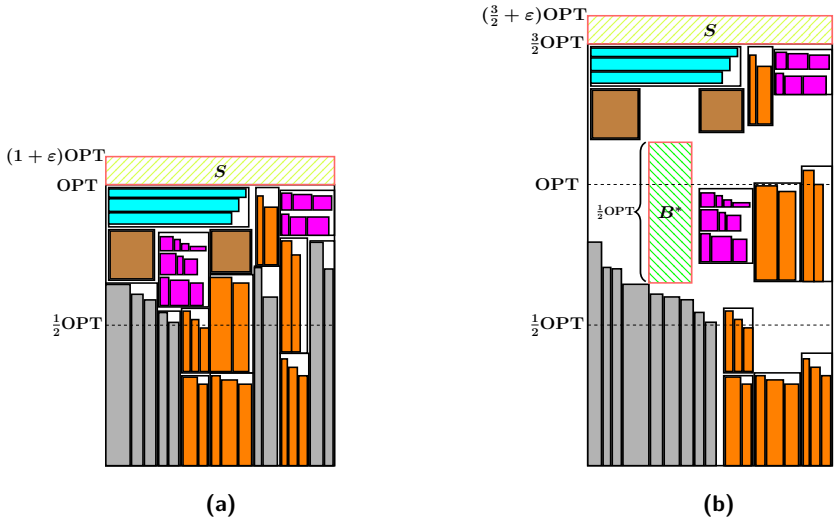


Figure 2 (a) A guillotine separable structured packing (for the pseudo-polynomial time approximation scheme) where all the items are packed nicely in containers. The tall items (dark-gray) are stacked next to each other just like the vertical items (orange); the horizontal items (blue) are stacked on top of each other, the small items (pink) are packed according to NFDH, and the large containers contain single large items (brown). (b) A guillotine separable structured packing for the polynomial time $(3/2+\varepsilon)$-approximation, where the packing from (a) is rearranged such that the tall items are *bottom-left-flushed* and there is an extra empty box B^* to accommodate some of the vertical items which we are unable to pack in polynomial time in the rest of the guessed boxes. This arises from the NP-hardness of the PARTITION problem. The yellow rectangular strip S on top of both the packings is used for packing the medium and leftover horizontal and small items.

always exists such a packing with a height of less than $5/4 \cdot \mathrm{OPT}$. If we could prove this, we could approximate the problem in pseudo-polynomial time with a better ratio than $5/4$, which is NP-hard [25].

To construct our structured packing, we start with an optimal packing and use the techniques in [32] to obtain a packing in which each item is *nicely packed* in one of a constant number of boxes and L-shaped *compartments*. We increase the height of our packing by $\varepsilon\mathrm{OPT}$ in order to round the heights of the packed items and get some leeway within the packing. Then, we rearrange the items placed inside the L-shaped compartments. Here, we crucially exploit that the items in the initial packing are guillotine separable. In particular, this property allows us to identify certain sets of items that we can swap, e.g., items on the left and the right of a vertical guillotine cut to simplify the packing, and reduce the number of boxes to $O(1)$. Then, using standard techniques, we compute a solution with this structure in pseudo-polynomial time and hence with a packing height of at most $(1+\varepsilon)\mathrm{OPT}$ (see Figure 2 (a)).

Note that we do not obtain a $(1+\varepsilon)$-approximation algorithm in polynomial time in this way. The reason is that when we pack the items into the rectangular boxes, we need to solve a generalization of PARTITION: there can be several boxes in which vertical items are placed side by side, and we need that the widths of the items in each box sum up to at most the width of the box. If there is only a single item that we cannot place, then we would need to place it on top of the packing, which can increase our packing height by up to OPT.

For our polynomial time $(3/2+\varepsilon)$-approximation algorithm, we, therefore, need to be particularly careful with the items whose height is larger than $\mathrm{OPT}/2$, which we call the *tall items*. We prove a different structural result which is the main technical contribution of

this paper: we show that there is always a $(3/2+\varepsilon)$-approximate packing in which the tall items are packed together in a *bottom-left-flushed* way, i.e., they are ordered non-increasingly by height and stacked next to each other with their bottom edges touching the base of the strip. All remaining items are nicely packed into $O_\varepsilon(1)$ boxes, and there is also an empty strip of height OPT/2 and width $\Omega_\varepsilon(W)$, see Figure 2 (b). Thus, it is very easy to pack the tall items correctly according to this packing. We pack the remaining items with standard techniques into the boxes. In particular, the mentioned empty strip allows us to make slight mistakes while we pack the vertical items that are not tall; without this, we would still need to solve a generalization of PARTITION.

In order to obtain our structural packing for our polynomial time $(3/2+\varepsilon)$-approximation algorithm, we build on the idea of the packing for the pseudo-polynomial time $(1+\varepsilon)$-approximation. Using that it is guillotine separable, we rearrange its items further. First, we move the items such that all tall items are at the bottom. To achieve this, we again argue that we can swap certain sets of items, guided by the guillotine cuts. Then, we shift certain items up by OPT/2, which leaves empty space between the shifted and the not-shifted items, see Figure 2 (b). Inside this empty space, we place the empty box of height OPT/2. Also, we use this empty space in order to be able to reorder the tall items on the bottom by their respective heights. During these changes, we ensure carefully that the resulting packing stays guillotine separable.

It is possible that also for (general) SP there always exists a structured packing of height at most $(3/2+\varepsilon)$OPT, similar to our packing. This would yield an essentially tight polynomial time $(3/2+\varepsilon)$-approximation for SP and thus solve the long-standing open problem to find the best possible polynomial time approximation ratio for SP. We leave this as an open question.

1.2 Other related work

In the 1980s, Baker et al. [2] initiated the study of approximation algorithms for strip packing, by giving a 3-approximation algorithm. After a sequence of improved approximations [13, 44], Steinberg [45] and Schiermeyer [41] independently gave 2-approximation algorithms. For asymptotic approximation, Kenyon and Rémila [29] settled SP by providing an APTAS.

SP has rich connections with important geometric packing problems [9, 30] such as 2D bin packing (2BP) [4, 33], 2D geometric knapsack (2GK) [19, 28], dynamic storage allocation [7], maximum independent set of rectangles (MISR) [22, 1], sliced packing [14, 18], etc.

In 2BP, we are given a set of rectangles and square bins, and the goal is to find an axis-aligned non-overlapping packing of all items into a minimum number of bins. The problem admits no APTAS [3], and the present best approximation ratio is 1.406 [4]. In 2GK, we are given a set of rectangular items and a square knapsack. Each item has an associated profit, and the goal is to pack a subset of items in the knapsack such that the profit is maximized. The present best polynomial time approximation ratio is 1.89 [19]. There is a pseudo-polynomial time $(4/3+\varepsilon)$-approximation [21] for 2GK. In MISR, we are given a set of (possibly overlapping) rectangles we need to find the maximum cardinality non-overlapping set of rectangles. Recently, Mitchell [36] gave the first constant approximation algorithm for the problem. Then Gálvez et al. [22] obtained a $(2+\varepsilon)$-approximation algorithm for MISR. Their algorithms are based on a recursive geometric decomposition of the plane, which can be viewed as a generalization of guillotine cuts, more precisely, to cuts with $O(1)$ bends. Pach and Tardos [38] even conjectured that for any set of n non-overlapping axis-parallel rectangles, there is a guillotine cutting sequence separating $\Omega(n)$ of them.

2BP and 2GK are also well-studied in the guillotine setting [39]. Caprara et al. [8] gave an APTAS for 2-stage SP and 2-stage BP. Later, Bansal et al. [5] showed an APTAS for guillotine 2BP. Bansal et al. [4] conjectured that the worst-case ratio between the best guillotine 2BP and the best general 2BP is 4/3. If true, this would imply a $(\frac{4}{3}+\varepsilon)$-approximation algorithm for 2BP. For guillotine 2GK, Khan et al. [32] recently gave a pseudo-polynomial time approximation scheme.

2 Pseudo-polynomial time approximation scheme

In this section, we present our pseudo-polynomial time approximation scheme (PPTAS) for GSP.

Let $\varepsilon > 0$ and assume w.l.o.g. that $1/\varepsilon \in \mathbb{N}$. We denote by OPT the height of the optimal solution. We classify the input items into a few groups according to their heights and widths similar to the classification in [32]. For two constants $1 \geq \delta > \mu > 0$ to be defined later, we classify each item $i \in I$ as:

- *tall* if $h_i > \mathrm{OPT}/2$;
- *large* if $w_i > \delta W$ and $\mathrm{OPT}/2 \geq h_i > \delta\mathrm{OPT}$;
- *horizontal* if $w_i > \delta W$ and $h_i \leq \mu\mathrm{OPT}$;
- *vertical* $w_i \leq \delta W$ and $\mathrm{OPT}/2 \geq h_i > \delta\mathrm{OPT}$;
- *medium if*
  - either $\delta\mathrm{OPT} \geq h_i > \mu\mathrm{OPT}$;
  - or $\delta W \geq w_i > \mu W$ and $h_i \leq \mu\mathrm{OPT}$;
- *small* if $w_i \leq \mu W$ and $h_i \leq \mu\mathrm{OPT}$;

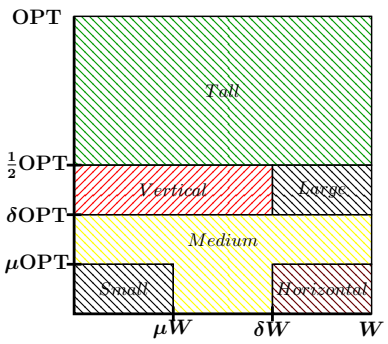


Figure 3 Item Classification: x-axis represents width and y-axis represents height.

See Figure 3 for a picture of item classification. Let $I_{tall}, I_{large}, I_{hor}, I_{ver}, I_{medium}, I_{small}$ be the set of tall, large, horizontal, medium, and small rectangles in I, respectively.

Using the following lemma, one can appropriately choose μ, δ such that the medium items occupy a marginal area. This effectively allows us to ignore them in our main argumentation.

▶ **Lemma 2.1** ([37]). *Let $\varepsilon > 0$ and $f(.)$ be any positive increasing function such that $f(x) < x$ for all $x \in (0, 1]$. Then we can efficiently find $\delta, \mu \in \Omega_\varepsilon(1)$, with $\varepsilon \geq f(\varepsilon) \geq \delta \geq f(\delta) \geq \mu$ so that the total area of medium rectangles is at most $\varepsilon(\mathrm{OPT} \cdot W)$.*

We will specify how we choose the function $f(x)$ later. In our PPTAS, we will use a packing, which is defined solely via boxes.

▶ **Definition 2.2.** *A* box B *is an axis-aligned open rectangle that satisfies* $B \subseteq [0, W] \times [0, \infty)$. *We denote by* $h(B)$ *and* $w(B)$ *the height and the width of* B*, respectively.*

Inside each box B, we will place the items *nicely,* meaning that they are either stacked horizontally or vertically, or B contains a single large item, or only small items, or only medium items. This is useful since in the first two cases, it is trivial to place a given set of items into B, and in the last two cases, it will turn out that it suffices to pack the items greedily using the Next Fit Decreasing Height (NFDH) algorithm [13] and Steinberg's algorithm [45], respectively. There will be one box with height at most $2\varepsilon\mathrm{OPT}$ that contains all medium items.

▶ **Definition 2.3** (Nice packing)**.** *Let* B *be a box and let* $I_B \subseteq I$ *be a set of items that are placed non-overlappingly inside* B*. We say that the packing of* I_B *in* B *is* nice *if the items in* I_B *are guillotine separable and additionally*

- I_B *contains only one item, or*
- $I_B \subseteq I_{hor}$ *and the items in* I_B *are stacked on top of each other inside* B*, or*
- $I_B \subseteq I_{tall} \cup I_{ver}$ *and the items in* I_B *are placed side by side inside* B*, or*
- $I_B \subseteq I_{medium}$*, or*
- $I_B \subseteq I_{small}$ *and for each item* $i \in I_B$ *it holds that* $w_i \leq \varepsilon \cdot w(B)$ *and* $h_i \leq \varepsilon \cdot h(B)$.

We will use the term *container* to refer to a box B that contains a nice packing of some set of items I_B. See Figure 4 for nice packings in different types of containers. We say that a set of boxes $\mathcal{B}$ is *guillotine separable* if there exists a sequence of guillotine cuts that separates them and that does not intersect any box in $\mathcal{B}$.

We now state the structural lemma for the PPTAS. Intuitively, it states that there exists a $(1+\varepsilon)$-approximate solution in which the input items are placed into $O_\varepsilon(1)$ boxes such that within each box the packing is nice. We remark that we will crucially use that in the optimal packing the items in I are guillotine separable. In fact, if one could prove that there exists such a packing with $O_\varepsilon(1)$ boxes and a height of $\alpha\mathrm{OPT}$ for some $\alpha < \frac{5}{4}$ also in the non-guillotine case (where neither the optimal solution nor the computed solution needs to be guillotine separable), then one would obtain a pseudo–polynomial time $(\alpha+\varepsilon)$-approximation algorithm also in this case, by using straightforward adaptations of the algorithms in, e.g., [37, 20, 27] or our algorithm in section 2.2. However, this is not possible for $\alpha < \frac{5}{4}$, unless $\mathsf{P} = \mathsf{NP}$ [25].

▶ **Lemma 2.4** (Structural lemma 1)**.** *Assume that* μ *is sufficiently small compared to* δ*. Then there exists a set* $\mathcal{B}$ *of* $O_\varepsilon(1)$ *pairwise non-overlapping and guillotine separable boxes all placed inside* $[0, W] \times [0, (1 + 16\varepsilon)\mathrm{OPT})$ *and a partition* $I = \bigcup_{B \in \mathcal{B}} I_B$ *such that for each* $B \in \mathcal{B}$ *the items in* I_B *can be placed nicely into* B*.*

We choose our function f due to Lemma 2.1 such that μ is sufficiently small compared to δ, as required by Lemma 2.4. We will prove Lemma 2.4 in the next subsection. In its packing, let $\mathcal{B}_{hor}, \mathcal{B}_{ver}, \mathcal{B}_{tall}, \mathcal{B}_{large}, \mathcal{B}_{small}$ and $\mathcal{B}_{med}$ denote the set of boxes for the horizontal, vertical, tall, large, small and medium items, respectively. Let $\mathcal{B}_{tall+ver} := \mathcal{B}_{tall} \cup \mathcal{B}_{ver}$.

2.1 Proof of Structural Lemma 1

In this section we prove Lemma 2.4. We have omitted a few proofs which can be found in the full version [31]. Our strategy is to start with a structural lemma from [32] that guarantees the existence of a structured packing of all items in $I_{hard} := I_{tall} \cup I_{large} \cup I_{hor} \cup I_{ver}$. This packing uses boxes and **L**-compartments. Note that, for now we ignore the items I_{small}. We will show how to pack them later.

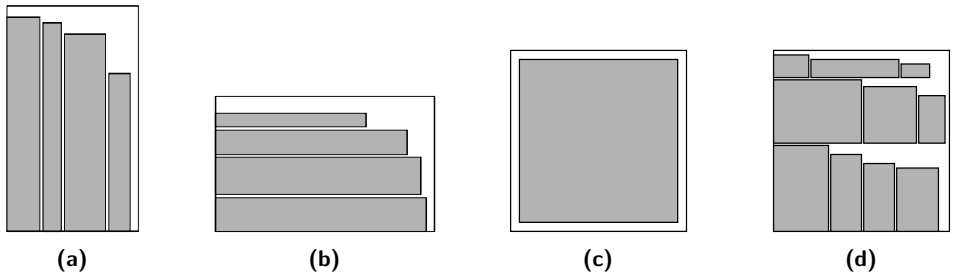


Figure 4 Nice packing of vertical, horizontal, large and small items in their respective containers.

▶ **Definition 2.5** (**L**-compartment). *An* **L***-compartment L is an open sub-region of $[0, W] \times [0, \infty)$ bounded by a simple rectilinear polygon with six edges $e_0, e_1, \ldots, e_5$ such that for each pair of horizontal (resp. vertical) edges e_i, e_{6-i} with $i \in \{1, 2\}$ there exists a vertical (resp. horizontal) line segment ℓ_i of length less than $\delta \frac{\mathrm{OPT}}{2}$ (resp. $\delta \frac{W}{2}$) such that both e_i and e_{6-i} intersect ℓ_i but no other edges intersect ℓ_i.*

Note that for an **L**-compartment, no item $i \in I_{hor}$ can be packed in its vertical arm and similarly, no item $i \in I_{ver} \cup I_{tall}$ can be packed in its horizontal arm.

The next lemma follows immediately from a structural insight in [32] for the guillotine two-dimensional knapsack problem. It partitions the region $[0, W] \times [0, \mathrm{OPT}]$ into non-overlapping boxes and **L**-compartments that admit a *pseudo-guillotine cutting sequence*. This is a sequence of cuts in which each cut is either a (normal) guillotine cut, or a special cut that cuts out an **L**-compartment L from the current rectangular piece R in the cutting sequence, such that $R \setminus L$ is a rectangle, see Figure 6. So intuitively L lies at the boundary of R.

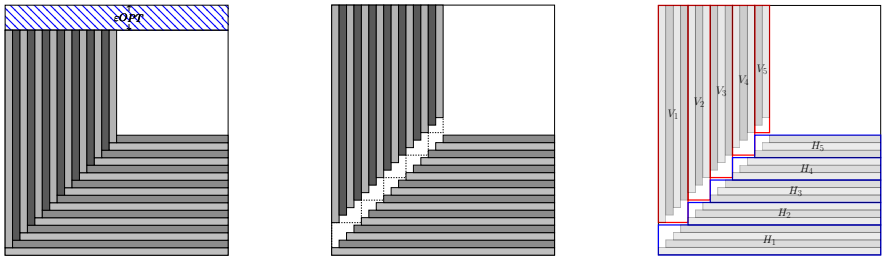


Figure 5 Using an extra $\varepsilon\mathrm{OPT}$ height, we convert a packing of items I in an **L**-compartment into another packing such that the items in I are packed in boxes $\mathcal{C}' = \mathbf{V} \cup \mathbf{H}$, which are guillotine separable and $|\mathcal{C}'| = O_\varepsilon(1)$, where $\mathbf{V} = \cup_{i=1}^{i=5} V_i$ and $\mathbf{H} = \cup_{i=1}^{i=5} H_i$.

▶ **Lemma 2.6** ([32]). *There exists a partition of $[0, W] \times [0, \mathrm{OPT}]$ into a set $\mathcal{B}_1$ of $O_\varepsilon(1)$ boxes and a set $\mathcal{L}$ of $O_\varepsilon(1)$* **L***-compartments such that*

- *the boxes and* **L***-compartments in $\mathcal{B}_1 \cup \mathcal{L}$ are pairwise non-overlapping,*
- *$\mathcal{B}_1 \cup \mathcal{L}$ admits a pseudo-guillotine cutting sequence,*
- *the items in I_{hard} can be packed into $\mathcal{B}_1 \cup \mathcal{L}$ such that for each $B \in \mathcal{B}_1$ it either contains only items $i \in I_{tall} \cup I_{large} \cup I_{ver}$ or it contains only items $i \in I_{hor}$.*

Our strategy is to take the packing due to Lemma 2.6 and transform it step by step until we obtain a packing that corresponds to Lemma 2.4. First, we round the heights of the tall, large, and vertical items such that they are integral multiples of $\delta^2\mathrm{OPT}$. Formally, for each item $i \in I_{tall} \cup I_{large} \cup I_{ver}$ we round its height to $h'_i := \left\lceil \frac{h_i}{\delta^2 \mathrm{OPT}} \right\rceil \delta^2 \mathrm{OPT}$. Let I_{hard} denote

the resulting set of items. By a shifting argument, we will show that we can still pack I_{hard} into $O_\varepsilon(1)$ guillotine separable boxes and **L**-compartments if we can increase the height of the packing by a factor $1+\varepsilon$ which also does not violate guillotine separability. Then, we increase the height of the packing by another factor $1+\varepsilon$. Using this additional space, we shift the items inside each **L**-compartment L such that we can separate the vertical items from the horizontal items (see Figure 5). Due to this separation, we can partition L into $O_\varepsilon(1)$ boxes such that each box contains only horizontal or only vertical and tall items. Note however, that they might not be packed nicely inside these boxes.

▶ **Lemma 2.7.** *There exists a partition of* $[0,W]\times[0,(1+2\varepsilon)\mathrm{OPT}]$ *into a set* $\mathcal{B}_2$ *of* $O_\varepsilon(1)$ *boxes such that*

- *the boxes in* $\mathcal{B}_2$ *are pairwise non-overlapping and admit a guillotine cutting sequence,*
- *the items in* I_{hard} *can be packed into* $\mathcal{B}_2$ *such that they are guillotine separable and each box* $B\in\mathcal{B}_2$ *either contains only items from* $I_{tall}\cup I_{large}\cup I_{ver}$*, or contains only items from* I_{hor}*.*
- *Any item* $i\in I_{tall}\cup I_{large}\cup I_{ver}$ *has height* $h'_i=k_i\delta^2\mathrm{OPT}$ *for integer* k_i*,* $k_i\leq 1/\delta^2+1$*.*

Let $\mathcal{B}_2$ be the set of boxes due to Lemma 2.7. Consider a box $B\in\mathcal{B}_2$ and let $I_{hard}(B)$ denote the items from I_{hard} that are placed inside B in the packing due to Lemma 2.7. Our goal is to partition B into $O_\varepsilon(1)$ smaller containers, i.e., the items in $I_{hard}(B)$ are packed nicely into these smaller boxes. If B contains horizontal items, then this can be done using standard techniques, e.g., by 1D resource augmentation (only in height) in [32]. This resource augmentation procedure maintains guillotine separability.

▶ **Lemma 2.8** ([32])**.** *Given a box* $B\in\mathcal{B}_2$ *such that* B *contains a set of items* $I_{hard}(B)\subseteq I_{hor}$*. There exists a partition of* B *into* $O_{\varepsilon'}(1)$ *containers* $\mathcal{B}'$ *and one additional box* B' *of height at most* $\varepsilon' h(B)$ *and width* $w(B)$ *such that the containers* $\mathcal{B}'$ *are guillotine separable and the containers* $\mathcal{B}'\cup\{B'\}$ *contain* $I_{hard}(B)$*.*

We apply Lemma 2.8 to each box $B\in\mathcal{B}_2$ that contains a horizontal item. Consider the items which are contained in their respective boxes B'. In order to avoid any confusions between constants of our algorithm and resource augmentation, we denote the constant used for resource augmentation as ε'. We choose $\varepsilon'=\varepsilon$ and then their total area is at most $\varepsilon\mathrm{OPT}\cdot W$ and therefore, all such items can be packed in a box of height at most $2\varepsilon\mathrm{OPT}$ and width W using Steinberg's algorithm [45]. But since this will possibly not result in a nice packing we apply resource augmentation (only along height) again to ensure that we get a nice packing of such horizontal items in $O_\varepsilon(1)$ containers which can all be packed in a box of height at most $3\varepsilon\mathrm{OPT}$ and width W.

Consider now a box $B\in\mathcal{B}_2$ that contains at least one item from $I_{tall}\cup I_{large}\cup I_{ver}$. Let $I_{hard}(B)\subseteq I_{tall}\cup I_{large}\cup I_{ver}$ denote the items packed inside B. We argue that we can rearrange the items in $I_{hard}(B)$ such that they are nicely placed inside $O_\varepsilon(1)$ containers. In this step we crucially use that the items in $I_{hard}(B)$ are guillotine separable.

Consider the guillotine cutting sequence for $I_{hard}(B)$. It is useful to think of these cuts as being organized in *stages:* in the first stage we do vertical cuts (possibly zero cuts). In the following stage, we take each resulting piece and apply horizontal cuts. In the next stage, we again take each resulting piece and apply vertical cuts, and so on. Since the heights of the items in $I_{hard}(B)$ are rounded to multiples of $\delta^2\mathrm{OPT}$ we can assume w.l.o.g. that the y-coordinates of the horizontal cuts are all integral multiples of $\delta^2\mathrm{OPT}$ (possibly moving the items a little bit). Assume here for the sake of simplicity that $t=1/\delta^2$ is an integer. Because of the rounding of heights of the items in $I'_{hard}(B)$, there are at most $(1/\delta^2-1)$ y-coordinates for making a horizontal cut. For a horizontal stage of cuts, for a rectangular

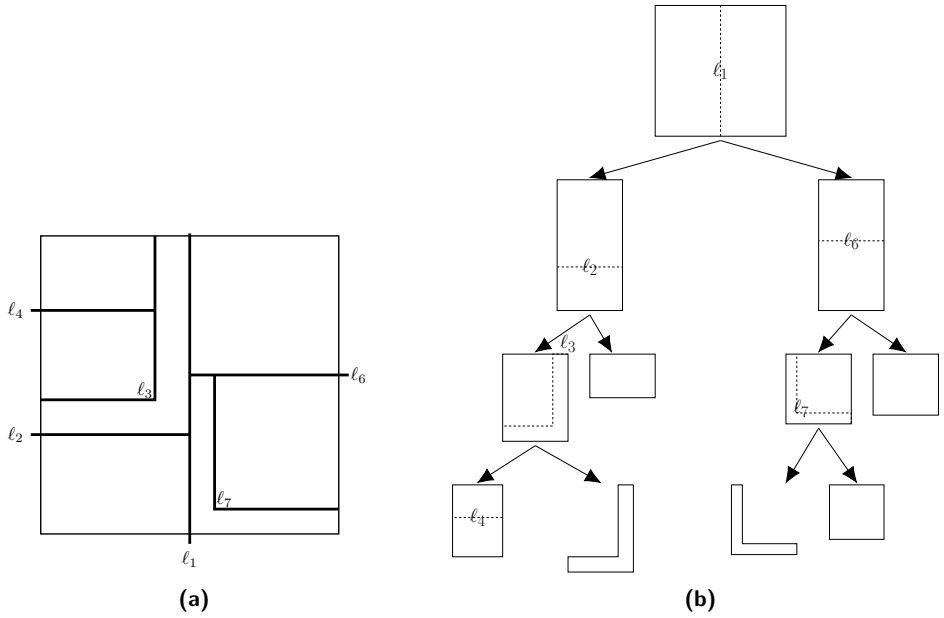


Figure 6 (a) A pseudo-guillotine cutting sequence. The first cut is l_1, and then the resulting left piece is further subdivided by ℓ_2, ℓ_3 and ℓ_4. Similarly, ℓ_6, ℓ_7 subdivide the right piece. Note that ℓ_3 and ℓ_7 are not guillotine cuts, but they cut out the corresponding **L**-compartments. (b) Step by step pseudo-guillotine cutting sequence corresponding to Figure (a). Dashed line at each level indicates a partition of a rectangle into two regions (two boxes, or one box and one **L**).

piece we define a *configuration vector* $(x_1, ..., x_{t-1})$: For each $i \in [t-1]$ if there is a horizontal cut in the piece at $y = t \cdot i$, then $x_i = 1$, otherwise $x_i = 0$. Consider $y = 0$ to be the bottom of the rectangular piece. Therefore, in each horizontal stage, for each piece there are at most $K := (2^{(1/\delta^2)})$ possible configurations. Consider the first stage (which has vertical cuts). If there are more than K vertical cuts then in two of the resulting pieces, in the second stage the same configuration of horizontal cuts is applied (see Figure 7).

We reorder the resulting pieces and their items such that pieces with the same configuration of horizontal cuts are placed consecutively. Therefore, in the first stage we need only K vertical cuts and we can have at most $(\frac{1}{\delta}2^{(1/\delta^2)})$ resulting pieces. We apply the same transformation to each stage with vertical cuts. Now observe that there can be at most $O(1/\delta)$ stages since there are at most $1/\delta$ possible tall, vertical or large items stacked on top of the other and thus at most $1/\delta$ stages with horizontal cuts. Therefore, after our transformations, we apply only $(\frac{1}{\delta}2^{(1/\delta^2)})^{\frac{1}{\delta}}$ cuts in total, in all stages in all resulting pieces. Thus, we obtain $O_\varepsilon(1)$ boxes at the end, in which the items are nicely packed. This leads to the following lemma.

▶ **Lemma 2.9.** *Given a box $B \in \mathcal{B}_2$ such that B contains a set of items $I_{hard}(B) \subseteq I_{tall} \cup I_{large} \cup I_{ver}$. There exists a partition of B into $O_\varepsilon(1)$ containers $\mathcal{B}'$ such that the containers $\mathcal{B}'$ are guillotine separable and contain the items $I_{hard}(B)$.*

We apply Lemma 2.9 to each box $B \in \mathcal{B}_2$ that contains an item from $I_{tall} \cup I_{large} \cup I_{ver}$. Thus, we obtain a packing of I_{hard} into a set of $O_\varepsilon(1)$ guillotine separable containers in which these items are nicely placed; we denote these containers by $\mathcal{B}_{hard}$. This yields directly a packing for the (original) items I_{hard} (without rounding). Finally, we partition the empty space of the resulting packing into more boxes, and one additional box that we place on top

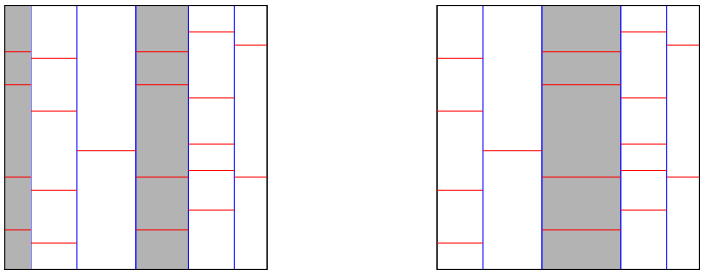


Figure 7 (a) 2 stages of guillotine cuts for a box containing vertical rectangles. (b) Since rounded heights of vertical rectangles are integral multiples of δ^2, merge configurations with same set of horizontal cuts to get $O_\delta(1)$ configurations.

of the current packing. We pack the items in I_{small} inside all these boxes. We might not be able to use some parts of the empty space, e.g., if two boxes are closer than μW to each other horizontally; however, if μ is sufficiently small compared to the number of boxes, this space is small and compensated by the additional box.

▶ **Lemma 2.10.** *Assume that μ is sufficiently small compared to δ. There exists a set of $O_\varepsilon(1)$ boxes $\mathcal{B}_{small}$, all contained in $[0, W] \times [0, (1+14\varepsilon)\mathrm{OPT}]$, such that the boxes in $\mathcal{B}_{hard} \cup \mathcal{B}_{small}$ are non-overlapping and guillotine separable and the items in I_{small} can be placed nicely into the boxes $\mathcal{B}_{small}$.*

Finally, we show the following lemma by using the fact that the medium items have area at most $\varepsilon(\mathrm{OPT} \cdot W)$ and by applying Steinberg's algorithm [45]. This completes the proof of Lemma 2.4.

▶ **Lemma 2.11.** *In time $n^{O(1)}$ we can find a nice placement of all items in I_{medium} inside one container B_{med} of height $2\varepsilon\mathrm{OPT}$ and width W.*

2.2 Algorithm

We describe now our algorithm that computes a packing of height at most $(1 + O(\varepsilon))\mathrm{OPT}$. First, we guess OPT and observe that there are at most $n \cdot h_{\max}$ possibilities, where $h_{\max} := \max_{i \in I} h_i$. Then, we guess the set of containers $\mathcal{B}$ due to Lemma 2.4 and their placement inside $[0, W] \times [0, (1 + O(\varepsilon))\mathrm{OPT})$. For each container $B \in \mathcal{B}$ we guess which case of Definition 2.3 applies to B, i.e., whether I_B contains only one item, $I_B \subseteq I_{hor}$, $I_B \subseteq I_{tall} \cup I_{ver}$, $I_B \subseteq I_{medium}$, or $I_B \subseteq I_{small}$. For each box $B \in \mathcal{B}$ for which I_B contains only one item $i \in I$, we guess i. Observe that for the remaining containers this yields independent subproblems for the sets I_{hor}, $I_{tall} \cup I_{ver}$, I_{medium}, and I_{small}. We solve these subproblems via similar routines as in [37, 20, 27].

We pack all medium items in I_{medium} into one single container B_{med} of height $2\varepsilon\mathrm{OPT}$ by Lemma 2.11. Then, for the sets I_{hor} and $I_{tall} \cup I_{ver}$ we pack their respective items into their containers using a standard pseudo-polynomial time dynamic program; we denote these containers by $\mathcal{B}_{hor}$ and $\mathcal{B}_{tall+ver}$, respectively. We crucially use that $|\mathcal{B}_{hor}| \leq O_\varepsilon(1)$ and $|\mathcal{B}_{tall+ver}| \leq O_\varepsilon(1)$. See the full version [31] for the details of packing of items in I_{hor} and $I_{tall} \cup I_{ver}$.

Finally, we pack the small items. From the proof of Lemma 2.10, apart from some items $I'_{small} \subset I_{small}$ which have area at most $\varepsilon\mathrm{OPT} \cdot W$, the other items can be packed nicely in the containers in $\mathcal{B}_{small} \setminus B_{small}$, where B_{small} has height $9\varepsilon\mathrm{OPT}$ and width W. Thus,

we use NFDH for packing the remaining small items. It can be shown that the small items which remain unpacked can be packed nicely in B_{small}, which is placed on the top of our packing.

▶ **Theorem 2.12.** *There is a $(1+\varepsilon)$-approximation algorithm for the guillotine strip packing problem with a running time of $(nW)^{O_\varepsilon(1)}$.*

3 Polynomial time $(\frac{3}{2}+\varepsilon)$-approximation

In this section, we first present the structural lemma for our polynomial time $(3/2+\varepsilon)$-approximation algorithm for guillotine strip packing. Then we describe our algorithm. We have omitted a few proofs which can be found in the full version [31].

To derive our structural lemma, we start with the packing due to Lemma 2.4. The problem is that with a polynomial time algorithm (rather than a pseudo-polynomial time algorithm) we might not be able to pack all tall items in their respective boxes. If there is even one single tall item i that we cannot pack, then we need to place i on top of our packing, which can increase the height of the packing by up to OPT.

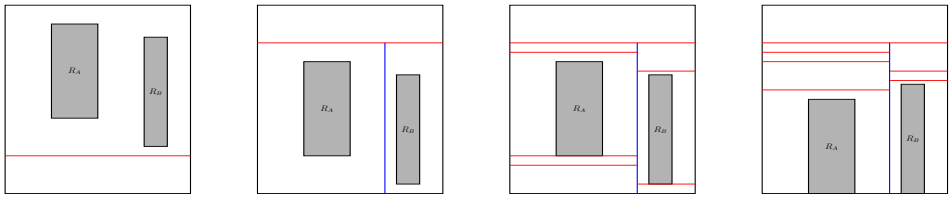


Figure 8 R_A and R_B are tall containers and by swapping the respective boxes (forming as a results of guillotine cuts) that contain them, they can be packed such that the bottoms of both containers intersect the bottom of the strip.

Therefore, we make our packing more robust to small errors when we pack the items into their boxes. In our changed packing, the tall items are *bottom-left-flushed* (see Figure 9(f)), the remaining items are packed into $O_\varepsilon(1)$ boxes, and there is one extra box B^* of height OPT/2 and width $\Omega_\varepsilon(W)$ which is empty. We will use the extra box B^* in order to compensate small errors when we pack the vertical items.

Formally, we say that in a packing, a set of items I' is *bottom-left-flushed* if they are ordered non-increasingly by height and stacked next to each other in this order within the strip $[0,W]\times[0,\infty)$ starting at the left edge of the strip, such that the bottom edge of each item $i\in I'$ touches the line segment $[0,W]\times\{0\}$. We now state the modified structural lemma for our polynomial time $(3/2+\varepsilon)$-approximation algorithm formally.

▶ **Lemma 3.1** (Structural lemma 2). *There exists a packing of the items I within $[0,W]\times[0,(3/2+O(\varepsilon))\mathrm{OPT})$ such that*

- *The items I_{tall} are bottom-left-flushed,*
- *There is a set $\mathcal{B}$ of $O_\varepsilon(1)$ containers that are pairwise non-overlapping and do not intersect the items in I_{tall},*
- *There is a partition of $I\setminus I_{tall}=\bigcup_{B\in\mathcal{B}} I_B$ such that for each $B\in\mathcal{B}$ the items in I_B can be placed nicely into B,*
- *There is a container $B^*\in\mathcal{B}$ of height OPT/2 and width $\varepsilon_1 W$ such that $I_{B^*}=\emptyset$,*
- *The items I_{tall} and the containers $\mathcal{B}$ together are guillotine separable.*

We now prove Lemma 3.1 in the following subsection.

3.1 Proof of Structural Lemma 2

We start with the packing due to Lemma 2.4 and transform it step by step. To obtain our packing, we first argue that we can ensure that all tall items are placed on the bottom of the strip, i.e., their bottom edges touch the bottom edge of the strip. Here we use that the initial packing is guillotine separable. Then we place the box B^* as follows. Suppose that there are initially C containers that cross the horizontal line with $y = \mathrm{OPT}/2$. Note that $C = O_\varepsilon(1)$ and $C \geq 1$ since at least one container is required to pack given non-zero number of items. Then, by an averaging argument we can show that there is a line segment l^* of length at least $\Omega(\frac{W}{C})$ which is the top edge of one of the containers B in the packing at some height $h^* \geq \mathrm{OPT}/2$. We push all the containers which completely lie above the line $y = h^*$ vertically upward by $\mathrm{OPT}/2$ and this creates enough space to pack B^* on top of B. After that, we take advantage of the gained extra space in order to ensure that the tall items are bottom-left-flushed.

Now we describe the proof formally. First, we define some constants. Let $g(\delta, \varepsilon) = O_\varepsilon(1)$ denote an upper bound on the number of containers in the packing obtained using Lemma 2.4, depending on ε and δ. Let $\varepsilon_1 = \frac{1}{3g(\delta,\varepsilon)}$, $\varepsilon_2 = \frac{\varepsilon}{4|\mathcal{B}_{hor}|}$, $\varepsilon_3 = \frac{\varepsilon_1}{4|\mathcal{B}_{ver}|}$, $\varepsilon_4 = \mu$, $\varepsilon_5 = \frac{\varepsilon_1\delta}{6}$, $\varepsilon_6 = \frac{\varepsilon\delta}{6}$.

Our first goal is to make sure that the tall items are all placed on the bottom of the strip $[0, W] \times [0, \infty)$. For this, we observe the following: suppose that in the guillotine cutting sequence a horizontal cut is placed. This cut separates the current rectangular piece R into two smaller pieces R_1 and R_2. Suppose that R_1 lies on top of R_2. Then only one of the two pieces R_1, R_2 can contain a tall item. Also, we obtain an alternative guillotine separable packing if we swap R_1 and R_2 – together with the items contained in them – within R. We perform this swap if R_1 contains a tall item. We apply this operation to each horizontal cut in the guillotine cutting sequence. As a result, we obtain a new packing in which all tall items are placed on the bottom of the strip (but possibly not yet bottom-left-flushed) as shown in Fig 8.

▶ **Lemma 3.2.** *There exists a set $\mathcal{B}$ of $O_\varepsilon(1)$ pairwise non-overlapping and guillotine separable boxes that are all placed inside $[0, W] \times [0, (1 + 16\varepsilon)\mathrm{OPT})$ and a partition $I = \bigcup_{B\in\mathcal{B}} I_B$ such that for each $B \in \mathcal{B}$ the items in I_B can be placed nicely into B. Also, for each box $B \in \mathcal{B}$ with $I_B \cap I_{tall} \neq \emptyset$ we have that the bottom edge of B intersects the line segment $[0, W] \times \{0\}$.*

Let $\mathcal{B}$ be the set of containers due to Lemma 3.2. We want to move some of them up in order to make space for the additional box B^*. To this end, we identify a horizontal line segment ℓ^* in the following lemma.

▶ **Lemma 3.3.** *There is a horizontal line segment ℓ^* of width at least $\varepsilon_1 W$ that does not intersect any container in $\mathcal{B}$, and such that the y-coordinate of ℓ^* is at least $\mathrm{OPT}/2$.*

Proof. Consider the containers in $\mathcal{B}$ that intersect with the horizontal line segment $\ell := [0, W] \times \{\mathrm{OPT}/2\}$ and let $p_1, ...p_k$ be the maximally long line segments on ℓ that do not intersect any container. Since the line segments $\{p_1, ...p_k\}$ are between containers in $\mathcal{B}$, we have that $k \leq |\mathcal{B}| + 1$. Therefore by an averaging argument we can find a horizontal line segment ℓ^* of width at least $\frac{W}{2(g(\delta,\varepsilon))+1} \geq \frac{W}{3g(\delta,\varepsilon)} \geq \varepsilon_1 W$ that either contains the top edge of one of these containers such that ℓ^* does not intersect any other container in $\mathcal{B}$ or ℓ^* is one of the line segments in the set $\{p_1, ..., p_k\}$. Hence, the y-coordinate of ℓ^* is at least $\mathrm{OPT}/2$. ◀

Let h^* be the y-coordinate of ℓ^*. We take all containers in $\mathcal{B}$ that lie "above h^*", i.e., that lie inside $[0, W] \times [h^*, \infty)$. We translate them up by $\mathrm{OPT}/2$. We define a container B^* which has height $\mathrm{OPT}/2$ and width $\varepsilon_1 W$ to be packed such that ℓ^* is the bottom edge of B^* (see Figure 9(b)). We then make the following claim about the resulting packing of $\mathcal{B} \cup \{B^*\}$ (we call this packing P_1).

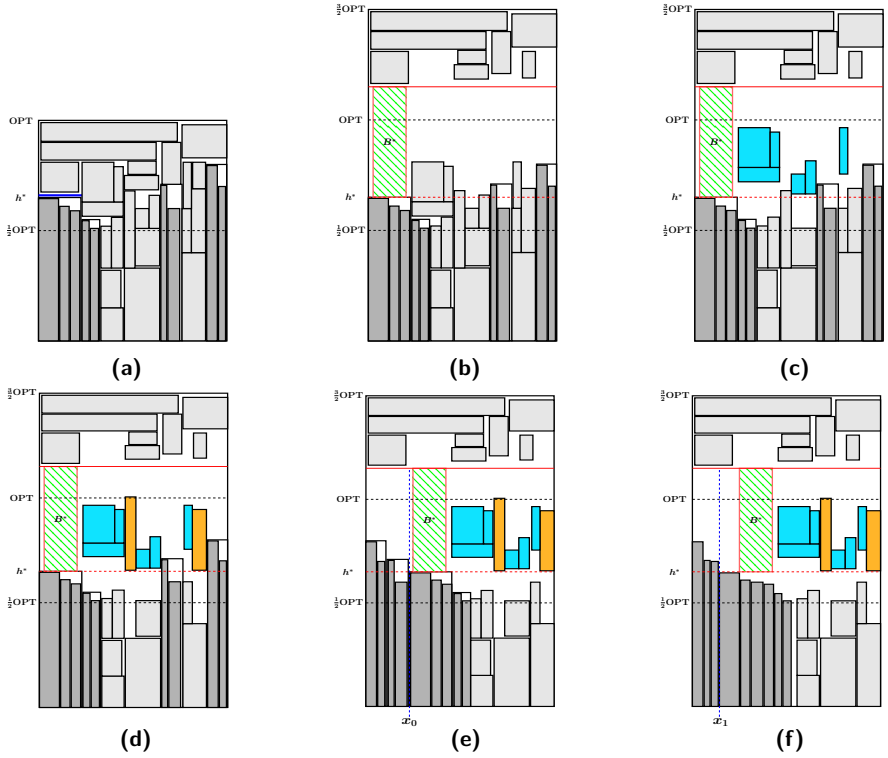


Figure 9 (a) A guillotine separable packing with items nicely packed in containers. The gray colored rectangles are the tall items and the light-gray rectangles are containers with items nicely packed inside. The blue line segment indicates l^* at height $h^\star$. (b) Items completely packed in $[h^*, \mathrm{OPT}]$ are shifted by $\frac{1}{2}\mathrm{OPT}$ vertically upward. The thick red line indicates $y = h^* + \frac{1}{2}\mathrm{OPT}$ which separates the items shifted up from the items below. The dashed red line indicates the height h^* and B^* is packed in the strip of sufficient width and lowest height h^*. (c) The containers of type 1 (colored blue) are moved accordingly so they do not intersect $y = h^\star$. (d) The containers of type 2 (colored yellow) are moved accordingly so they do not intersect $y = h^\star$. (e) The containers in $\mathcal{B}_{tall}$ are *bottom-left-flushed* while other non-tall containers are moved accordingly to the right. The blue vertical dashed line $x = x_0$ separates containers in $\mathcal{B}^+_{tall}$ to its left hand side from other containers to the right. (f) Final packing where tall items are *bottom-left-flushed* and the blue vertical dashed line $x = x_1$ separates items $i \in I_{tall}$ with $h_i > h^*$ to the left from other items and containers to the right.

▶ **Lemma 3.4.** *The packing P_1 is feasible, guillotine separable and has height* $(3/2 + O(\varepsilon))\mathrm{OPT}$.

Proof. Since $h^* > \mathrm{OPT}/2$, observe that no containers are intersecting the line $[0, W] \times \{h^* + \mathrm{OPT}/2\}$. This is because any containers which were lying above the line $[0, W] \times \{h^*\}$ before were pushed up by $\mathrm{OPT}/2$ and the height of such containers is at most $\mathrm{OPT}/2$. Thus, the first guillotine cut is applied at $y = h^* + \mathrm{OPT}/2$ so that we get two pieces R and R_{top}. For the guillotine separability of the top piece R_{top}, we use the fact that the packing to begin with was guillotine separable and we have moved a subset of the items in the initial packing vertically upwards by the same height. For the bottom piece R, which has a subset of the initial packing, we have packed B^* on the top edge (which is part of the line $[0, W] \times \{h^*\}$) of another container (say B) whose width is more than the width of B^*. In the guillotine cutting sequence of this piece without the addition of B^*, consider the horizontal cuts at height at least h^*. Note that there is no container lying completely above the line $[0, W] \times \{h^*\}$ in R.

Hence, we can remove such horizontal cuts and extend the vertical cuts that were intercepted by these horizontal cuts until they hit the topmost horizontal edge of R. Now, if we follow this new guillotine cutting sequence, we would finally have a rectangular region with only the container B. As there is no container in the region $[left(B), right(B)] \times [h^*, h^* + \mathrm{OPT}/2]$, we can pack B^* in this region without violating the guillotine separability condition. Now, observe that the height of the piece R_{top} is $(1 + O(\varepsilon))\mathrm{OPT} - h^*$ and height of the piece R is $h^* + \mathrm{OPT}/2$. Hence the height of the packing P_1 is $(3/2 + O(\varepsilon))\mathrm{OPT}$. ◀

Our next goal is to rearrange the tall items and their containers such that the tall items are bottom-left flushed. Let $\mathcal{B}_{tall} \subseteq \mathcal{B}$ denote the containers in $\mathcal{B}$ that contain at least one tall item. Consider the line segment $\ell := [0, W] \times \{h^*\}$ and observe that it might be intersected by containers in $\mathcal{B}_{tall}$. Let $\ell_1, \ell_2, ..., \ell_t$ be the connected components of $\ell \setminus \bigcup_{B \in \mathcal{B}_{tall}} B$. For each $j \in \{1, ..., t\}$ we do the following. Consider the containers in $\mathcal{B} \setminus \mathcal{B}_{tall}$ whose bottoms are contained in $\ell_j \times [\mathrm{OPT}/2, h^*]$ (we call them *type* 1 *containers*). We move them up by $h^* - \mathrm{OPT}/2$ units. There is enough space for them since the top edge of any of these containers lies below the line segment $[0, W] \times \{h^* + \mathrm{OPT}/2\}$ after shifting.

Then we take all containers in $\mathcal{B} \setminus \mathcal{B}_{tall}$ that intersect ℓ_j and also the line segment $[0, W] \times \{\mathrm{OPT}/2\}$ (*type* 2 *containers*). We move them up such that their respective bottom edges are contained in ℓ_j. Again there is enough space for this since the containers have height at most $\mathrm{OPT}/2$ and hence, their top edges cannot cross the line segment $[0, W] \times \{h^* + \mathrm{OPT}/2\}$. Note that in this step we do not necessarily move the affected containers uniformly. See Figure 9(c) and Figure 9(d) for a sketch. Note that due to the way ℓ^* is defined, no type 1 or type 2 container after being shifted overlaps with the region occupied by B^*.

One can show that the resulting packing is still guillotine separable. In particular, there is such a sequence that starts as follows: the first cut of this sequence is a horizontal cut with y-coordinate $h^* + \mathrm{OPT}/2$. For the resulting bottom piece R, there are vertical cuts that cut through the vertical edges of the containers in $\mathcal{B}_{tall}$ whose height is strictly greater than h^*, denote these containers by $\mathcal{B}^+_{tall}$. Let $R_1, ..., R_{t'}$ denote the resulting partition of R. We can rearrange our packing by reordering the pieces $R_1, ..., R_{t'}$. We reorder them such that on the left we place the pieces containing one container from $\mathcal{B}^+_{tall}$ each, sorted non-increasingly by their heights. Then we place the remaining pieces from $\{R_1, ..., R_{t'}\}$ (which hence, do not contain any containers in $\mathcal{B}^+_{tall}$), denote their union by R'. Let the left end of R' be $x = x_0$. We can assume that the guillotine cutting sequence places a vertical cut that separates R' from the other pieces in $\{R_1, ..., R_{t'}\}$ at $x = x_0$. From Lemma 3.3, we know that there is a container B (or possibly the case when $h^* = \mathrm{OPT}/2$ and we have a line segment ℓ' of width at least $\varepsilon_1 W$ on top of which we can pack B^*) whose top is at height h^*, has width at least ℓ^* which now lies to the right of x_0 in R'. Thus, the region $[left(B), left(B) + \varepsilon_1 W] \times [h^*, h^* + \mathrm{OPT}/2]$ is empty and can be used to place B^*.

We change now the placement of the containers within R'. Due to our rearrangements, no container inside R' intersects the line segment $[0, W] \times \{h^*\}$, so we can assume that R' is cut by the horizontal cut $[0, W] \times \{h^*\}$, let R'' be the resulting bottom piece and R''' be the piece above. We first show why R''' is guillotine separable. First, we separate B^* using vertical guillotine cuts at its left and right edges. Then we prove that the shifting operation for type 2 and type 1 containers does not violate guillotine separability of the packing for any region defined by some horizontal segment l_j for $j \in [t]$. Consider any type 2 container B'. Its top edge was initially lying above $y = h^*$ and its bottom below $\mathrm{OPT}/2$. Hence, before shifting this container no item could have been packed such that it was in the region $[0, W] \times [h^*, h^* + \mathrm{OPT}/2]$ and was intersecting the vertically extended line segments from the left and right edges of B' because any item packed in $[0, W] \times [h^*, \infty)$ initially was shifted

upward by OPT/2. Hence, after shifting B' such that its bottom touches $y = h^*$, after considering the cut $y = h^*$ in l_j, extend its left and right edges vertically upward to separate B' using guillotine cuts. For the type 1 containers, after the aforementioned cuts observe that all such containers have been shifted by an equal amount vertically upward and using the fact that they were guillotine separable initially, we claim that they are guillotine separable afterward. This is proved by considering the initial guillotine cuts that were separating such items and shifting the horizontal cuts upward by $h^* - \text{OPT}/2$ (equal to the distance the type 1 containers were shifted upward by).

To show that R'' is guillotine separable, observe that due to our rearrangements there are no containers that are completely contained in $R'' \cap ([0, W] \times [\text{OPT}/2, h^*])$. Therefore, we can assume that the next cuts for R'' are vertical cuts that contain all vertical edges of the boxes in $\mathcal{B}_{tall}$ that are contained in R''. Let $R''_1, ..., R''_{t''}$ denote the resulting pieces. Like above, we change our packing such that we reorder the pieces in $R''_1, ..., R''_{t''}$ non-increasingly by the height of the respective box in $\mathcal{B}_{tall}$ contained in them, and at the very right we place the pieces from $R''_1, ..., R''_{t''}$ that do not contain any container from $\mathcal{B}_{tall}$ (see Figure 9(e))

Finally, we sort the tall items inside the area $\bigcup_{B \in \mathcal{B}_{tall}} B$ non-increasingly by height so that they are bottom-left-flushed, and we remove the containers $\mathcal{B}_{tall}$ from $\mathcal{B}$ (see Figure 9(f)). We now prove that the tall items can be sorted inside the area $\bigcup_{B \in \mathcal{B}_{tall}} B$ non-increasingly by height without violating guillotine separability and feasibility. Note that the area $\bigcup_{B \in \mathcal{B}_{tall}} B$ can possibly contain some vertical items. Now, we reorder the tall items within R' such that they are sorted in non-increasing order of their heights. We do the same for all the tall items on the left of R'. There may be tall items (or vertical items) on the left hand side of R' such that for any such item, its height is less than the the tallest tall item in R'. Note that such tall items have to have a height of at most h^*. Such items can be repeatedly swapped with their neighboring tall item till they are in the correct position according to the *bottom-left-flushed* packing of the tall items, while maintaining guillotine separability. Such a swap operation between consecutive tall items ensures that all of the tall items and possibly some vertical items which were initially packed in tall containers remain inside the area $\bigcup_{B \in \mathcal{B}_{tall}} B$. We ensure that the vertical items which were packed to the left of R' get swapped so that they are packed on the right of all the tall items in a container. Now, to prove that guillotine separability of the packing is maintained after all such swapping operations, that is, after all tall items are sorted according to their heights in a non-increasing order consider the x-coordinate (say x_1) of the right edge of the shortest tall item which has height strictly greater than h^*. Observe that there were no tall containers of height strictly greater than h^* beyond $x = x_0$, which implies $x_1 \leq x_0$ and hence, now, for the guillotine cutting sequence, we can have a vertical guillotine cut at $x = x_1$ instead of at $x = x_0$, the rest being the same as mentioned before. This yields the packing claimed by Lemma 3.1.

3.2 Algorithm for polynomial time $(\frac{3}{2} + \varepsilon)$-approximation

First we guess a value OPT' such that $\text{OPT} \leq \text{OPT}' \leq (1 + \varepsilon)\text{OPT}$ in $n^{O_\varepsilon(1)}$ time (see the full version [31] for the details). In order to keep the notation light we denote OPT' by OPT. We want to compute a packing of height at most $(\frac{3}{2} + O(\varepsilon))\text{OPT}$ using Lemma 3.1.

Intuitively, we first place the tall items in a bottom-left-flushed way. Then we guess approximately the sizes of the boxes, place them in the free area, and place the items inside them via guessing the relatively large items, solving an instance of the generalized assignment problem (GAP), using NFDH for the small items, and invoking again Lemma 2.11 for the medium items. This is similar as in, e.g., [19, 28].

Formally, first we place all items in I_{tall} inside $[0, W] \times [0, (3/2 + \varepsilon)\text{OPT})$ such that they are bottom-left-flushed. Then, we guess approximately the sizes of the containers in $\mathcal{B}$. Note that in polynomial time we cannot guess the sizes of the containers exactly. Let $B \in \mathcal{B}$. Depending on the items packed inside B, we guess different quantities for B.

- If there is only one single large item $i \in I$ packed inside B then we guess i.
- If B contains only items from I_{hor} then we guess the widest item packed inside B. This defines our guessed width of B. Also, we guess all items packed inside B whose height is at least $\varepsilon_2\text{OPT}$ (at most $O(1/\varepsilon_2)$ many), denote them by I'_B. We guess the total height of the remaining items $I_B \setminus I'_B$ approximately by guessing the quantity $\hat{h}(B) := \left\lfloor \frac{h(I_B \setminus I'_B)}{\varepsilon_2 \text{OPT}} \right\rfloor \varepsilon_2\text{OPT}$. Our guessed height for B is then $\sum_{i \in I'_B} h(i) + \hat{h}(B)$.
- Similarly, if B contains only items from I_{ver} then we guess the highest item packed inside B, which defines our guessed height of B. Also, we guess all items packed inside B whose width is at least $\varepsilon_3 W$ (at most $O(1/\varepsilon_3)$ many), denote them by I'_B. We guess the total width of the remaining items $I_B \setminus I'_B$ approximately by guessing the quantity $\hat{w}(B) := \left\lfloor \frac{w(I_B \setminus I'_B)}{\varepsilon_3 \text{OPT}} \right\rfloor \varepsilon_3\text{OPT}$ and our guessed width of B is then $\sum_{i \in I'_B} w(i) + \hat{w}(B)$.
- If B contains only small items, then our guessed heights and widths of B are $\left\lfloor \frac{h(B)}{\varepsilon_4 \text{OPT}} \right\rfloor \varepsilon_4\text{OPT}$ and $\left\lfloor \frac{w(B)}{\varepsilon_4 W} \right\rfloor \varepsilon_4 W$, respectively.

Note that here $\varepsilon_2 = \frac{\varepsilon}{4|\mathcal{B}_{hor}|}$, $\varepsilon_3 = \frac{\varepsilon_1}{4|\mathcal{B}_{ver}|}$ and $\varepsilon_4 = \mu$ are chosen so that the unpacked horizontal items, unpacked vertical items and unpacked small items due to container rounding can be packed in containers B_{hor} (defined below), B^* and B_{small}, respectively.

We have at most $O_\varepsilon(1)$ containers and for each container $B \in \mathcal{B}$ we guess the type of container B and its respective width and height (depending on the type) in $n^{O_\varepsilon(1)}$ time.

Additionally, we guess three containers B_{med} of height $2\varepsilon\text{OPT}$, B_{hor} of height εOPT, and B_{small} of height $27\varepsilon\text{OPT}$ and width W each that we will use to place all medium items, and to compensate errors due to inaccuracies of our guesses for the sizes of the containers for horizontal and small items, respectively. Let $\mathcal{B}'$ denote the guessed containers (including B_{med}, B_{hor}, and B_{small}). Since $|\mathcal{B}'| = O_\varepsilon(1)$ and the containers in $\mathcal{B}'$ are not larger than the containers in $\mathcal{B}$, we can guess a placement for the containers $\mathcal{B}'$ such that together with I_{tall} they are guillotine separable. We place the containers B_{med}, B_{hor}, and B_{small} on top of the packing of rest of the containers in $\mathcal{B}'$, and I_{tall}.

▶ **Lemma 3.5.** *In time $n^{O_\varepsilon(1)}$ we can compute a placement for the containers in $\mathcal{B}'$ such that together with the items I_{tall}, they are guillotine separable.*

Next, we place the vertical items. Recall that for each container $B \in \mathcal{B}$ containing items from I_{ver} we guessed the items packed inside B whose width is at least $\varepsilon_3 W$. For each such container B we pack these items into the container $B' \in \mathcal{B}'$ that corresponds to B. With a similar technique as used for the generalized assignment problem (GAP) [19], we place all but items with width at most $\varepsilon_3 W$ for each container in I_{ver}. Further using the PTAS for this variant of GAP, we can ensure that items of total area at most $3\varepsilon_5 \cdot \text{OPT} \cdot W$ are not packed. Hence, items of total width at most $(3\varepsilon_5/\delta)W$ remain unpacked as each such item has height at least δOPT. We pack these remaining items into B^*, using that each of them has a height of at most $\text{OPT}/2$ and that their total width is at most $|\mathcal{B}'| \cdot 2\varepsilon_3 W + (3\varepsilon_5/\delta)W \leq \varepsilon_1 W = w(B^*)$. In other words, we fail to pack some of the vertical items since we guessed the widths of the containers only approximately and since our polynomial time approximation algorithm for GAP might not find the optimal packing. We use a similar procedure for the items in I_{hor} where instead of B^* we use B_{hor} in order to place the unassigned items.

▶ **Lemma 3.6.** *In time $n^{O_\varepsilon(1)}$ we can compute a placement for all items in $I_{ver} \cup I_{hor}$ in B^*, B_{hor}, and their corresponding boxes in $\mathcal{B}'$.*

For the medium items we invoke again Lemma 2.11 and we place B_{med} on top of the containers in $\mathcal{B}$ which increases the height of the packing only by $2\varepsilon\mathrm{OPT}$.

Finally, we use NFDH again to pack the small items into their corresponding containers in $\mathcal{B}'$, which we denote by $\mathcal{B}'_{small}$, and B_{small}. We need B_{small} due to inaccuracies of NFDH and of our guesses of the container sizes.

▶ **Lemma 3.7.** *In time $n^{O(1)}$ we can compute a placement for all items in I_{small} in $\mathcal{B}'_{small}$ and B_{small}.*

▶ **Theorem 3.8.** *There is a $(3/2+\varepsilon)$-approximation algorithm for the guillotine strip packing problem with a running time of $n^{O_\varepsilon(1)}$.*

4 Conclusion and Open problems

We were able to show essentially tight approximation algorithms for GSP in both the polynomial and the pseudo-polynomial settings. This was possible due to the structure of the respective optimal packings since they are guillotine separable. However, it is unclear how to obtain such a structured packing in the general case of SP, and the question remains to close the gap between the best approximation guarantee of $(5/3+\varepsilon)$ and the lower bound of $3/2$. Another interesting open problem related to guillotine cuts is to find out whether there exists a PTAS for the 2D guillotine geometric knapsack (2GGK) problem.

References

1 Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. ACM*, 66(4):29:1–29:40, 2019.

2 Brenda S Baker, Edward G Coffman, Jr, and Ronald L Rivest. Orthogonal packings in two dimensions. *SIAM Journal on computing*, 9(4):846–855, 1980.

3 Nikhil Bansal, Jose R Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31:31–49, 2006.

4 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *SODA*, pages 13–25, 2014.

5 Nikhil Bansal, Andrea Lodi, and Maxim Sviridenko. A tale of two dimensional bin packing. In *FOCS*, pages 657–666, 2005.

6 István Borgulya. An eda for the 2d knapsack problem with guillotine constraint. *Central European Journal of Operations Research*, 27(2):329–356, 2019.

7 Adam L. Buchsbaum, Howard J. Karloff, Claire Kenyon, Nick Reingold, and Mikkel Thorup. OPT versus LOAD in dynamic storage allocation. *SIAM J. Comput.*, 33(3):632–646, 2004.

8 Alberto Caprara, Andrea Lodi, and Michele Monaci. Fast approximation schemes for two-stage, two-dimensional bin packing. *Mathematics of Operations Research*, 30(1):150–172, 2005.

9 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

10 Nicos Christofides and Charles Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30–44, 1977.

11 François Clautiaux, Ruslan Sadykov, François Vanderbeck, and Quentin Viaud. Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. *Discrete Optimization*, 29:18–44, 2018.

12 François Clautiaux, Ruslan Sadykov, François Vanderbeck, and Quentin Viaud. Pattern-based diving heuristics for a two-dimensional guillotine cutting-stock problem with leftovers. *EURO Journal on Computational Optimization*, 7(3):265–297, 2019.

13 Edward G. Coffman, Jr, Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980.

14 Max A Deppert, Klaus Jansen, Arindam Khan, Malin Rau, and Malte Tutas. Peak demand minimization via sliced strip packing. In *APPROX/RANDOM*, volume 207, pages 21:1–21:24, 2021.

15 Alessandro Di Pieri. Algorithms for two-dimensional guillotine packing problems. Master's thesis, University of Padova, Italy, 2013.

16 Mohammad Dolatabadi, Andrea Lodi, and Michele Monaci. Exact algorithms for the two-dimensional guillotine knapsack. *Computers & Operations Research*, 39(1):48–53, 2012.

17 Fabio Furini, Enrico Malaguti, and Dimitri Thomopulos. Modeling two-dimensional guillotine cutting problems via integer programming. *INFORMS Journal on Computing*, 28(4):736–751, 2016.

18 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, and Kamyar Khodamoradi. Approximation algorithms for demand strip packing. In *APPROX/RANDOM*, volume 207, pages 20:1–20:24, 2021.

19 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via L-packings. *ACM Trans. Algorithms*, 17(4):33:1–33:67, 2021.

20 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, and Arindam Khan. Improved pseudo-polynomial-time approximation for strip packing. In *FSTTCS*, pages 9:1–9:14, 2016.

21 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved approximation algorithms for 2-dimensional knapsack: Packing into multiple l-shapes, spirals, and more. In *SoCG*, volume 189 of *LIPIcs*, pages 39:1–39:17, 2021.

22 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A (2+ϵ)-approximation algorithm for maximum independent set of rectangles. *CoRR*, abs/2106.00623, 2021. `arXiv:2106.00623`.

23 P. C. Gilmore and Ralph E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations research*, 13(1):94–120, 1965.

24 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A (5/3 + ϵ)-approximation for strip packing. *Computational Geometry*, 47(2):248–267, 2014.

25 Sören Henning, Klaus Jansen, Malin Rau, and Lars Schmarje. Complexity and inapproximability results for parallel task scheduling and strip packing. *Theory of Computing Systems*, 64(1):120–140, 2020.

26 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

27 Klaus Jansen and Malin Rau. Closing the gap for pseudo-polynomial strip packing. In *ESA*, volume 144 of *LIPIcs*, pages 62:1–62:14, 2019.

28 Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *SODA*, pages 204–213, 2004.

29 Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000.

30 Arindam Khan. *Approximation algorithms for multidimensional bin packing*. PhD thesis, Georgia Institute of Technology, 2015.

31 Arindam Khan, Aditya Lonkar, Arnab Maiti, Amatya Sharma, and Andreas Wiese. Tight approximation algorithms for two dimensional guillotine strip packing. *arXiv preprint*, 2022. `arXiv:2202.05989`.

32 Arindam Khan, Arnab Maiti, Amatya Sharma, and Andreas Wiese. On guillotine separable packings for the two-dimensional geometric knapsack problem. In *SoCG*, volume 189, pages 48:1–48:17, 2021.

33 Arindam Khan and Eklavya Sharma. Tight approximation algorithms for geometric bin packing with skewed items. In *APPROX/RANDOM*, volume 207 of *LIPIcs*, pages 22:1–22:23, 2021.

34 Andrea Lodi, Michele Monaci, and Enrico Pietrobuoni. Partial enumeration algorithms for two-dimensional bin packing problem with guillotine constraints. *Discrete Applied Mathematics*, 217:40–47, 2017.

35 Michael L McHale and Roshan P Shah. Cutting the guillotine down to size. *PC AI*, 13:24–26, 1999.

36 Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. *CoRR*, abs/2101.00326, 2021. `arXiv:2101.00326`.

37 Giorgi Nadiradze and Andreas Wiese. On approximating strip packing with a better ratio than 3/2. In *SODA*, pages 1491–1510, 2016.

38 János Pach and Gábor Tardos. Cutting glass. In *SoCG*, pages 360–369, 2000.

39 Enrico Pietrobuoni. *Two-dimensional bin packing problem with guillotine restrictions*. PhD thesis, University of Bologna, Italy, 2015.

40 Jakob Puchinger, Günther R Raidl, and Gabriele Koller. Solving a real-world glass cutting problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 165–176. Springer, 2004.

41 Ingo Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In *European Symposium on Algorithms*, pages 290–299. Springer, 1994.

42 W Schneider. Trim-loss minimization in a crepe-rubber mill; optimal solution versus heuristic in the 2 (3)-dimensional case. *European Journal of Operational Research*, 34(3):273–281, 1988.

43 Steven S. Seiden and Gerhard J. Woeginger. The two-dimensional cutting stock problem revisited. *Mathematical Programming*, 102(3):519–530, 2005.

44 Daniel Dominic Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Inf. Process. Lett.*, 10(1):37–40, 1980.

45 A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

46 Paul E. Sweeney and Elizabeth Ridenour Paternoster. Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

47 Lijun Wei and Andrew Lim. A bidirectional building approach for the 2d constrained guillotine knapsack packing problem. *European Journal of Operational Research*, 242(1):63–71, 2015.

A Study of Weisfeiler–Leman Colorings on Planar Graphs

Sandra Kiefer
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Daniel Neuen
School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

The Weisfeiler–Leman (WL) algorithm is a combinatorial procedure that computes colorings on graphs, which can often be used to detect their (non-)isomorphism. Particularly the 1- and 2-dimensional versions 1-WL and 2-WL have received much attentionattention, due to their numerous links to other areas of computer science.

Knowing the expressive power of a certain dimension of the algorithm usually amounts to understanding the computed colorings. An increase in the dimension leads to finer computed colorings and, thus, more graphs can be distinguished. For example, on the class of planar graphs, 3-WL solves the isomorphism problem. However, the expressive power of 2-WL on the class is poorly understood (and, in particular, it may even well be that it decides isomorphism).

In this paper, we investigate the colorings computed by 2-WL on planar graphs. Towards this end, we analyze the graphs induced by edge color classes in the graph. Based on the obtained classification, we show that for every 3-connected planar graph, it holds that: a) after coloring all pairs with their 2-WL color, the graph has fixing number 1 with respect to 1-WL, or b) there is a 2-WL-definable matching that can be used to transform the graph into a smaller one, or c) 2-WL detects a connected subgraph that is essentially the graph of a Platonic or Archimedean solid, a prism, a cycle, or a bipartite graph $K_{2,\ell}$. In particular, the graphs from case (a) are identified by 2-WL.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Finite Model Theory; Theory of computation → Graph algorithms analysis

Keywords and phrases Weisfeiler-Leman algorithm, planar graphs, edge-transitive graphs, fixing number

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.81

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2206.10557`

1 Introduction

The Weisfeiler–Leman (WL) algorithm [41] is a combinatorial procedure that given a graph G, computes a coloring on G which respects (and sometimes also detects) the symmetries in the graph. Its most prominent application is in theoretical [6, 8, 31] and practical approaches [2, 10, 30, 35, 36] to the graph isomorphism problem. The original algorithm by Weisfeiler and Leman is the 2-dimensional version and it colors pairs of vertices. Its generalization yields for every natural number k the k-dimensional WL algorithm k-WL, which iteratively refines a coloring of vertex k-tuples by aggregating local structural information encoded in the colors. Its final output is a coloring that is *stable* with respect to the criterion for partitioning the color classes, and graphs with different final colorings are never isomorphic.

Over the decades, fascination for the algorithm has persisted. This is to a large extent due to the discovery of numerous connections to other areas in computer science that are still being explored. For example, the algorithm has close links to linear and semidefinite

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 81; pp. 81:1–81:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

programming [4, 5, 25], homomorphism counting [11, 12], and machine learning [1, 21, 37, 39, 43]. Its expressive power can be characterized via winning strategies for the players in a particular type of Ehrenfeucht-Fraïssé game [8, 27]. Moreover, it is known that two graphs receive different final colorings with respect to k-WL if and only if the graphs can be distinguished via a formula in the counting-logic fragment C^{k+1} [8, 29].

In this work, we focus on the original version 2-WL, as introduced by Weisfeiler and Leman [41]. Besides the connections outlined for k-WL above, 2-WL has a precise correspondence to coherent configurations (see, e.g., [9]). Despite the simple and very natural concept behind the algorithm, its behavior is not well-understood and there is an extensive line of study to capture its expressive power. For example, one branch of research aims at understanding which graph properties can be detected by 2-WL. In this direction, Fürer [16] as well as Arvind et al. and Fuhlbrück et al. [3, 15] obtained insights concerning the ability of 2-WL to detect and count small subgraphs. Furthermore, the algorithm is able to detect 2-separators in graphs and implicitly computes the decomposition of a graph into its 3-connected components [32].

A related line of research analyzes which graphs are *identified* by 2-WL, i.e., on which graphs 2-WL serves as a complete isomorphism test. Positive examples include interval graphs [13] and distance-hereditary graphs [17] as well as almost all regular graphs [7]. In the light of the upper bound of 3 on the dimension of the algorithm needed to identify all planar graphs [34], there is hope that the class of planar graphs can eventually be added to the list. Towards a complete characterization of the expressive power of 2-WL, Fuhlbrück, Köbler, and Verbitsky [14] developed an algorithmic characterization of the graphs of color class size at most 4 that are identified by 2-WL.

Our Contribution. In this work, we investigate 2-WL on planar graphs. We are interested in analyzing the stable output coloring computed by 2-WL and deducing symmetries and other properties of the input graph from properties of the coloring.

As a starting point, we precisely characterize the planar graphs in which all edges receive the same color with respect to 2-WL. Since the coloring that 2-WL computes is preserved by automorphisms, edge-transitive planar graphs clearly fall into this category. As our first main result, we prove the converse of this statement: every planar graph in which all edges receive the same color with respect to 2-WL is edge-transitive. To show the implication, we reprove the classification of edge-transitive planar graphs (see, e.g., [26]) building solely on the 2-WL coloring.

Using the classification, we continue to analyze the WL coloring on general planar graphs. Since, by [32], the algorithm 2-WL implicitly computes the graph decomposition into 3-connected components, understanding 2-WL on planar graphs essentially amounts to a study of 3-connected planar graphs. Here, we can exploit a theorem due to Whitney [42], which says that all embeddings of a 3-connected planar graph are combinatorially equivalent.

Our focus lies on the following three tasks: (i) classify the subgraphs induced by edges of the same 2-WL color that can occur, (ii) analyze how these subgraphs interleave, and (iii) establish connections to properties of the entire graph G.

Let G be a 3-connected planar graph and let $C_E(G)$ denote the set of 2-WL colors that correspond to edges of G. For every $c \in C_E(G)$, denote by $G[c]$ the subgraph induced by all edges of 2-WL color c. To describe our results, it turns out to be useful to partition edge colors into three types depending on the number of faces per connected component of $G[c]$. We say that c has *Type I* if every connected component of $G[c]$ has one face, *Type II* if every connected component of $G[c]$ has two faces, and *Type III* if every connected component of $G[c]$ has at least three faces. (By the properties of 2-WL, these types indeed cover all cases that can occur.)

First, we analyze the graphs induced by edge colors c of Type III. It is not hard to see that every edge in such a graph $G[c]$ receives the same 2-WL color (when applying 2-WL to $G[c]$), and thus, by our classification, $G[c]$ is edge-transitive. However, it turns out that much stronger statements are possible, since, in the end, many edge-transitive planar graphs cannot appear as a graph $G[c]$. For example, we show that $G[c]$ is always connected. As our central result for colors of Type III, we obtain a precise classification of the possible graphs $G[c]$. An interesting consequence of this classification is that the automorphism group $\mathrm{Aut}(G)$ of G is always isomorphic to a subgroup of $\mathrm{Aut}(G[c])$. More precisely, we show that fixing the images of all vertices of $G[c]$ uniquely determines the image of every vertex of G under any automorphism of G. Hence, by only looking at the subgraph induced by a single edge color of Type III, we obtain strong insights about the symmetries of the entire graph.

On the other side of the spectrum, we prove that if all edge colors are of Type I, then G has *fixing number* at most 1, where the fixing number is the minimum number of vertices that need to be fixed pointwise so that the identity mapping is the only automorphism of G. It is known that 3-connected planar graphs have fixing number at most 3, and there is a complete characterization of those graphs of fixing number exactly 3 [34]. In our analysis of 3-connected planar graphs G in which all edge colors are of Type I, we only use 1-WL to prove that G has no non-trivial automorphisms after fixing a certain single vertex. This implies that 2-WL identifies all such graphs.

If neither of the above cases applies, then there is an edge color of Type II. Let us first remark that the graphs of many Archimedean solids fall into this category (while the edge colors in the graph of all Platonic solids are of Type III). In such a situation, the graph of the Archimedean solid is defined by edge colors c, d where one of the two colors has Type II. With this in mind, towards solving task (ii), we analyze how edge colors of Type II interleave with other edge colors. More precisely, similarly as for Type III, we aim at identifying a connected subgraph defined by two colors $c, d \in C_E(G)$, where c has Type II, that corresponds to one of the Archimedean solids, or stems from a small number of infinite graph families. We remark that, similar to the case of edge colors of Type III, if we have such a subgraph $G[c, d]$, then $\mathrm{Aut}(G)$ is isomorphic to a subgroup of $\mathrm{Aut}(G[c, d])$.

We show that either this goal can be achieved, or G has fixing number 1 or there is a *WL-definable matching*. Such a matching is given by an edge color c such that $G[c]$ is a matching graph (i.e., every vertex has degree 1) and the endpoints of every edge receive different colors. Such matchings also play a crucial role in the analysis of 2-WL on graphs of color class size 4 [14], and contracting all matching edges preserves many crucial properties related to WL such as the stable coloring, identifiability by WL, as well as the automorphism group of G. As a result, finding a WL-definable matching is beneficial since we can proceed to a smaller graph without affecting the problem at hand.

Towards the WL Dimension of Planar Graphs. The WL dimension of a graph class $\mathcal{C}$ is the minimal k such that k-WL identifies every graph from $\mathcal{C}$, i.e., k-WL serves as a complete isomorphism test for the class $\mathcal{C}$. Many classes of graphs are known to have a finite WL dimension, for example, interval graphs [13], graphs of bounded rank-width [24] as well as graphs of bounded genus [22] and, more generally, all graph classes that exclude a fixed graph as a minor [19, 20].

For planar graphs, the quest for bounds on their WL dimension was initiated by Immerman already over three decades ago [28]. In a first step, Grohe [18] proved that the dimension is finite. Analyzing Grohe's proof in detail, Redies [38] showed an upper bound of 14 on the WL dimension of planar graphs. This was further improved in [34], where it is shown

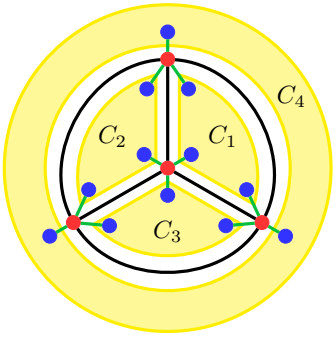


Figure 1 The visualization shows a 3-connected planar graph G and an edge color $c \in C_E(G)$ (shown in black) such that $G[c]$ is isomorphic to K_4. Also, C_1, C_2, C_3, C_4 are the vertex sets of the connected components of $G - V(G[c])$. Using regularity constraints specific to the case $G[c] \cong K_4$, one can show that the graphs induced by the C_i are all indistinguishable by 2-WL. More strongly, it actually holds for all $i, j \in [4]$, $v \in N(C_i)$ and $w \in N(C_j)$ that $\{\{\chi(v, v') \mid v' \in N(v) \cap C_i\}\} = \{\{\chi G(w, w') \mid w' \in N(w) \cap C_j\}\}$, where χ denotes the 2-WL coloring. (In the picture, this multiset contains one green edge, i.e., every $v \in V(G[c])$ has exactly one neighbor via a green edge in each adjacent C_i.)

that already 3-WL identifies all planar graphs, thus narrowing down the WL dimension of planar graphs to 2 or 3. Moreover, it was recently shown that a constant dimension of the WL algorithm suffices to identify all planar graphs in a logarithmic number of refinement rounds [23], extending previous results for 3-connected planar graphs [40]. Still, the task to determine the precise WL-dimension of the class of planar graphs remains open. A central motivation for our work is to find out whether 2-WL identifies every planar graph.

Our results suggest an inductive approach to this question. Indeed, building on the fact that 2-WL is able to detect the decomposition into 3-connected components [32], we can restrict our attention to 3-connected graphs. Given a 3-connected planar graph G, by combining the results described above, we always obtain that G has one of the following:

- **(A)** fixing number 1 under 1-WL, i.e., individualizing a single vertex and performing 1-WL (after coloring all pairs with their 2-WL color) results in a discrete coloring,
- **(B)** a WL-definable matching, or
- **(C)** a connected subgraph induced by at most two edge colors that corresponds to a Platonic or Archimedean solid or stems from a small number of infinite graph families.

In Case A, the graph G is identified by 2-WL. In Case B, we can follow the strategy outlined in [14] and move to a smaller graph by contracting the definable matching. Therefore, determining the WL dimension of the class of planar graphs boils down to defeating Case C. In this case, we obtain a connected subgraph H that is defined by at most two edge colors c and d and which we can classify precisely. Let $C_1, \dots, C_s$ denote the vertex sets of the connected components of $G - V(H)$, the graph G with the vertices in H removed (see also Figure 1). Also, let G' be a second graph that cannot be distinguished from G by 2-WL. Let H' denote the subgraph of G' induced by c and d and let $C'_1, \dots, C'_s$ denote the vertex sets of the connected components of $G' - V(H')$. Presupposing by induction that the statement holds for smaller graphs, we may assume that 2-WL identifies the subgraphs induced by $C_1, \dots, C_s$. This implies that $G[C_i]$ is isomorphic to $G'[C'_i]$ for all $i \in [s]$ (possibly after reordering the sets $C'_1, \dots, C'_s$). It is not hard to see that 2-WL identifies H and, thus, H is isomorphic to H'. Now, ideally, we want to glue all these partial isomorphisms together to obtain a global isomorphism from G to G'. Towards this end, it is our intuition that the

options for the interplay between H and the sets C_i are extremely limited due to G being planar and H being defined by few edge colors, which enforces strong regularity conditions on the interaction between H and the surrounding graph. A formalization of such a strategy for all subcases that can appear in Case C should yield that 2-WL identifies every planar graph.

2 Preliminaries

Graphs. An (undirected) *graph* is a pair $G = (V(G), E(G))$ of a finite *vertex set* $V(G)$ and an *edge set* $E(G) \subseteq \big\{\{u, v\} \,\big|\, u \neq v \in V(G)\big\}$. Unless stated explicitly otherwise, graphs are undirected. For a directed graph G', we write $\mathsf{undir}(G')$ to denote its undirected version. For $v, w \in V(G)$, we also write vw as a shorthand for $\{v, w\}$. The *neighborhood* of v in G is denoted by $N_G(v)$ and the *degree* of v in G is $\deg_G(v) \coloneqq |N_G(v)|$. If the graph G is clear from the context, we usually omit the index and simply write $N(v)$ and $\deg(v)$. For $W \subseteq V(G)$, we also define $N(W) \coloneqq \big(\bigcup_{v \in W} N(v)\big) \setminus W$. We denote by $G[W]$ the *induced subgraph* of G on the vertex set W, and define $G - W \coloneqq G[V(G) \setminus W]$. A set $S \subseteq V(G)$ is a *separator* of G if $G - S$ has more connected components than G. A *k-separator* of G is a separator of G of size k. The graph G is *k-connected* if it is connected and has no separator of size at most $k - 1$.

In our definitions of vertex sets of graphs, we use the notation $\uplus$ to denote a formal disjoint union. More precisely, for sets V and W, the set $V \uplus W$ contains $|V| + |W|$ vertices, one distinct copy of each vertex in V and one distinct copy of each vertex in W. (For ease of notation, we refer to the vertices by their original names in V and W instead of renaming them first.)

A *vertex-colored graph* is a tuple (G, λ) where G is a graph and $\lambda\colon V(G) \to C$ is a *vertex coloring*, a mapping from $V(G)$ into some set C of colors. We define the set of *arcs* of a graph G as $A(G) \coloneqq \{(v, v) \mid v \in V(G)\} \cup \{(v, w) \mid \{v, w\} \in E(G)\}$. Observe that for each $vw \in E(G)$, there are the two arcs (v, w), (w, v). An *arc-colored graph* is a tuple (G, λ), where G is a graph and $\lambda\colon A(G) \to C$ is a mapping from $A(G)$ into some set C of colors. Similarly, a *pair-colored graph* is a tuple (G, λ), where G is a graph and $\lambda\colon (V(G))^2 \to C$ is a mapping into some set of colors C.

Typically, the set C is chosen to be an initial segment $[n]$ of the natural numbers. We say a coloring λ is *discrete* if it is injective, i.e., all color classes have size 1. Finally, for a coloring λ and distinct vertices $v_1, \dots, v_\ell$, we denote by $(G, \lambda, v_1, \dots, v_\ell)$ the colored graph where each v_i for $i \in [\ell]$ is individualized. To be more precise, if λ is a vertex coloring, then $(G, \lambda, v_1, \dots, v_\ell) \coloneqq (G, \widetilde{\lambda})$ where $\widetilde{\lambda}(v_i) = (1, i)$ for all $i \in [\ell]$, and $\widetilde{\lambda}(v) = (0, \lambda(v))$ for all $v \in V(G) \setminus \{v_1, \dots, v_\ell\}$. The definitions for arc and pair colorings are analogous. We generally assume that all graphs are arc-colored even if not explicitly stated. Every (uncolored) graph can be interpreted as an arc-colored graph by assigning to every diagonal arc (v, v) the color 1 and assigning to every non-diagonal arc the color 2.

A graph is called *planar* if it can be embedded into the plane $\mathbb{R}^2$. A *plane graph* is a graph embedded into the plane. As the following statement shows, all plane realizations of a planar graph have the same number of faces, i.e., regions bounded by edges.

▶ **Theorem 1** (Euler's formula). *Let G be a connected plane graph with n vertices, m edges, and f faces. Then $n - m + f = 2$.*

We will also fall back on the following famous theorem due to Whitney.

▶ **Theorem 2** (Whitney's theorem [42]). *Up to homeomorphism, a 3-connected planar graph has a unique embedding into the plane.*

The theorem allows us to speak about faces of 3-connected planar graphs as abstract objects, since it implies that in a 3-connected planar graphs, the set of faces does not depend on a specific embedding and thus, the faces can be viewed as combinatorial objects associated with G and are uniquely defined by their sets of vertices $V(F)$ and the edges $E(F)$ bounding F. We will therefore not draw a clear distinction between this combinatorial view and the topological view of F as a region and just use whichever is most suitable for our purpose.

The Weisfeiler–Leman Algorithm. Let $\chi_1, \chi_2 \colon (V(G))^k \to C$ be colorings of the k-tuples of vertices of a graph G. We say χ_1 *refines* χ_2, denoted $\chi_1 \preceq \chi_2$, if $\chi_1(\bar{v}) = \chi_1(\bar{w})$ implies $\chi_2(\bar{v}) = \chi_2(\bar{w})$ for all $\bar{v}, \bar{w} \in (V(G))^k$. The colorings χ_1 and χ_2 are *equivalent*, denoted $\chi_1 \equiv \chi_2$, if $\chi_1 \preceq \chi_2$ and $\chi_2 \preceq \chi_1$.

Given a graph G, the algorithm 1-WL iteratively computes an isomorphism-invariant coloring of the vertices of G. In this work, we actually require an extension of 1-WL, which also takes arc colors into account. For an arc-colored graph (G, λ), we define the initial coloring computed by the algorithm via $\chi^1_0[G](v) \coloneqq \lambda(v, v)$ for all $v \in V(G)$. This coloring is refined via $\chi^1_{i+1}[G](v) \coloneqq (\chi^1_i[G](v), \mathcal{M}_i(v))$, where $\mathcal{M}_i(v)$ is a multiset defined as

$$\mathcal{M}_i(v) \coloneqq \left\{\!\!\left\{ \left(\chi^1_i[G](w), \lambda(v, w), \lambda(w, v)\right) \;\middle|\; w \in N_G(v) \right\}\!\!\right\}.$$

By definition, $\chi^1_{i+1}[G] \preceq \chi^1_i[G]$ holds for all $i \geq 0$. Hence, there is a minimal value i_∞ such that $\chi^1_{i_\infty}[G] \equiv \chi^1_{i_\infty+1}[G]$. We call $\chi^1_{i_\infty}[G]$ the *stable* coloring of G and denote it by $\chi^1_{\mathsf{WL}}[G]$. The algorithm 1-WL takes an arc-colored graph (G, λ) as input and returns $\chi^1_{\mathsf{WL}}[G]$.

We can also apply 1-WL to a pair-colored graph (G, λ). This can be done by defining $\widetilde{\lambda}(v_1, v_2) \coloneqq (1, \lambda(v_1, v_2))$ for all $v_1, v_2 \in V(G)$ with $v_1v_2 \in E(G)$, and $\widetilde{\lambda}(v_1, v_2) \coloneqq (0, \lambda(v_1, v_2))$ for all $v_1, v_2 \in V(G)$ with $v_1v_2 \notin E(G)$. Then we define $\chi^1_{\mathsf{WL}}[G, \lambda] \coloneqq \chi^1_{\mathsf{WL}}[H, \widetilde{\lambda}]$ where H is a complete graph on vertex set $V(G)$.

Next, we describe the *k-dimensional Weisfeiler–Leman algorithm* (k-WL) for $k \geq 2$. For an input graph G, let $\chi^k_0[G] \colon (V(G))^k \to C$ be the coloring where each tuple is colored with the isomorphism type of its underlying ordered subgraph. $v_i = v_j \Leftrightarrow v'_i = v'_j$ and $v_iv_j \in E(G) \Leftrightarrow v'_iv'_j \in E(G)$. If the graph comes equipped with a coloring, the initial coloring $\chi^k_0[G]$ also takes the input coloring into account. More formally, for an arc coloring λ, for $\chi^k_0[G](v_1, \ldots, v_k) = \chi^k_0[G](v'_1, \ldots, v'_k)$ to hold, we have the additional conditions $\lambda(v_i, v_j) = \lambda(v'_i, v'_j)$ for all $i, j \in [k]$ with $(v_i, v_j) \in A(G)$. For a pair coloring λ, we have the additional conditions $\lambda(v_i, v_j) = \lambda(v'_i, v'_j)$ for all $i, j \in [k]$.

We then recursively define the coloring $\chi^k_i[G]$ obtained after i rounds of the algorithm. For $\bar{v} = (v_1, \ldots, v_k) \in (V(G))^k$, let $\chi^k_{i+1}[G](\bar{v}) \coloneqq \big(\chi^k_i[G](\bar{v}), \mathcal{M}_i(\bar{v})\big)$, where

$$\mathcal{M}_i(\bar{v}) \coloneqq \left\{\!\!\left\{ \left(\chi^k_i[G](\bar{v}[w/1]), \ldots, \chi^k_i[G](\bar{v}[w/k])\right) \;\middle|\; w \in V(G) \right\}\!\!\right\}$$

and $\bar{v}[w/i] \coloneqq (v_1, \ldots, v_{i-1}, w, v_{i+1}, \ldots, v_k)$ is the tuple obtained from substituting the i-th entry of $\bar{v}$ with w. Again, there is a minimal i_∞ such that $\chi^k_{i_\infty}[G] \equiv \chi^k_{i_\infty+1}[G]$, and we set $\chi^k_{\mathsf{WL}}[G] \coloneqq \chi^k_{i_\infty}[G]$.

The algorithm k-WL takes a (pair- or arc-)colored graph G as input and returns $\chi^k_{\mathsf{WL}}[G]$. Given graphs G and H, the algorithm *distinguishes* G and H if $\{\!\{\chi^k_{\mathsf{WL}}[G](\bar{v}) \mid \bar{v} \in (V(G))^k\}\!\} \neq \{\!\{\chi^k_{\mathsf{WL}}[H](\bar{w}) \mid \bar{w} \in (V(H))^k\}\!\}$. Also, k-WL *identifies* G if it distinguishes G from every other non-isomorphic graph.

▶ **Definition 3.** *Let G be a graph and let $k \geq 2$. Then k-*WL determines arc orbits on *G if for every $(v_1, v_2) \in A(G)$, every graph H, and every $(w_1, w_2) \in A(H)$ such that $\chi^k_{\mathsf{WL}}[G](v_1, v_2, \ldots, v_2) = \chi^k_{\mathsf{WL}}[H](w_1, w_2, \ldots, w_2)$, there is an isomorphism $\varphi\colon G \cong H$ such that $\varphi(v_i) = w_i$ holds for both $i \in \{1, 2\}$.*

*Moreover, k-*WL determines pair orbits of *G if for all $v_1, v_2 \in V(G)$, every graph H, and all $w_1, w_2 \in V(H)$ such that $\chi^k_{\mathsf{WL}}[G](v_1, v_2, \ldots, v_2) = \chi^k_{\mathsf{WL}}[H](w_1, w_2, \ldots, w_2)$, there is an isomorphism $\varphi\colon G \cong H$ such that $\varphi(v_i) = w_i$ holds for both $i \in \{1, 2\}$.*

Observe that if k-WL determines arc or pair orbits of G, then it identifies G. Indeed, if for a second graph H, there is no isomorphism from G to H, the multisets of χ^k_{WL}-colors in the two graphs must be disjoint by Definition 3.

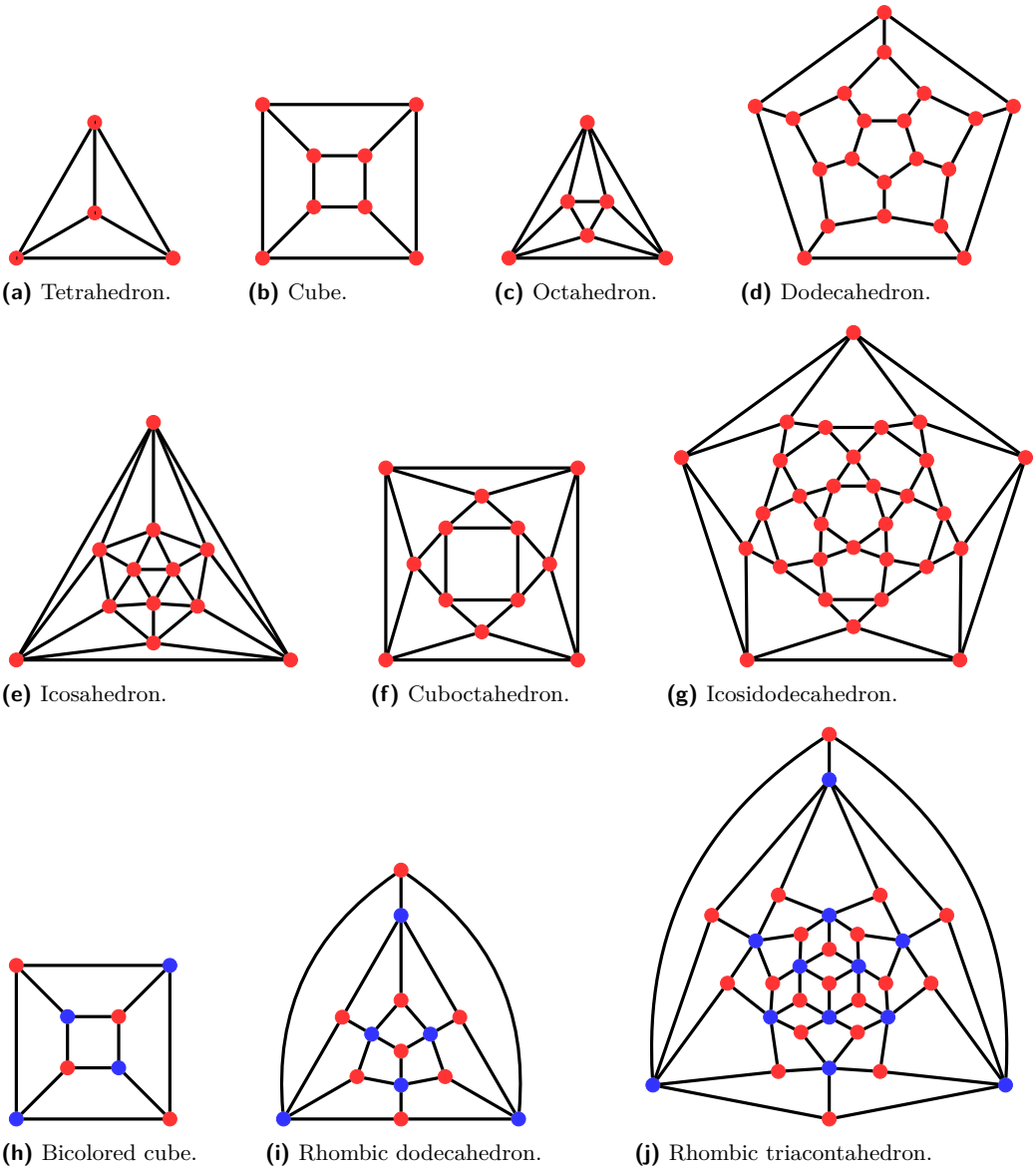
3 Edge-Transitive Planar Graphs

In this section, we classify planar graphs where all edges receive the same color with respect to 2-WL. We call an undirected graph G *edge-transitive* if for all $uv, u'v' \in E(G)$, there is an automorphism $\varphi\colon V(G) \to V(G)$ with $\varphi(u) = u'$ and $\varphi(v) = v'$. It is well-known that there are only nine edge-transitive connected planar graphs of minimum degree 3 [26]. Based on this result, one can easily classify all edge-transitive planar graphs. Clearly, all of these graphs have the property that all edges receive the same color with respect to 2-WL. In this section, we show the converse of this statement, i.e., every planar graph in which all edges receive the same color with respect to 2-WL is edge-transitive. Towards this goal, we reprove the classification from [26] relying only on 2-WL colors. More precisely, the main result in this section is the following theorem (see also Figure 2). Since 2-WL colors directed pairs and it may happen that a pair (u, v) receives a different color than (v, u), it is more convenient to consider directed graphs and demand that all directed edges receive the same color (rather than saying the pair of colors for both orientations is the same for all undirected edges).

▶ **Theorem 4.** *Let G be a connected planar (directed or undirected) graph of minimum degree at least 3 such that $\chi^2_{\mathsf{WL}}[G](v_1, w_1) = \chi^2_{\mathsf{WL}}[G](v_2, w_2)$ for all $(v_1, w_1), (v_2, w_2) \in E(G)$. Then one of the following holds:*

- **(A)** *G is isomorphic to a tetrahedron (Figure 2a), a cube (Figure 2b), a dodecahedron (Figure 2d), or an icosahedron (Figure 2e),*
- **(B)** *the undirected version $\mathsf{undir}(G)$ is isomorphic to an octahedron (Figure 2c), a cuboctahedron (Figure 2f), or an icosidodecahedron (Figure 2g), or*
- **(C)** *the undirected version $\mathsf{undir}(G)$ is isomorphic to a cube (Figure 2h), a rhombic dodecahedron (Figure 2i), or a rhombic triacontahedron (Figure 2j).*

Note that the classification includes the graphs of all Platonic solids. To prove the theorem, we distinguish two cases. Let $\chi := \chi^2_{\mathsf{WL}}[G]$ and let $C_V(G, \chi) := \{\chi(v, v) \mid v \in V(G)\}$ denote the set of *vertex colors*. Since $\chi(u, u) = \chi(u', u')$ and $\chi(v, v) = \chi(v', v')$ whenever $\chi(u, v) = \chi(u', v')$, we conclude that $1 \leq |C_V(G, \chi)| \leq 2$. First suppose $|C_V(G, \chi)| = 1$. Then $\mathsf{undir}(G)$ is d-regular for some $d \geq 3$. Since G is planar, $d \leq 5$ and thus, $d \in \{3, 4, 5\}$. A deep analysis of these three cases leads to the graphs listed in Parts A and B. Let us remark at this point that obtaining such a classification is much more challenging than for edge-transitive graphs. Indeed, the proofs for edge-transitive graphs highly exploit that the multiset of sizes of faces incident to an edge (and a vertex, respectively) is always the same. However, we cannot immediately deduce information about the size of faces from considering WL-colors and hence, we cannot directly rely on this type of argument. Instead, our arguments exploit the fact that 2-WL can detect 2-separators [32] as well as the existence of certain short cycles [15].



(a) Tetrahedron. **(b)** Cube. **(c)** Octahedron. **(d)** Dodecahedron.

(e) Icosahedron. **(f)** Cuboctahedron. **(g)** Icosidodecahedron.

(h) Bicolored cube. **(i)** Rhombic dodecahedron. **(j)** Rhombic triacontahedron.

Figure 2 All edge-transitive connected planar graphs of minimum degree 3.

Also, note that the graphs listed in Part A are always undirected since d is odd. On the other hand, every graph listed in Part B also has at least one directed version that is also edge-transitive (we refer the reader to the full version [33] for details).

Finally, for the case $|C_V(G, \chi)| = 2$, it is possible to perform a reduction to the first case by defining an auxiliary graph on one of the two vertex-color classes. This results in the graphs listed in Part C. Here, it is notable that the cube appears for a second time because it is bipartite and directing all edges from one bipartition class to the other one also leads to an edge-transitive graph.

In Theorem 4, we restrict ourselves to graphs that are connected and have minimum degree at least 3. Both of these restrictions can easily be lifted as follows. Let us first consider the restriction on the degree and let G be a connected planar graph such that

$\chi^2_{\mathsf{WL}}[G](v_1, w_1) = \chi^2_{\mathsf{WL}}[G](v_2, w_2)$ holds for all $(v_1, w_1), (v_2, w_2) \in E(G)$. If G has maximum degree 2 or contains a vertex of degree at most 1, then it is easy to see that G is either a cycle or isomorphic to a star $K_{1,h}$ for some $h \geq 0$ ($h = 0$ covers the special case that G consists of a single vertex).

▶ **Definition 5.** *Let H be a graph and $s \geq 1$. The* s-subdivision of H *is the graph $H^{(s)}$ obtained from H by replacing each edge with s parallel paths of length 2. Formally, $H^{(s)}$ is the graph with vertex set $V(H^{(s)}) \coloneqq V(H) \uplus (E(H) \times [s])$ and edge set*

$$E(H^{(s)}) \coloneqq \left\{ v(e,i) \;\middle|\; e \in E(H), v \in e, i \in [s] \right\}.$$

In the remaining case, G has maximum degree at least 3 and minimum degree 2. Then it is easy to see that G is one of the graphs from Theorem 4, or an s-subdivision of one of the graphs from Parts A and B for some $s \geq 1$, a cycle C_ℓ for some $\ell \geq 3$, or the complete graph on two vertices K_2.

Finally, if G is not connected, then it is isomorphic to the disjoint union of ℓ copies of one of its connected components for some $\ell \geq 2$, because all graphs listed above can be distinguished from each other by 2-WL. Actually, it can be checked that all of the graphs are even identified by 2-WL. Overall, this gives the following corollary.

▶ **Corollary 6.** *Let G be a directed planar graph such that $\{\{\chi^2_{\mathsf{WL}}[G](v,w), \chi^2_{\mathsf{WL}}[G](w,v)\}\} = \{\{\chi^2_{\mathsf{WL}}[G](v',w'), \chi^2_{\mathsf{WL}}[G](w',v')\}\}$ holds for all $(v,w), (v',w') \in E(G)$. Then 2-WL determines arc orbits on G. In particular, G is edge-transitive.*

4 Graphs Induced by a Single Edge Color

After considering planar graphs with a single edge color with respect to 2-WL, we now wish to analyze the 2-WL coloring of arbitrary planar graphs. Since, by [32], the algorithm 2-WL implicitly computes the decomposition of a graph into 3-connected components[1], understanding 2-WL on planar graphs essentially amounts to a study of 3-connected planar graphs. Hence, we restrict our attention to those.

4.1 Edge Types

Let G be a 3-connected planar graph and set $\chi \coloneqq \chi^2_{\mathsf{WL}}[G]$. To analyze the coloring χ, we focus on subgraphs induced by a single edge color. Towards this end, let $C_V \coloneqq C_V(G,\chi) = \{\chi(v,v) \mid v \in V(G)\}$ denote the set of *vertex colors*. Similarly, let $C_E \coloneqq C_E(G,\chi) = \{\chi(v,w) \mid vw \in E(G)\}$ be the set of *edge colors*. For $C \subseteq C_E$, we define the graph $G[C]$ with

$$V(G[C]) \coloneqq \{v_1, v_2 \mid \chi(v_1,v_2) \in C\} \quad \text{and} \quad E(G[C]) \coloneqq \{v_1v_2 \mid \chi(v_1,v_2) \in C\}.$$

In case $C = \{c_1, \dots, c_\ell\}$, we also write $G[c_1, \dots, c_\ell]$ instead of $G[\{c_1, \dots, c_\ell\}]$. Observe that $G[C]$ is defined as an undirected graph. However, it may be that $\chi(v_1,v_2) \neq \chi(v_2,v_1)$ holds for some $v_1v_2 \in E(G)$. Since this information turns out to be relevant in some cases, we always assume that $G[C]$ is equipped with an arc coloring where colors are inherited from χ.

As indicated, we are particularly interested in the case $C = \{c\}$ for a single color c. Observe that the ends of c-colored edges have the same vertex color, i.e., if $\chi(v_1,w_1) = \chi(v_2,w_2) = c$, then $\chi(v_1,v_1) = \chi(v_2,v_2)$ and $\chi(w_1,w_1) = \chi(w_2,w_2)$. This implies that $1 \leq |C_V(G[c],\chi)| \leq 2$. We say that $G[c]$ is *unicolored* if $|C_V(G[c],\chi)| = 1$. Otherwise, we say that $G[c]$ is *bicolored*.

[1] For the formal and quite technical definition of this notion, we refer to [32].

To analyze 2-WL on 3-connected planar graphs, we consider the graphs $G[c]$ for suitable edge colors $c \in C_E$. Towards this end, it turns out to be useful to group the graphs $G[c]$ according to the number of faces of each connected component of $G[c]$. Note that 2-WL detects connected components of graphs. More precisely, it holds that

- all connected components in $G[c]$ have the same size because 2-WL detects, for every $w \in V(G[c])$, the set of vertices reachable in the arc-colored graph (G, χ) from w via edges uv of color c (i.e., $\chi(u,v) = c$ or $\chi(v,u) = c$), and
- all connected components in $G[c]$ have the same multiset of vertex degrees in $G[c]$ since otherwise the vertex colors and, thus, also the arc colors would be different.

In combination, all connected components of $G[c]$ have the same number of vertices and edges and hence, by Euler's formula, they also have the same number of faces. We distinguish between three types in C_E.

Type I. For the first category, we consider those graphs $G[c]$ that have only one face. To be more precise, we say that $c \in C_E$ has *Type I* if $(G[c])[A]$ has a single face for every vertex set A of a connected component of $G[c]$. It is not difficult to see that $G[c]$ is isomorphic to a disjoint union of stars $K_{1,h}$ for $h \in [n]$.

Type II. For the second category, we consider those graphs $G[c]$ where every connected component has exactly two faces. Formally, we say that $c \in C_E$ has *Type II* if $(G[c])[A]$ has exactly two faces for every vertex set A of a connected component of $G[c]$. In this case, $G[c]$ is a disjoint union of cycles of the same length. Also, it is not difficult to see that every connected component of $G[c]$ is either a directed cycle (i.e., $\chi(v_1, v_2) \neq \chi(v_2, v_1)$ holds for every edge $v_1v_2 \in E(G[c])$), or an undirected cycle in which all vertices have the same color with respect to 2-WL, or an undirected cycle with two vertex colors that alternate along the cycle.

Type III. Finally, for the last category, we consider those graphs $G[c]$ where each connected component has at least three faces. Again, to be precise, we say that $c \in C_E$ has *Type III* if $(G[c])[A]$ has at least three faces for every vertex set A of a connected component of $G[c]$.

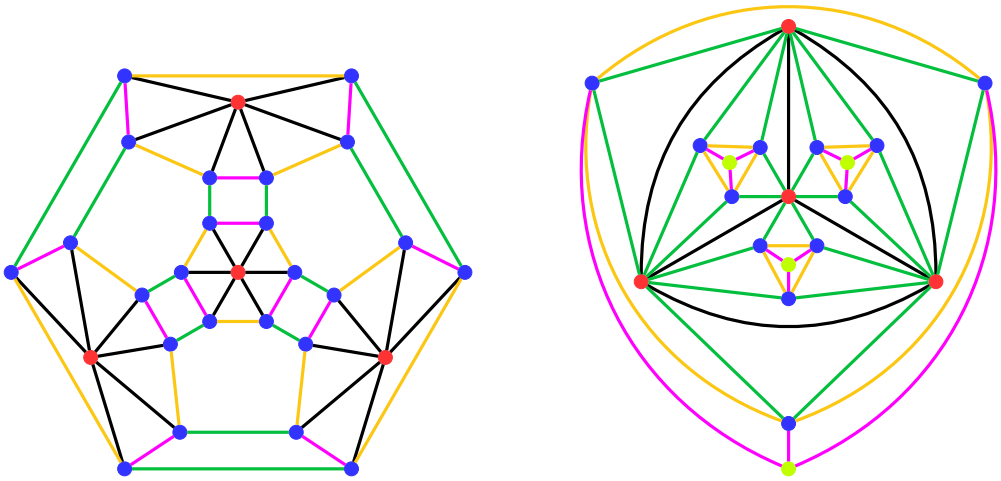
Also, we define the *type* of an edge $v_1v_2 \in E(G)$ as the type of its color $\chi(v_1, v_2)$ (note that the type of $\chi(v_1, v_2)$ is equal to the type of $\chi(v_2, v_1)$).

In the following, we derive several properties of the graphs $G[c]$ depending on the type of c, as well as properties of G depending on which types of edge colors occur. Towards this end, we also define the *type of* G as the maximal type of any edge color $c \in C_E$. So we say that G has Type III if there is some $c \in C_E$ of Type III. The graph G has Type II if there is some $c \in C_E$ of Type II, but there is no $c' \in C_E$ of Type III. Lastly, G has Type I if every $c \in C_E$ has Type I. Two example graphs are displayed in Figure 3.

4.2 Graphs of Fixing Number One

We start by investigating 3-connected planar graphs of Type I (see Figure 3a for an example). It turns out that such graphs have fixing number 1 with respect to 1-WL (after coloring all edges with their 2-WL colors), which in particular implies that 2-WL identifies all graphs of Type I. The proof is based on the following result.

▶ **Theorem 7** ([34, Lemma 23]). *Let G be a 3-connected planar graph and suppose $v_1, v_2, v_3 \in V(G)$ are pairwise distinct vertices lying on a common face of G. Then $\chi^1_{\mathsf{WL}}[G, v_1, v_2, v_3]$ is discrete.*



(a) A graph G of Type I. Each edge color $c \in C_E$ defines a graph $G[c]$ that is isomorphic to a disjoint union of stars. Individualizing an arbitrary blue vertex and performing 1-WL results in a discrete coloring. Hence, the graph is identified by 2-WL.

(b) A graph G of Type III. The edge colors black and green have Type III, yellow has Type II, and pink has Type I. Note that $G[c]$ is connected for every edge color c of Type III whereas the other edge colors induce non-connected subgraphs.

Figure 3 Two 3-connected planar graphs where all vertices and edges are colored by their 2-WL color. For visualization purposes, we only color edges and do not distinguish between potentially different colors of two arcs (v, w) and (w, v).

Here, $\chi^1_{\mathsf{WL}}[G, v_1, v_2, v_3]$ denotes the coloring computed by 1-WL after individualizing v_1, v_2, and v_3. For a vertex coloring $\lambda\colon V \to C$ and $v \in V$, we define $[v]_\lambda := \{w \in V \mid \lambda(v) = \lambda(w)\}$ as the color class of v and $\mathrm{Singles}(\lambda) := \{v \in V \mid |[v]_\lambda| = 1\}$. For a graph G and vertices $v_1, \dots, v_\ell \in V(G)$, we define

$$\mathrm{Singles}_G(v_1, \dots, v_\ell) := \mathrm{Singles}(\chi^1_{\mathsf{WL}}[G, \chi^2_{\mathsf{WL}}[G], v_1, \dots, v_\ell]).$$

In other words, $\mathrm{Singles}_G(v_1, \dots, v_\ell)$ is the set of all vertices appearing in a singleton color class after performing 1-WL on G where every pair is colored with its 2-WL-color, and where $v_1, \dots, v_\ell$ are individualized.

▶ **Lemma 8.** *Let G be a graph and let $v_1, \dots, v_\ell \in V(G)$ such that $\mathrm{Singles}_G(v_1, \dots, v_\ell) = V(G)$. Also define $k := \max\{2, \ell + 1\}$. Then k-WL determines pair orbits in G.*

The following lemma provides a sufficient condition for a 3-connected planar graph to have fixing number 1.

▶ **Lemma 9.** *Let G be a 3-connected planar graph and suppose there is a face F such that every edge $e \in E(F)$ has Type I. Then there is a vertex $v \in V(G)$ such that $\mathrm{Singles}_G(v) = V(G)$.*

Proof. Let H be a directed graph with vertex set $V(H) := V(G)$ and edge set

$$E(H) := \{(v, w) \mid vw \in E(G) \wedge \deg_{G[\chi^2_{\mathsf{WL}}[G](v,w)]}(v) = 1\}.$$

Intuitively speaking, we add a directed edge (v, w) to the graph H if w is the only neighbor of v reachable via an edge of color $\chi^2_{\mathsf{WL}}[G](v, w)$. In particular, if v is individualized, then w is also fixed after performing 1-WL.

For every edge $vw \in E(G)$ of Type I, it holds that $(v,w) \in E(H)$ or $(w,v) \in E(H)$. Hence, there are three vertices $v_1, v_2, v_3 \in V(G)$ lying on the face F of G such that $(v_1,v_2),(v_2,v_3) \in E(H)$ or $(v_1,v_2),(v_1,v_3) \in E(H)$.

Now consider the coloring $\lambda := \chi^1_{\mathsf{WL}}[G, \chi^2_{\mathsf{WL}}[G], v_1]$. Let $c := \chi^2_{\mathsf{WL}}[G](v_1,v_2)$. By definition of the edge set of H, it holds that v_2 is the only neighbor of v_1 which is adjacent via an edge of color c. Hence, $|[v_2]_\lambda| = 1$. By the same argument, $|[v_3]_\lambda| = 1$. Since v_1, v_2, v_3 all lie on the face F, it follows from Theorem 7 that λ is discrete. In other words, $\mathrm{Singles}_G(v_1) = V(G)$. ◀

▶ **Corollary 10.** *Let G be a 3-connected planar graph of Type I. Then there is a vertex $v \in V(G)$ such that $\mathrm{Singles}_G(v) = V(G)$. In particular, 2-WL determines pair orbits of G.*

We also record the following useful lemma, which is another consequence of Theorem 7.

▶ **Lemma 11.** *Let G be a 3-connected planar graph. Suppose $v_1, \dots, v_\ell \in V(G)$ form a cycle in G, i.e., $v_1v_\ell \in E(G)$ and $v_iv_{i+1} \in E(G)$ holds for all $i \in [\ell{-}1]$. Let $w \in V(G) \setminus \{v_1, \dots, v_\ell\}$. Then $\mathrm{Singles}_G(v_1, \dots, v_\ell, w) = V(G)$.*

The lemma says that in a 3-connected planar graph G, it suffices to fix a cycle and one additional vertex in order to fix the entire graph. For example, this allows us to extract from the presence of certain 2-WL-detectable subgraphs bounds on the fixing number of the entire 3-connected planar graph G. Note that in the case that the fixing number in the subgraph is 1, Lemma 8 yields that 2-WL determines pair orbits in G.

4.3 Three Faces

We now turn to edge colors of Types II and III. For both types, it is not difficult to see that it is impossible to bound the fixing number by 1 in general. Instead, our focus here lies on investigating how edge colors of the corresponding type can appear within a 3-connected planar graph.

We first focus on edge colors of Type III (see Figure 3b for an example). Let G be a 3-connected planar graph and let $c \in C_E(G,\chi)$ be an edge color of Type III, where $\chi := \chi^2_{\mathsf{WL}}[G]$. By Corollary 6, the graph $G[c]$ is edge-transitive, which already puts severe restrictions on $G[c]$. However, as it turns out, due to the planarity and 3-connectedness of G, many edge-transitive graphs can in fact not appear as subgraphs $G[c]$. In the following, we classify the graphs $G[c]$ induced by an edge color c of Type III. The following lemma is a useful tool for the proof of our classification in Theorem 13, but also an interesting insight by itself, since it also yields restrictions on how different colors c, c' of Type III can appear together in one graph G.

▶ **Lemma 12.** *Let G be a 3-connected planar graph and let $c \in C_E(G, \chi^2_{\mathsf{WL}}[G])$ be an edge color of Type III. Then $G[c]$ is connected. Moreover, for every edge color $c' \in C_E(G, \chi^2_{\mathsf{WL}}[G])$ of Type III, it holds that $V(G[c]) \cap V(G[c']) \neq \emptyset$.*

Proof Idea. We focus on the first part of the lemma. The second part can be proved using similar arguments. Let $c \in C_E(G, \chi^2_{\mathsf{WL}}[G])$ be an edge color of Type III and suppose for simplicity that $G[c]$ is unicolored. Also let $A_1, \dots, A_\ell$ denote the vertex sets of the connected components of $G[c]$ and suppose towards a contradiction that $\ell \geq 2$. Consider the auxiliary graph H with vertex set $V(H) := \{A_1, \dots, A_\ell\}$ and edges A_iA_j whenever there is a path from a vertex $v_i \in A_i$ to a vertex $v_j \in A_j$ that is internally disjoint from $A_1 \cup \dots \cup A_\ell$. Note that H is connected because G is connected. Also, we have $\chi^2_{\mathsf{WL}}[H](A_i,A_i) = \chi^2_{\mathsf{WL}}[H](A_j,A_j)$ for all $i,j \in [\ell]$, by exploiting known properties of 2-WL (see, e.g., [9, Theorem 3.1.11]). So H is 2-connected with [32, Theorem 3.15].

Now, consider the graph $G - A_1$. Since H is 2-connected, all sets $A_2, \dots, A_\ell$ are contained in the same connected component of $G - A_1$. Let X denote the vertex set of this component. We claim that $N_G(X) = A_1$. Note that $N_G(X) \neq \emptyset$ since G is connected. Let $v \in N_G(X)$ and let $w \in A_1$. Since $G[c]$ is unicolored, we conclude that $\chi^2_{\mathsf{WL}}[G](v,v) = \chi^2_{\mathsf{WL}}[G](w,w)$. Also, since $v \in N_G(X)$, there is a path from v to another vertex $v' \in A_2 \cup \dots \cup A_\ell$ that is internally disjoint from $A_1 \cup \dots \cup A_\ell$. Since this property can be detected by 2-WL, such a path also exists starting in w. But this is only possible if $w \in N_G(X)$. So overall, $N_G(X) = A_1$.

By contracting the set X to a single vertex, we obtain that adding a universal vertex to $(G[c])[A_1]$ still results in a planar graph. However, by Theorem 4, we have that $(G[c])[A_1]$ is isomorphic to one of the graphs from Figure 2a – 2g. But this gives a contradiction since it is not possible to add a universal vertex to any of those graphs while preserving planarity. So $\ell = 1$, which means that $G[c]$ is connected. ◀

▶ **Theorem 13.** *Let G be a 3-connected planar graph and let $c \in C_E(G, \chi^2_{\mathsf{WL}}[G])$ be of Type III. Then one of the following holds.*

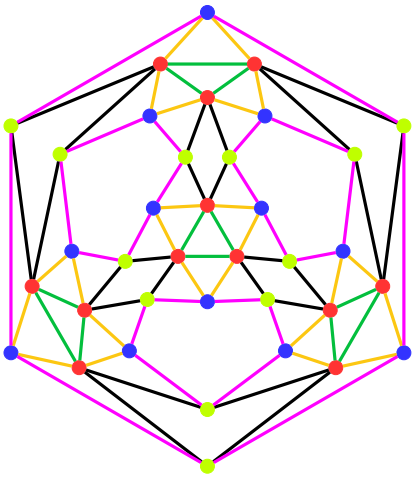
1. *$G[c]$ is bicolored and isomorphic to $K_{2,\ell}$ for some $\ell \geq 3$,*
2. *$G[c]$ is bicolored and isomorphic to a 2-subdivision of a cycle C_ℓ for some $\ell \geq 3$,*
3. *$G[c]$ is bicolored and isomorphic to a graph from Fig. 2h – 2j,*
4. *$G[c]$ is unicolored and isomorphic to a graph from Fig. 2a – 2g,*
5. *$G[c]$ is bicolored and isomorphic to a 1-subdivision of a graph from Fig. 2a – 2g, or*
6. *$G[c]$ is bicolored and isomorphic to a 2-subdivision of a graph from Fig. 2a – 2e.*

Observe that this classification is optimal in the sense that every graph listed in the theorem can actually appear as a graph $G[c]$ for some edge color c within a 3-connected planar graph. An easy way to see this is to take one of the graphs listed in the theorem, embed this graph H in the plane, place a fresh vertex v_F into every face F and connect it to all vertices lying on F. The resulting graph G is 3-connected and planar, and $H = G[c]$ for some edge color $c \in C_E(G, \chi^2_{\mathsf{WL}}[G])$.

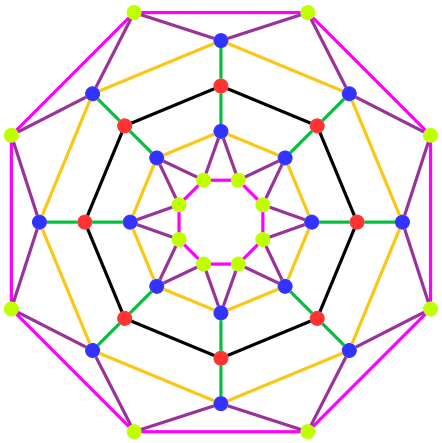
Also note that for a 3-connected planar graph G and an edge color $c \in C_E(G, \chi^2_{\mathsf{WL}}[G])$ of Type III, the automorphism group $\mathrm{Aut}(G)$ is isomorphic to a subgroup of $\mathrm{Aut}(G[c])$. Indeed, every automorphism $\gamma \in \mathrm{Aut}(G)$ naturally restricts to an automorphism $\gamma|_{V(G[c])}$ of $G[c]$ since the coloring computed by 2-WL is invariant. This gives rise to a homomorphism $\varphi\colon \mathrm{Aut}(G) \to \mathrm{Aut}(G[c])\colon \gamma \mapsto \gamma|_{V(G[c])}$. By Theorem 13 and Lemma 11, we obtain that $\mathrm{Singles}_G(w_1, \dots, w_\ell) = V(G)$ where $\{w_1, \dots, w_\ell\} = V(G[c])$. This implies that the kernel of φ is trivial, which implies that $\mathrm{Aut}(G)$ is isomorphic to a subgroup of $\mathrm{Aut}(G[c])$.

5 Disjoint Unions of Cycles

In this section, we consider 3-connected planar graphs of Type II. Let G be a 3-connected planar graph and let $\chi \coloneqq \chi^2_{\mathsf{WL}}[G]$ be the coloring computed by 2-WL. Suppose that G has Type II, i.e., there is an edge color $c \in C_E(G, \chi)$ of Type II, but there is no edge color of Type III. As before, we wish to understand in which ways edge colors of Type II can occur in G. More precisely, similarly to the case where G has Type III, our goal is to identify and classify connected subgraphs defined by few edge colors. Towards this end, we define three subcategories of edge colors of Type II. Let $c \in C_E(G, \chi)$ be of Type II. If $G[c]$ is unicolored and $\{(v,w) \mid \chi(v,w) = c\} \neq \{(v,w) \mid \chi(w,v) = c\}$ (i.e., $G[c]$ is a disjoint union of directed cycles), then we say that c has Type IIc. If c does not have Type IIc, but $G[c]$ is connected, then we say that c has Type IIb. If c does not have Type IIc, and $G[c]$ is not connected, then we say that c has Type IIa.



(a) A graph G of Type IIa where each edge color has Type IIa. The black and green edges induce a connected subgraph that is isomorphic to a parallel subdivision of a truncated tetrahedron.



(b) A graph G of Type IIb. The color black has Type IIb, yellow, violet and pink have Type IIa, and green has Type I. Note that $\mathrm{Aut}(G)$ is isomorphic to $(\mathrm{Aut}(G[\text{black}])) \times \mathbb{Z}_2$ because, after individualizing all red vertices, we can only swap the "interior" and the "exterior" region of the black cycle.

Figure 4 Two 3-connected planar graphs of Type II. All vertices and edges are colored by their 2-WL color. For visualization purposes, we only color edges and do not distinguish between potentially different colors of two arcs (v, w) and (w, v).

Let us remark that the main point of the subtypes is to distinguish between edge colors c of Type II that induce non-connected subgraphs (Type IIa) and those that induce connected subgraphs (Type IIb). The reason why we additionally single out the directed cycles (Type IIc) is that the existence of an edge color of Type IIc almost always (i.e., with the exception of one graph family) implies that the graph has fixing number 1, because individualizing a single vertex in a directed cycle fixes all other vertices on the cycle as well. In particular, we can show that every graph that contains an edge color of Type IIc is identified by 2-WL.

We also point out that the existence of an edge color of Type IIb immediately puts severe restrictions on the structure of G. For example, if $c \in C_E(G, \chi)$ is an edge of Type IIb, it can be shown that $\mathrm{Aut}(G)$ is isomorphic to a subgroup of $(\mathrm{Aut}(G[c])) \times \mathbb{Z}_2$ because, after individualizing all vertices of $G[c]$, an automorphism of G can only swap the "interior" and the "exterior" region of the cycle (see also Figure 4b).

Recall that the type of G is defined as the maximal type of any of its edge colors. We extend this definition to subtypes in the natural way. For example, G has Type IIb if there is an edge color $c \in C_E(G, \chi)$ of Type IIb, but no edge color of Type III or IIc. Two example graphs of Type II are given in Figure 4.

5.1 Directed Cycles

We start by analyzing graphs that contain an edge color c of Type IIc. As indicated above, this is a particularly well-behaved case because we can precisely classify those graphs that do not have fixing number 1. The *bipyramid (of order $m \geq 3$)* is the graph P_m^* with vertex set $V(P_m^*) \coloneqq \{u_1, u_2\} \cup \{v_i \mid i \in [m]\}$ and edge set $E(P_m^*) = \{u_i v_j \mid i \in [2], j \in [m]\} \cup \{v_i v_{i+1} \mid i \in [\ell - 1]\} \cup \{v_1 v_\ell\}$.

▶ **Lemma 14.** *Let G be a 3-connected planar graph. Also let $c \in C_E(G, \chi^2_{\mathsf{WL}}[G])$ be an edge color of Type IIc. Then there is a vertex $v \in V(G)$ such that $\mathrm{Singles}_G(v) = V(G)$, or G is isomorphic to a bipyramid.*

(a) Bicolored cube. **(b)** Chamfered tetrahedron.

Figure 5 A chamfered tetrahedron is obtained from a bicolored cube by truncating all red vertices.

▶ **Corollary 15.** *Let G be a 3-connected planar graph and suppose G contains an edge color of Type IIc. Then 2-WL determines pair orbits in G.*

Proof. If there is a vertex $v \in V(G)$ such that $\mathrm{Singles}_G(v) = V(G)$, then 2-WL determines pair orbits in G by Lemma 8. Otherwise, G is a bipyramid and it is easy to check that 2-WL determines pair orbits in G. ◀

5.2 Connected Substructures

In the remainder of this section, we analyze graphs of Type IIa and, similarly as before, aim at finding highly regular connected substructures. (Observe that if G has Type IIb, then a witnessing edge color already provides such an object). Unfortunately, we need to allow for two further possible outcomes. First, we are again satisfied with finding a vertex $v \in V(G)$ such that $\mathrm{Singles}_G(v) = V(G)$, which in particular implies that 2-WL determines pair orbits on G (see Lemma 8). As the other potential outcome, we consider *definable matchings*.

▶ **Definition 16.** *Let G be a graph and let $\chi := \chi^2_{\mathsf{WL}}[G]$. A color $c \in C_E(G,\chi)$* defines a matching *if for every $(v,w) \in \chi^{-1}(c)$, it holds that $\chi(v,v) \neq \chi(w,w)$, $\{v' \in V(G) \mid \chi(v',w) = c\} = \{v\}$, and $\{w' \in V(G) \mid \chi(v,w') = c\} = \{w\}$.*

Suppose there is some $c \in C_E(G,\chi)$ that defines a matching. Such a situation is generally beneficial since we can simply contract all edges of color c and move to a smaller graph. This operation neither changes the 2-WL coloring (see, e.g., [9, Theorem 3.1.11]) nor identification of the graph by 2-WL, as shown in the next lemma (see also [14]).

Let $c \in C_E(G,\chi)$ and let $A_1, \dots, A_\ell$ be the vertex sets of the connected components of $G[c]$. We define G/c as the graph obtained from contracting every set A_i to a single vertex. Formally, $V(G/c) := \{\{v\} \mid v \in V(G) \setminus V(G[c])\} \cup \{A_1, \dots, A_\ell\}$ and $E(G/c) := \{X_1X_2 \mid X_1, X_2 \in V(G/c), \exists v_1 \in X_1, v_2 \in X_2 \colon v_1v_2 \in E(G)\}$. We also define the pair coloring χ/c by setting $(\chi/c)(X_1,X_2) := \{\{\chi(v_1,v_2) \mid v_1 \in X_1, v_2 \in X_2\}\}$ for all $X_1, X_2 \in V(G/c)$.

▶ **Lemma 17.** *Let G be a graph and let $\chi := \chi^2_{\mathsf{WL}}[G]$. Also, let $c \in C_E(G,\chi)$ be an edge color that defines a matching. Suppose that 2-WL determines arc orbits (resp. pair orbits) on the arc-colored graph $(G/c, \lambda)$, where $\lambda(X_1,X_2) := (\chi/c)(X_1,X_2)$ for all $(X_1,X_2) \in A(G/c)$. Then 2-WL determines arc orbits (resp. pair orbits) on G.*

We now provide the main classification result of this section. We start by defining the graphs that appear in it. Let G be a 3-connected planar graph and let $W \subseteq V(G)$ be a set of vertices. We define H to be the graph obtained from G as follows. Let w be a vertex in W. First, subdivide each edge incident to w once. This gives $\ell := \deg_G(w)$ new vertices, which we call $w_1, \dots, w_\ell$ according to the unique cyclic order in any embedding of G. We then remove all edges ww_i and insert edges $w_iw_{(i+1) \bmod \ell}$ for $i \in [\ell]$, turning $w_1, \dots, w_\ell$ into a cycle. Each w_i inherits the color of w. We call these steps the *truncation* of w. One by

one, we then truncate each vertex in W. (Note that their order does not matter for the final result.) The obtained graph is H; see Figure 5 for an example. Now, we can define the following graphs.

- For every 3-connected planar graph G, the *truncated* G is obtained from G by truncating all vertices.
- A *chamfered tetrahedron* (Figure 5b) is obtained from a bicolored cube (Figure 5a) by truncating all red vertices[2].
- A *chamfered cube* is obtained from a rhombic dodecahedron (Figure 2i) by truncating all blue vertices.
- A *chamfered octahedron* is obtained from a rhombic dodecahedron (Figure 2i) by truncating all red vertices.
- A *chamfered dodecahedron* is obtained from a rhombic triacontahedron (Figure 2j) by truncating all blue vertices.
- A *chamfered icosahedron* is obtained from a rhombic triacontahedron (Figure 2j) by truncating all red vertices.

Next, let G be a planar graph. We define the *C_4-subdivision of G* to be the graph obtained from G by replacing each edge $vw \in E(G)$ with four vertices $(vw,1)$, $(vw,2)$, $(vw,3)$, $(vw,4)$ and edges $(vw,1)(vw,2)$, $(vw,2)(vw,3)$, $(vw,3)(vw,4)$, $(vw,4)(vw,1)$ and $v(vw,1)$, $w(vw,3)$.

Also, for $m \geq 2$, we define the graph C_m^* with vertex set $V(C_m^*) \coloneqq [m] \times [4]$ and edge set

$$\begin{aligned} E(C_m^*) \coloneqq \quad & \{(i,1)(i,2),(i,2)(i,3),(i,3)(i,4),(i,4)(i,1) \mid i \in [m]\} \\ & \cup \{(i,3)(i+1,1) \mid i \in [m-1]\} \cup \{(m,3)(1,1)\}. \end{aligned}$$

Finally, for $h \geq 3$, we define $K_{2,h}^*$ to be the graph with vertex set $V(K_{2,h}^*) \coloneqq \{u_1,u_2\} \uplus ([h] \times [4])$ and edge set

$$\begin{aligned} E(K_{2,h}^*) \coloneqq \quad & \{(i,1)(i,2),(i,2)(i,3),(i,3)(i,4),(i,4)(i,1) \mid i \in [h]\} \\ & \cup \{u_1(i,1) \mid i \in [h]\} \cup \{u_2(i,3) \mid i \in [h]\}. \end{aligned}$$

Examples for the last three constructions can be found in Figure 6.

Let H be a graph and $f\colon E(H) \to \mathbb{N}$ be a function. The *f-subdivision of H* is the graph $H^{(f)}$ obtained from H by replacing each edge e with $f(e)$ parallel paths of length 2 (if $f(e)=0$, the edge e remains unaltered). Formally, $H^{(f)}$ is the graph with vertex set $V(H^{(f)}) \coloneqq V(H) \uplus \{(e,i) \mid e \in E(H), i \in [f(e)]\}$ and edge set $E(H^{(s)}) \coloneqq \{e \in E(H) \mid f(e)=0\} \cup \{v(e,i) \mid e \in E(H), v \in e, i \in [f(e)]\}$. A graph G is a *parallel subdivision of H* if there is a function $f\colon E(H) \to \mathbb{N}$ such that G is isomorphic to $H^{(f)}$.

▶ **Theorem 18.** *Let G be a 3-connected planar graph of Type IIa and let $\chi \coloneqq \chi^2_{\mathsf{WL}}[G]$ be the coloring computed by 2-WL. Then one of the following options holds.*

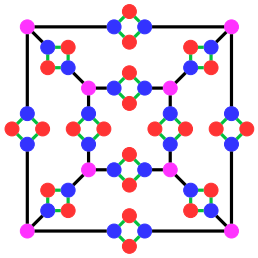
(A) *There is a vertex $v \in V(G)$ such that $\mathrm{Singles}_G(v) = V(G)$,*

(B) *there is an edge color $c \in C_E(G,\chi)$ that defines a matching, or*

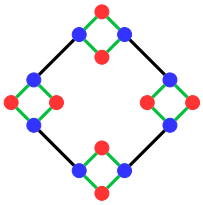
(C) *there are colors $c,d \in C_E(G,\chi)$ such that $G[c,d]$ is isomorphic to a parallel subdivision of one of the following graphs:*

1. *a truncated tetrahedron, a truncated cube, a truncated octahedron, a truncated dodecahedron, a truncated icosahedron,*
2. *an m-side prism for $m \geq 3$,*

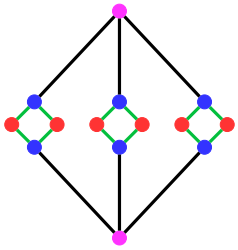
[2] The name *chamfered tetrahedron* comes from an alternative construction that obtains a chamfered tetrahedron by truncation of all edges of a tetrahedron.



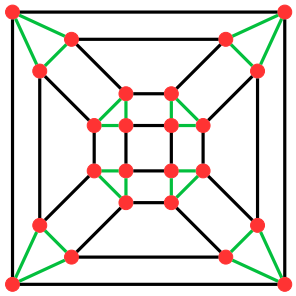
(a) C_4-subdivision of a cube.



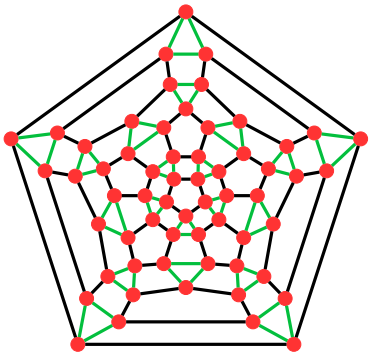
(b) C_4^*.



(c) $K_{2,3}^*$.



(d) Rhombicuboctahedron.



(e) Rhombicosidodecahedron.

Figure 6 Examples for the constructions from the classification for graphs of Type IIa.

3. *a cuboctahedron (with two edge colors), a rhombicuboctahedron, a rhombicosidodecahedron,*
4. *a C_4-subdivision of one of the graphs from Figure 2a – 2e,*
5. *a C_m^* for $m \geq 2$,*
6. *a $K_{2,h}^*$ for $h \geq 3$,*
7. *a chamfered tetrahedron, a chamfered cube, a chamfered octahedron, a chamfered dodecahedron, or a chamfered icosahedron.*

▶ Remark 19. There are four Archimedean solids that are explicitly listed neither in Theorem 13 nor in Theorem 18. These are the truncated cuboctahedron, the truncated icosidodecahedron, the snub cube, and the snub dodecahedron. The graphs corresponding to these solids have fixing number 1 under 1-WL and hence, they implicitly appear in Theorem 18, Option A.

Note that for Option C from Theorem 18, using the same arguments as for edge colors of Type III, we obtain that $\mathrm{Aut}(G)$ is isomorphic to a subgroup of $\mathrm{Aut}(G[c,d])$.

Overall, by combining Lemmas 9 and 14 and Theorems 13 and 18, we obtain that every 3-connected planar graph G satisfies one of the following options.

(A) There is some $v \in V(G)$ such that $\mathrm{Singles}_G(v) = V(G)$, which implies that 2-WL determines pair orbits of G by Lemma 8,

(B) there is an edge color $c \in C_E(G, \chi^2_{\mathsf{WL}}[G])$ that defines a matching, or

(C) there is a set $C \subseteq C_E(G, \chi^2_{\mathsf{WL}}[G])$ such that $|C| \leq 2$ and $G[C]$ is essentially a Platonic or Archimedean solid, or stems from a small number of infinite families of connected graphs.

Option C contains the graphs listed in Theorems 13 and 18, as well as the class of bipyramids from Lemma 14 and the class of cycles to cover graphs of Type IIb.

It remains an important open question whether 2-WL identifies every planar graph. With the structural insights from this paper, it now suffices to focus on Case C and, as explained above, the classification of the subgraphs $G[C]$ appearing in this case should be a crucial step to determining the WL dimension of planar graphs.

References

1 Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *Mach. Learn.*, 92(1):91–132, 2013. doi:10.1007/s10994-013-5385-0.

2 Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In Martin Farach-Colton and Sabine Storandt, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pages 73–84. SIAM, 2021. doi:10.1137/1.9781611976472.6.

3 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On Weisfeiler–Leman invariance: Subgraph counts and related graph properties. *J. Comput. Syst. Sci.*, 113:42–59, 2020. doi:10.1016/j.jcss.2020.04.003.

4 Albert Atserias and Elitza N. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. *SIAM J. Comput.*, 42(1):112–137, 2013. doi:10.1137/120867834.

5 Albert Atserias and Joanna Ochremiak. Definable ellipsoid method, sums-of-squares proofs, and the isomorphism problem. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 66–75. ACM, 2018. doi:10.1145/3209108.3209186.

6 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

7 Béla Bollobás. Distinguishing vertices of random graphs. In *Graph theory (Cambridge, 1981)*, volume 62 of *North-Holland Math. Stud.*, pages 33–49. North-Holland, Amsterdam-New York, 1982. doi:10.1016/S0304-0208(08)73545-X.

8 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Comb.*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

9 Gang Chen and Ilia N. Ponomarenko. Lectures on coherent configurations. Lecture notes available at http://www.pdmi.ras.ru/~inp/ccNOTES.pdf, 2019.

10 Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for CNF. In Sharad Malik, Limor Fix, and Andrew B. Kahng, editors, *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, pages 530–534. ACM, 2004. doi:10.1145/996566.996712.

11 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász meets Weisfeiler and Leman. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.40.

12 Zdenek Dvorák. On recognizing graphs by numbers of homomorphisms. *J. Graph Theory*, 64(4):330–342, 2010. doi:10.1002/jgt.20461.

13 Sergei Evdokimov, Ilia N. Ponomarenko, and Gottfried Tinhofer. Forestal algebras and algebraic forests (on a new class of weakly compact graphs). *Discret. Math.*, 225(1-3):149–172, 2000. doi:10.1016/S0012-365X(00)00152-7.

14 Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. Identifiability of graphs with small color classes by the Weisfeiler–Leman algorithm. *SIAM J. Discret. Math.*, 35(3):1792–1853, 2021. doi:10.1137/20M1327550.

15 Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. Local WL invariance and hidden shades of regularity. *Discret. Appl. Math.*, 305:191–198, 2021. doi:10.1016/j.dam.2021.08.037.

16 Martin Fürer. On the combinatorial power of the Weisfeiler–Lehman algorithm. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 260–271, 2017. doi:10.1007/978-3-319-57586-5_22.

17 Alexander L. Gavrilyuk, Roman Nedela, and Ilia N. Ponomarenko. The Weisfeiler–Leman dimension of distance-hereditary graphs. *CoRR*, abs/2005.11766, 2020. arXiv:2005.11766.

18 Martin Grohe. Fixed-point logics on planar graphs. In *Thirteenth Annual IEEE Symposium on Logic in Computer Science, Indianapolis, Indiana, USA, June 21-24, 1998*, pages 6–15. IEEE Computer Society, 1998. doi:10.1109/LICS.1998.705639.

19 Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *J. ACM*, 59(5):27:1–27:64, 2012. doi:10.1145/2371656.2371662.

20 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Association for Symbolic Logic, Ithaca, NY; Cambridge University Press, Cambridge, 2017. doi:10.1017/9781139028868.

21 Martin Grohe. The logic of graph neural networks. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–17. IEEE, 2021. doi:10.1109/LICS52264.2021.9470677.

22 Martin Grohe and Sandra Kiefer. A linear upper bound on the Weisfeiler–Leman dimension of graphs of bounded genus. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 117:1–117:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.117.

23 Martin Grohe and Sandra Kiefer. Logarithmic Weisfeiler–Leman identifies all planar graphs. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 134:1–134:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.134.

24 Martin Grohe and Daniel Neuen. Canonisation and definability for graphs of bounded rank width. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785682.

25 Martin Grohe and Martin Otto. Pebble games and linear equations. *J. Symb. Log.*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.

26 Branko Grünbaum and Geoffrey C. Shephard. Edge-transitive planar graphs. *J. Graph Theory*, 11(2):141–155, 1987. doi:10.1002/jgt.3190110204.

27 Lauri Hella. Logical hierarchies in PTIME. *Inf. Comput.*, 129(1):1–19, 1996. doi:10.1006/inco.1996.0070.

28 Neil Immerman. Expressibility as a complexity measure: results and directions. In *Proceedings of the Second Annual Conference on Structure in Complexity Theory, Cornell University, Ithaca, New York, USA, June 16-19, 1987*, pages 194–202. IEEE Computer Society, 1987.

29 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

30 Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.13.

31 Sandra Kiefer. The Weisfeiler–Leman algorithm: an exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020. doi:10.1145/3436980.3436982.

32 Sandra Kiefer and Daniel Neuen. The power of the Weisfeiler–Leman algorithm to decompose graphs. *SIAM J. Discret. Math.*, 36(1):252–298, 2022. doi:10.1137/20M1314987.

33 Sandra Kiefer and Daniel Neuen. A study of Weisfeiler—Leman colorings on planar graphs. *CoRR*, abs/2206.10557, 2022. arXiv:2206.10557.

34 Sandra Kiefer, Ilia N. Ponomarenko, and Pascal Schweitzer. The Weisfeiler–Leman dimension of planar graphs is at most 3. *J. ACM*, 66(6):44:1–44:31, 2019. doi:10.1145/3333003.

35 Brendan D. McKay. Practical graph isomorphism. *Congr. Numer.*, 30:45–87, 1981.

36 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

37 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33014602.

38 Joachim Redies. Defining PTIME problems on planar graphs with few variables. Master's thesis, RWTH Aachen University, 2014.

39 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler–Lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011. URL: http://dl.acm.org/citation.cfm?id=2078187.

40 Oleg Verbitsky. Planar graphs: Logical complexity and parallel isomorphism tests. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 682–693. Springer, 2007. doi:10.1007/978-3-540-70918-3_58.

41 Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series 2*, 1968. English translation by G. Ryabov available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

42 Hassler Whitney. Congruent Graphs and the Connectivity of Graphs. *Amer. J. Math.*, 54(1):150–168, 1932. doi:10.2307/2371086.

43 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

Beating Matrix Multiplication for $n^{1/3}$-Directed Shortcuts

Shimon Kogan
Weizmann Institute of Science, Rehovot, Israel

Merav Parter
Weizmann Institute of Science, Rehovot, Israel

Abstract

For an n-vertex digraph $G = (V, E)$ and integer parameter D, a *D-shortcut* is a small set H of directed edges taken from the transitive closure of G, satisfying that the diameter of $G \cup H$ is at most D. A recent work [Kogan and Parter, SODA 2022] presented shortcutting algorithms with improved diameter vs. size tradeoffs. Most notably, obtaining linear size D-shortcuts for $D = \widetilde{O}(n^{1/3})$, breaking the $\sqrt{n}$-diameter barrier. These algorithms run in $O(n^\omega)$ time, as they are based on the computation of the transitive closure of the graph.

We present a new algorithmic approach for D-shortcuts, that matches the bounds of [Kogan and Parter, SODA 2022], while running in $o(n^\omega)$ time for every $D \geq n^{1/3}$. Our approach is based on a reduction to the min-cost max-flow problem, which can be solved in $\widetilde{O}(m + n^{3/2})$ time due to the recent breakthrough result of [Brand et al., STOC 2021].

We also demonstrate the applicability of our techniques to computing the minimal chain covers and dipath decompositions for directed acyclic graphs. For an n-vertex m-edge digraph $G = (V, E)$, our key results are:

- An $\widetilde{O}(n^{1/3} \cdot m + n^{3/2})$-time algorithm for computing D-shortcuts of linear size for $D = \widetilde{O}(n^{1/3})$, and an $\widetilde{O}(n^{1/4} \cdot m + n^{7/4})$-time algorithm for computing D-shortcuts of $\widetilde{O}(n^{3/4})$ edges for $D = \widetilde{O}(n^{1/2})$.
- For a DAG G, we provide $\widetilde{O}(m + n^{3/2})$-time algorithms for computing its minimum chain covers, maximum antichain, and decomposition into dipaths and independent sets. This improves considerably over the state-of-the-art bounds by [Caceres et al., SODA 2022] and [Grandoni et al., SODA 2021].

Our results also provide a new connection between shortcutting sets and the seemingly less related problems of minimum chain covers and the maximum antichains in DAGs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Directed Shortcuts, Transitive Closure, Width

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.82

Category Track A: Algorithms, Complexity and Games

Funding This project is partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 949083).

1 Introduction

This paper is concerned with time-efficient algorithms for computing directed shortcut sets. For a given n-vertex digraph $G = (V, E)$ and an integer parameter D, a D-shortcut set H is a subset of edges from the transitive closure of G, denoted as $TC(G)$, satisfying that the diameter of $G \cup H$ is at most D. The diameter of the digraph is length of the longest u-v shortest path in G over any pair $(u, v) \in TC(G)$. The key objective in this setting is to optimize the diameter vs. size tradeoff. Since their introduction by Ullman and Yannakakis [27] and Thorup [26], shortcutting sets have been studied extensively due to

their wide-range of applications for parallel, distributed, dynamic and streaming algorithms [18, 14, 11, 22, 10, 13, 4]. Their applicability is also demonstrated by the recent breakthrough results, e.g., [22, 10, 13] that use shortcuts as their core component.

Diameter vs. Size Tradeoffs of Shortcuts. Thorup conjectured [26] that any n-vertex digraph with m edges, has an D-shortcut of size $\widetilde{O}(m)$ for[1] $D = \widetilde{O}(1)$. This has been shown to hold for a restricted class of graphs, such as planar [26]. Hesse [15] refuted this conjecture for general graphs by presenting a construction of an n-vertex digraph with $m = \Theta(n^{19/17})$ edges that requires $\Omega(mn^{1/17})$ shortcut edges to reduce its diameter to below $n^{1/17}$. Up to very recently, the only known size vs. diameter tradeoff was given by a folklore randomized algorithm, attributed to Ullman and Yannakakis [27], that for every integer $D \geq 1$, provides a D-shortcut of size $O((n/D)^2 \log^2 n)$. Berman et al. [3] improved the size bound to $O((n/D)^2)$. Setting $D = O(\sqrt{n})$ provides D-shortcuts of linear size in the number of vertices of the graph. This should be compared with the lower bound result by Huang and Pettie [16] that admits an n-vertex graph for which any linear shortcut (in the number of vertices of the graph) provides a diameter of $\Omega(n^{1/6})$. Lu, Vassilevska-Williams, Wein and Xu have recently improved the lower bound for D-shortcuts with $O(m)$ edges to $D = \Omega(n^{1/8})$.

Very recently, the authors [19] presented an improved tradeoff, breaking the $\sqrt{n}$ diameter barrier for linear-size shortcuts. For any $D \leq n^{1/3}$, they provide D-shortcuts of $\widetilde{O}(n^2/D^3)$ edges. For any $D > n^{1/3}$, they provide D-shortcuts with $\widetilde{O}((n/D)^{3/2})$ edges, hence of sublinear size. The algorithms of [19] are critically based on the precomputation the transitive closure of the graph, whose $O(n^\omega)$-time computation dominates the runtime of their shortcut constructions.

Time-Efficient Shortcut Computation. Due to the algorithmic importance of shortcuts, e.g., for reachability computation, much focus has been devoted to their time-efficient computation, in almost any classical computational setting, e.g., sequential, parallel and distributed, etc. In this algorithmic context, the primary objective is to compute shortcuts *faster* than computing the transitive closure itself. The reason is that shortcuts are usually used to provide faster algorithms for reachability related problems. The latter can be trivially solved by computing the transitive closure. Hence, for shortcuts to become algorithmically applicable, say in the sequential setting, their construction should run in $o(n^\omega)$ time. In a sequence of breakthrough results, Liu, Jambulapati and Sidford [22] extended the framework of Fineman [10] to compute, in nearly linear time, a shortcut set of $\widetilde{O}(n)$ edges that reduces the diameter of the graph to $D = O(n^{1/2+o(1)})$ (i.e., almost as obtained by the folklore algorithm). They also provide a parallel implementation of their algorithm leading to the first parallel reachability algorithm with near linear work and $n^{1/2+o(1)}$ depth.

The algorithms of [19] are based on computing the transitive closure of the graph. Towards making these structures applicable in the algorithmic context, e.g., for reachability, we ask:

▶ **Question 1.1.** *Is it possible to break the $\sqrt{n}$ diameter barrier of shortcuts in $o(n^\omega)$ time?*

We answer this question in the affirmative by providing a new algorithmic approach for shortcuts, that is based on a reduction to the *minimum-cost maximum-flow* problem. Luckily, the time complexity of the latter problem has only recently improved from $\widetilde{O}(\sqrt{n}m)$ (by Lee and Sidford [21]) to $\widetilde{O}(m + n^{3/2})$ by Brand et al. [29]. We observe that the key algorithmic challenge is in computing a collection of ℓ vertex-disjoint dipaths $\mathcal{P}$ (possibly in $TC(G)$)

[1] The $\widetilde{O}(.)$ notation hides poly-log n factors.

satisfying that the length of the longest path contained in the transitive closure induced on the set of *uncovered* vertices, namely, $U = V \setminus \bigcup_{P \in \mathcal{P}} V(P)$, is bounded by $\Theta(n/\ell)$. Our min-cost max-flow based approach for computing these dipaths finds broader applications for the seemingly less related problems of minimum chain covers and maximum antichains in directed acyclic graphs. Finally, we note our time complexity of shortcuts cannot be expressed *only* as a function of the time complexity of the min-cost max-flow problem, due to other computational steps that require $\widetilde{O}(m + n^{3/2})$ time.

Chains and Antichains: Covers and Decompositions. A *chain* is a dipath in the transitive closure $TC(G)$, and an antichain is subset of vertices $I \subseteq V$ that form an independent set in $TC(G)$. The closely related notions of chains and antichains give rise to four classical graph theoretical problems that have been studied thoroughly in the literature:

- **Minimum Chain Covers (MCC):** A minimum set of chains covering all vertices in G.
- **Maximum Antichain (MA):** An antichain of maximum size[2].
- **Minimum Antichain Covers (MAC):** A minimum set of antichains covering all vertices in G.
- **Longest Chain (LC):** A dipath of largest length in $TC(G)$.

The cardinality of the MCC is also known as the *width* of the graph, denoted by $\omega(G)$. Dilworth [9] showed that the size of the MA is the same as the size of the MCC, namely, $\omega(G)$. In a somewhat dual manner, Mirsky [24] showed that the length of the LC equals to the size of the MAC. Interestingly, while there are linear time algorithms for computing the MAC and LC (e.g., by dynamic programming, [24]), no such algorithms are known for the MCC and MA problems, at least for graphs with *non-constant* width. To date, the time complexity of the existing algorithms has an inherent dependency in $\omega(G)$. For example, for the MCC problem, the classical approach computes the chains one-by-one by solving $\omega(G)$ max-flow instances. Recent work by Cáceres et al. [5, 6] has studied MCC algorithms whose complexity is parametrized on $\omega(G)$. The state-of-the-art is due to [6] that runs in $\widetilde{O}(\omega(G)^2 n + m)$ time (for solving both MCC and MA). In light of the current gap in time complexity gap for solving MAC and LC on the one hand, and for MCC and MA, on the other hand, we ask:

▶ **Question 1.2.** *Is it possible to* also *compute MCC and MA in (near) linear time?*

We answer this question, at least partially, in the affirmative by presenting linear time MCC and MA algorithms for moderately dense graphs. It is also known that MCC and MA are at least as hard as computing maximum matching in bipartite graphs. Interestingly, our algorithms use the shortcut algorithms in a black-box manner! That is, an interesting take-home message is that the MCC and MA computations can be made faster by reducing first the the diameter of the digraph.

1.1 Our Results

We provide new algorithms for diameter shortcutting that break the $\sqrt{n}$ barrier without using any matrix multiplication, running in $o(n^\omega)$ time[3]. For example, for sparse graphs, our algorithms run in $o(n^2)$ for any diameter $D = \widetilde{\Omega}(1)$. Our first result, which in fact also serves as a building block later on, computes $\sqrt{n}$-shortcuts in near linear time:

[2] I.e., maximum independent set in $TC(G)$.
[3] The state-of-the-art value for the matrix multiplication constant is $\omega = 2.372$ due to Alman and Vassilevska Williams [1]

Table 1 Prior work on MCC. $(*)$ indicates that the MCC result is implicit.

Computation time	Citation
$O(n^3)$	[17]
$O(n^2+\omega(G)^{3/2}n)$	[7]
$O(\max(m\sqrt{n},\omega(G)^{3/2}n))$	[8]
$O(\omega(G)m\log n)$	[23](*)
$O(\omega(G)^2 n\log n+m)$	[6](*)
$\widetilde{O}(m+n^{3/2})$	**(this paper)**

▶ **Theorem 1.3** ($\Theta(\sqrt{n})$-Shortcuts of Linear Size). *There exists a randomized algorithm that for every n-vertex digraph G with m edges computes, w.h.p., in time $\widetilde{O}(m+n^{3/2})$ a shortcut set $H\subseteq TC(G)$ satisfying that $|E(H)|=\widetilde{O}(n)$ and $\texttt{Diam}(G\cup H)=O(\sqrt{n})$.*

For dense graphs, with $\Omega(n^{3/2})$ edges, this improves the diameter bound of Liu, Jambulapati and Sidford [22] that yields D-shortcuts for diameter $D=n^{1/2+o(1)}$. We next use this improved algorithm for computing D-shortcuts for any $D=O(\sqrt{n})$, to provide the following:

▶ **Theorem 1.4** ($o(\sqrt{n})$-Shortcuts). *There exists a randomized algorithm that for every n-vertex m-edge digraph G and $D=O(\sqrt{n})$, computes, w.h.p., in time $\widetilde{O}(m\cdot n/D^2+n^{3/2})$ a D-shortcut set $H\subseteq TC(G)$ with $|E(H)|=\widetilde{O}(n^2/D^3+n)$ edges.*

For example, for $D=n^{1/3}$, we get D-shortcuts of near-linear size in $\widetilde{O}(m\cdot n^{1/3}+n^{3/2})$ time, improving on the time complexity of $O(n^\omega)$ by [19].

We also consider faster algorithms for computing shortcuts with sublinear number of edges. This turns out to be quite technically involved, as many of the tools used in Theorem 1.4 are based on adding a linear number of edges, which we cannot afford in the sublinear regime. We show:

▶ **Theorem 1.5** ($\omega(n^{1/3})$-Shortcuts of Sublinear Size). *There exists a randomized algorithm that for every n-vertex m-edge digraph G and $D=\omega(n^{1/3})$, computes, w.h.p., in time $\widetilde{O}((m+n^{3/2})\sqrt{n/D})$ a D-shortcut set $H\subseteq TC(G)$ with $|E(H)|=\widetilde{O}((n/D)^{3/2})$ edges.*

For example, for $D=n^{1/2}$, this provides D-shortcuts with $\widetilde{O}(n^{3/4})$ edges in $\widetilde{O}(m\cdot n^{1/4}+n^{7/4})$ time, improving on the time complexity of $O(n^\omega)$ by [19].

Application to Decompositions and Covers of Chains and Antichains. Recall that a chain (resp., antichain) is a dipath (resp., independent set) in $TC(G)$. A minimum chain cover (MCC) for a DAG $G=(V,E)$ is a collection of vertex-disjoint chains that cover $V(G)$. The maximum antichain is the maximum independent set in $TC(G)$. We provide the first nearly linear algorithms for moderately dense graphs, that in contrast to prior algorithms, do not depend on the width of the graph.

▶ **Theorem 1.6.** *For every n-vertex m-edge DAG $G=(V,E)$, one can compute the minimum chain cover, as well as, the maximum antichain in time $\widetilde{O}(m+n^{3/2})$, w.h.p.*

This improves over that very recent state-of-the-art running time of $O((\omega(G))^2 n\log n+m)$ by [6] for graphs of large width $\omega(G)=\widetilde{\Omega}(n^{1/4})$. More conceptually this provides a near linear-time algorithm for dense graphs, narrowing the gap to the related problems of MAC and ML for which linear time algorithms are folklore. Finally, we consider the decomposition of G into a collection of vertex-disjoint dipaths $\mathcal{P}$ and vertex-disjoint independent sets[4].

[4] Grandoni et al. [12] called this *chain and antichain decomposition*. Since this decomposition is in G and the notions of chain and antichains are usually defined in $TC(G)$, we use the terminology of dipaths

▶ **Definition 1.7** (Slight restatement of Def. 3.1 of [12]). *An (a, b)-decomposition of a directed acyclic graph (DAG) G consists of a collection $\mathcal{P} = \{P_1, \ldots, P_a\}$ of a vertex-disjoint directed paths in G, and a collection $\mathcal{Q} = \{Q_1, \ldots, Q_b\}$ of b vertex-disjoint independent sets of G such that $\bigcup_{i=1}^{a} V(P_i) \cup \bigcup_{j=1}^{b} Q_j = V(G)$.*

Grandoni et al. [12] presented an $O(n^2)$-time algorithm for computing an $(\ell, 2n/\ell)$ decomposition in G. In fact, the algorithm of [19] employed this decomposition on the transitive closure of G. We observe that by reducing to the min-cost max-flow problem, one can break the $O(n^2)$-time complexity. We show:[5]

▶ **Theorem 1.8.** *Let $G = (V, E)$ be an n-vertex DAG. For every $\ell \in [1, n]$, there is an $\widetilde{O}(|E| + n^{3/2})$-time randomized algorithm for computing an $(\ell, 2 \cdot n/\ell)$-decomposition of G.*

1.2 Technical Overview

1.2.1 A New Algorithmic Approach for Shortcuts

For the sake of presentation, we focus on the computation of a $D = O(n^{1/3})$-shortcut H of near-linear size. We start by providing a succinct description of the Kogan-Parter algorithm in [19]. Throughout, we assume, w.l.o.g., that the input graph G is a DAG, and denote its transitive closure by $TC(G)$. We note that in context of computing shortcuts of *sublinear* size, we provide an alternative reduction to DAG[6]. For a dipath P, let $H(P)$ be a 2-shortcut set for P satisfying that $\texttt{Diam}(P \cup H(P)) \leq 2$. It is well-known that one can compute $H(P)$ in nearly-linear time and that $|H(P)| = O(|P| \log |P|)$. For a path collection $\mathcal{P}$, let $V(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} V(P)$. Finally, for a set of elements X and probability $p \in [0, 1]$, let $X[p]$ be the subset obtained by sampling each element of X into $X[p]$ independently w.p. p.

A Quick Recap of the $O(n^\omega)$-Time Algorithm by [19]. The algorithm starts by computing $TC(G)$ in $O(n^\omega)$-time. It then applies the decomposition algorithm Grandoni et al. [12] to partition $V(TC(G))$ into $n^{2/3}$ vertex-disjoint dipaths $\mathcal{P}$ (denoted as chains), and $2 \cdot n^{1/3}$ vertex-disjoint antichains $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$. I.e., $V = V(\mathcal{P}) \cup V(\mathcal{Q})$. Letting $U = V(\mathcal{Q})$, the key property that we use in the diameter argument is that the length of a longest path of the graph[7] $TC(G)[U]$ is bounded by $k = n^{1/3}$, which indeed follows by the definition of $\mathcal{Q}$.

For a vertex v and a dipath P, let $e(v, P)$ be the edge connecting v to the *first* vertex in $V(P)$ that is reachable from v (if such exists). Let $\mathcal{P}' = \mathcal{P}[p]$ and $V' = V[p]$ for $p = n^{-1/3}$ (i.e., subsamples of dipaths and vertices). Then the output shortcut set is $H = H_1 \cup H_2$, where $H_1 = \bigcup_{P \in \mathcal{P}} P \cup H(P)$ and $H_2 = \{e(v, P) \mid v \in V', P \in \mathcal{P}'\}$. Summarizing, the key algorithmic steps can be summarized by:

- **Step 1.** Computing a collection $\mathcal{P}$ of (at most) $n^{2/3}$ *vertex-disjoint* dipaths such that the length of the longest dipath of $TC(G)[U]$ is of length $O(n^{1/3})$, where $U = V \setminus V(\mathcal{P})$.
- **Step 2.** Computing the edges $e(v, P)$ for every $v, P \in \mathcal{P}' \times V'$.

The implementation of [19] for both steps uses $TC(G)$ in a strong manner. Before explaining our approach, we sketch the size and diameter bounds of the above construction.

---

and independent sets.

[5] We provide this as an independent observation. As this decomposition is w.r.t G (rather than $TC(G)$), it is not useful for our shortcut algorithms.

[6] An alternative reduction is needed since the folklore approach of contracting each strongly connected component in G introduces linear number of edges to the shortcut.

[7] I.e., the transitive closure $TC(G)$ induced on the uncovered vertices U, namely, vertices that do not appear on $\mathcal{P}$.

Size and Diameter Bound. The size bound is immediate: the $\mathcal{P}$ are vertex-disjoint, thus $|H_1| = \widetilde{O}(n)$. W.h.p., $|V'| = \widetilde{O}(n^{2/3})$ and $|\mathcal{P}'| = \widetilde{O}(n^{1/3})$, therefore, $|H_2| = \widetilde{O}(n)$, as well.

Consider a u-v shortest path P in $G \cup H_1$ and denote its $7n^{1/3}$-length prefix (resp., suffix) by P', P'', respectively. By Chernoff, w.h.p., there exists some sampled vertex $u^* \in V(P') \cap V'$. The challenge is in showing that P'' intersects with $\Omega(n^{1/3})$ distinct paths in $\mathcal{P}$, and thus, w.h.p., intersects with at least one sampled path in $\mathcal{P}'$. This is shown by observing that (i) P'' contains at most 3 vertices from each dipath $Q \in \mathcal{P}$ (due to the addition of the shortcut $H(Q)$), and that (ii) P'' contains at most $n^{1/3}$ vertices in U. Altogether, we have some $Q^* \in \mathcal{P}'$ satisfying that $V(Q^*) \cap V(P'') \neq \emptyset$. The edge $e(u^*, Q^*) \in H_2$ then provides us an $O(n^{1/3})$-hop length dipath from u^* to $w \in V(Q^*) \cap V(P'')$.

New Approach: Shortcuts via Min-Cost Max-Flows. We focus on Step (1), as Step (2) can be implemented in $O(m \cdot n^{1/3})$ time using e.g., dynamic programming. Our goal is to show that Step (1) can be implemented in $\widetilde{O}(m + n^{3/2})$ time, which establishes Theorem 1.4 for $D = n^{1/3}$. The challenge boils down into computing a *nice* collection of dipaths which also has further independent applications, as illustrated in this paper. Denote by $\mathrm{LP}(G)$ to be the length of a longest simple dipath in G.

> **Key Task:** Compute in $o(n^\omega)$-time a collection of (at most) $n^{2/3}$ vertex-disjoint dipaths $\mathcal{P}$ in $TC(G)$, such that $\mathrm{LP}(G') = O(n^{1/3})$ where $G' = TC(G)[U]$ and $U = V \setminus V(\mathcal{P})$.

Note that the above requires to bound the length of the longest path in the graph[8] $TC(G)[U]$. This task can also be viewed as a relaxed chain and antichain decomposition (see Def. 1.7) for $\ell = n^{2/3}$, where it is required to output only chain part, while guaranteeing the remaining uncovered vertices, U, can be decomposed into $n^{1/3}$ anti-chains[9]. Clearly the challenge is that it is required to compute such (partial) decomposition in $TC(G)$ *without ever* computing it explicitly! While our goal is to solve it in $\widetilde{O}(m + n^{3/2})$ time, the decomposition algorithm of [12] runs in $O(n^2)$ even when *given* $TC(G)$. Our challenge is therefore two-folds.

We introduce ℓ-*covers* which for a given input parameter $\ell \in [1, n]$ provide a *multiset* $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ of ℓ dipaths in G. Letting, $\mathrm{TotLen}(\mathcal{P}) = \sum_{P_i \in \mathcal{P}} |P_i|$ and $U = V \setminus V(\mathcal{P})$ (i.e., the total dipath lengths), then the ℓ-cover satisfies:

1. $\mathrm{TotLen}(\mathcal{P}) \leq \min\{\ell, \mathtt{Diam}(G)\} \cdot n$,
2. $\mathrm{LP}(G') \leq 2n/\ell$ where $G' = TC(G)[U]$.

Note that an ℓ-cover for $\ell = n^{2/3}$, almost fits the key task, up to one major caveat: the total path length, $\mathrm{TotLen}(\mathcal{P})$, might be super-linear, hence leading to shortcuts of super-linear size[10]. We put this technicality aside for a moment, explain first how to compute ℓ-covers *without* computing $TC(G)$, and then show how to use them to solve the key task.

Computing ℓ-Covers in Time $\widetilde{O}(m+n^{3/2}+\min\{\ell, \mathtt{Diam}(G)\}\cdot n)$. The algorithm is based on a reduction to the min-cost max-flow problem. We define a flow-instance $\widetilde{G} = (\widetilde{V}, \widetilde{E}, u, c)$ corresponding to G, where $|\widetilde{V}| = O(n), |\widetilde{E}| = O(m)$, $u \in \mathbb{Z}_{\geq 0}^{\widetilde{E}}$ and $c \in \mathbb{Z}^{\widetilde{E}}$ are the capacity and cost functions, respectively. We then apply the recent algorithm of van den Brand et al. [29] to compute the min-cost max-flow s-t flow in $\widetilde{G}$ for a given pair $s, t \in \widetilde{V}$. In their breakthrough result, [29] provided an $\widetilde{O}(m + n^{3/2})$-time randomized algorithm for this problem, provided that the edge capacities and costs are *integral*.

---

[8] The transitive closure of G induced on the vertices of U.
[9] By [24], a bounded longest path of $TC(G)[U]$ indeed guarantees the antichain decomposition of U.
[10] As we include in the output shortcut, the union of path shortcuts $H(P)$ for every $P \in \mathcal{P}$.

The desired ℓ-cover collection is then obtained by computing the flow-decomposition of the output integral flow. Since G is a DAG, this decomposition can be done in linear time in the total length of the output path collection. Finally, we translate the collection of dipaths in $\widetilde{G}$ into a collection of dipaths in G, which provides the desired ℓ-cover $\mathcal{P}$. Importantly, our flow-instance deviates from the classical flow-reductions[11] by incorporating a special gadget per vertex $v_i \in V$. We carefully set the capacity and the cost values in a way that on the one hand, allows a given vertex to participate in multiple paths, while at the same time bounding the total length of $\mathcal{P}$, and most importantly bounds the longest path length of the transitive closure induced on the uncovered vertices U, $TC(G)[U]$. Altogether, the computation of the ℓ-cover $\mathcal{P}$ takes $\widetilde{O}(m + n^{3/2} + \text{TotLen}(\mathcal{P}))$ time which by Property (1) is bounded by $\widetilde{O}(m + n^{3/2} + n \min\{\ell, \texttt{Diam}(G)\})$.

ℓ-Covers in $\widetilde{O}(m + n^{3/2})$ Time. To beat this dependency in $\text{TotLen}(\mathcal{P})$, we take the following strategy. First, we reduce the diameter of G to $d = O(\sqrt{n})$ by computing a d-shortcut set H_0 of linear size. We show that this can be done using ℓ-covers for $\ell = \sqrt{n}$, hence taking $O(m + n^{3/2})$ time. We then apply the ℓ-cover algorithm on the graph $G \cup H_0$. Since the diameter of $G \cup H$ is $O(\sqrt{n})$, we have that $\text{TotLen}(\mathcal{P}) = O(n^{3/2})$, and the entire computation takes $\widetilde{O}(m + n^{3/2})$ time.

Implementing Step (1) in $\widetilde{O}(m + n^{3/2})$ Time. So-far, we have computed a collection of $O(n^{2/3})$ dipaths $\mathcal{P}$ (in $G \cup H_0$) of total length $O(n^{3/2})$, and such that the uncovered diameter is at most $O(n^{1/3})$. It remains to turn $\mathcal{P}$ into vertex-disjoint dipaths in $TC(G)$. This can be easily done by iterating over the dipaths of $\mathcal{P}$, one by one, shortcutting the given dipath by cutting out the vertices that appear in the currently collection of vertex-disjoint dipaths. This works in $O(\text{TotLen}(\mathcal{P})) = O(n^{3/2})$ time.

Shortcuts of Sublinear-Size. In [19], it was shown that for any $D = \omega(n^{1/3})$, one can compute D-shortcuts of size $\widetilde{O}((n/D)^{3/2})$. It will be instructive to consider the case where $D = O(\sqrt{n})$ for which the construction provides shortcuts with $\widetilde{O}(n^{3/4})$ edges. The algorithm of [19] for this setting is based on sampling a collection of landmarks $V' = V[p]$ for $p = n^{1/4}$. It then defines the graph $G' = (V', E')$ where $E' = \{(u, v) \mid \text{dist}_G(u, v) \leq n^{1/4}\}$, and applies the above mentioned algorithm for computing $D' = |V'|^{1/3}$-shortcut H of size $O(|V'|)$. Since $|V'| = O(n^{3/4})$, w.h.p., this provides a shortcut of $O(n^{3/4})$ edges.

In our context, computing the graph G' takes $O(|V'|m)$ time which is too costly for our purposes. Another challenge in provide a *sublinear* reduction into DAGs. So-far, we assumed w.l.o.g. that G is a DAG, by using the following folklore reduction: shortcut each strongly connected component C to diameter 2, by connecting the all vertices of C to center vertex $v \in C$. This allows one to restrict attention to the DAG obtained by contracting each component. This reduction, however, adds $\Theta(n)$ shortcut edges, which is above our budget. Our algorithm provides an alternative reduction. Given G, we compute a DAG G' of similar size and show that any D-shortcut set H' for G' can be translated into a D-shortcut set H for G, where $|H| = O(|H'|)$. This allows us to restrict attention to DAGs. We believe this alternative reduction should be of general interest.

Our shortcut algorithm calls for a collection of additional refined tools. We introduce partial ℓ-covers that provide the desired properties (1,2), up to some slack, with respect to a given *subset* of vertices. The shortcut algorithm is then based on iteratively computing collections of partial ℓ-covers, to mimic, in some way, the effect of the single application of a shortcut computing on the virtual graph G'.

[11] E.g, where each vertex $v_i \in V$ has two copies v_i^{in} and v_i^{out} connected by a directed edge of capacity 1.

1.2.2 Minimum Chain Covers, Maximum Antichain, and More

Recall that a chain is a dipath in $TC(G)$ and an antichain is an independent set in $TC(G)$. The cardinality of the minimum chain cover (MCC) of a DAG G is denoted as the width of G, $\omega(G)$. An minimum path cover (MPC) is a minimum set of dipaths (not necessarily disjoint) in G that cover V. Prior algorithms for computing the MCC are based on iteratively solving a maximum s-t flow instance in the residual graph, each iteration provides one additional dipath to the output collection. This dipath-by-dipath approach requires solving $\omega(G)$ flow computations. Note that computation of the value of the width, $\omega(G)$, can be easily done using only $O(\log n)$ max-flow computations (by binary search). Hence, the main challenge is in the computation of chain cover itself.

Our approach is based on having a single application of the min-cost max-flow algorithm. The starting observation is that thanks to our shortcut algorithms, we can assume, w.l.o.g., that the diameter of G is $O(\sqrt{n})$. This is because we can apply the MCC algorithm on the graph $G \cup H_0$ where H_0 is the $O(\sqrt{n})$-shortcut set for G, obtained by Thm. 1.3. Since we aimed at computing *chains*, rather than dipaths in G, we can safely work on $G \cup H_0$.

Using our flow reduction, we are able to compute a collection $\mathcal{P}$ of $\omega(G)$ dipaths (these are dipaths in $G \cup H_0$), in time $\widetilde{O}(m + n^{3/2} + \text{TotLen}(\mathcal{P}))$. By the way that we set the cost of the edges in our instance, $\mathcal{P}$ covers all vertices in G. I.e., it is an MPC in $G \cup H_0$. This by itself is not enough as the dipaths of $\mathcal{P}$ are might not be vertex-disjoint, possibly with large total length, $\text{TotLen}(\mathcal{P})$.

To overcome this, we first provide an algorithm for computing a dipath cover of *minimum* total length. We introduce the notion of *minimum-length cover* (MLC): a collection of $\omega(G)$ dipaths, that covers all vertices while minimizing the *total length* of the dipaths.

Then, we provide a combinatorial argument that shows that any n-vertex DAG admits an MPC $\mathcal{P}$ of length $\text{TotLen}(\mathcal{P}) \leq n \cdot \texttt{Diam}(G)$. Since $\texttt{Diam}(G \cup H_0) = O(\sqrt{n})$, we end up with having a collection of dipaths of total length $O(n^{3/2})$. These dipaths can be transformed in vertex-disjoint chains in time $O(n^{3/2})$, in a brute-force manner (Lemma 1.15). This completes the high-level approach.

1.3 Preliminaries

Graph Notations. For an n-vertex digraph G, let $TC(G)$ denote its transitive closure. We also denote $TC(G)$. For a vertex pair $u, v \in V(G)$ where $(u, v) \in TC(G)$, let $\text{dist}_G(u, v)$ be the length (measured by the number of edges) of the shortest dipath from u to v. For $(u, v) \notin TC(G)$, $\text{dist}_G(u, v) = \infty$. For a subset $V' \subseteq V$, let $G[V']$ be the induced graph on V'. The graph *diameter* is denoted by $\texttt{Diam}(G) = \max_{(u,v)\in TC(G)} \text{dist}_G(u, v)$. We say that $u \rightsquigarrow_G v$ if there is a directed path from u to v in G, i.e., $(u, v) \in TC(G)$. A *shortcut* edge is an edge in $TC(G)$. For a vertex $v \in G$, let $N_{in}(v, H) = \{u \mid (u, v) \in G\}$ denote the incoming neighbors of v in G. The set of outgoing neighbors $N_{out}(v, G)$ is defined in an analogous manner. For a collection of paths $\mathcal{P}$, the vertices of $\mathcal{P}$ is denoted by $V(\mathcal{P}) = \bigcup_{P\in\mathcal{P}} V(P)$. For a dipath $P = [u_0, \ldots, u_k] \subseteq G$, for $j \leq q$, let $P[u_j, u_q]$ denote the path segment $P[u_j, u_{j+1}, \ldots, u_q]$. In the case where $u_j = u_0$ (resp., $u_q = u_k$), we simply write $P[\cdot, u_q]$ (reps., $P[u_j, \cdot]$). For an a-b dipath P and an b-c dipath P' the concatenation of the paths is denoted by $P \circ P'$. Let $|P|$ denote the number of edges on P (unless mentioned otherwise). For a set of elements X and $p \in [0, 1]$, let $X[p]$ be the set obtained by taking each element of X into $X[p]$ independently with probability p.

▶ **Definition 1.9.** *For a digraph G, $H \subseteq TC(G)$ is* D-shortcut *if* $\texttt{Diam}(G \cup H) \leq D$.

The following lemma computes linear 2-shortcuts for directed paths:

▶ **Lemma 1.10** (Lemma 1.1, [25]). *Given a directed path P, one can compute in $O(|P|\log|P|)$ time, a 2-shortcut $H(P)$ for P where $|H| = O(|P|\log|P|)$ edges.*

Given a digraph G, by G^+ we denote the result of contracting all strong components in G. Note that G^+ is acyclic directed graph (DAG) and can be computed in linear time.

▶ **Proposition 1.11** (Lemma 3.2, [25]). *Let H^+ be a shortcut set for G^+. One can compute in $O(|E(G)|+|H|)$-time, a shortcut set H for G such that $|H| = O(|H^+|+n)$ and $\texttt{Diam}(G\cup H) = O(\texttt{Diam}(G^+ \cup H^+))$.*

Algorithmic Tools for Flow Computation. In the *minimum-cost maximum-flow* problem, given is a connected directed graph $G = (V, E, u, c)$ with edges capacities $u \in \mathbb{R}^E_{\geq 0}$ and costs $c \in \mathbb{R}^E$ (which can be negative). The vector $x \in \mathbb{R}^E$ is an *s-t flow* for $s, t \in V$ if $x(e) \in [0, u(e)]$ for all e in E, and for each vertex $v \notin \{s, t\}$ the amount of flow entering v equals the amount of flow leaving v, i.e., $\sum_{e=(a,v)} x(e) = \sum_{e=(v,b)} x(e)$. The *cost* of a flow x is defined by $c(x) = \sum_e c(e)x(e)$. The *value* of an s-t flow is the amount of flow leaving s, i.e., $val(x) = \sum_{e=(s,v)} x_e$ (or equivalently, entering t, i.e., $\sum_{e=(v,t)} x_e$). The objective is to compute a maximum s-t flow of minimum cost denoted by $\sum_{e\in E} c_e x_e$. The following theorem was proven in [29].

▶ **Theorem 1.12** (Theorem 1.4 of [29]). *There is an algorithm* MinCostFlow(G, s, t) *that, given a n-vertex m-edge digraph $G = (V, E, u, c)$, integral edge capacities $u \in \mathbb{Z}^E_{\geq 0}$ and costs $c \in \mathbb{Z}^E$, w.h.p., computes an integral minimum cost maximum flow in time*

$$\widetilde{O}\left(m\log(||u||_\infty||c||_\infty) + n^{3/2}\log^2(||u||_\infty||c||_\infty)\right) .$$

We observe that even more recently, there have been additional breakthrough results [2, 28] that provide improved running time for sparse graphs, this however, does not effect our final time complexity.

▶ **Definition 1.13.** *For a digraph $G = (V, E)$ and a given valid s-t flow vector $x \in \mathbb{N}^E_{\geq 0}$, a* flow decomposition *is a multiset of s-t dipaths in G given by $\mathcal{Q} = \{P_1, \ldots, P_k\}$, such that for every $e \in E$, it holds that $x(e) = |\{P_i \in \mathcal{Q} \mid e \in P_i\}|$. I.e., $x(e)$ equals the number of paths in the multiset containing e.*

▶ **Lemma 1.14.** *Let $G = (V, E)$ be an n-vertex m-edge DAG and let $x \in \mathbb{N}^E_{\geq 0}$ be an integral s-t flow of value k for $s, t \in V$, then one can compute a flow decomposition of x, denoted by $\mathcal{Q}$ in $O(m + n + \sum_{e\in E} x(e))$ time.*

Dipaths to Chains. Throughout, we make use of the following procedure to translate a collection of dipaths in G into a collection of vertex-disjoint chains. The proof of the next lemma is in the full version.

▶ **Lemma 1.15.** *Let $\mathcal{P}$ be a collection of dipaths in G, there is an algorithm* DisjointChains *that in time $O(\sum_{P\in\mathcal{P}} |P|)$ computes a collection of* vertex-disjoint *chains $\mathcal{C}$ satisfying that: (i) $|\mathcal{C}| \leq |\mathcal{P}|$, (ii) $V(\mathcal{C}) = V(\mathcal{P})$ and (iii) every $C \in \mathcal{C}$ is a dipath in $TC(G)$.*

Roadmap. In Sec. 2, we introduce ℓ-covers, that serve the key algorithmic part for computing our shortcuts. In Sec. 3.1, we compute $\sqrt{n}$-shortcuts of linear size (proof of Thm. 1.3). In Sec. 3.2, we provide an algorithm that computes $n^{1/3}$-shortcut of linear size in $\widetilde{O}(mn^{1/3} + n^{3/2})$ time, and more generally prove Thm. 1.4. Then Sec. 3.3 considers the constructions of shortcuts with sublinear number of edges establishing Thm. 1.5.

2 New Notions of Dipath Decompositions

2.1 ℓ-Covers

We introduce a new notion of dipath decomposition which we denote by *ℓ-covers*. For a given parameter ℓ, the ℓ-cover is given by a multiset of ℓ dipaths in G, $\mathcal{P} = \{P_1, \ldots, P_\ell\}$, whose vertices $V(\mathcal{P})$ dominate long dipaths in G, in a way that becomes formal in Def. 2.1. As we will see, this notion is strong enough to substitute the path collection obtained by computing the chain-antichain decomposition in $TC(G)$. Missing proofs of this section are deferred to the full version. Recall that for a multiset of dipaths $\mathcal{P}$, $\text{TotLen}(\mathcal{P}) = \sum_{P_i \in \mathcal{P}} |P_i|$.

▶ **Definition 2.1** (ℓ-Covers). *For a given n-vertex digraph $G = (V, E)$ and an integer parameter $\ell \in [1, n]$, an ℓ-cover for G is a multiset of ℓ paths $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ (possibly consisting of single vertices or empty), satisfying the following:*

1. $\text{TotLen}(\mathcal{P}) \leq \min(\ell \cdot n, \texttt{Diam}(G) \cdot n)$,
2. *For any dipath $P \subseteq G$, it holds that $|V(P) \cap (V \setminus V(\mathcal{P}))| \leq 2n/\ell$.*

Property (2) implies that $\text{LP}(TC(G)[U]) \leq 2n/\ell$, that is the following holds:

▶ **Corollary 2.2.** *Each path in $TC(G)[V \setminus V(\mathcal{P}]$ contains at most $2n/\ell$ vertices.*

▶ **Theorem 2.3.** *Let G be an n-vertex DAG with m edges, then for every $\ell \geq 1$, there is a randomized algorithm for computing an ℓ-cover of G, w.h.p, in time $\widetilde{O}(m + n^{3/2} + n \cdot \min\{\texttt{Diam}(G), \ell\})$.*

For our purposes of computing shortcuts, we employ the algorithm of Thm. 2.3 only in settings where $\min\{\texttt{Diam}(G), \ell\} = O(\sqrt{n})$, therefore spending only $\widetilde{O}(m + n^{3/2})$ time.

Path Decomposition via Min-Cost Max-Flow. The decomposition algorithm is based on a reduction to the *minimum cost maximum flow* problem (see Sec. 1.3). For the given DAG G, the algorithm computes a corresponding min-cost flow instance $\widetilde{G} = (\widetilde{V}, \widetilde{E}, u, c)$. At the high level, for each $v_i \in V(G)$, the graph $\widetilde{G}$ contains three inter-connected copies $v_i^{in}, v_i^{out}, v_i'$. The out-copy of the incoming G-neighbors of v_i are connected to the in-copy of v_i. In addition, $\widetilde{V}$ includes also three additional vertices, s, s' and t, where s and t are the source and target vertices given to the min-cost flow algorithms. The precise definition of $\widetilde{G}$ are specified in the pseudocode below, see also Fig. 1.

Let $x \in \mathbb{N}_{\geq 0}^{|\widetilde{E}|}$ be the output flow solution for the instance $\widetilde{G}$ obtained by applying Alg. MinCostFlow from Theorem 1.12. The algorithm then decomposes the flow x into a multiset of s-t paths $\widetilde{\mathcal{P}} = \{P_1', \ldots, P_\ell'\}$. Since without loss of generality the flow values are integrals and since $\widetilde{G}$ is a DAG, the flow decomposition of x can be computed efficiently by applying Lemma 1.14.

Finally, the output ℓ-cover $\mathcal{P}$ is obtained by mapping each $\widetilde{G}$-dipath $P_j' \in \widetilde{\mathcal{P}}$ into a corresponding G-dipath P_j. This mapping is done by applying Procedure Translate on every $P_j' \in \widetilde{\mathcal{P}}$. The output G-path $P_j = \textsf{Translate}(P_j')$ is defined as follows. The vertices s, s' and t are omitted, and each appearance of a vertex $u_j \in v_i^{in}, v_i^{out}, v_i'$ is replaced by its G-vertex v_i, we omit multiple successive occurrences of a vertex in this translation. For example, for $P' = [s, s', v_i^{in}, v_i^{out}, v_j^{in}, t]$, $\textsf{Translate}(P') = [v_i, v_j]$. For a path $P'' = [s, s', t]$, $\textsf{Translate}(P'') = \emptyset$.

The ℓ-cover is then given by the multiset $\mathcal{P} = \{\textsf{Translate}(P_j') \mid j \in \{1, \ldots, \ell\}\}$. We need the following definition. For $P_j \in \mathcal{P}$, denote $\textsf{Translate}^{-1}(P_j) = P_j'$ where $P_j' \in \widetilde{\mathcal{P}}$. Note that while each path $P_j' \in \widetilde{\mathcal{P}}$ is mapped to a *unique* G-path (given by $\textsf{Translate}(P_j')$), it

might be the case that two distinct paths $P'_i, P'_j \in \widetilde{\mathcal{P}}$ map to the same G-path. For example, $P'_i = [s, s', v_i^{in}, v'_i, v_i^{out}, t]$ and $P'_j = [s, s', v_i^{in}, v_i^{out}, t]$ both translate to the single vertex $\{v_i\}$. Therefore, we consider the $\mathsf{Translate}$ function as a bijection from the *multiset* $\widetilde{\mathcal{P}}$ to the *multiset* $\mathcal{P}$. I.e., each path $P_i \in \mathcal{P}$ is uniquely mapped to a path $P'_i = \mathsf{Translate}^{-1}(P_i)$ in the flow decomposition $\widetilde{\mathcal{P}}$ and vice-versa.

Algorithm PathCover:

Input: An n-vertex DAG G and a parameter $\ell \in \mathbb{N}_{\geq 1}$.
Output: An ℓ-cover $\mathcal{P}$ of G.

1. **Transformation to Min-Cost Max-Flow Instance.** The instance $\widetilde{G} = (\widetilde{V}, \widetilde{E}, u, c)$ with the designated source s and target t vertices are defined as follows.

   - For each vertex $v_i \in V(G)$, include three copies $v_i^{in}, v_i^{out}, v'_i$ in $\widetilde{V}$. These copies are connected by the edges $E_{0,i} = \{(v_i^{in}, v_i^{out}), (v_i^{in}, v'_i), (v'_i, v_i^{out})\}$.
     In addition, add three designated vertices s, s' and t. So overall,

     $$\widetilde{V} = \{v_i^{in}, v_i^{out}, v'_i \mid v_i \in V(G)\} \cup \{s, s', t\} .$$

   - Connect the out-copy v_j^{out} of every incoming neighbor $v_j \in N_{in}(v_i, G)$ to v_i^{in}. In the same manner, connect the in-copy v_k^{in} of every $v_k \in N_{out}(v_i, G)$ to v_i^{out}. Formally,

     $$E_1 = \{(v_j^{out}, v_i^{in}) \mid v_j \in N_{in}(v_i, G)\} \text{ and } E_2 = \{(v_j^{in}, v_i^{out}) \mid v_j \in N_{out}(v_i, G)\} .$$

   - Let $E_3 = \{(s', v_i^{in}) \mid v_i \in V(G)\} \cup \{(v_i^{out}, t) \mid v_i \in V(G)\} \cup \{(s, s'), (s', t)\}$. Then

     $$\widetilde{E} = \bigcup_i E_{0,i} \cup E_1 \cup E_2 \cup E_3.$$

   - The edge capacities $u \in \mathbb{Z}_{\geq 0}^{\widetilde{E}}$ and costs $c \in \mathbb{Z}^{\widetilde{E}}$ are defined by:
     a. $u((s, s')) = \ell$ and $u(v_i^{in}, v_i^{out}) = 1$ for every $v_i \in V$. All remaining edges e in $\widetilde{E}$ have capacity of $u(e) = \ell$.
     b. $c(v_i^{in}, v_i^{out}) = -n^3$ and $c(v_i^{in}, v'_i) = 1$ for every $v_i \in V$. All remaining edges e in $\widetilde{E}$ have cost of $c(e) = 0$.
2. Apply Algorithm $\mathsf{MinCostFlow}(\widetilde{G}, s, t)$ (Theorem 1.12), and let $x \in \mathbb{R}^{|\widetilde{E}|}$ be the output flow.
3. **Flow Decomposition.** Decompose $x \in \mathbb{R}^{|\widetilde{E}|}$ into s-t paths $\widetilde{\mathcal{P}}$ in $\widetilde{G}$ by Lem. 1.14.
4. The ℓ-cover is given by the multiset $\mathcal{P} = \{\mathsf{Translate}(P') \mid P' \in \widetilde{\mathcal{P}}\}$ (where $|\mathcal{P}| = \ell$ as each path corresponds to one unit flow in $\widetilde{G}$).

Analysis. For a path $P' \subseteq \widetilde{G}$, denote the cost of P' by $c(P') = \sum_{e \in P'} c(e)$. We start with observing that any path P' in the flow decomposition $\widetilde{\mathcal{P}}$ must have a non-positive cost. Intuitively, this holds as one can replace P', with the zero-cost path $[s, s', t]$ that has a sufficiently large capacity. Hence we have the following observation.

▶ **Observation 2.4.** *Every path $P' \in \widetilde{\mathcal{P}}$ has a non-positive cost, $c(P') \leq 0$.*

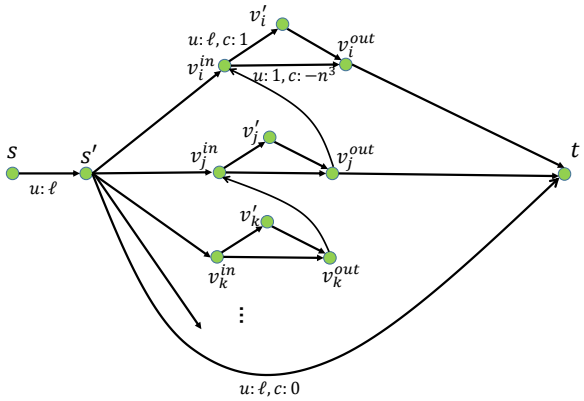


Figure 1 An illustration for the min-cost max-flow reduction. The default values for the edge capacities and costs are ℓ and 0, respectively.

Recall that $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ where P_i might be equal to P_j. A vertex v_i in a path $P_j \in \mathcal{P}$ is denoted as a *principal vertex* of P_j if its corresponding flow path $\mathsf{Translate}^{-1}(P_j) \in \widetilde{\mathcal{P}}$ (i.e., the j^{th} path in $\widetilde{\mathcal{P}}$) contains the edge (v_i^{in}, v_i^{out}). We denote the number of principal vertices in $P_j \in \mathcal{P}$ by $n(P_j)$. Formally,

$$n(P_j) = |\{v_i \in V \ \mid \ (v_i^{in}, v_i^{out}) \in \mathsf{Translate}^{-1}(P_j)\}\ . \tag{1}$$

Since in our flow instance $\widetilde{G}$, the capacity of any edge $e = (v_i^{in}, v_i^{out})$ is 1, i.e., $u(e) = 1$, we have that a vertex $v_i \in V$ can be a principal vertex of at most *one* path in $\mathcal{P}$, hence:

▶ **Observation 2.5.** $\sum_{P_j \in \mathcal{P}} n(P_j) \leq n$.

▶ **Lemma 2.6.** $|\mathcal{P}| = \ell$ *and* $\mathrm{TotLen}(\mathcal{P}) \leq \min(\ell \cdot n, \mathtt{Diam}(G) \cdot n)$.

Proof. The bound of the number of paths simply follows by the fact that x is an integral flow of value ℓ, and by the definition of flow decomposition (Def. 1.13). Furthermore as G is a DAG every path of G is of length at most $n-1$. Thus $\mathrm{TotLen}(\mathcal{P}) \leq \ell \cdot n$. To show that $\mathrm{TotLen}(\mathcal{P}) \leq \mathtt{Diam}(G) \cdot n$ we make the following observation. Consider any $P_j \in \mathcal{P}$ and let u and v be two consecutive principal vertices on P_j, in the sense that the only principal vertices on the segment $P_j[u, v]$ are u and v. We then claim that $|P_j[u, v]| \leq \mathtt{Diam}(G)$.

Assume for the sake of contradiction that $|P_j[u, v]| > \mathtt{Diam}(G)$, we will show that in this case, we can replace the path $Q = \mathsf{Translate}^{-1}(P_j)$ with an alternative path $Q^* \subseteq \widetilde{G}$, that provides a strictly lower cost and the same value of flow, hence leading to a contradiction. That is, in the alternative flow solution x' we will add one unit flow over Q^* and omit one unit flow from Q. Letting $R = [u_0 = u, \ldots, u_k = v]$ be the u-v shortest path in G, then the alternative path Q^* is defined as follows:

$$Q^* = Q[., u^{out}] \circ (u^{out}, u_1^{in}) \circ T_1 \circ (u_1^{out}, u_2^{in}) \circ T_2 \circ (u_2^{out}, u_3^{in}) \circ \ldots \circ T_{k-1} \circ (u_{k-1}^{out}, u_k^{inn}) \circ Q[v^{in}, .]\ ,$$

where $T_j = (u_j^{in}, u_j') \circ (u_j', u_j^{out})$ for every $j \in \{1, \ldots, k-1\}$. Consider the flow solution x' defined by:

$$\begin{cases} x'(e) = x(e) - 1, & \text{for every } e \in Q, \\ x'(e) = x(e) + 1, & \text{for every } e \in Q^*, \\ x'(e) = x(e), & \text{otherwise .} \end{cases}$$

We first show that x' is a legal flow, and with the same value as x. This holds as all edges on the paths of T_j for $j \in \{1, \ldots, k-1\}$, have a sufficiently large capacity. Next, we show that $\sum_e c(e)x'(e) < \sum_e c(e)x(e)$, which will be in contradiction to the optimality of x.

By the definition of u and v, $c(Q[u^{out}, v^{in}]) = |P_j[u,v]| - 1$. In the same manner, $c(Q^*[u^{out}, v^{in}]) = |R| - 1$. Since $|R| < |P_j[u,v]|$, we conclude that $c(Q) > c(Q^*)$, and consequently that $\sum_e c(e)x'(e) < \sum_e c(e)x(e)$ as desired. We therefore have that $|P_j[u,v]| \leq \texttt{Diam}(G)$, and $|P_j| \leq \texttt{Diam}(G) \cdot n(P)$. Combining with Obs. 2.5, we have $\sum_{i=1}^{\ell} |P_i| \leq \texttt{Diam}(G) \cdot \sum_{i=1}^{\ell} n(P_i) \leq \texttt{Diam}(G) \cdot n$. ◀

We next show that each long dipath in G is dominated by the vertices of $V(\mathcal{P})$ (i.e., contains a bounded number of uncovered vertices).

▶ **Lemma 2.7.** *Any dipath in G contains at most $2n/\ell$ vertices of $V \setminus V(\mathcal{P})$.*

Proof. Assume by contradiction that there exists a dipath $Q \subseteq G$ that contains at least $2n/\ell$ vertices of $U = V \setminus V(\mathcal{P})$. Observe that as $|\mathcal{P}| = \ell$, by Obs. 2.5, there exists a path $P^* \in \mathcal{P}$ with $n(P^*) \leq n/\ell$ principal vertices. Therefore $c(\mathsf{Translate}^{-1}(P^*)) \geq -n^3 \cdot n/\ell$ (recall that $c((v_i^{in}, v_i^{out})) = -n^3$ for any $i \in \{1, \ldots, n\}$).

We next show that the path Q implies the existence of an s-t path Q' in $\widetilde{G}$ that has a lower cost than that of $\mathsf{Translate}^{-1}(P^*)$. The path Q' is defined based on $Q = [v_1, v_2, \ldots, v_q]$, as follows:

$$Q' = (s, s') \circ (s', v_1^{in}) \circ T_1 \circ (v_1^{out}, v_2^{in}) \circ T_2 \circ (v_2^{out}, v_3^{in}) \circ \ldots (v_{q-1}^{out}, v_q^{in}) \circ T_q \circ (v_q^{out}, t) \ ,$$

where $T_j = (v_j^{in}, v_j') \circ (v_j', v_j^{out})$ for every $v_j \in V(Q) \setminus U$, and $T_j = (v_j^{in}, v_j^{out}))$ for every $v_j \in V(Q) \cap U$. By the definition of U, $x((v_i^{in}, v_i^{out})) = 0$ for every $v_i \in U$. In addition,

$$c(Q') \leq -n^3 \cdot 2n/\ell + n \leq -n^3 \cdot n/\ell - n^3 + n < c(\mathsf{Translate}^{-1}(P^*)) \ .$$

Consider an alternative flow solution x' obtained by replacing the path P^* with the path Q' in the flow decomposition. That is, let $x'(e) = x(e) - 1$ for every $e \in \mathsf{Translate}^{-1}(P^*)$, and $x'(e) = x(e) + 1$ for every $e \in Q'$. Note that since $x((v_i^{in}, v_i', v_i^{out})) = 0$ for every $v_i \in U$, we have that x' is a legal flow of the same value as that of x, but of a strictly lower cost, as $c(\mathsf{Translate}^{-1}(P^*)) > c(Q')$. Ending with a contradiction to the optimality of x. ◀

2.2 Partial ℓ-Covers

The construction of shortcut sets of sublinear size of Theorem 1.5 calls for a variant of the ℓ-covers that we call *partial ℓ-covers*. These covers are defined w.r.t. a given subset V' of vertices that we wish to cover, in the following sense:

▶ **Definition 2.8** (Partial ℓ-Covers). *Given an n-vertex digraph $G = (V, E)$, a subset $V' \subseteq V$, and a parameter ℓ, a* partial ℓ-cover *for G w.r.t. V' is a multiset of ℓ paths $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ (possibly, singletons or empty), satisfying the following:*

1. $\text{TotLen}(\mathcal{P}) \leq 8|V(\mathcal{P}) \cap V'|$,
2. *For any long dipath $P \subseteq G$, i.e., such that $|P| \geq 8|V' \cap V(\mathcal{P})|/\ell$, it holds that*

$$|V(P) \cap (V' \setminus V(\mathcal{P}))| \leq |V(P)|/4 \ .$$

▶ **Theorem 2.9.** *For any given n-vertex m-edge DAG $G = (V, E)$, $\ell \in [1, n]$ and $V' \subseteq V$, there is a randomized algorithm $\mathsf{PartialPathCover}(G, V', \ell)$ for computing an ℓ-partial cover of G w.r.t. V', w.h.p, in time $\widetilde{O}(m + n^{3/2})$.*

Algorithm PartialPathCover. The algorithm is almost equivalent to Algorithm PathCover, up to small adaptations of the min-cost max flow instance. Specifically, we define $\widetilde{G} = (\widetilde{V}, \widetilde{E}, u', c')$, with the same edge and vertex set as in Algorithm PathCover. Letting, u, c be the capacity (resp., cost) functions of Alg. PathCover, then in our reduction, we define: $c'((v_i^{in}, v_i^{out})) = -7$ for every $v_i \in V$, and $c'(e) = c(e)$ for every other edge in $\widetilde{G}$. In addition, $u'((v_i^{in}, v_i^{out})) = 0$ for every $v_i \in V \setminus V'$, and $u'(e) = u(e)$ for every other edge in $\widetilde{G}$.

The remaining algorithm is identical as Algorithm PathCover. It applies Algorithm MinCostFlow$(\widetilde{G}, s, t)$ using Theorem 1.12. The output flow solution is given by x. Then, it computes the flow-decomposition $\widetilde{\mathcal{P}}$ by applying Lemma 1.14 on x. Finally, the paths $\widetilde{\mathcal{P}}$ are translated into a *multiset* of G-paths $\mathcal{P}$, which provides the output partial ℓ-cover w.r.t. V'. The function Translate provides the bijection from the multiset $\widetilde{\mathcal{P}}$ to the multiset $\mathcal{P}$.

3 Shortcut Algorithms via Path Covers

3.1 Shortcutting to Diameter $D = O(n^{1/2})$

We start by proving Theorem 1.3, by presenting an algorithm for computing linear-size D-shortcuts for $D = O(\sqrt{n})$. This algorithm will be used later on in order to provide improved diameter bounds. By Proposition 1.11, it is sufficient to prove Theorem 1.4 for DAGs. Before presenting the improved shortcut algorithm FasterShortcutSqrtN, we also need the following definition. For a given directed path $P = [u_1, \ldots, u_k]$ and a vertex v, let u_i be the *first* vertex on P satisfying that $(v, u_i) \in TC(G)$. The edge (v, u_i) is denoted as *the first incoming edge* from v to P, represented by $e(v, P) = (v, u_i)$. Note that the augmented path $P \cup \{e(v, P)\}$ provides a directed path from v to every vertex $u \in P$ such that $(v, u) \in TC(G)$. The following result provided in [20].

▶ **Lemma 3.1** (Restatement of Lemma 2 in [20]). *Let $\mathcal{Q}$ be a collection of directed paths in a DAG. Then, one can compute the edge set $\{e(v, Q) \mid v \in V, Q \in \mathcal{Q}\}$ within $O(|\mathcal{Q}| \cdot m)$ time.*

Algorithm FasterShortcutSqrtN:

Input: An n-vertex DAG G,
Output: A shortcut set $H \subseteq TC(G)$ such that $|E(H)| = \widetilde{O}(n)$ and $\mathtt{Diam}(G \cup H) = O(\sqrt{n})$.

1. Compute ℓ-cover $\mathcal{P} = \mathsf{PathCover}(G, \ell)$, for $\ell = \Theta(\sqrt{n})$ (using Theorem 2.3).
2. Let $\mathcal{C} = \mathsf{DisjointChains}(\mathcal{P})$ (using Lemma 1.15).
3. For every $C_i \in \mathcal{C}$, let $H_i = H(C_i)$ be a shortcut set for reducing the diameter of C_i to 2 as obtained by Lemma 1.10.
4. Output $H = \bigcup_{C_i \in \mathcal{C}} C_i \cup H_i$.

▶ **Lemma 3.2** (Runtime and Size). *Algorithm* FasterShortcutSqrtN *can be implemented in time $\widetilde{O}(m + n^{3/2})$. In addition, $E(H) = \widetilde{O}(n)$, w.h.p.*

Finally, we complete the diameter argument and establish Theorem 1.3.

▶ **Lemma 3.3.** *The diameter of $G \cup H$ is at most $O(\sqrt{n})$, w.h.p.*

3.2 Shortcutting to Diameter $D = o(n^{1/2})$

In this subsection, we extend the algorithm of Sec. 3.1 to provide D-shortcuts for $D = o(n^{1/2})$. Obtaining a smaller diameter bound will entitle a cost both in terms of the running time and in the size of the shortcut set. Again, we assume that G is a DAG.

Algorithm FasterShortcutSmallDiam:

Input: An n-vertex DAG G and a diameter bound $D = o(n^{1/2})$.
Output: A shortcut set $H \subseteq TC(G)$ satisfying that $\texttt{Diam}(G \cup H) \leq D$.

1. Let $H_0 \leftarrow \mathsf{FasterShortcutSqrtN}(G)$, (using Theorem 1.3).
2. Compute ℓ-cover $\mathcal{P} = \mathsf{PathCover}(G \cup H_0, \ell)$ for $\ell = 16n/D$, (using Theorem 2.3).
3. Let $\mathcal{C} = \mathsf{DisjointChains}(\mathcal{P})$ (using Lemma 1.15).
4. For every $C_i \in \mathcal{C}$, let $H(C_i)$ be a 2-shortcut set for C_i obtained by Lemma 1.10.
5. Let $V' = V[p]$ and $\mathcal{C}' = \mathcal{C}[p]$ where $p = \Theta(\log n/D)$.
6. Applying Lemma 3.1 to compute $\widehat{H} = \{e(v, C_i) \mid v \in V', C_i \in \mathcal{C}'\}$.
7. Output $H = H_0 \cup \bigcup_{C_i \in \mathcal{C}}(C_i \cup H(C_i)) \cup \widehat{H}$.

▶ **Lemma 3.4** (Runtime and Size). *Algorithm* FasterShortcutSmallDiam *can be implemented in time* $\widetilde{O}(m \cdot n/D^2 + n^{3/2})$. *In addition,* $E(H) = \widetilde{O}(n^2/D^3)$, *w.h.p.*

▶ **Lemma 3.5** (Diameter Bound). $\texttt{Diam}(G \cup H) \leq D$, *w.h.p.*

Consider a u-v shortest path P in $G' = G \cup H_0 \cup \bigcup(C_i \cup H(C_i))$. Let P', P'' be the $(D/4)$-length prefix (resp., suffix) of P.

▶ **Lemma 3.6.** P'' *contains at most* $D/8$ *vertices from* $V \setminus V(\mathcal{C})$.

By the exact same argument as in the proof of Lemma 3.3, we also have:

▶ **Observation 3.7.** $|V(P'') \cap V(C_i)| \leq 3$ *for every* $C_i \in \mathcal{C}$.

Therefore by Lemma 3.6, P'' contains representatives vertices from $\Omega(D)$ distinct paths in $\mathcal{C}$. That is, $|\{C_i \in \mathcal{C} \mid V(C_i) \cap V(P'') \neq \emptyset\}| = \Omega(D)$. As each chain in $\mathcal{C}$ is sampled into $\mathcal{C}'$ independently with probability p, by the Chernoff bound, we have that w.h.p., P'' contains at least one vertex $y \in C_i$ for some sampled chain $C_i = [a_1, \ldots, a_k] \in \mathcal{C}'$. In addition, w.h.p., the prefix P' contains at least one sampled vertex $x \in V'$. We then have that the edge $e(x, C_i)$ is in $\widehat{H} \subseteq H$. Let $e(x, C_i) = (x, z)$ for $z \in C_i[a_1, y]$. Therefore the augmented graph $G \cup H$ contains a u-v path $Q = P[u, x] \circ (x, z) \circ C_i[z, y] \circ P[y, v]$ where

$$\begin{aligned} \text{dist}_{G \cup H}(u, v) &\leq |Q| = |P[u, x]| + 1 + |C_i[z, y]| + |P[y, v]| \\ &\leq |P'| + 1 + \text{dist}_{C_i \cup H(C_i)}(z, y) + |P''| \\ &\leq D/4 + 1 + 2 + D/4 \leq D/4 + D/4 + 3 \leq D \ . \end{aligned}$$

This concludes the proof of Theorem 1.4.

3.3 Shortcut Algorithms for Diameter $D = \Omega(n^{1/3})$

In this section, we provide a proof for Theorem 1.5 and compute D-shortcuts of sublinear size for any input $D = \omega(n^{1/3})$. We start by showing that in this case, as well, one can assume that the input graph G in Theorem 1.5 is a DAG. The standard reduction to a DAG is based on adding $\Theta(n)$ shortcut edges, thus too costly for our setting. Then, we present an algorithm that given a collection of dipaths $\mathcal{P}$ computes a D'-shortcut for each

path such that the total size of the union of shortcuts in *sublinear*. Again, as we aimed at obtaining sublinear shortcuts we cannot simply use Lemma 1.10 as is. Finally, we present Alg. FasterShortcutLargeD that computes the shortcuts given these tools.

Tool 1: Reduction to DAGs with Sublinear Size. In the full version, we show:

▶ **Theorem 3.8.** *Given a digraph $G = (V, E)$, one can compute in time $O(|V| + |E|)$ a DAG $G' = (V', E')$ such that $|V'| = O(|V|)$ and $|E'| = O(|E|)$, and in addition the following holds: Given any D-shortcut set $H' \subseteq TC(G')$ for G', one can compute in time $O(|H'|)$ a D-shortcut set $H \subseteq TC(G)$ for G such that $|H| \leq |H'|$.*

Tool 2: Shortcutting Dipaths with Sublinear Number of Edges. A crucial tool in our prior constructions is based on computing 2-shortcuts of nearly linear size for dipaths. For our purposes, we next provide an alternative scheme that provides shortcuts of sublinear size at the cost of increasing the diameter.

▶ **Lemma 3.9** (Shortcuts of Dipaths with Sublinear Size)**.** *There exists an algorithm* PathShortcut *that given a DAG G, dipath collection $\mathcal{P}$ and integer parameter d, outputs $H \subseteq TC(G)$ satisfying that for all $1 \leq i \leq t$, we have $\texttt{Diam}(P_i \cup H[V(P_i)]) = O(d)$ and $|H| = \widetilde{O}(\sum_{j=1}^{t} |P_i|/d)$. The running time is $\widetilde{O}(\sum_{P \in \mathcal{P}} |V(P|)$.*

The Shortcut Algorithm. We are now ready to present Algorithm FasterShortcutLargeD. The algorithm is given as input a DAG G and diameter bound $D = \omega(n^{1/3})$. Define:

$$t_D = \sqrt{n/D} \quad \text{and} \quad s_D = D^{3/2}/\sqrt{n} \,. \tag{2}$$

The algorithm consist of $i^* \leq t_D$ iterations (as will be shown in the analysis), where in each iteration i, the algorithm computes a partial t_D-cover $\mathcal{P}_i$ with respect to the set $V_i = V \setminus \bigcup_{j=0}^{i-1} \mathcal{P}_i$, namely, the set of vertices that are not yet covered by the current collection of paths $\bigcup_{j=0}^{i-1} \mathcal{P}_i$. It then computes an s_D-shortcut $H(P)$ for each $P \in \mathcal{P}_i$ by applying Algorithm PathShortcut of Lemma 3.9, and adds $\bigcup_{P \in \mathcal{P}_i} H(P)$ to the output shortcut H.

At the end of these iterations, we obtain $i^* \leq t_D$ path collections $\mathcal{P}_0, \ldots, \mathcal{P}_{i^*-1}$, where $|\mathcal{P}_{j-1}| \leq t_D$ for every $j \in \{1, \ldots, i^*\}$. The algorithm then defines a collection of $\Theta(t_D \log n)$ paths $\mathcal{R}$ as follows. The dipaths of the last path collection, namely, $\mathcal{P}_{i^*-1}$ is taken *entirely* into $\mathcal{R}$. In addition, each other dipath in $\bigcup_{j=1}^{i^*-2} \mathcal{P}_j$ is sampled into $\mathcal{R}$ independently with probability $p = \Theta(\log n / t_D)$. By the Chernoff bound we get that, w.h.p., $|\mathcal{R}| = \Theta(t_D \log n)$ as desired. The algorithm then adds to H, the collection of $e(v, P)$ edges for every $v, P \in V[p'] \times \mathcal{R}$ where $p' = \Theta(\log n / D)$. This completes the description of the algorithm.

Algorithm FasterShortcutLargeD:

Input: An n-vertex DAG $G = (V, E)$ and a diameter bound $D = \omega(n^{1/3})$.
Output: A shortcut set $H \subseteq TC(G)$ satisfying that $\texttt{Diam}(G \cup H) = O(D)$ and $|H| = \widetilde{O}((n/D)^{3/2})$.

1. Set $V_0 = V$, $U_0 = V$, $i = 0$, $p' = 10 \log n \cdot D^{-1}$ and $p = 10 \log n \cdot t_D^{-1}$.
2. While $|U_i| \geq \frac{n}{t_D}$ do the following:
   a. Set $\mathcal{P}_i = \text{PartialPathCover}(G, V_i, t_D)$.
   b. Set $H_i = \text{PathShortcut}(\mathcal{P}_i, s_D)$.
   c. $U_{i+1} \leftarrow V(\mathcal{P}_i) \cap V_i$, $V_{i+1} \leftarrow V_i \setminus V(\mathcal{P}_i)$, and $i \leftarrow i + 1$.

3. Set $\mathcal{R} = \bigcup_{j=0}^{i-2} \mathcal{P}_j$, $\mathcal{R}' = \mathcal{R}[p] \cup \mathcal{P}_{i-1}$ and $V' = V[p']$.
4. $\widehat{H} = \{e(v, P_i) \mid v \in V', P_i \in \mathcal{R}'\}$ (using Lemma 3.1).
5. Output $H = \bigcup_{j=0}^{i-1} H_i \bigcup \widehat{H}$.

Analysis. Recall that the index i^* indicates the largest index of the application, where $|U_{i^*}| < n/t_D$ leading to the termination of the algorithm.

▶ **Observation 3.10.** $V = \bigcup_{j=0}^{i^*-1} V(\mathcal{P}_j) \cup V_{i^*}$.

▶ **Observation 3.11.** *For every $Q \in \bigcup_{j=0}^{i^*-1} \mathcal{P}_j$, for every u-v shortest path $P \subseteq G \cup \bigcup_{j=0}^{i^*-1} H_j$ it holds that $|V(Q) \cap V(P)| = O(s_D)$.*

▶ **Lemma 3.12** (Runtime and Size). *Algorithm* FasterShortcutLargeD *can be implemented in time $\widetilde{O}((m + n^{3/2}) \cdot t_D)$. In addition, $|E(H)| = \widetilde{O}((n/D)^{3/2})$, w.h.p.*

Proof. We first claim that the While loop is applied at most t_D times. By definition, $U_j \cap V_j = \emptyset$ and $U_j \cup V_j = V_{j-1}$. In addition, since $U_j \subseteq V_{j-1}$, it holds that $U_1, U_2, \ldots, U_i$ are vertex-disjoint. Since the While loop continues only provided that $|U_i| \geq n/t_D$, there can be at most t_D applications of this loop. We will need the following observation.

▶ **Observation 3.13.** $\text{TotLen}(\mathcal{R} \cup \mathcal{P}_{i^*-1}) = O(n)$.

Runtime. We focus on a single application of the While loop and show that it can be implemented in $\widetilde{O}(m + n^{3/2})$ time, since there are at most t_D applications, the final runtime is bounded by $\widetilde{O}((m + n^{3/2})t_D)$. Step (a) is implemented in $\widetilde{O}(m + n^{3/2})$ time by Theorem 2.9. Step (b) runs in $\widetilde{O}(\text{TotLen}(\mathcal{R} \cup \mathcal{P}_{i-1})) = \widetilde{O}(n)$, by Lemma 3.9 and Observation 3.13 (we note that in fact this is the total running time of Step (b) over all the iterations of the While loop). Step (c) runs in $O(n)$ time. The runtime bound of the t_D applications of the While loop follows. We next analyze the remaining steps of the algorithm. By Observation 3.13, Step 3 is implemented in $O(n)$ time. Since $|\mathcal{R}'| = |\mathcal{R}[p]| + |\mathcal{P}_{i-1}|$, w.h.p., we have that $|\mathcal{R}'| = \widetilde{O}(t_D)$ as $|\mathcal{R}[p]| = \widetilde{O}(t_D)$ (this follows from the fact that $|\mathcal{R}| \leq t_D^2$ as there are at most t_D iterations of line 2(a) and at each iterations at most t_D paths are created) and $|\mathcal{P}_{i-1}| \leq t_D$. Using Lemma 3.1, Step 4 is implemented in $\widetilde{O}(m \cdot t_D)$.

Size. By the Chernoff bound, w.h.p., it holds that $|\widehat{H}| = |V'| \cdot |\mathcal{R}'| = |V'| \cdot (|\mathcal{R}[p]| + |\mathcal{P}_{i^*-1}|) = \widetilde{O}\left(\frac{n}{D} \cdot (t_D + t_D)\right) = \widetilde{O}((n/D)^{3/2})$. Furthermore, we also have $\sum_{j=0}^{i^*-1} |H_j| = O((n/D)^{3/2})$. This follows from the fact proven in Observation 3.13 that $\sum_{P \in \mathcal{R} \cup \mathcal{P}_{i^*-1}} |V(P)| = O(n)$. Hence, by Lemma 3.9 and Step 2(b) of the algorithm, w.h.p., we have that

$$\sum_{j=0}^{i^*-1} |H_j| = \widetilde{O}\left(s_D^{-1} \cdot \sum_{P \in \mathcal{R} \cup \mathcal{P}_{i-1}} |V(P)|\right) = \widetilde{O}(s_D^{-1} \cdot n) = \widetilde{O}((n/D)^{3/2}).$$

We conclude that $|H| = |\bigcup_{j=0}^{i^*-1} H_j \bigcup \widehat{H}| = \widetilde{O}((n/D)^{3/2})$, w.h.p. ◀

▶ **Lemma 3.14** (Diameter Bound). $\texttt{Diam}(G \cup H) = O(D)$, *w.h.p.*

Proof. Consider a u-v shortest dipath P in the graph $G' = \bigcup_{j=0}^{i^*-1} H_j \cup G$. Let P', P'' be the $8D$-length prefix and suffix of P, respectively. By the Chernoff bound, it holds that $V(P') \cap V' \neq \emptyset$, and let u^* be some arbitrary vertex in $V(P') \cap V'$. Recall that $\mathcal{P}_{i^*-1}$ is the last path collection obtained by applying Alg. PartialPathCover We next consider two cases.

Case 1: $V(P'') \cap V(\mathcal{P}_{i^*-1}) \neq \emptyset$. Let v^* be some vertex in $V(P'') \cap V(\mathcal{P}_{i^*-1})$ and let $Q \in \mathcal{P}_{i^*-1}$ be some path satisfying that $v^* \in V(Q)$. Since $Q \in \mathcal{R}'$, the edge $e(u^*, Q)$ is in $\widehat{H}$, and thus also in the output shortcut H. Let $e(u^*, Q) = (u^*, z)$ where z is the first vertex on Q such that there is a dipath from u^* to z in G. We therefore have that:

$$\begin{aligned} \mathrm{dist}_{G \cup H}(u, v) &\leq \mathrm{dist}_{G'}(u, u^*) + \mathrm{dist}_{G' \cup \widehat{H}}(u^*, z) + \mathrm{dist}_{G \cup H_{i^*-1}}(z, v^*) + \mathrm{dist}_{G'}(v^*, v) \quad (3) \\ &\leq |P'| + 1 + O(s_D) + |P''| = O(D) \ . \quad (4) \end{aligned}$$

This holds as $\mathrm{dist}_{G \cup H_{i-1}}(z, v^*) = O(s_D)$ by Lemma 3.9 and $s_D = D \cdot \sqrt{D/n} \leq D$.

Case 2: $V(P'') \cap V(\mathcal{P}_{i^*-1}) = \emptyset$. We will need the following observations.

- Observation 1: $V(P'') \subseteq \bigcup_{j=0}^{i^*-2} \mathcal{P}_j \cup V_{i^*}$. This observation follows from Obs. 3.10, as $V(P'') \cap V(\mathcal{P}_{i^*-1}) = \emptyset$.
- Observation 2: $8|U_{i^*}|/t_D \leq 8D$. This observation follows from the fact that $|U_{i^*}| \leq n/t_D$.

Hence by property (2) of Definition 2.8 of the partial ℓ-cover $\mathcal{P}_{i^*-1}$ w.r.t. V_{i^*-1} (with $\ell = t_D$), we have that:

$$\begin{aligned} |V(P'') \cap V_{i^*-1}| &= |V(P'') \cap V_{i^*}| && \text{By Observation 1.} \\ &\leq |V(P'')|/4 && \text{By property (2) of Definition 2.8 and Observation 2.} \end{aligned}$$

By Obs. 3.11, it holds that $|V(Q) \cap V(P'')| = O(s_D)$ for every $Q \in \bigcup_{j=0}^{i^*-2} \mathcal{P}_j$. Therefore, P'' must contain vertices from at least $\Omega(D/s_D) = \Omega(t_D)$ distinct paths in $\mathcal{P}' = \bigcup_{j=0}^{i^*-2} \mathcal{P}_j$. Formally, let $\mathcal{X} = \{Q \in \mathcal{P}' \mid V(Q) \cap V(P'') \neq \emptyset\}$, then $|\mathcal{X}| = \Omega(t_D)$.

Since each path in $\mathcal{X}$ is sampled into $\mathcal{R}'$ independently with probability of $p = \Theta(\log n \cdot t_D^{-1})$, by the Chernoff bound, w.h.p., $|\mathcal{X} \cap \mathcal{R}'| \neq \emptyset$. Letting $Q^* \in \mathcal{X} \cap \mathcal{R}'$, we have that there is a vertex $y \in V(Q^*) \cap V(P'')$ and in addition, the edge $e(u^*, Q^*) = (u^*, z)$ is in H, where z is the first vertex on Q^* from which there is an incoming path from u^*. Altogether by the same argument as in Case 1, we have that $\mathrm{dist}_{G \cup H}(u, v) = O(D)$. ◀

Due to lack of space, the applications of our ℓ-covers and shortcut constructions (Theorems 1.6, 1.8) appear in the full version.

References

1. Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.
2. Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Faster sparse minimum cost flow by electrical flow localization. *CoRR*, abs/2111.10368, 2021. `arXiv:2111.10368`.
3. Piotr Berman, Sofya Raskhodnikova, and Ge Ruan. Finding sparser directed spanners. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 424–435. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.
4. Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental SSSP in dense weighted digraphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1112–1122. IEEE, 2020.

5 Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I. Tomescu. A linear-time parameterized algorithm for computing the width of a DAG. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Graph-Theoretic Concepts in Computer Science - 47th International Workshop, WG 2021, Warsaw, Poland, June 23-25, 2021, Revised Selected Papers*, volume 12911 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2021.

6 Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I Tomescu. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. *arXiv preprint arXiv:2107.05717*, 2021.

7 Yangjun Chen and Yibin Chen. An efficient algorithm for answering graph reachability queries. In Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen, editors, *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico*, pages 893–902. IEEE Computer Society, 2008. doi:10.1109/ICDE.2008.4497498.

8 Yangjun Chen and Yibin Chen. On the graph decomposition. In *2014 IEEE Fourth International Conference on Big Data and Cloud Computing, BDCloud 2014, Sydney, Australia, December 3-5, 2014*, pages 777–784. IEEE Computer Society, 2014. doi:10.1109/BDCloud.2014.118.

9 RP Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, pages 161–166, 1950.

10 Jeremy T. Fineman. Nearly work-efficient parallel algorithm for digraph reachability. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1197850.

11 Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 686–697. IEEE Computer Society, 2018.

12 Fabrizio Grandoni, Giuseppe F. Italiano, Aleksander Lukasiewicz, Nikos Parotsidis, and Przemyslaw Uznanski. All-pairs LCA in dags: Breaking through the $O(n^{2.5})$ barrier. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 273–289. SIAM, 2021. doi:10.1137/1.9781611976465.18.

13 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2542–2561. SIAM, 2020.

14 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Improved algorithms for decremental single-source reachability on directed graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 725–736. Springer, 2015.

15 William Hesse. Directed graphs requiring large numbers of shortcuts. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 665–669. ACM/SIAM, 2003.

16 Shang-En Huang and Seth Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, volume 101 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

17 H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Trans. Database Syst.*, 15(4):558–598, 1990. doi:10.1145/99935.99944.

18 Philip N. Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *J. Algorithms*, 25(2):205–220, 1997.

19 Shimon Kogan and Merav Parter. New diameter-reducing shortcuts and directed hopsets: Breaking the $\sqrt{n}$ barrier. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2022*, 2022.

20 Anna Kuosmanen, Topi Paavilainen, Travis Gagie, Rayan Chikhi, Alexandru I. Tomescu, and Veli Mäkinen. Using minimum path cover to boost dynamic programming on dags: Co-linear chaining extended. In Benjamin J. Raphael, editor, *Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018, Paris, France, April 21-24, 2018, Proceedings*, volume 10812 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 2018.

21 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433. IEEE Computer Society, 2014.

22 Yang P. Liu, Arun Jambulapati, and Aaron Sidford. Parallel reachability in almost linear work and square root depth. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1664–1686. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00098.

23 Veli Mäkinen, Alexandru I. Tomescu, Anna Kuosmanen, Topi Paavilainen, Travis Gagie, and Rayan Chikhi. Sparse dynamic programming on dags with small width. *ACM Trans. Algorithms*, 15(2):29:1–29:21, 2019. doi:10.1145/3301312.

24 Leon Mirsky. A dual of dilworth's decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971.

25 Sofya Raskhodnikova. Transitive-closure spanners: A survey. In Oded Goldreich, editor, *Property Testing - Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*, pages 167–196. Springer, 2010. doi:10.1007/978-3-642-16367-8_10.

26 Mikkel Thorup. On shortcutting digraphs. In Ernst W. Mayr, editor, *Graph-Theoretic Concepts in Computer Science, 18th International Workshop, WG '92, Wiesbaden-Naurod, Germany, June 19-20, 1992, Proceedings*, volume 657 of *Lecture Notes in Computer Science*, pages 205–211. Springer, 1992.

27 Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. doi:10.1137/0220006.

28 Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. *CoRR*, abs/2112.00722, 2021. arXiv:2112.00722.

29 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 859–869. ACM, 2021. doi:10.1145/3406325.3451108.

Monotone Arithmetic Complexity of Graph Homomorphism Polynomials

Balagopal Komarath
Indian Institute of Technology Gandhinagar, India

Anurag Pandey
Department of Computer Science, Universität des Saarlances, Saarland Informatics Campus, Saarbrücken, Germany

Chengot Sankaramenon Rahul
School of Mathematics and Computer Science, Indian Institute of Technology Goa, India

Abstract

We study homomorphism polynomials, which are polynomials that enumerate all homomorphisms from a pattern graph H to n-vertex graphs. These polynomials have received a lot of attention recently for their crucial role in several new algorithms for counting and detecting graph patterns, and also for obtaining natural polynomial families which are complete for algebraic complexity classes VBP, VP, and VNP. We discover that, in the monotone setting, the formula complexity, the ABP complexity, and the circuit complexity of such polynomial families are exactly characterized by the treedepth, the pathwidth, and the treewidth of the pattern graph respectively.

Furthermore, we establish a single, unified framework, using our characterization, to collect several known results that were obtained independently via different methods. For instance, we attain superpolynomial separations between circuits, ABPs, and formulas in the monotone setting, where the polynomial families separating the classes all correspond to well-studied combinatorial problems. Moreover, our proofs rediscover fine-grained separations between these models for constant-degree polynomials.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Computational complexity and cryptography; Theory of computation → Circuit complexity; Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Homomorphism polynomials, Monotone complexity, Algebraic complexity, Graph algorithms, Fine-grained complexity, Fixed-parameter algorithms and complexity, Treewidth, Pathwidth, Treedepth, Graph homomorphisms, Algebraic circuits, Algebraic branching programs, Algebraic formulas

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.83

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.04778`

Acknowledgements We thank Christian Engels and Marc Roth for bringing missing and incorrect references to our attention. We thank Markus Bläser for valuable discussions. We also thank anonymous reviewers for their valuable comments on the draft that helped improve the presentation of our results.

1 Introduction

This work is a culmination of exploration of four themes in combinatorics, algorithm design, and algebraic complexity – graph algorithms, homomorphism polynomials, graph parameters, and monotone computations. While each of these themes are of independent interest, a strong interplay among them has become quite apparent in recent years, and has lead to several new advancements in algorithms and complexity.

The first theme is *graph algorithms*, where the algorithms that are relevant to this work are those corresponding to pattern detection and counting. Loosely speaking, in such problems, we look for the "presence" of a graph H, called the *pattern graph* in another graph G, called the *host graph*[1]. The notions of presence of one graph in another graph that have been the most prevalent are subgraph isomorphism, induced subraph isomorphism, and homomorphism. In detection and counting algorithms for subgraph isomorphism (resp. induced subgraph isomorphism), we want to detect and count subgraphs (resp. induced subraphs) of an n-vertex host graph which is *isomorphic* to the pattern graph. Whereas, while looking for the occurrence of the pattern graph in the host graph, if we relax the mapping to allow multiple vertices in the pattern graph to be mapped to a single vertex in the host graph, while preserving the edge relations, we get a *homomorphism* of the pattern graph in the host graph. When the notion is homomorphism, we are interested in detecting and counting homomorphisms from the pattern graph to the host graph.

All the above mentioned problems have found many applications, both in theory and practice. For instance, detecting and counting (induced) subgraph isomorphisms are used in extremal graph theory in the study of dense graphs and quasirandom graphs [10, 29, 30], and in many applications which boil down to analyzing real-world graphs. This includes finding protein interactions in biomolecular networks [1, 34], finding active chemicals of interest in the research for drug synthesis [4], advertisement targeting by finding and counting certain social structure patterns in the analysis of social networks [24, 46], and also finding user patterns for recommender services on platforms like Amazon and Yelp [47]. The homomorphism counting problem appears for instance, in statistical physics via partition functions, in graph property testing, and extremal graph theory (see [5] for a survey). When both the host graph and the pattern graph are parts of the input, then it can be shown that all these pattern detection problems are NP-complete, since they generalize the CLIQUE problem, whereas the corresponding counting problems are all known to be #P-complete. On the other hand, in almost all the real-world applications pointed out above, we have a fixed pattern graph which we are trying to detect or count in a given host graph. Thus, in this work, we focus on the setting when the pattern graph is a fixed graph, and the host graph is a part of the input. Here, in the word-RAM model with $O(\log(n))$-bit words, since the pattern graph is of a fixed size, say its number of vertices is k, all the above problems can be solved using a trivial algorithm, based on exhaustive search, that runs in time $O(n^k)$ and space $O(1)$ where n is the number of vertices in the host graph. However, in almost all the real-world applications pointed out above, the host graph is massive. Hence, one desires faster algorithms, preferably linear or even sub-linear algorithms. Hence, there is a lot of interest and advances in improving upon this trivial algorithm using ideas from combinatorics, algebraic circuits, and machine learning (see [33] for a survey, also see [3, 28] and references therein). These problems are also very interesting from the perspective of complexity theory. For instance, in the Fixed Parameter Tractability community, it is conjectured that the best known algorithms for variants of these pattern detection and counting problems cannot be further improved. If true, this would imply $\mathsf{P} \neq \mathsf{NP}$ in a rather strong way (see [27, 32, 38, 26]). Recently, it was discovered by Curticapean, Dell and Marx [11] (also see [13]) that counting subgraph isomorphisms corresponds to counting linear combinations of homomorphisms, and hence establishing that it is sufficient to just consider the homomorphism counting problem. Several advances have been attained by considering homomorphism problems [3, 23, 25, 17, 44].

[1] In this paper, the pattern graph and the host graph are always simple, undirected graphs.

Our second theme – *homomorphism polynomials*, corresponds to one of the most successful ways by which progress has been made towards homomorphism problems, and hence towards (induced) subgraph isomorphism problems too. For a pattern graph H and a host graph G, the *homomorphism polynomial* is the polynomial whose monomials enumerate homomorphisms from H to G. For using homomorphism polynomials to obtain new algorithms for the above graph problems for a pattern graph H, it suffices to consider the homomorphism polynomial from H to K_n, the clique on n vertices. Thus, we call the homomorphism polynomial from H to K_n as the *homomorphism polynomial of* H (see Definition 10). It turns out that efficient *arithmetic circuit* (see Section 2 for the definition) constructions of homomorphism polynomials can be used to obtain almost all known best algorithms for detecting induced subgraph isomorphisms as well (See [3], also [23, 25, 17, 44]). Therefore, the study of homomorphism polynomials is a crucial area of study within graph pattern detection and counting. Homomorphism polynomials also turned out to be extremely useful in algebraic complexity theory in the quest for finding natural complete polynomial families for the complexity class VP (the algebraic complexity analog of P. See [39]). Homomorphism polynomials yield polynomial families that are complete for important algebraic complexity classes such as VBP, VP, and VNP through a single framework, simply by considering different pattern graphs [15, 31, 9], making it important not only from the perspective of algorithm design, but also from the perspective of complexity theory (also see [16], where homomorphism polynomials yield several dichotomy theorems in algebraic complexity theory). Homomorphism polynomials are thus the central focus of this work too.

In our third theme, that is of *graph parameters*, the parameters relevant to this work are treewidth, pathwidth, and treedepth of graphs. Treewidth, loosely speaking, is a measure of how far a graph is from being a tree; similarly the pathwidth measures how far a graph is from being a path; and treedepth measures how far a graph is from being a star (see Section 2 for definitions). The connection between the parameters treewidth and pathwidth and counting homomorphisms was first explored in [13] (also see [3, 31]), where it is shown that when H has bounded treewidth, then there are small-sized arithmetic circuits for homomorphism polynomials, whereas when H has small pathwidth, then the corresponding homomorphism polynomials have small-sized *algebraic branching programs* (ABPs, defined in Section 2). These improved constructions for circuits and ABPs are one of the main source of advancement in finding improved algorithms for these pattern detection and counting problems [3, 23, 25, 17, 44]. The connection between the subgraph isomorphism problem and treedepth has also been explored in the context of Boolean computational models in [27] and [26], where they discuss treewidth and treedepth based upper bounds and lower bounds for counting subgraph isomorphisms for Boolean circuits and formulas. In the context of parameterized counting complexity of these problems, a stronger connection between treewidth and the complexity of counting homomorphisms was established by Dalmau and Jonsson [12] who showed a dichotomy theorem stating that when the pattern graph has bounded treewidth, then we can count homomorphisms in polynomial time. Otherwise, we can show #W[1]-hardness. Now we turn our attention to the other major application of homomorphism polynomials. It turns out that the parameters treewidth and pathwidth played a crucial role in the construction of natural complete polynomials for $\mathsf{VP}, \mathsf{VBP}$, and VNP too. More specifically, the complete polynomials for VP and VBP were homomorphism polynomials corresponding to specific pattern graphs of bounded treewidth and bounded pathwidth respectively. Thus, one sees that in the context of these pattern detection and counting problems, and in the homomorphism polynomials of interest, these graph parameters of the pattern graph ubiquitously pop up over and over again as the crucial complexity parameters. This made us wonder, to what extent do these parameters dictate the complexity of these problems and, in particular, the homomorphism polynomials.

Towards this, the starting point of our work is the observation that all the improved arithmetic circuit constructions of homomorphism polynomials based on treewidth and pathwidth, do not use any negative constants in the circuit. This brings us to our final theme, that is, of *monotone computation*.

In the Boolean setting, monotone computations refer to computations that do not involve the negation operation, the NOT gate. Similarly, in the algebraic setting for computing polynomials, monotone arithmetic circuits correspond to circuits that do not use negative constants or subtraction gates, that is, there is no cancellation involved in the circuit. Monotone computations are interesting for three main reasons. First reason is that monotone operations have favorable stability properties with respect to rounding errors [40]. Secondly, an improved monotone circuit requires combinatorial insights since it does not use any cancellations and algebraic identities. Such constructions often reveal interesting combinatorial structure about the problem at hand and lead to new combinatorial algorithms. for the problem at hand. For instance, Grenet's monotone algebraic branching program construction of the permanent polynomial [19]. Finally, the weakness of monotone models resulting from the lack of cancellations, makes them significantly easier to understand. Their computations are much better understood, both in the Boolean and the algebraic setting, and hence have been used as a starting point towards understanding the general model. Indeed, we know exponential lower bounds as well as separations between important complexity classes when we restrict ourselves to the monotone setting, in contrast to the embarrassingly poor understanding that we have in the general setting. In particular, Schnorr [40] showed an exponential lower bound for the clique polynomial, which happens to be a special case of homomorphism polynomial, where the pattern graph is a clique. Moreover, we know superpolynomial separations between complexity classes[2] mVF and mVBP [41], mVBP and mVP [20], and finally, very recently, between mVP and mVNP [45]. None of these separations are settled in the non-monotone setting. Moreover, the best known circuit lower bound in the non-monotone setting is just $\Omega(n \log n)$ [2]. Thus, it makes sense to study monotone models, both from the perspective of upper bounds – due to special interests in algorithms based on improved monotone circuits – as well as lower bounds – as a first step towards getting lower bounds for the general model. Many times, the best upper bounds come first in the monotone setting. For instance, the best known upper bound for the permanent polynomial for ABPs is a monotone construction [19]. We note that for computing homomorphism polynomials, Curticapean, Dell, and Marx gave an $O(n^{\omega})$ upper bound for all pattern graphs with treewidth $= 2$ [11], and an $O(n^{k\omega/3})$ upper bound is known for all k-vertex graphs [35], where ω is the exponent of matrix multiplication. However, for sufficiently large pattern graphs with treewidth > 2, which is almost all pattern graphs, the monotone constructions are the best known.

This motivated us to pursue the concrete question of whether these treewidth and pathwidth based improved monotone arithmetic circuits for homomorphisms polynomials can further be improved, and to what extent would these graph parameters dictate it. The answer that we discovered turns out to be conceptually quite satisfying, and settles the problem completely.

[2] $\mathsf{mVF}, \mathsf{mVBP}$ and mVP refer to the class of polynomial families computable by poly-sized monotone arithmetic formulas, monotone algebraic branching programs, and monotone arithmetic circuits, respectively. mVNP refers to the monotone analog of the complexity class VNP.

1.1 Our contributions

In this work, we fully solve the question of monotone complexity of homomorphism polynomials by showing that they are completely determined by the aforementioned graph parameters.

For arithmetic circuits, we discover that the treewidth of the pattern graph exactly determines the monotone complexity of its homomorphism polynomial, by establishing that the treewidth based upper bound in [13, 31, 3] is also a lower bound.

In what follows, the monotone arithmetic circuit (resp. ABP or formula) complexity of a polynomial refers to the size of the smallest monotone arithmetic circuit (resp. ABP or formula) computing the polynomial. Here *size* of an arithmetic circuit (resp. ABP or formula) is the number of edges in the circuit (resp. ABP or formula). See Section 2 for detailed definitions. The underlying field is $\mathbb{Q}$.

▶ **Theorem 1.** *The monotone arithmetic circuit complexity of the homomorphism polynomial for a pattern graph H is $\Theta(n^{tw(H)+1})$, where $tw(H)$ is the treewidth of H.*

In the case of algebraic branching programs, we find that the pathwidth of the pattern graph is the parameter controlling the monotone complexity of its homomorphism polynomial, again, by proving a lower bound that exactly matches the pathwidth based upper bound in [13, 31, 3].

▶ **Theorem 2.** *The monotone ABP complexity of the homomorphism polynomial for a pattern graph H is $\Theta(n^{pw(H)+1})$, where $pw(H)$ is the pathwidth of H.*

Finally, for monotone formulas, it is the treedepth of the pattern graph which governs the complexity of its homomorphism polynomial.

▶ **Theorem 3.** *The monotone formula complexity of the homomorphism polynomial for a pattern graph H is $\Theta(n^{td(H)})$, where $td(H)$ is the treedepth of H.*

Hence, we resolve the question of monotone complexity of homomorphism polynomials completely by showing that treewidth, pathwidth and treedepth exactly characterize the complexity of homomorphism polynomials for arithmetic circuits, ABPs, and arithmetic formulas respectively. This also answers the conceptual question raised earlier asking to what extent do these graph parameters dictate the complexity of these homomorphism polynomials.

The characterization, in addition to giving several new lower bounds, in particular, also allows us to collect, through a unified framework, several classical and recent results for the monotone complexity of polynomials, obtained independently using different methods, over several decades. This is simply because the above theorems imply that for every family of pattern graph with high treewidth, pathwidth, or treedepth, the corresponding homomorphism polynomial family will have high circuit complexity, ABP complexity, or formula complexity, respectively.

- As a first example, applying the observations made in the proofs of Theorem 1 and Theorem 2 to polynomial families that count colored isomorphisms from pattern families (as opposed to fixed size pattern graphs), we obtain an alternative proof for the superpolynomial separation between monotone arithmetic circuits and monotone algebraic branching programs, first discovered by Hrubes and Yehudayoff [20]. We believe that our proofs are conceptually much simpler.

- In the same spirit, we also manage to reattain the superpolynomial separation between algebraic branching programs and arithmetic formulas in the monotone setting, using simple applications of Theorem 2 and Theorem 3. This was previously known as a consequence of a result by Snir [41].
- Our characterization, in particular, also rediscovers the fine-grained separations between all these models for constant-degree polynomials. That is, for every constant d, we get a polynomial family such that it is computable by linear-sized monotone arithmetic circuits, but any monotone ABP computing it must be of size at least n^d. Analogously, for every constant d, we get a polynomial family computable by linear-sized monotone ABP but any monotone arithmetic formula computing it must be of size at least n^d. Earlier, such fine-grained separations could be obtained by applying the results of [41] and [20] together.
- Another simple application of Theorem 1 yields an exponential lower bound against monotone arithmetic circuits for the clique polynomial $\mathrm{CL}_{2n,n}$, which enumerates all cliques of size n in a $2n$-vertex clique, first proved by Schnorr [40, Theorem 4.4].

▶ Remark 4. Note that, Theorems 1, 2, and 3 have pattern graph H fixed and hence of constant size. Thus, the theorems pertain to constant degree polynomials and they do not yield superpolynomial lower bounds on homomorphism polynomials. Unlike the works that show exponential lower bounds on monotone polynomials, for instance [6, 7, 8, 42], our main goal is to obtain *tight* lower bounds on *homomorphism polynomials* for *fixed size* pattern graphs. Our proofs, however, go via the colored subgraph isomorphism polynomials for which we also get superpolynomial lower bounds by considering pattern graphs of non-constant size, see Definition 19, Theorem 20, and Theorem 21.

1.2 Limitations of known techniques

It is worth noting that in the Boolean setting, while there are known upper bounds and lower bounds based on these graph parameters for circuits and formulas for (colored) subgraph isomorphism problem, there is still an asymptotic gap between the upper bound and the corresponding lower bounds in these Boolean models (see [27, 26]). Thus, we cannot hope to translate the lower bounds in Boolean setting to the algebraic setting to get the lower bounds exactly matching the upper bounds.

An overwhelming majority of monotone arithmetic lower bounds exploit the so-called decomposition theorem (also refered to as normal form or representation theorem; see for e.g. [39, Lemma 8.7]) including the exponential monotone lower bounds for permanent [21, 37], separation between monotone VP and monotone VNP [45], as well as the superpolynomial separation between monotone arithmetic circuits and monotone ABPs [20]. However, a direct application of such a decomposition theorem fails to yield the *exact* characterization that the above theorems manage to achieve. We argue in Section 3 why these methods give asymptotically weaker lower bounds than what we desire to obtain.

For ABPs, Fournier, Malod, Szusterman, and Tavenas [18] recently gave a rank-based lower bound method for the monotone setting, inspired by [36] given in the non-commutative setting. However, in their models, the edge labels are homogeneous linear forms, which makes their method unsuitable for the tight lower bounds that we were looking for.

Finally, an approach that seems different from all these aforementioned approaches, including ours, is the one based on the concept of separating sets introduced by Schnorr [40], which also gave an exponential lower bound on the clique polynomial $\mathrm{CL}_{2n,n}$, which enumerates all cliques of size n in a $2n$-vertex clique [40, Theorem 4.4]. When the pattern is

a clique, his technique gives the optimal lower bound. However, it can be shown that, for other pattern graphs H, his proof technique, too, falls short in proving the lower bound that matches the upper bound (see full version for details). In fact, unlike Schnorr's lower bound techniques, our proofs, at a very high level, follow an abstract argument that can be shown to always give the optimal lower bound on the number of addition gates (see full version for details).

1.3 Proof ideas

At a high level, proofs to all our main theorems follow the same strategy. We discuss the case of circuits, since it already illuminates the overall idea.

The starting point towards the discovery of our proof is the fact that the upper bounds for homomorphism polynomials are all monotone and they use an optimal *tree decomposition* (see Definition 13) of the pattern graph to build the circuit. Our initial observation, which is a simple consequence of the above fact, is that *every* parse tree of such a circuit reflects a tree decomposition used to build the circuit. Moreover, since in these upper bound constructions (see [3, 13]), the whole circuit construction stems from a single tree-decomposition, all the parse trees correspond to the same tree decomposition of the pattern graph.

The above observation led us to the key question that sets the path towards our proofs:

▶ **Problem 1.** *Does an arbitrary parse tree of an arbitrary monotone circuit computing the homomorphism polynomial for an arbitrary pattern H allows us to recover a tree-decomposition of H?*

It turns out that answering the above question for homomorphism polynomials in the affirmative is difficult and most likely false. This is due of presence of non-multilinear monomials in homomorphism polynomials which correspond to those homomorphic images of the pattern graphs which are simpler than the pattern graph itself (see Definition 10). These monomials may loose the information about the pattern graph altogether, since such monomials may also result from homomorphism polynomials of other patterns. Thus, in order to be able to recover the tree decomposition, it seems imperative to work with the subgraph isomorphism polynomial instead. At a very high-level, our proof successfully answers the question – *in how many monomial computations can a single gate participate?* To give a precise bound on this, we need an even more structured polynomial – *the colored subgraph isomorphism polynomial*, where a monomial apart from encoding the pattern graph we begin with, also encodes the precise mapping from the pattern graph to the host graph that gives rise to the monomial (see Definition 11).

Before going further into the proof, one might wonder whether by going from homomorphism polynomial to the colored subgraph isomorphism polynomial, we have made the polynomial too structured as compared to the homomorphism polynomial. That is, whether the subgraph isomorphism polynomial is much harder than the homomorphism polynomial and a lower bound against the former would not yield a desired lower bound for the latter. Our first technical contribution is establishing that this is not so. We show that, as long as the the pattern graph is of constant size, the colored[3] homomorphism polynomial and colored subgraph isomorphism polynomial have the same monotone circuit complexity, monotone ABP complexity and monotone formula complexity (Lemma 15). For this, we first note

[3] It can be easily shown that the homomorphism polynomial and its colored counterpart have the same complexity.

that going from colored subgraph isomorphism polynomial to homomorphism polynomial is straight-forward. For the other direction, we note that the colored homomorphism polynomial contains a superset of the set of monomials of the colored subgraph isomorphism polynomial. To remove the other monomials present, we carefully introduce weights corresponding to edge variables and then use partial derivatives w.r.t those weight variables. We finally show that such partial derivatives can indeed be done efficiently in the the monotone circuits, monotone ABPs, as well as monotone formulas; the case of formulas require the most care.

Having established that we can indeed work with the colored subgraph isomorphism polynomial, we move on to answer Problem 1 for the colored subraph isomorphism polynomial in affirmative. Our main technical contribution is to show that, given an arbitrary parse tree of an arbitrary circuit computing the colored subgraph isomorphism polynomial for a pattern graph, there exists simple algorithm to recover its tree decomposition. The algorithm works in a bottom up fashion starting with the leaves of the parse tree and moves towards the root using a precise way to combine the *bags* of the gates it encounters along the way (see Theorem 16).

Once we have the tree decomposition, we use this tree decomposition to show that if the number of vertices in H is k, and its treewidth is $tw(H)$, a gate can participate in the computation of at most $n^{k-tw(H)-1}$ monomials. We show this by using a weakness of monotone computation that, due to the absence of cancellations, any circuit computing a polynomial cannot compute an invalid submonomial[4] at any intermediate gate along the way. This weakness is exploited in almost all known monotone lower bounds, and dates back to Jerrum and Snir [21].

For proving the lower bounds in Theorem 2 and 3 for formulas and ABPs, we are able to use the same framework as above. Instead of constructing a tree decomposition using the parse trees, we construct a *path decomposition* (Definition 13) in case of ABPs, and an *elimination tree* (Definition 14) in case of formulas. If we start with an ABP instead, it is easy to see that the same algorithm for tree decomposition yields a path instead, and, hence, we get a path decomposition (see Theorem 17). For formulas, we give a different simple algorithm to show that an elimination forest of H can be constructed using the parse tree (see Theorem 18). Using the same weakness of monotone computation, we conclude that the number of monomials whose computation a gate can participate in is upper bounded by $n^{k-pw(H)-1}$ in case of ABPs, and by $n^{k-td(H)}$ in case of formulas, where $pw(H)$ and $td(H)$ denote the pathwidth and the treedepth of H respectively.

To obtain the upper bounds claimed in Theorem 1, 2 and 3, we note that the upper bounds for circuits and ABPs that were already known are both monotone constructions, which go via the tree decomposition and the path decomposition of the pattern graph in case of circuits and ABPs respectively [13, 3, 31]. We give the formula upper bound using the elimination tree of the pattern graph. A treedepth based monotone formula upper bound is folklore in the Boolean setting.

To prove the separation between monotone complexity classes, we first note that the lower bounds on colored subgraph isomorphism polynomials also hold for pattern grpahs H of non-constant size. Consequently, pattern graphs with high pathwidth but low treewidth yield superpolynomial separation between circuits and ABPs (discussed in Theorem 20), whereas pattern graphs with high treedepth but low pathwidth give superpolynomial separation between ABPs and formulas (described in Theorem 21). For the first separation, we use a

[4] An invalid submonomial is a monomial m which does not divide any of the monomial of the target polynomial.

tree as the pattern graph, whereas for the second separation, we use path as the pattern graph. For an exponential lower bound on the clique polynomial, we simply apply Theorem 1 in combination with the fact that the treewidth of a clique on n vertices is $n-1$.

To simplify the presentation, we assume that the pattern graph is connected. The complexity for a disconnected pattern is the maximum of the complexity of its connected components.

2 Preliminaries

For basic notions in graph theory, we refer the readers to [43, 14]. We first give some definitions that set up our objects of computation and the models of computation.

▶ **Definition 5.** *A polynomial over $\mathbb{Q}$ is called* monotone *if all its coefficients are non-negative.*

Compact representations of polynomials such as the following are usually used by algorithms.

▶ **Definition 6.** *An* arithmetic circuit *over the variables $x_1, \ldots, x_n$ is a rooted DAG where each source node (also called an input gate) is labeled by one of the variables x_i or a constant $a \in \mathbb{Q}$. All other nodes (called gates) are labeled with either $+$ (addition) or $\times$ (multiplication). The circuit computes a polynomial over $\mathbb{Q}[x_1, \ldots, x_n]$ in the usual fashion. The circuit is called* monotone *if all constants are non-negative. The circuit is a* skew circuit *if for all $\times$ gates, at least one of the inputs is a variable or a constant. The circuit is a* formula *if all gates have out-degree at most one. The* size *of a circuit or skew circuit or formula is the number of edges in the circuit. The* depth *of a circuit is the number of edges in the longest path from the root to an input gate.*

Instead of skew circuits, a model that is equivalent in terms of power is usually studied in algebraic complexity.

▶ **Definition 7.** *An* Algebraic Branching Program (ABP) *is a DAG with a unique source node s and a unique sink node t. Each edge is labeled with a variable from $x_1, \ldots, x_n$ or a constant $a \in \mathbb{Q}$. Each path in the DAG from s to t corresponds to a term obtained by multiplying all the edge labels on that path. The polynomial computed by the ABP is the sum of all terms over all paths from s to t. The ABP is called* monotone *if all constants are non-negative. The* size *of the ABP is the number of edges.*

It is well-known that for any (monotone) polynomial, the size of the smallest (monotone) ABP and the size of the smallest (monotone) skew circuit are within constant factors of each other. In this paper, we will use the skew circuit definition in our proofs.

In this paper, we look at families of polynomials $(p_n)_{n \geq 0}$ and the optimal size and depth of the models computing them. In this case, the size and depth are functions of n and we are only interested in the asymptotic growth rate of these functions.

For simplicity, when proving lower bounds, we assume that all non-input gates have exactly two incoming edges (both may be from the same gate). This assumption increases the size by at most a constant factor and may increase the depth by at most a logarithmic factor.

The families of polynomials that we look at in this paper enumerate graph homomorphisms or colored isomorphisms. We first define these notions.

▶ **Definition 8.** *For graphs H and G, a* homomorphism *from H to G is a function $\phi : V(H) \mapsto V(G)$ such that $\{i,j\} \in E(H)$ implies $\{\phi(i), \phi(j)\} \in E(G)$. For an edge $e = \{i,j\}$ in H, we use $\phi(e)$ to denote $\{\phi(i), \phi(j)\}$.*

▶ **Definition 9.** *Let H be a k-vertex graph where its vertices are labeled by $[k]$ and let G be a graph where each vertex has a color in $[k]$. Then, a* colored isomorphism *of H in G is a subgraph of G isomorphic to H such that all vertices in the subgraph have different colors and for each edge $\{i, j\}$ in H, there is an edge in the subgraph between vertices colored i and j.*

Now, we are ready to define our main object of computation, the homomorphism polynomial.

▶ **Definition 10.** *For a pattern graph H on k vertices, the n-th* homomorphism polynomial *for H is a polynomial on $\binom{n}{2}$ variables x_e where $e = \{u, v\}$ for $u, v \in [n]$.*

$$\mathsf{Hom}_{H,n} = \sum_{\phi} \prod_{e \in E(H)} x_{\phi(e)}$$

where ϕ ranges over all homomorphisms from H to K_n.

Next, we define the colored isomorphism polynomial which will be crucial in our proofs. This polynomial enumerates all colored isomorphisms from a pattern to a host graph where there are n vertices of each color. This polynomial can be used to count colored isomorphisms in n-vertex host graphs by setting the variables corresponding to edges not in the host graph to 0.

▶ **Definition 11.** *For a pattern graph H on k vertices, the n-th* colored isomorphism polynomial *for H is a polynomial on $|E(H)|n^2$ variables x_e where $e = \{(i, u), (j, v)\}$ for $u, v \in [n]$ and $\{i, j\} \in E(H)$.*

$$\mathsf{ColIso}_{H,n} = \sum_{u_1,\ldots,u_k \in [n]} \prod_{\{i,j\} \in E(H)} x_{\{(i,u_i),(j,u_j)\}}$$

We notice that the labeling of H does not affect the complexity of ColIso_H. Given the polynomial ColIso_H for some labeling of H and if ψ is a relabeling of H, then the polynomial ColIso_H for the new labeling can be obtained by the substitution $x_{\{(i,u),(j,v)\}} \mapsto x_{\{(\psi(i),u),(\psi(j),v)\}}$.

For our main proofs, we need a way to analyze how a monomial is being computed in a circuit, an ABP, or a formula. For this, we use the notion of parse trees.

▶ **Definition 12.** *Let g be a gate in a circuit C. A* parse tree *rooted at g is any rooted tree which can be obtained by the following procedure, duplicating gates in C as necessary to preserve the tree structure.*

1. *The gate g is the root of the tree.*
2. *If there is a multiplication gate g in the tree, include all its children in the circuit as its children in the tree.*
3. *If there is an addition gate g in the tree, pick an arbitrary child of g in the circuit and include it in the tree.*

If the root gate of a parse tree is not mentioned, then it is assumed to be the output gate of the circuit. A parse tree witnesses the computation of some term. We note that if C is a formula, then any gate can occur at most once in any parse tree in C.

Given a parse tree T that contains a gate g, we use T_g to denote the subtree of T rooted at g. The tree obtained by removing T_g from T is called the tree outside T_g in T. Note that we can replace T_g in T with any parse tree rooted at g to obtain another parse tree.

Similarly, if we have two parse trees T and T' that both contain the same multiplication gate g from the circuit, then we can replace the left or right subtree of T_g with the left or right subtree of T'_g to obtain another parse tree. This is because both the left and right child of g in both parse trees are the same and therefore we can apply the aforementioned replacement.

Now, we define the graph parameters most crucial to our work. They are the parameters that turn out to exactly characterize the complexity of the above polynomial families in the models of computations defined above.

▶ **Definition 13.** *A* tree decomposition *of H is a tree where each vertex (called a* bag*) in the tree is a subset of vertices of H. This tree must satisfy two properties.*

1. *For every edge $\{i, j\}$ in H, there must be at least one bag in the tree that contains both i and j.*
2. *For any vertex i in H, the subgraph of the tree decomposition induced by all bags containing i must be a subtree. This subtree is called the* subtree induced by *i.*

The size of a tree decomposition is the size of the largest bag minus one. The treewidth *of H is the size of a smallest tree decomposition of H.*

A tree decomposition is called a path decomposition *if it is a path. The* pathwidth *of H is the size of a smallest path decomposition.*

▶ **Definition 14.** *For a connected graph H, an* elimination tree *of H is a rooted, directed tree that can be constructed by arbitrarily picking a vertex u in H and adding edges from the roots of elimination trees of connected components of $H - u$ to the root vertex labeled u. In particular, if H is a single vertex, then the elimination tree of H is the same single vertex graph.*

The depth *of an elimination tree is the number of vertices in the longest path from a leaf to the root. The* treedepth *of H is the depth of the smallest depth elimination tree of H.*

We note that for any graph H, its elimination tree contains exactly $|V(H)|$ vertices. All edges in the tree are directed towards the root and the vertices of the tree are uniquely labeled with the vertices of H. If T is an elimination tree for the connected graph H, then all edges in H are between vertices that are in an ancestor-descendant relationship in T.

We now state some basic facts about treewidth, pathwidth, and treedepth. We combine these facts with the lower bounds in Section 3 to obtain, for any constant k, constant-degree polynomial families having linear size circuits (ABPs resp.) but requiring $\Omega(n^k)$ size ABPs (formulas resp.), thus giving us a fine-grained separation between these models. Note that, for constant-degree polynomials, such polynomial factor separations are the best one could ask for between formulas, ABPs, and circuits, since all constant-degree polynomials are computable by polynomial-sized formulas. Moreover, we use these facts to obtain superpolynomial separations between these models for high degree polynomials.

▶ **Fact 1.** *For all graphs H, we have $tw(H) \leq pw(H) \leq td(H) - 1$.*

▶ **Fact 2.** *For any number $p \geq 1$, there is a tree X_k on $k = \frac{1}{6}(5 \cdot 3^p - 3)$ vertices that have pathwidth p.*

Proof. The tree X_2 is simply an edge. The tree X_k for $p \geq 2$ is obtained by connecting three copies of the constructed trees with pathwidth $p - 1$ to a new root vertex. ◀

▶ **Fact 3.** *All paths have pathwidth 1 and the k-vertex path has treedepth $\lceil \log_2(k + 1) \rceil$.*

3 Algebraic complexity of homomorphism polynomials

In this section, we prove Theorems 1, 2 and 3. thus achieving our main result, that is, the exact characterization for the monotone complexity of homomorphism polynomials.

▶ **Lemma 15.** *For any fixed pattern H, both the homomorphism polynomial for H and the colored isomorphism polynomial for H have the same (asymptotically) monotone arithmetic circuit complexity, monotone ABP complexity, and monotone formula complexity.*

Proof. First, we show how to obtain a circuit that computes Collso_H from one that computes Hom_H. We introduce new variables w_e for each $e \in E(H)$. Let C be a circuit that computes Hom_H over the vertex set $[k] \times [n]$. We substitute $x_{\{(i,u),(j,v)\}}$ with $x_{\{(i,u),(j,v)\}} w_{\{i,j\}}$ for all i, j, u, v if $\{i, j\} \in E(H)$. Otherwise, we set the variable to 0. Let C' be the resulting circuit. We then compute $\frac{\partial^{|E(H)|}}{\partial w_{e_1} \dots \partial w_{e_{|E(H)|}}} C'$ using the sum and product rule for partial derivatives. Then, we set $w_e = 0$ for all e. The partial differentiation ensures that all monomials must have at least one edge of each color. Setting $w_e = 0$ ensures that all monomials can only have at most one edge of each color. Therefore, the remaining monomials contain each edge in H exactly once. These are exactly those homomorphisms that correspond to colored isomorphisms. For each colored isomorphism, there are $|aut(H)|$ ways to relabel the vertices of H to obtain the same edge set. So we divide by $|aut(H)|$ to obtain Hom_H exactly. Each partial differentiation increases the size of the circuit at most by a factor of 3. Therefore, if C has size s, the final circuit has size at most $c3^{|E(H)|}s$ for some constant c.

For the other direction, given a circuit that computes Collso_H, we can replace each $x_{\{(i,u),(j,v)\}}$ with $x_{\{u,v\}}$ if $u \neq v$ and 0 otherwise. The circuit now computes Hom_H because the monomial corresponding to the homomorphism ϕ is generated by the corresponding colored isomorphism $i \mapsto (i, \phi(i))$. Also, every colored isomorphism on vertices (i, u_i) for $1 \leq i \leq k$ corresponds to a homomorphism to K_n as long as $u_i \neq u_j$ for $\{i, j\} \in E(H)$.

Notice that both constructions preserve monotonicity and yield a monotone ABP when the original circuit is a monotone ABP.

A straightforward application of the sum and product rule to compute the partial derivative does not necessarily yield a formula from a formula. While computing the partial derivative, we have to compute both f and $\partial f/\partial x$ for every sub-formula f. If $f = g + h$ for some formulas g and h, then we can simply compute $\partial f/\partial x = \partial g/\partial x + \partial h/\partial x$ without using any sub-formula more than once. When the formula $f = gh$ for some formulas g and h, we have to compute $\partial f/\partial x = g\partial h/\partial x + h\partial g/\partial x$. Therefore, we are using g and h twice, once for computing f and once for computing $\partial f/\partial x$. We can convert the resulting circuit into a formula by duplicating the sub-formulas g and h. For general formulas, this leads to a quadratic blowup in size. But, for monotone formulas computing constant-degree polynomials, we show that the size increases only by a constant factor when duplicating sub-formulas in this fashion.

First, we eliminate all gates in the formula that computes 0 and replace all gates that compute a constant by an input gate labeled with that constant. We call a multiplication gate f *trivial* if $f = ag$ for some constant a and sub-formula g. In this case, we have $\partial f/\partial x = a\partial g/\partial x$. Therefore, we do not have to make a copy of the non-constant sub-formula g. We make a copy of the input gate a here. But, we do not have to make additional copies of this input gate a later for any trivial gate encountered on the path from f to root. This is because f is not a constant and therefore it must be the other input to the trivial gate that is constant. We claim that if the monotone formula computes a polynomial of degree at most d, for any gate g, there are at most d non-trivial multiplication gates on the path

from g to the root gate. This is because each non-trivial gate increases the degree by at least one and no cancellations can occur in a monotone formula. Therefore, we can compute the partial derivative using at most $d+2$ copies of each gate in the original formula. Since the degree of the polynomials we are differentiating is at most $2|E(H)|$ and we perform $|E(H)|$ differentiations, the final formula has size $O\big((2|E(H)|+2)^{|E(H)|}s\big)$ if the original formula has size s. ◀

From now on, we can use either Collso_H or Hom_H to prove our results. For a pattern graph H, we say that an edge $\{i,j\}$ or a vertex i in H is present in a monomial over the variables of the colored isomorphism polynomial if there is a variable $x_{\{(i,u),(j,v)\}}$ for some u and v in that monomial. We also say that the edge $\{(i,u),(j,v)\}$ or the vertex (i,u) is present in that monomial in that case.

In our proofs, we restrict our attention to the multiplication gates and the input gates in parse trees. This is because our proofs consider the computation of individual monomials separately and in this case, the addition gates play no role in the computation as all of them have exactly one child in the parse tree. While proving lower bounds, we also assume that all addition and multiplication gates have exactly two incoming edges. Since we define the size of a circuit as the number of edges, this transformation changes the size only by a constant factor. In addition, with this restriction, the number of gates and the number of edges are related by constant factors. Therefore, we may lower bound either to lower bound the size of the circuit.

▶ **Theorem 16** (Theorem 1 restated). *The monotone arithmetic circuit complexity of* Hom_H *is* $\Theta\big(n^{tw(H)+1}\big)$.

Proof. The upper bound is already known (See [13], [3]).

Let H be of treewidth t. For proving the lower bound, we consider the n-th colored isomorphism polynomial for H, $\mathsf{Collso}_{H,n} = \sum_{u_1,\ldots,u_k} \prod_{i,j} x_{\{(i,u_i),(j,u_j)\}}$, where $u_1,\ldots,u_k \in [n]$ and $\{i,j\} \in E(H)$ (see Definition 11). Consider any monomial m on the vertices (i,u_i) for $1 \le i \le k$ and its associated parse tree. We assume without loss of generality that both sides of any multiplication gate computes a non-constant term (If they compute a constant, it has to be 1 and we can simply ignore this subtree). We build a tree decomposition of H from this parse tree in a bottom-up fashion. In this tree decomposition, each gate is associated with one or two bags in the decomposition. Exactly one of these bags is designated as the *root bag* of that gate. An example of this construction is shown in Figure 1 where the 4-vertex cycle is the pattern.

1. For an input gate $x_{\{(i,u),(j,v)\}}$. Add the bag $\{i,j\}$ as a leaf of the tree decomposition. For example, in Figure 1, the input gate labeled $x_{\{(1,u),(2,v)\}}$ is associated with the bag $\{1,2\}$.
2. For a multiplication gate g. Let A and B be the contents of the root bags of its left and right subtrees. Add a bag containing $A \cup B$ as the parent of those roots. For example, in Figure 1, the gate g is first associated with a bag containing $\{1,2,3\} = \{1,2\} \cup \{2,3\}$.
3. If at any gate, there are vertices (i,u) such that the monomial computed at that gate includes all edges incident on (i,u). Then add a new bag that excludes exactly all such vertices i from the current root bag and add it as a parent of the current root bag. This new bag is now considered the root bag associated with the gate g. For example, in Figure 1, we observe that at gate g, all edges incident on the vertex $(2,v)$ are included and therefore we add a new bag that removes 2 and make it the root bag associated with g.

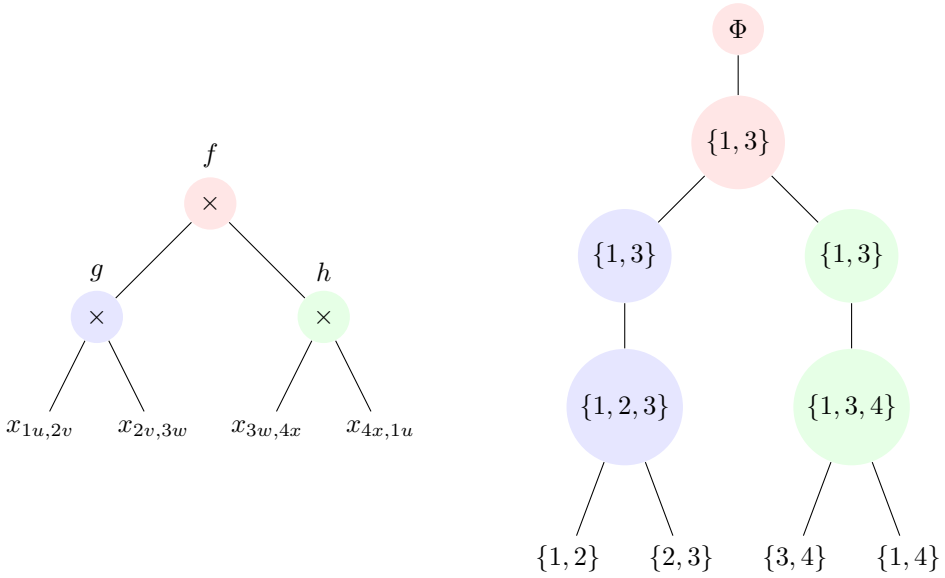


Figure 1 A tree decomposition of C_4 from a parse tree. $u, v, w, x \in [n]$.

The result is a tree decomposition of H. All edges of H are covered because all edges of H must be present in the monomial. Since a vertex is forgotten only after all edges incident on it have been multiplied, the subgraph of the tree decomposition induced by any vertex in H is a subtree.

Since H has treewidth t, the tree decomposition that we constructed must have a bag with at least $t+1$ vertices. We consider some gate g in the parse tree that is associated with a bag that contains at least $t+1$ vertices. In the following proof, we only consider the case where the bag contains exactly $t+1$ vertices. If there are more than $t+1$ vertices, then we get a better lower bound. We assume without loss of generality that these vertices are $1, \ldots, t+1$. We now claim that the gate g can only be present in parse trees of monomials m' such that m' contains vertices (i, u_i) for $1 \leq i \leq t+1$. Suppose for contradiction there is a monomial m' with a parse tree T' that contains g and has vertex (i, v_i) where $v_i \neq u_i$ for some such i. Let T be the parse tree for m. There are two cases:

1. The vertex i is forgotten at a bag associated with g: This means that both left and right subtrees of T_g compute monomials that contain (i, u_i). Now if T' contains (i, v_i) on the left subtree of T'_g, then replace right subtree of T'_g with right subtree of T_g. Else, the tree T' contains (i, v_i) on the right subtree of T'_g or outside T'_g. In both cases we replace the left subtree of T'_g with the left subtree of T_g.
2. Vertex i is not forgotten at a bag associated with g: This means that (i, u_i) appears in at least one of the subtrees of T_g and outside T_g in T. In T', if (i, v_i) appears in T'_g, then replace the tree T_g with T'_g in T. Otherwise, the vertex (i, v_i) must appear outside T'_g in T'. In this case, replace T'_g with T_g in T'.

All these new parse trees yield monomials that contain both (i, u_i) and (i, v_i). A contradiction. Notice that we can obtain exactly n^{k-t-1} colored isomorphisms that fix $t+1$ vertices. Therefore, at most n^{k-t-1} monomials of the polynomial can contain g in its parse tree. Since there are n^k monomials, this gives the required lower bound. ◀

▶ **Theorem 17** (Theorem 2 restated). *The monotone algebraic branching program complexity of* Hom_H *is* $\Theta\left(n^{pw(H)+1}\right)$.

Proof. The upper bound is already known (See [13], [3]). For the lower bound, we modify the proof of Theorem 16 to obtain a path decomposition instead. Recall that for any (monotone) polynomial, the size of the smallest (monotone) ABP and the size of the smallest (monotone) skew circuit are within constant factors of each other. We consider the parse tree for a monomial m in a skew circuit computing Hom_H and start building the path decomposition from a deepest input gate.

We give a detailed proof in the full version. ◀

▶ **Theorem 18** (Theorem 3 restated). *The monotone formula complexity of* Hom_H *is* $\Theta\left(n^{td(H)}\right)$.

Proof. We first prove the upper bound[5]. Let T be an elimination tree of depth d for H. We show how to construct a formula of size n^d for $\mathsf{Collso}_{H'}$ in a bottom-up fashion where H' is the graph obtained from T by adding all possible edges $\{i,j\}$ where i is an ancestor of j in T. Note that H' and H has the same treedepth by construction. The polynomial Collso_H can be obtained by setting extra variables in this polynomial to 1.

Let i be the label of a node in T such that the path from root to i is labeled $i_1,\ldots,i_p$ and the children of i are labeled $\ell_1,\ldots,\ell_s$. We use the notation $(i_j,u_j)_j$ where $j\in[p]$ to denote the p pairs $(i_1,u_1),\ldots,(i_p,u_p)$. We construct the formula:

$$f_i^{\{(i_j,u_j)_j\}} = \sum_u \left(\left(\prod_j x_{\{(i_j,u_j),(i,u)\}}\right) \prod_t f_{\ell_t}^{\{(i,u),(i_j,u_j)_j\}} \right)$$

where $j\in[p]$ and $t\in[s]$.

We use induction on the height c of the node i in the elimination tree to prove the size upper bound. We claim that the formula that corresponds to such a node has size $O(n^c)$ (constants hidden by the O notation depend only on H). For the base case, if i is a leaf node, this formula has size $O(n)$. If i has depth c in T, then this formula has size $O(n)\times O(n^{c-1}) = O(n^c)$ by the induction hypothesis.

If $\{i,j\}$ is not an edge in H, then we set all $x_{\{(i,u),(j,v)\}}$ to 1 in the formula. The formula corresponding to the root node in T is the required polynomial. To prove this, consider the colored subgraph isomorphism containing the vertices (i,u_i) for $1\le i\le k$. This monomial has the parse tree obtained by setting the variable u in the outermost summation for each f_i^* to u_i. For the other direction, the parse tree obtained by setting u to u_i in the outermost summation of each f_i^* generates the monomial that corresponds to the colored subgraph isomorphism on vertices (i,u_i).

We now prove the lower bound. Let d be the treedepth of H. We consider a parse tree for a monomial m in a formula computing Collso_H and build an elimination tree for H from it. An example of this construction is shown in Figure 2 where the 4-vertex path is the pattern. We associate a set of rooted trees with each gate g as follows: If the gate g is the lowest gate in the parse tree such that all edges incident on vertices $i_1,\ldots,i_r$ are present in the monomial computed at g, then make i_1 the parent of all roots of trees from the children of g. Now, make i_{j+1} the parent of i_j for $1\le j\le r-1$. We call the vertices $i_1,\ldots,i_r$ to be associated with g. If there are no such vertices for a gate g, then the forest associated

[5] We believe this construction is already known as folklore. But we present it here for the sake of completion since we couldn't find a reference for the construction.

with g is simply the union of the forests of its children. We start with empty forests initially. For any edge $\{i,j\} \in E(H)$, the gates that corresponds to i and j in this tree must belong to the path from the input gate for this edge to the root. This shows that this tree is an elimination tree for H.

In Figure 2, the input gate labeled z is such that the only edge incident on $(1,u)$ is already multiplied in at z. Therefore, we associate the vertex 1 with z and the set of trees associated with z is just the one-vertex tree 1. Also, at gate f, we have multiplied in all edges incident on $(3,w)$ and therefore, we associate 3 with f. Also, note that this vertex 3 is the parent of the roots of elimination trees from g and h.

For proving the lower bound, we consider some gate g in the parse tree of m such that g is associated with a leaf at a depth of at least d in the elimination tree. We can assume that the depth is exactly d. If it is more, then we obtain a better lower bound. Let this vertex be d. Assume without loss of generality that $1,\ldots,d$ are the vertices on the path from the root to d in the elimination tree. Let $(1,u_1),\ldots,(d,u_d)$ be the corresponding vertices in m. We claim that any monomial m' for which g appears in its parse tree must also have vertices u_j of color j for $1 \leq j \leq d$.

Suppose for contradiction that there is a monomial m' with g in its parse tree and m' has vertex (i,v_i) where $u_i \neq v_i$ for some $1 \leq i \leq d$. Let g' be the gate in T such that i is associated with g'. Now, recall that if T is an elimination tree for the connected graph H, then all edges in H are between vertices that are in an ancestor-descendant relationship in T. Thus, g' must be an ancestor of g (g' could be the same as g). Let T' be the parse tree for m'. Then, the tree T' must contain g' as well because it contains g and in a formula there is a unique path from any gate to the root. There are two cases:

1. The gate g' is an input gate: In this case, $u_i = v_i$ because a monomial cannot have two different vertices of the same color and (i,u_i) is present in g'.
2. The gate g' is a multiplication gate: In this case, the vertex (i,u_i) must appear in both subtrees of $T_{g'}$. If the vertex (i,v_i) appears in the right (left) subtree of $T'_{g'}$, then we replace the left (right) subtree of $T'_{g'}$ with the left (right) subtree of $T_{g'}$. Otherwise, the vertex (i,v_i) appears outside $T'_{g'}$ in T'. In this case, we replace the subtree $T'_{g'}$ with the subtree $T_{g'}$.

In all cases, we obtain a monomial that contains both vertices (i,u_i) and (i,v_i). A contradiction. Therefore, at most n^{k-d} monomials of the polynomial can contain the gate g in their parse tree. Since there are a total of n^k monomials, the lower bound follows.

In Figure 2, we can infer using the above argument that the input gate z can be a part of parse trees of at most n different monomials. ◀

We show how to use our characterizations to prove superpolynomial separations between various monotone models. The polynomials are *not* based on fixed pattern graphs but a natural extension of our polynomials to patterns that grow in size with the host graph.

▶ **Definition 19.** *Let $H = (H_n)$ be a family of* pattern graphs *where H_n is some graph on n vertices. Then, we define the n-th polynomial in the family* CollSo_H *as:*

$$\mathsf{CollSo}_{H_n,n} = \sum_{u_1,\ldots,u_n} \prod_{\{i,j\}} x_{\{(i,u_i),(j,u_j)\}}$$

where $u_1,\ldots,u_n \in [n]$ and $\{i,j\} \in E(H_n)$.

It is easy to see that our lower bounds for monotone circuits, monotone ABPs, and monotone formulas hold for CollSo_H for any family of pattern graphs H, where the lower bound for the n-th polynomial is given by the treewidth, pathwidth, and treedepth of H_n

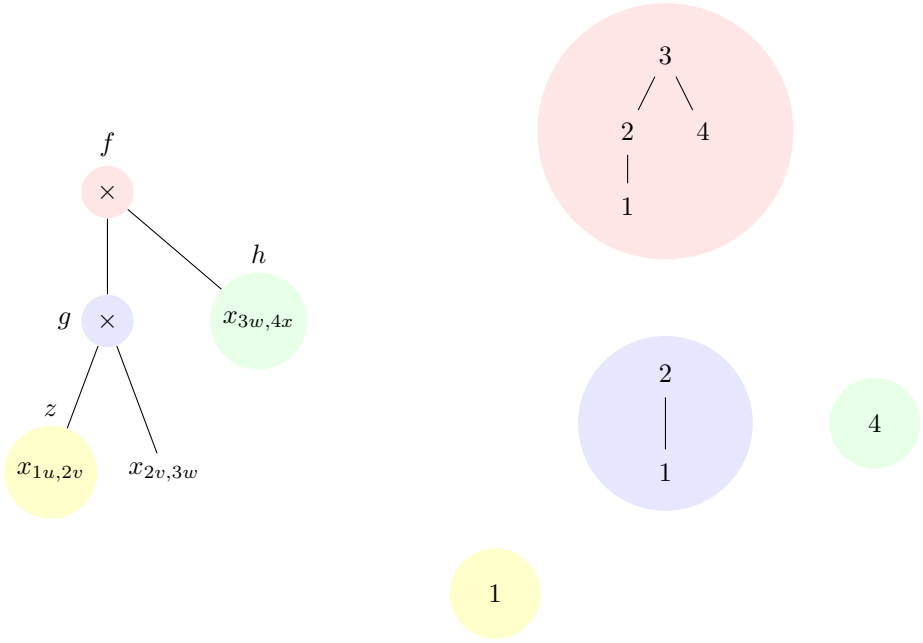


Figure 2 An elimination tree for P_4 from a parse tree. $u, v, w, x \in [n]$.

respectively. The lower bounds for Hom_H, however, does not hold because Lemma 15 causes a size blow-up that is exponential in the size of the pattern graph while constructing $\mathsf{Collso}_{H_n,n}$ from $\mathsf{Hom}_{H_n,n}$.

We now show that there are pattern families that prove a super-polynomial separation between the size complexity of monotone circuits and monotone ABPs.

▶ **Theorem 20** (Compare [20, Theorem 1, Proposition 13]). *For any $p \geq 1$ and $n = \frac{1}{6}(5 \cdot 3^p - 3)$, there is a polynomial family of degree $n-1$ on $n^3 - n$ variables such that the family has linear size monotone circuits but require $n^{\log_2(n+1)}$ size monotone ABPs.*

For the separation, the n-th polynomial in the family is simply Collso_{X_n}, where X_n are the trees whose existence is guaranteed from Fact 2. See full version for a detailed proof.

We now show that there are pattern families that prove a super-polynomial separation between the size complexity of monotone circuits and monotone ABPs.

▶ **Theorem 21** (See [41, Theorem 3.2]). *For $n \geq 2$, there is a polynomial family of degree $n-1$ on $n^3 - n$ variables such that the family has linear size monotone ABPs but require $n^{\lceil \log_2(n+1) \rceil}$ size monotone formulas.*

The n-th polynomial in the family is simply Collso_{P_n}, the path on n vertices. See full version for a detailed proof.

4 Discussion

We note that our lower bounds for colored isomorphism polynomials also hold for *counting circuits* studied by Jukna [22]. Such circuits produce the same evaluation as arithmetic circuits when variables are substituted from $\{0, 1\}$. Informally, a counting circuit is an arithmetic circuit that computes a polynomial modulo the axiom $x^2 = x$. Note that this is sufficient for counting homomorphisms or colored isomorphisms. However, the counting circuit lower

bounds do not extend to the homomorphism polynomials since the reduction between the colored isomorphism polynomial and the homomorphism polynomial in Lemma 15 use partial derivatives.

References

1 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and Süleyman Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. In *Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, 2008*, pages 241–249, 2008. doi:10.1093/bioinformatics/btn163.

2 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

3 Markus Bläser, Balagopal Komarath, and Karteek Sreenivasaiah. Graph pattern polynomials. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 18:1–18:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.18.

4 C. Borgelt and M. R. Berthold. Mining molecular fragments: finding relevant substructures of molecules. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 51–58, 2002.

5 Christian Borgs, Jennifer Chayes, László Lovász, Vera T Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, pages 315–371. Springer, 2006.

6 Bruno Pasqualotto Cavalar, Mrinal Kumar, and Benjamin Rossman. Monotone circuit lower bounds from robust sunflowers. In *LATIN 2020: Theoretical Informatics: 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings*, pages 311–322, 2021. doi:10.1007/978-3-030-61792-9_25.

7 Arkadev Chattopadhyay, Rajit Datta, and Partha Mukhopadhyay. Negations provide strongly exponential savings. *Electron. Colloquium Comput. Complex.*, page 191, 2020. URL: https://eccc.weizmann.ac.il/report/2020/191.

8 Arkadev Chattopadhyay, Rajit Datta, and Partha Mukhopadhyay. *Lower Bounds for Monotone Arithmetic Circuits via Communication Complexity*, pages 786–799. Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3406325.3451069.

9 Prasad Chaugule, Nutan Limaye, and Aditya Varre. Variants of homomorphism polynomials complete for algebraic complexity classes. In *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 2019. doi:10.1007/978-3-030-26176-4_8.

10 F. R. K. Chung, R. L. Graham, and R. M. Wilson. Quasi-random graphs. *Combinatorica*, 9(4):345–362, 1989. doi:10.1007/BF02125347.

11 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

12 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoret. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

13 Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Counting h-colorings of partial k-trees. *Theor. Comput. Sci.*, 281(1-2):291–309, 2002. doi:10.1016/S0304-3975(02)00017-8.

14 R. Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006. URL: https://books.google.de/books?id=aR2TMYQr2CMC.

15 Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. Homomorphism polynomials complete for VP. *Chic. J. Theor. Comput. Sci.*, 2016, 2016. URL: `http://cjtcs.cs.uchicago.edu/articles/2016/3/contents.html`.

16 Christian Engels. Dichotomy theorems for homomorphism polynomials of graph classes. *Journal of Graph Algorithms and Applications*, 20(1):3–22, 2016. `doi:10.7155/jgaa.00382`.

17 Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. *SIAM J. Discrete Math.*, 29(3):1322–1339, 2015. `doi:10.1137/140978211`.

18 Hervé Fournier, Guillaume Malod, Maud Szusterman, and Sébastien Tavenas. Nonnegative Rank Measures and Monotone Algebraic Branching Programs. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.FSTTCS.2019.15`.

19 Bruno Grenet. An upper bound for the permanent versus determinant problem, 2012.

20 Pavel Hrubes and Amir Yehudayoff. On isoperimetric profiles and computational complexity. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 89:1–89:12, 2016. `doi:10.4230/LIPIcs.ICALP.2016.89`.

21 Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29(3):874–897, July 1982. `doi:10.1145/322326.322341`.

22 Stasys Jukna. Lower bounds for monotone counting circuits. *Discrete Applied Mathematics*, 213:139–152, 2016.

23 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000. `doi:10.1016/S0020-0190(00)00047-8`.

24 Xiangnan Kong, Jiawei Zhang, and Philip S. Yu. Inferring anchor links across multiple heterogeneous social networks. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 179–188, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2505515.2505531`.

25 Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discrete Math.*, 27(2):892–909, 2013. `doi:10.1137/110859798`.

26 Deepanshu Kush and Benjamin Rossman. Tree-depth and the formula complexity of subgraph isomorphism. *CoRR*, abs/2004.13302, 2020. `arXiv:2004.13302`.

27 Yuan Li, Alexander A. Razborov, and Benjamin Rossman. On the ac^0 complexity of subgraph isomorphism. *SIAM J. Comput.*, 46(3):936–971, 2017. `doi:10.1137/14099721X`.

28 Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 1959–1969, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3394486.3403247`.

29 László Lovász. *Large networks and graph limits*, volume 60 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 2012. `doi:10.1090/coll/060`.

30 László Lovász and Vera T. Sós. Generalized quasirandom graphs. *J. Combin. Theory Ser. B*, 98(1):146–163, 2008. `doi:10.1016/j.jctb.2007.06.005`.

31 Meena Mahajan and Nitin Saurabh. Some complete and intermediate polynomials in algebraic complexity theory. *Theory Comput. Syst.*, 62(3):622–652, 2018. `doi:10.1007/s00224-016-9740-y`.

32 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6:85–112, 2010. `doi:10.4086/toc.2010.v006a005`.

33 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 542–553. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.542.

34 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

35 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: http://eudml.org/doc/17394.

36 Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 410–418, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/103418.103462.

37 Ran Raz and Amir Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. *J. Comput. System Sci.*, 77(1):167–190, 2011. doi:10.1016/j.jcss.2010.06.013.

38 Benjamin Rossman. Lower bounds for subgraph isomorphism. In *Proceedings of the International Congress of Mathematicians—Rio de Janeiro 2018. Vol. IV. Invited lectures*, pages 3425–3446. World Sci. Publ., Hackensack, NJ, 2018.

39 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. *Github survey*, 2015.

40 C.P. Schnorr. A lower bound on the number of additions in monotone computations. *Theoretical Computer Science*, 2(3):305–315, 1976. doi:10.1016/0304-3975(76)90083-9.

41 Marc Snir. On the size complexity of monotone formulas. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 621–631, Berlin, Heidelberg, 1980. Springer Berlin Heidelberg.

42 Srikanth Srinivasan. Strongly exponential separation between monotone vp and monotone vnp. *ACM Trans. Comput. Theory*, 12(4), September 2020. doi:10.1145/3417758.

43 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.

44 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.

45 Amir Yehudayoff. Separating monotone vp and vnp. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 425–429, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316311.

46 J. Zhang and G. Wu. Targeting social advertising to friends of users who have interacted with an object associated with the advertising, Dec. 15 2010. US Patent App. 12/968,786.

47 Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. Meta-graph based recommendation fusion over heterogeneous information networks. In *KDD*, pages 635–644, 2017. doi:10.1145/3097983.3098063.

Exact Recovery Algorithm for Planted Bipartite Graph in Semi-Random Graphs

Akash Kumar
School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

Anand Louis
Computer Science and Automation Department, IISc, Bangalore, India

Rameesh Paul
Computer Science and Automation Department, IISc, Bangalore, India

Abstract

The problem of finding the largest induced balanced bipartite subgraph in a given graph is NP-hard. This problem is closely related to the problem of finding the smallest Odd Cycle Transversal.

In this work, we consider the following model of instances: starting with a set of vertices V, a set $S \subseteq V$ of k vertices is chosen and an arbitrary d-regular bipartite graph is added on it; edges between pairs of vertices in $S \times (V \setminus S)$ and $(V \setminus S) \times (V \setminus S)$ are added with probability p. Since for $d = 0$, the problem reduces to recovering a planted independent set, we don't expect efficient algorithms for $k = o\left(\sqrt{n}\right)$. This problem is a generalization of the planted balanced biclique problem where the bipartite graph induced on S is a complete bipartite graph; [46] gave an algorithm for recovering S in this problem when $k = \Omega\left(\sqrt{n}\right)$.

Our main result is an efficient algorithm that recovers (w.h.p.) the planted bipartite graph when $k = \Omega_p\left(\sqrt{n \log n}\right)$ for a large range of parameters. Our results also hold for a natural semi-random model of instances, which involve the presence of a monotone adversary. Our proof shows that a natural SDP relaxation for the problem is integral by constructing an appropriate solution to it's dual formulation. Our main technical contribution is a new approach for construction the dual solution where we calibrate the eigenvectors of the adjacency matrix to be the eigenvectors of the dual matrix. We believe that this approach may have applications to other recovery problems in semi-random models as well.

When $k = \Omega\left(\sqrt{n}\right)$, we give an algorithm for recovering S whose running time is exponential in the number of small eigenvalues in graph induced on S; this algorithm is based on subspace enumeration techniques due to the works of [42, 8, 41].

2012 ACM Subject Classification Theory of computation → Semidefinite programming; Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases SDP duality, Planted models, Semi-random models, Exact recovery, Threshold rank, Spectral embedding, Subspace enumeration

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.84

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2205.03727

Funding *Akash Kumar*: Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759471). *Anand Louis*: Supported in part by SERB Award ECR/2017/003296 and a Pratiksha Trust Young Investigator Award.

1 Introduction

Given a graph $G = (V, E)$, the problem of finding the largest induced bipartite subgraph of G is well known to be NP-hard [64]. The problem is equivalent to the Odd Cycle Transversal problem. The problem is also related to the balanced biclique problem, where the task is

that of finding the largest induced balanced complete bipartite subgraph. This problem has a lot of practical application in computational biology [19], bioinformatics [65] and VLSI design [7].

For the worst-case instance of the problem, the work [3] gives an algorithm that computes a set with at least $\left(1-\mathcal{O}\left(\varepsilon\sqrt{\log n}\right)\right)$ fraction of vertices which induces a bipartite graph, when it is promised that the graph contains an induced bipartite graph having $(1-\varepsilon)\,n$ fraction of the vertices. The work [32] gives an efficient randomized algorithm that computes an induced bipartite subgraph having $\left(1-\mathcal{O}\left(\sqrt{\varepsilon\log d}\right)\right)$ fraction of the vertices where d is the bound on the maximum degree of the graph. They also give a matching (up to constant factors) Unique Games hardness for certain regimes of parameters. We refer to Section 1.2 for more details about these related problems.

In an effort to better understand the complexity of various computationally intractable problems, a lot of work has been focused on the special cases of the problem, and towards studying the problem in various *random* and *semi-random* models. Here, one starts with solving the problem for random instances (for graph problems this is often $G_{n,p}$ Erdős-Rényi graphs[1]). The analysis in random instances is often much simpler, and one can give algorithms with "good" approximation guarantees. The next goal in this direction is to plant a solution that is "clearly optimal" in an ambient random graph and then attempt to recover this planted solution. We, therefore, build towards the worst-case instances of the problem by progressively weakening our assumptions. We refer to the book [59] for a more detailed discussion of these models in the context of other problems like planted clique, planted bisection, k-coloring, Stochastic Block Models, and Matrix completion problems.

We start our discussion with the problem of computing a maximum clique/independent set, since it has been extensively studied in such planted models. In the planted clique/independent set problem we plant a clique/independent set of size k in an otherwise random $G_{n,p}$ graph. The work [6] presents an algorithm, which, given a graph $G \sim \mathcal{G}(n,1/2)$ with a planted clique/independent set of size k, recovers the planted clique when $k > c_1\sqrt{n}$ (where c_1 is a constant). We will refer to the planted independent set/clique problem at various points throughout the introduction

Such *random planted models* have been studied in context of other problems as well such as the planted 3-coloring problem [14, 5], planted dense subgraph problem [34, 35, 36], planted bisection and planted Stochastic Block models [16, 22, 37, 17, 21, 2], to state a few. We define a similar *random planted model* to study our problem, as stated below.

▶ **Definition 1** (Random planted model)**.** *Given n, k, d, p, our planted bipartite graph is constructed as follows,*

1. *Let V be a set of n vertices. Fix an arbitrary subset $S \subset V$ such that $|S| = k$.*
2. *Add edges arbitrarily inside S such that the resulting graph is a connected d-regular bipartite graph. Let S_1, S_2 denote the bipartite components.*
3. *For each pair of vertices in $S \times (V \setminus S)$, add an edge independently with probability p.*
4. *For each pair of vertices in $(V \setminus S) \times (V \setminus S)$, add an edge independently with probability p.*

For planted cliques, a lot of work has been done in the special case of $p = 1/2$. However, people have studied other problems such as the planted bisection problems [24], and exact recovery problems in Stochastic Block Models [2] in the harder $p = o(1)$ regimes. Therefore, we also aim to solve our problem in $p = o(1)$ regimes.

[1] For each pair of vertices, an edge is added independently with probability p.

We note that this problem is a generalization of the planted independent set and the planted balanced biclique problem. For $d = 0$, it reduces to recovering a planted independent set and hence we do not expect efficient algorithms for $k = o(\sqrt{n})$ [28, 11]. For $k = \Omega(\sqrt{n})$, both these special cases i.e the planted independent set problem [6, 26], and the planted balanced biclique problem [46] admit a polynomial-time recovery algorithm. So it is natural to consider $k = \Omega(\sqrt{n})$ as a benchmark for recovery and look for algorithms in this regime. The other consideration for interesting regimes to study the problem comes by viewing this problem as a special case of the densest k-subgraph (DkS) problem. When $d \gg pk$, the problem can be viewed as the densest k-subgraph (DkS) and for $d \ll pk$, the problem can be viewed as sparsest k-subgraph problem (studying the complement of this graph would be an instance of DkS problem). However, this general DkS problem is information-theoretically unsolvable for $d = pk$ [18]. Formally, this follows from Theorem 2.1 in the work [18], by setting $d = qk = pk$ and setting $r = 1$ where q is the edge probability within the vertices of planted subgraph and a p is the edge probability when at least one of the vertex does not belong to the planted subgraph and r is the number of clusters. Therefore we focus our attention to the case when $d \approx pk$ (also including $d = pk$). In our problem, we can hope to use the specifics of the bipartite structure in hand and recover the planted set exactly.

1.1 Our models and results

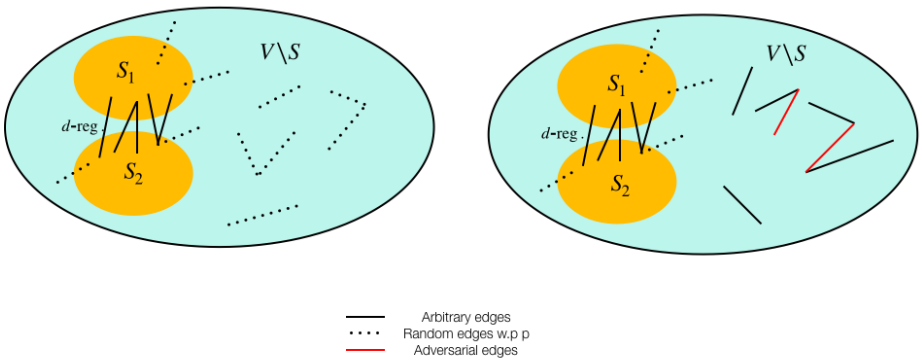


Figure 1 Random Planted model Definition 1 (left) and semi-random model Definition 2 (right).

We start by introducing our semi-random model which attempts to robustify the random planted model from Definition 1.

▶ **Definition 2** (Semi-random model). *Fix n, k, d, p, we now describe how a graph G from our semi-random model is generated,*

1. *Let V be a set of n vertices. Fix an arbitrary subset $S \subset V$ such that $|S| = k$.*
2. *Add edges arbitrarily inside S such that the resulting graph is a connected d-regular bipartite graph. Let S_1, S_2 denote the bipartite components.*
3. *For each pair of vertices in $S \times (V \setminus S)$, add an edge independently with probability p.*
4. *Arbitrarily add edges in $(V \setminus S) \times (V \setminus S)$ such that smallest eigenvalue of the matrix $\left(A_{(V\setminus S)\times(V\setminus S)} - p\mathbb{1}_{V\setminus S}\mathbb{1}_{V\setminus S}^T\right)$ is greater than $-((1/2-\alpha)/(1/2+\alpha))\,d$ where α is a small[2] positive constant (throughout this paper we assume $\alpha \leq 1/6$).*
5. *Allow a monotone adversary to add edges in $(V \setminus S) \times (V \setminus S)$ arbitrarily.*

[2] Note that the smaller the value of α, the weaker is this assumption.

▶ **Observation 3.** *Definition 2 also captures Definition 1; since in the case when $V \setminus S$ is chosen to be a $G_{(n-k),p}$ random graph, $\left(A_{(V\setminus S)\times(V\times S)} - p\mathbb{1}_{V\setminus S}\mathbb{1}_{V\setminus S}^T\right) = A_{(V\setminus S)\times(V\setminus S)} - \mathbb{E}\left[A_{(V\setminus S)\times(V\setminus S)}\right]$, and therefore the smallest eigenvalue of $\left(A_{(V\setminus S)\times(V\times S)} - p\mathbb{1}_{V\setminus S}\mathbb{1}_{V\setminus S}^T\right)$ is greater than $-2\sqrt{n}$ (as follows from the work [63]).*

Models stronger than *random planted models* have also been considered in the literature for planted problems. The work [26] studies the planted clique problem in what they call the "sandwich model". The model is constructed as per the random planted model in Definition 1, but an adversary is allowed to act on the top of that in a fashion similar to step 5 of Definition 2.

The work [24] introduced a strong adversarial *semi-random* model (referred to as the *Fiege and Kilian* model). They gave recovery algorithms for the planted clique ($k = \Omega(n)$ regimes) and for the planted bisection and planted k-coloring in this model. The work [54] further shows that one can recover the planted clique for $k = \Omega_p\left(n^{2/3}\right)$ [3] in [24] model.

In the Feige-Kilian model, step 4 allows for any arbitrary graph in $(V \setminus S) \times (V \setminus S)$. However, with no further assumptions on graph induced on $V \setminus S$, even for the special case of planted independent set problem ($d = 0$), the best known algorithm [54] works only for $k = \Omega_p\left(n^{2/3}\right)$. However, since our benchmark is $k = \Omega\left(\sqrt{n}\right)$, we look at a model with stronger assumptions than the Feige-Kilian model. In order to uniquely identify the planted graph, we need to assume that $V \setminus S$ is far from having any induced bipartite subgraphs of degree at least d. Our condition in step 4 implies that this indeed holds. This is because if the smallest eigenvalue is greater than $-d/2 + 2\sqrt{n}$, the graph is indeed far from having an induced bipartite subgraph of smallest degree d. Since otherwise, a vector having entries 1 for one side of the bipartition and -1 on the other side and 0 elsewhere achieves a Rayleigh Quotient of value $-d$ (and hence the smallest eigenvalue is at most $-d$).

We now present our main result which holds for both the random planted model (Definition 1) and semi-random model (Definition 2).

▶ **Theorem 4.** *For n, k, d, p satisfying $k = \Omega_p\left(\sqrt{n \log n}\right)$ and $p = \Omega\left(\log k/k\right)^{1/6}$ and $d \geq 2pk/3$, there exists a deterministic algorithm that takes as input an instance generated by Definition 2, and recovers the arbitrary planted set S exactly, in polynomial time and with high probability (over the randomness of the input).*

Achieving exact recovery for $k = \Omega_p\left(\sqrt{n}\right)$ is still an open problem. To the best of our knowledge, nothing is known about this problem in full generality. For the planted clique problem, recovery for $k = \Omega\left(\sqrt{n \log n}\right)$ is trivial [43]. However, such techniques don't work for our problem when $d = pk$. We prove Theorem 4 by showing that an SDP relaxation for the problem is integral, by constructing an optimal dual solution. We give an outline of the proof in Section 1.4. We leave the proof of a formal version of this theorem to the full version of the paper.

Our proofs use the spectral properties of bipartite graphs and random graphs to show the existence of an optimal dual solution having large rank. Our main technical contribution is a new approach for constructing a dual solution where we *calibrate the eigenvectors* of the adjacency matrix to be the eigenvectors of the dual matrix. We believe that this approach may have applications to other recovery problems in semi-random models as well.

[3] Ω_p hides $\mathsf{poly}\left(1/p\right)$ factors.

▶ **Theorem 5.** *For n, k, d, p, satisfying $k = \Omega_p(\sqrt{n})$, there exists a deterministic algorithm that takes as input an instance generated as per Definition 1, and recovers the arbitrary planted set S exactly with high probability (over the randomness of the input) in time exponential in the number of small eigenvalues of the adjacency matrix (eigenvalues smaller than $-d/2 + 2\sqrt{n}$) of the graph induced on S.*

We leave the proof of a formal version of this theorem to the full version of the paper.

▶ **Observation 6.** *For and many special classes of instances such as, (i) when the probability $p = \Omega(1)$, (ii) when the planted graph is a complete bipartite graph like in the balanced biclique problem (iii) when the planted bipartite graph is a d-regular random graph or (iv) more generally when the planted graph is a d-regular expander graph; the number of these small eigenvalues is a constant in the regimes of $d = \Omega(pk)$ and Theorem 5 allows efficient recovery (running time of the algorithm is polynomial in n).*

1.2 Related Work

Odd Cycle Transversal problem

The odd cycle transversal problem asks to find the smallest set of vertices in the graph such that the set has an intersection with every odd cycle of the graph. Removing these vertices will result in a bipartite graph, and hence this problem is equivalent to finding the largest induced bipartite graph. Owing to the hereditary nature of the bipartiteness property, the problem is NP-hard, as follows from the work of Yannakakis [64]. The work [64] shows that for a broad class of problems that have a structure that is hereditary on induced subgraphs, finding such a structure is NP-Complete. The optimal long code test by Khot and Bansal [10] rules out any constant factor approximation for this problem. On the algorithmic front, casting the problem as a 2-CNF deletion problem, [4] gives a reduction to the min-multicut problem. This reduction gives us an $\mathcal{O}(\log n)$ approximation due to the work [30], which was further improved to $\mathcal{O}\left(\sqrt{\log n}\right)$ in the work [3]. The work [32] gives an efficient randomized algorithm that removes only $\mathcal{O}\left(n\sqrt{\mathsf{OPT}\log d}\right)$ vertices where d is the bound on the maximum degree of the graph and OPT denotes the fraction of vertices in the optimal set. They also give a matching (up to constant factors) Unique Games hardness for certain regimes of parameters.

The problem is equivalent to finding the largest 2-colorable subgraph of a given graph and is known as the partial 2-coloring problem. The work [33] studies the problem in the Feige-Kilian semi-random model [24], where a 2-colorable graph of size $(1-\varepsilon)n$ is planted. They give an algorithm that outputs a set $\mathcal{S}'$ such that $|\mathcal{S}'| \geq \left(1 - \varepsilon c/p^2\right) n$ for $p = \Omega\left(\sqrt{\log n/n}\right)$ and $\varepsilon \leq p^2$ where c is a positive constant. Their algorithm is a partial recovery algorithm and works for the regimes when ε is small. Our results in Theorem 4 hold when $1-\varepsilon$ is small and give complete recovery for a large range of p. However, since our model in Definition 2 makes stronger assumptions than the [24] model, we don't make any comparisons.

Balanced Biclique problem

In the balanced complete bipartite subgraph problem (also called the balanced biclique problem), we are given a graph on n vertices and a parameter k, and the problem then asks whether there is a complete bipartite subgraph that is balanced with k vertices in each of the bipartite components. The problem was studied when the underlying graph is a bipartite graph, and shown to be NP-complete by a reduction from the CLIQUE problem in

the works [29, 38]. They additionally note that the balanced constraint is what makes the problem hard. If we remove the balanced constraint, the problem can be reduced to finding a maximum independent set in a bipartite graph. The latter problem admits a polynomial-time solution using the matching algorithm. The work [25] shows that this problem of finding a maximum balanced biclique is hard to approximate within a factor of $2^{(\log n)^{\delta}}$ for some $\delta > 0$, under the assumption that $\mathsf{3SAT} \notin \mathsf{DTIME}\left(2^{n^{3/4+\varepsilon}}\right)$ for some $\varepsilon > 0$. Recently, the work [53] showed that one cannot find a better approximation than $n^{1-\varepsilon}$, assuming the *Small Set Expansion Hypothesis* and that $\mathsf{NP} \nsubseteq \mathsf{BPP}$ for every constant $\varepsilon > 0$.

A related problem is the maximum edge biclique problem, where we are asked to find whether G contains a biclique with at least k edges. This problem was also shown to be NP-hard in the work [58].

Given these intractability results for general graphs, there has been some success in special classes of graphs. In graphs with constant arboricity, the work [23] gives a linear time algorithm that lists all maximal complete bipartite subgraphs. In a degree bounded graph, the work [60] gives a combinatorial algorithm for the balanced biclique problem that runs in time $\mathcal{O}\left(n2^{d}\right)$. Another systematic approach, however, is to consider planted and semi-random models for the problem. In the work [46], they study the planted version of the problem, which, they call "hidden biclique problem". Their model is similar to our model in Definition 1; however, we consider an arbitrary d-regular bipartite graph instead of a complete bipartite graph. They give a linear-time combinatorial algorithm that finds the planted hidden biclique with high probability (over the randomness of the input instance) for $k = \Omega\left(\sqrt{n}\right)$. Their algorithm builds on the "Low Degree Removal" algorithm, due to Feige and Ron [27] which finds a planted clique in linear time.

Graph problems in Semi-random and Pseudorandom models

A wide variety of random graph models and their relaxations have been a rich source of algorithmic problems on graphs. Alon and Kahale [5] sharpened the results of Blum and Spencer [14] and gave algorithms that recover a planted 3-coloring in a natural family of random 3-colorable instances. [44] extended this result and showed how to recover a 3-coloring when the input graph is pseudorandom (has some mild expansion properties) and is known to admit a random like 3-coloring. A unified spectral approach by McSherry [55] gives a single shot recovery algorithm for many problems in these random planted models. One can use the [55] framework to recover a planted random bipartite graph; however, it is not known if it will work if S is an arbitrary bipartite graph.

On the other side, we have semi-random models. Notably, the Feige-Kilian model [24] is one of the strongest semi-random models. In [24], they also give recovery algorithms for planted clique, planted k-colorable, and planted bisection problem in this model. In [54], they give a recovery algorithm for the independent set problem for large regimes of parameters. The work [40] generalizes these results to r-uniform hypergraphs in this model. There are other works [51, 52, 49, 50] that study graph partitioning in semi-random models.

A host of work has been done in various random and semi-random models for the more general densest k-subgraph problem. The works by Hajek, Wu, and Xu [34, 35, 36] study the problem when the planted dense subgraph is random and gives algorithms for exact recovery using SDP relaxations for some range of parameters. They complement these results by providing information-theoretic limits for regimes where recovery is impossible. The work by [13] studies this problem when the planted graph is arbitrary. They analyze an SDP-based method to distinguish the dense graphs from the family of $G_{n,p}$ graphs when $k \geq \sqrt{n}$. The work [39] studies the problem of densest k-subgraph in some semi-random model and gives a partial recovery algorithm for some regimes of d, k, n, p.

SDP has been the tool of choice for exact recovery in semi-random models. Starting from the fundamental works of exact recovery for the planted clique problem [26], for the planted bisection problem [24], for Stochastic Block Models [2] etc., (and many other works as have been mentioned above), are based on SDP relaxations. A natural way to analyze these SDP relaxations is by constructing an optimal dual solution to prove integrality of the primal relaxation. This idea has been explored in the works of [24, 20, 13, 1, 2, 49], to state a few. We note that the task of constructing an optimal dual solution is problem-specific, and there is no generic way of doing this.

1.3 Preliminaries

We start with some essential notation to understand the proof overview and review some well-known facts about random perturbation matrices. Then, we write our SDP relaxation to the problem and the accompanying dual SDP. We follow this up with a discussion on some well known tools from spectral graph theory such as the *threshold rank* and *spectral embedding*. We will build on these ideas in our Proof Overview Section 1.4 to show that the primal SDP is an optimal one and the primal matrix is a rank-one matrix.

1.3.1 Notation

We let $[M]_{n\times n}$ denote a matrix M of size $n \times n$. For some set of indices $R_1, R_2 \subseteq [n]$, $M_{R_1\times R_2}$ denotes a matrix of size $n \times n$ constructed out of matrix M of size $n \times n$ by copying the entries for $(i, j) \in R_1 \times R_2$ and setting rest of the entries to be 0. We let $M|_{R_1\times R_2}$ denote the matrix of size $|R_1| \times |R_2|$ constructed from a matrix M of size $n \times n$ by taking rows corresponding to R_1 and columns corresponding to R_2. The eigenvalues of a matrix M are sorted as $\lambda_1(M) \leq \lambda_2(M) \leq \ldots \leq \lambda_n(M)$. We will drop the matrix M wherever it is clear from the context. The eigenvectors are also sorted by their corresponding eigenvalues.

1.3.2 Spectral bounds on Perturbation matrices

We let A denote the adjacency matrix of the graph obtained using Definition 1. We can express the matrix A as sum of "simpler" matrices,

$$A = A_{S\times S} + A_{V\setminus S\times V\setminus S} + p\left(\mathbb{1}\mathbb{1}^T - \mathbb{1}_S\mathbb{1}_S^T - \mathbb{1}_{V\setminus S}\mathbb{1}_{V\setminus S}^T\right) + R \quad \left(R_{ij} \stackrel{\text{def}}{=} A_{ij} - \mathbb{E}[A_{ij}]\right) \tag{1}$$

where $A_{S\times S}$ represents the matrix corresponding to the planted bipartite graph, the term $p\left(\mathbb{1}\mathbb{1}^T - \mathbb{1}_S\mathbb{1}_S^T - \mathbb{1}_{V\setminus S}\mathbb{1}_{V\setminus S}^T\right)$ is the expected adjacency matrix for the random graph and R as defined above is the perturbation matrix corresponding to the random part of the graph.

▶ **Proposition 7.** *For the perturbation matrix R as defined in equation* (1) *we have that $\|R\| \leq 2\sqrt{n}$ almost surely.*

Proof. R is a symmetric random matrix and the entries R_{ij} can be treated as random variables, bounded between -1 and 1, with expectation 0 and variance $p(1-p) \leq 1/4$. Also the entries R_{ij} are independent and hence, by Theorem 1.1 in the work [63], we have $\|R\| \leq 2\sqrt{n}$ almost surely. ◀

1.3.3 SDP Relaxation

Our main results are based on analyzing the following SDP relaxation SDP 8. We construct its dual SDP 9.

▶ **SDP 8** (Primal).

$$\min \sum_{\{i,j\}\in E} 2\langle \mathbf{x}_i, \mathbf{x}_j\rangle$$

subject to

$$\sum_{i\in V} \|\mathbf{x}_i\|^2 = 1 \tag{2}$$

$$\|\mathbf{x}_i\|^2 \le 1/k \qquad \forall i \in V \tag{3}$$

$$\langle \mathbf{x}_i, \mathbf{x}_j\rangle \le 0 \qquad \forall \{i,j\} \in E\,. \tag{4}$$

▶ **SDP 9** (Dual).

$$\max\ \beta - \sum_{i\in V} \gamma_i$$

subject to

$$Y = A - \beta I + k\sum_{i\in V} \gamma_i D_i + \sum_{\{i,j\}\in E} B_{ij}\left(\mathbb{1}_{ij} + \mathbb{1}_{ji}\right) \tag{5}$$

$$B_{ij} \ge 0, \qquad \forall \{i,j\} \in E \tag{6}$$

$$Y \succeq 0\,. \tag{7}$$

In SDP 9, the Lagrange multipliers β_i's,γ_i's and B_{ij}'s are our dual variables and Y is the dual SDP matrix. By $\mathbb{1}_{ij}$ we mean an indicator matrix which is one for (i,j) entry and zero elsewhere. Similarly, D_i is an indicator matrix which is one for (i,i) entry and zero elsewhere. For clarity, we will denote $\sum_{\{i,j\}\in E} B_{ij}\left(\mathbb{1}_{ij} + \mathbb{1}_{ji}\right)$ by a matrix B.

Intended solution

We denote the primal SDP matrix by X and let $\mathbf{x}_i$ denote the vector corresponding to vertex i such that $X_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j\rangle$.

Our intended integral solution to the SDP is $X = \mathbf{g}\mathbf{g}^T$, where $\mathbf{g} \in \mathbb{R}^n$ s.t $g_i = 1/\sqrt{k}$ for $i \in S_1$, $g_i = -1/\sqrt{k}$ for $i \in S_2$ and 0 otherwise. This solution is obtained by setting,

$$\mathbf{x}_i^* = \begin{cases} \hat{e}/\sqrt{k} & \text{if } i \in S_1 \\ -\hat{e}/\sqrt{k} & \text{if } i \in S_2 \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where $\hat{e}$ is some unit vector.

Weak Duality for fixing dual variables

Let $\mathsf{SDPOPT}(G)$ denote the optimal value of the primal SDP; then from the proposed integral solution we have that,

$$\mathsf{SDPOPT}(G) \le -2\sum_{\{i,j\}\in E} \left\langle \mathbf{x}_i^*, \mathbf{x}_j^*\right\rangle = \left\langle A, \mathbf{g}\mathbf{g}^T\right\rangle = \mathbf{g}^T A\mathbf{g} = -d\,.$$

For any feasible solution to the dual SDP 9, by weak duality, we know that

$$\beta - \sum_{i\in V} \gamma_i \le \mathsf{SDPOPT}(G) \le -d\,.$$

We note that the upper bound is achievable by setting $\beta = -d$ and $\gamma_i = 0, \forall i \in V$.

We will show later that the remaining dual variables B_{ij}'s can be chosen in a way that the choice of $\beta = -d$ and $\gamma_i = 0, \forall i \in V$ yields a feasible dual solution.

▶ **Fact 10** (Folklore, also see Lemma 2.3 in [49]). *The primal solution $X = \mathbf{g}\mathbf{g}^T$ is the unique solution to SDP 8 if there exists a dual matrix Y such that it satisfies constraints in SDP 9, with $\beta = -d$ and $\gamma_i = 0, \forall i \in V$ and having* $\mathsf{rank}(Y) = n - 1$ *(i.e. $\lambda_2(Y) > 0$).*

1.3.4 Threshold rank eigenvectors

▶ **Definition 11** (Threshold rank of a graph). *For $\tau \in [0, d]$, we define threshold rank of a graph with adjacency matrix G (denoted by* $\mathsf{rank}_{\leq -\tau}(G)$*) as,*

$$\mathsf{rank}_{\leq -\tau}(G) = |\{i : \lambda_i(G) \leq -\tau\}| \ .$$

We let $P_{-\tau} = \left\{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(L_{-\tau})}\right\}$ (the bottom $L_{-\tau}$ vectors) denote the set of orthonormal eigenvectors of $A|_{S\times S}$ with eigenvalues smaller than the threshold $-\tau$, breaking ties arbitrarily where $L_{-\tau} = \mathsf{rank}_{\leq -\tau}\left(A|_{S\times S}\right)$. We call these vectors as *τ-threshold rank eigenvectors* of $A|_{S\times S}$. Next, we recall a well known fact about the threshold rank of a graph.

▶ **Fact 12** (Folklore). $\mathsf{rank}_{\leq -\tau}\left(A|_{S\times S}\right) \leq \dfrac{kd}{2\tau^2}$.

Proof. Since, $A|_{S\times S}$ is the adjacency matrix of a bipartite graph, it's eigenvalue spectrum is symmetric around 0. Therefore the number of eigenvalues with absolute value greater then or equal to τ is given by $2\,\mathsf{rank}_{\leq -\tau}\left(A|_{S\times S}\right)$ and are bounded as,

$$2\tau^2 \mathsf{rank}_{\leq -\tau}(A|_{S\times S}) \leq \sum_i \lambda_i^2(A|_{S\times S}) \leq \left\|A|_{S\times S}\right\|_F^2 = kd. \qquad \blacktriangleleft$$

We note that a similar notion of threshold rank has appeared in other works [8, 9, 12, 31] etc.

1.3.5 Spectral embedding vectors

▶ **Definition 13** (Spectral embedding vectors). *Given the planted bipartite graph S and the matrix of bottom $L_{-\tau}$ orthonormal eigenvectors $W_{-\tau}^T = \begin{bmatrix}\mathbf{v}^{(1)} & \mathbf{v}^{(2)} & \ldots & \mathbf{v}^{(L_{-\tau})}\end{bmatrix}$, we define the spectral embedding of a vertex $i \in S$ as the $L_{-\tau}$-dimensional vector given by $\mathbf{w}^{(i)} = W_{-\tau}\mathbf{e}_i$ where $\mathbf{e}_i$ is a vector with one in the i^{th} coordinate and zero elsewhere.*

Informally, these are the vectors obtained by looking at the subspace of the columns of $W_{-\tau}^T$ where the vertex i is mapped to the i^{th} column of $W_{-\tau}^T$. These spectral embedding vectors have been explored in various works on graph partitioning as [57, 45, 48] etc. It is known that these spectral embedding vectors are "well spread", formally referred to as being in an isotropic[4] position. We define these set of vectors to be in an isotropic position if $\sum_{i\in S}\mathbf{w}^{(i)} = 0$ and $\sum_{i\in S}\mathbf{w}^{(i)}\mathbf{w}^{(i)^T} = I$ where I is an $L_{-\tau}\times L_{-\tau}$ sized identity matrix. The condition that $\sum_{i\in S}\mathbf{w}^{(i)}\mathbf{w}^{(i)^T} = I$ can equivalently be written as $\sum_{i\in S}\left\langle \mathbf{y}, \mathbf{w}^{(i)}\right\rangle^2 = 1, \forall \mathbf{y}$ with $\|\mathbf{y}\|_2 = 1$.

▶ **Lemma 14** (Folklore). *The spectral embedding vectors are in an isotropic position.*

For a proof, we refer the reader to the work [47].

[4] Typically, isotropicity is a property of distribution. We say a distribution is isotropic if the mean of a random variable sampled from the distribution is zero and it's covariance matrix is an identity matrix.

1.4 Proof Overview

For the sake of simplicity, we will assume that the graph is sampled as per the random planted model (Definition 1). We will also allow an action of a monotone adversary (as in step 5) on this model; but we analyze its action separately (in Section 1.4.6). The main ideas for the semi-random model (Definition 2) are essentially the same, and the additional steps to handle them is just a technical adjustment.

In this section we give an overview of how Theorem 4 and Theorem 5 are proven. Because of page limits, the detailed proofs are deferred to the full version of the paper. In what follows, we present the main ideas which go inside these proofs.

1.4.1 Spectral Approaches

We start with some natural spectral approaches for recovering the planted set. These approaches have found some success, e.g. in recovering planted cliques/independent sets, planted bisection, planted k-colorable graphs (refer work [55] for details). We recall from our earlier discussion, that the interesting regimes for this problem are $k = \Omega(\sqrt{n})$ and $d \approx pk$.

Detecting planted bipartitions and why it is easy

We note that the *detection problem* i.e. detecting the presence of bipartite graph as constructed in the random planted model (Definition 1) against the null hypothesis of Erdős-Rényi graph $G_{n,p}$, is easy when $k = \Omega(\sqrt{n})$. Formally one notes that given two distributions

$$H_0 : G \sim G(n,p) \text{ against } H_1 : G \sim G(n,k,d,p) \text{ as per Definition 1,}$$

the spectral test, which outputs H_1 when $\lambda_1(G) \leq -d$ and H_0 otherwise, is correct almost surely for $d \approx pk$ and $k \geq c\sqrt{n}/p$ where $c > 0$ is a large enough constant. This is because for a $G_{n,p}$ graph, the smallest eigenvalue is greater than $-2\sqrt{n}$ almost surely (Claim 7), while for a graph with planted bipartite subgraph, the smallest eigenvalue is smaller than $-d$ since the vector $\mathbb{1}_{S_1} - \mathbb{1}_{S_2}$ already achieves Rayleigh Quotient of value $-d$.

The challenges in exact recovery

However, as expected, the exact recovery problem is more challenging. There are some works that look at these planted problems on an individual basis ([15],[6]). They typically rely on the spectral bounds of perturbation matrices and the framework of Davis-Kahan theorem (refer [61]) to identify eigenvector(s) indicating the planted set. However, we need a sufficient eigengap[5] to apply these results from perturbation theory. Since our planted graph S in the random planted model is an arbitrary bipartite graph, it can have any number of eigenvalues close to the smallest eigenvalue $-d$ and hence we may not have such an eigengap.

A unified spectral framework for random planted models was given by McSherry [55] (further refined in the work [62]). Here, one can check that we cannot satisfy the conditions in Observation 11 of this work [55] if the planted set has size $o(n)$. Again, the reason is because the planted bipartite graph is arbitrary. Since the planted bipartite graph can have arbitrary rank we cannot get the constants γ_1 in [55] to be small enough to recover in $o(n)$ regimes. It is also easy to verify that this framework works if the planted bipartite graph is also a random graph, for regimes of $k = \Omega(\sqrt{n})$ and $d \approx pk$ (say by choosing edge probability for the random planted bipartite graph as $p' = 2p$).

[5] Typically around the bottom eigenvector(s) or the top eigenvector(s).

A subspace enumeration style approach

Another spectral approach, inspired from the works of [42, 8, 41, 44] is to apply the *subspace enumeration* technique to recover a large fraction of planted set S. Here we first identify that the vector $\mathbf{u} = \mathbb{1}_{S_1} - \mathbb{1}_{S_2}$ has a large projection on the space spanned by τ'-threshold rank eigenvectors of A (for choice of $\tau' = d/2$). Note that this vector $\mathbf{u}$ identifies the planted set (as well as the planted bipartition), and therefore we call it the *signed indicator vector*. We then do a standard ε-net construction to find a vector $\mathbf{y}$ close to $\mathbf{u}$ and use $\mathbf{y}$ to recover a large fraction of planted set S. We can recover the remaining set of vertices by an argument due to the work [33], where they distinguish vertices by the size of matching in induced neighborhoods. Putting all this together, we can prove Theorem 5.

The running time of the procedure described above is exponential in $L_{-\tau}$ where $L_{-\tau} = \mathcal{O}(1/p)$ for $\tau = \Omega(d)$ (follows from Fact 12). Therefore, for many special classes of instances such as, (i) when the probability $p = \Omega(1)$ and $d = \Omega(pk)$, (ii) when the planted graph is a complete bipartite graph (this is the balanced biclique problem) and $d = \Omega(pk)$, (iii) when the planted bipartite graph is d-regular random graph for $d = \Omega(pk)$ or (iv) more generally when the planted graph is a d-regular expander graph for $d = \Omega(pk)$ we have $L_{-\tau} = \mathcal{O}(1)$ and this already gives us a polynomial-time algorithm.

However, as stated earlier, we want to solve the problem in $p = o(1)$ regimes. To accomplish this, we shift our focus to the SDP formulation we mentioned in SDP 8. Also, for other problems in this literature (planted clique, planted bisection, planted k-colorable, Stochastic Block models etc), only SDP's have provable guarantees of working in the presence of such monotone adversaries (refer Chapter 10 in [59] for more intuition on this).

1.4.2 Traditional SDP Analysis

Now we overview our SDP-based approach to solving the problem. We will see that the difficulties in the spectral approach will translate to showing the feasibility of the dual SDP solution. However, we have more freedom here since we have the dual variables to work with and we can use them and try to enforce the optimality of the dual solution.

Characterizing dual variables through optimality conditions

A standard technique for analyzing SDP relaxations (like our SDP 8) is to show optimality by constructing a dual solution that matches the $\mathsf{SDPOPT}(G)$ value of the primal in a manner that the dual matrix Y is positive semi-definite and has rank $n-1$, (see Fact 10).

These impose a "wish list" of desired conditions, which can be used to characterize our dual variables

1. $\beta = -d$ (Optimal objective)
2. $\langle \mathbf{g}\mathbf{g}^T, Y \rangle = 0$ (Complementary slackness)
3. $Y \succeq 0$ (Dual feasibility)
4. $\lambda_2(Y) > 0$ (Strong duality).

Using weak duality we set $\beta = -d$ and $\gamma_i = 0, \forall i \in V$ to match the optimal primal objective value of $\mathsf{SDPOPT} = -d$. We expand upon the complementary slackness condition as,

$$\langle \mathbf{g}\mathbf{g}^T, Y \rangle = \mathbf{g}^T Y \mathbf{g} = \mathbf{g}^T (A + dI)\mathbf{g} + \mathbf{g}^T B \mathbf{g} = 0 + \mathbf{g}^T B \mathbf{g} = \mathbf{g}^T B \mathbf{g}.$$

Therefore the complementary slackness condition gives us that,

$$\sum_{i \in S} \sum_{\substack{j \in S \\ \{i,j\} \in E}} B_{ij} = 0 \tag{9}$$

and since the SDP dual requires that $B_{ij} \geq 0$, it implies that $B_{ij} = 0$ for all $(i, j) \in E(S_1, S_2)$. Now using the characterization of dual variables from conditions (1) and (2), one tries to show the feasibility of the dual and the strong duality rank condition. Typically, this characterization turns out to be rather weak. So, we refer to the dual variables set so far (ensuring condition (1) and (2) are satisfied) as *weakly characterized*.

Showing optimality of dual solution through weakly characterized dual variables

For certain problems in semi-random models, such as the planted clique problem [26], community detection in SBM [2], the weak characterization above suffices. We are able to show that the weakly characterized dual solution satisfies conditions (3) and (4). This is typically done by invoking some standard results for random matrix bounds and concentration inequalities. In our setup, satisfying condition (3) requires that

$$\lambda_{\min}(Y) \geq 0 \text{ which is implied if } \lambda_{\min}(A) + d + \lambda_{\min}(B) = \lambda_{\min}(A) + d \geq 0\,. \tag{10}$$

However, in our random planted model, the smallest eigenvalue of A can be smaller than $-d - 2\sqrt{pn}$ and condition (3) may not hold (as per choice of B_{ij}'s dictated from equation (9)). Thus we need a stronger characterization of dual variables to satisfy the conditions (3) and (4). In our problem, we need to make use of the large number of unused dual variables $B'_{ij}s$ for $\{i, j\} \in E \cap \{(V \times V) \setminus (S \times S)\}$

Guessing/Constructing the dual certificate

Now we discuss an approach of making the dual matrix satisfy conditions (3) and (4) by guessing the dual variables thus giving an explicit setting of dual variables. This is typically done by assigning some sort of meaning to dual variables and guessing their values based on the input instance. This approach has found reasonable success in other recovery problems like the planted bisection problem [24], coloring semi-random graphs [20], decoding binary node labels from censored edge measurements [1], and planted sparse vertex cuts [49].

Therefore, we may expect to guess a nice setting of dual variables that satisfy equation (10). However, if one takes a deeper look at this approach, the task again reduces to applying results from perturbation theory. Again, such an approach would work if the planted bipartite graph were also a random graph or an expander, since there would only be a single eigenvector whose corresponding eigenvalue disobeys equation (10), and one could choose the dual variables constructively to handle it and make it satisfy condition (3).

However, for an arbitrary planted bipartite graph, we can have a lot of eigenvalues in the interval $[-d, -d - 2\sqrt{n}]$ (and hence the entire graph A can have a lot of eigenvalues in the interval $[-d - 2\sqrt{n}, -d]$). Therefore, we need a more principled approach to deal with the corresponding eigenvectors of the planted graph having eigenvalue close to $-d$ (as we pointed out earlier, there can be $\mathcal{O}(1/p)$ such eigenvectors).

1.4.3 Calibrating the eigenvectors

Now we present our approach towards satisfying conditions (3) and (4), which is to *calibrate the eigenvectors*. We will see that, this calibration will further complicate our requirements on the dual variables, however we will argue in Section 1.4.4 on how we manage that.

Obtaining optimality of Primal SDP by assuming existence of a certifying B

It is now clear that our Achilles' heel are the eigenvalues (and corresponding threshold rank eigenvectors[6]) of the planted graph in the interval $[-d, -d - 2\sqrt{n}]$. If we were allowed to ignore these vectors it's easy to see that equation (10) and hence condition (3) holds.

Our core idea is to extend (by padding with 0's so that they are the right length) the threshold rank eigenvectors of $A|_{S\times S}$ to be the eigenvectors of the dual matrix Y. Recall, the eigenvalues of $A|_{S\times S}$ lie in the interval $[-d, d]$. Now take a threshold rank eigenvector of $A|_{S\times S}$ (say with eigenvalue λ_l). We wish to calibrate such a threshold rank eigenvector to be an eigenvector of Y with eigenvalue $d + \lambda_l$. If we are able to achieve this calibration, we need not bother about the $2\sqrt{n}$ term since now these eigenvectors have a non-negative quadratic form[7].

The only thing at our disposal for this calibration are the unused (so far) dual variables B_{ij}'s. Denote this set of threshold rank eigenvectors of $A|_{S\times S}$ as $P_{-\tau}$. Given this set $P_{-\tau}$, achieving this calibration can be expressed as satisfying the system of equations,

$$\sum_{i\in S} \mathbf{v}_i^{(l)} \left(A_{ij} + B_{ij}\right) = 0, \quad \forall j \in V \setminus S, \forall\, \mathbf{v}^{(l)} \in P_{-\tau}\,. \tag{11}$$

- L different system of equations $\mathcal{E}_l$, one for each $\mathbf{v}^{(l)}$.
- Each system $\mathcal{E}_l$ involves $|S| \times |V \setminus S|$ variables B_{ij} where $i \in S$ and $j \in V \setminus S$.

Now, all we need is a setting of B_{ij}'s such that the system of equations is satisfied. However, this will still not be enough. We note that if this system of equations were to have a solution, we would have set some of these B_{ij} variables to non-zero values. Therefore our equation (10) would now need to be modified to showing that $\lambda_{\min}(A) + d + \lambda_{\min}(B) \geq 0$.

The way we deal with this is by noting that we can tune $\tau > 0$ apriori to be sufficiently large for this calibration such that $\lambda_{\min}(A) + d \geq \eta$ where $\eta > 0$; and now impose an additional constraint on the matrix of dual variables that $\|B\| \leq \eta$. At this point, it seems highly suspicious as whether such B_{ij}'s exist. However, if we table these considerations aside and for choice of $\tau = 2d/3$ and $\eta = \tilde{\mathcal{O}}(pk)$ we can indeed show that condition (3) and condition (4) of our "wish list" are met and we get the desired integral primal solution.

1.4.4 Setting of dual variables

In this section we show that there exists a matrix of non-negative dual variables B_{ij}'s that satisfies the system of equations (11) and $\|B\|_2 = \tilde{\mathcal{O}}(pk)$.

An LP formulation and Farkas Lemma based approach

We start by observing that the condition $\|B\|_2 = \tilde{\mathcal{O}}(pk)$ is implied by a condition that $B_{ij} \leq t = \mathcal{O}_p\left(\sqrt{\log k/k}\right), \forall (i,j) \in (S \times (V \setminus S)) \cup ((V \setminus S) \times S)$. Also, since these system of equations (11) only concerns the non-negative dual variables B_{ij}'s with $\{i,j\} \in (S \times (V \setminus S)) \cup ((V \setminus S) \times S)$, we set the rest of them to 0.

[6] For an appropriate choice of τ, which we decide later, these will be the threshold rank eigenvectors.
[7] The quadratic form of vector $\mathbf{x}$ with a matrix Y is a number given by $\mathbf{x}^T Y \mathbf{x}$

We now reorganize our collection of linear systems in (11) as follows.

- For $j \in V \setminus S$, define a system of equations $\mathcal{F}_j$.
- In all, this gives a collection of systems $\{\mathcal{F}_j\}_{j \in V \setminus S}$. Each system contains $L_{-\tau} \times |S|$ variables. In particular, the system $\mathcal{F}_j$ is expressed in the standard form $W_{-\tau}\mathbf{x} = \mathbf{b}$, where $W_{-\tau} \in \mathbb{R}^{L_{-\tau} \times k}$ is a matrix formed by stacking the vectors $\mathbf{v}^{(l)} \in P_{-\tau}$ as rows.

Fix $j \in V \setminus S$ and consider the system $\mathcal{F}_j$. The vector $\mathbf{b}$ in this system is a row vector of size $L_{-\tau} \times 1$ and has entries given by $b_l = -\sum_{i \in S} A_{ij} v_i^{(l)}, \forall l \in [L_{-\tau}]$ and $\mathbf{x}$ here is a row vector of size $k \times 1$ where the entry $x_i = B_{ij}$ (recall that we have fixed a $j \in V \setminus S$). However since B_{ij}'s are not arbitrary variables but dual variables of SDP 9, these are required to be non-negative and should only be defined for $i \in N(j)$. Since the graph on $S \times (V \setminus S)$ is random, the choice of random edges while choosing $N(j)$ (in model construction) corresponds to setting those $B_{ij} = 0$ whenever the edge is not chosen. Let $\tilde{W}_{-\tau}$ denote the submatrix after removing the columns corresponding to $i \notin N(j)$ and recall t is our upper bound on the entries of B matrix as mentioned above. We then consider the following feasibility LP formulation for this problem of finding appropriate B_{ij}'s.

▶ **LP 15.**

$$\tilde{W}_{-\tau}\mathbf{x} = \mathbf{b} \tag{12}$$

$$0 \le \mathbf{x} \le t\mathbb{1}\,. \tag{13}$$

For simplicity, consider the case $p = 1$, i.e. when $A_{ij} = 1$ for all $i \in S, j \notin S$. Then we have that for any vector $\mathbf{y} \in \mathbb{R}^{L_{-\tau}}$,

$$\mathbf{b}^T\mathbf{y} = \sum_{r \in [L_{-\tau}]} b_r y_r = -\sum_{r \in [L_{-\tau}]} \sum_{i \in S} v_i^{(r)} y_r = -\sum_{i \in S} \sum_{r \in [L_{-\tau}]} v_i^{(r)} y_r \tag{14}$$

$$= -\sum_{i \in S} \sum_{r \in [L_{-\tau}]} w_r^{(i)} y_r = -\sum_{i \in S} \left\langle \mathbf{w}^{(i)}, y \right\rangle = -\left\langle \sum_{i \in S} \mathbf{w}^{(i)}, \mathbf{y} \right\rangle = 0\,. \tag{15}$$

Using the standard variant of Farkas' Lemma, this immediately implies the existence of a solution to equation (11). However, in general, for $p < 1$, we need to do more work here.

We apply a more general version of Farkas' Lemma, and we have that satisfying this LP in the general case corresponds to showing that for some $t > 0$, the following holds.

$$\forall\, \mathbf{y} \in \mathbb{R}^{L_{-\tau}}, \forall\, \mathbf{z} \ge 0,\ \tilde{W}_{-\tau}^T\mathbf{y} + \mathbf{z} \ge 0 \implies \mathbf{b}^T\mathbf{y} + t\,\langle \mathbf{z}, \mathbb{1} \rangle \ge 0\,. \tag{16}$$

The first term in the expression, $\mathbf{b}^T\mathbf{y}$, can be expanded as in equation (14) to obtain

$$\mathbf{b}^T\mathbf{y} = -\sum_{i \in N(j)} \left\langle \mathbf{w}^{(i)}, \mathbf{y} \right\rangle \text{ and using } \left(\tilde{W}_{-\tau}^T\mathbf{y}\right)_i = \left\langle \mathbf{w}^{(i)}, \mathbf{y} \right\rangle \text{ we have } z_i \ge -\left\langle \mathbf{w}^{(i)}, \mathbf{y} \right\rangle.$$

We can give a proof by contradiction for equation (16). By contradiction there exists a $\mathbf{y}$ and a $\mathbf{z}$ such that $\mathbf{b}^T\mathbf{y} + t\,\langle \mathbf{z}, \mathbb{1} \rangle < 0$. We choose $\mathbf{z}' \le \mathbf{z}$ by setting $z_i' = \max\left\{0, -\left\langle \mathbf{w}^{(i)} \right\rangle, \mathbf{y}\right\}$ and argue that it is enough to show contradiction for $t, \mathbf{y}$ and $\mathbf{z}'$. Using the expressions for $\mathbf{b}^T\mathbf{y}$ as above, this translates to showing that

$$\sum_{i \in N(j)} \left\langle \mathbf{w}^{(i)}, \mathbf{y} \right\rangle + t \sum_{i \in N(j)} \min\left\{0, \left\langle \mathbf{w}^{(i)}, \mathbf{y} \right\rangle\right\} > 0, \tag{17}$$

has no solution. We show that equation (17) does not hold for our desired choice of $t = \mathcal{O}_p\left(\sqrt{\log k/k}\right)$.

We note that the second term in equation (17) is ≥ 0, and we wish the inequality to not hold for as small a value of t as possible; therefore, we seek an upper bound on both terms.

Structure of threshold rank/spectral embedding vectors

To upper bound the first term, it might be helpful to understand the structure of the spectral embedding vectors $\mathbf{w}^{(i)}$. Since these are intimately connected to the threshold rank eigenvectors $\mathbf{v}^{(l)}$, we use these eigenvectors to characterize them. For convenience we let $\mathbf{v}^{(l)} \in P_{-\tau}$ have unit norm, then we show that $\left\|\mathbf{v}^{(l)}\right\|_\infty = \left\|\mathbf{w}^{(i)}\right\|_\infty \leq 2/\sqrt{d}$. Since $i \in N(j)$ are sampled randomly, we can use the Hoeffding bounds to upper bound the l_∞ norm for $\sum_{i\in N(j)} \mathbf{w}^{(i)}$ and hence upper bound our first term by $\mathcal{O}_p\left(\sqrt{\log k}\right)$ with high probability. We choose our parameters such that the vectors in $P_{-\tau}$ are orthogonal to $\mathbb{1}_S$. For the embedding vectors, this translates to saying that $\sum_{i\in S} \mathbf{w}^{(i)} = 0$.

Towards bounding the second term, we use these *spectral embedding vectors*. The spectral embedding vectors are isotropic for $p = 1$ (already where we can easily show that equation (17) does not hold and we are done). However, for $p < 1$, we have $i \in N(j)$ (and corresponding embedding vectors) being sampled randomly as per $G_{n,p}$ distribution. Here, we show that by using Matrix Bernstein concentration we can get close to isotropic vectors,

$$\sum_{i\in N(j)} \left\langle \mathbf{y}, \mathbf{w}^{(i)} \right\rangle^2 \geq p/2 \quad \text{(This shows that embedding vectors are } p/2\text{-isotropic .)} \tag{18}$$

Showing existence of a solution to LP 15

Now, we look at two cases; the first case where the negative terms dominate the summand in equation (17), then we use the eigenvector structure that $\left\|\mathbf{w}^{(i)}\right\| \leq 2/\sqrt{d}$ and we are done; for the other case where the positive terms dominate, we relate the positive terms to the negative terms again using the bound we obtained from the eigenvector structure $\left\|\sum_{i\in N(j)} \mathbf{w}^{(i)}\right\| = \mathcal{O}_p\left(\sqrt{\log k}\right)$.

Therefore, we argue that we can upper bound the second term by $-\Omega_p\left(\sqrt{k}\right)$. Therefore for a choice of t as we obtained above of $t = \mathcal{O}_p\left(\sqrt{\log k/k}\right)$, equation (17) does not hold. As discussed earlier this implies that the there exists a dual such that conditions (1)-(4), equation (11) and $\|B\|_2 = \tilde{\mathcal{O}}\left(pk\right)$ holds which further implies that the primal SDP is feasible. Further, if the graph is connected; the *signed indicator vector* would be the only eigenvector with eigenvalue $-d$ (after padding to make these the eigenvectors of Y), this would be the only eigenvector of Y with eigenvalue 0. Using Fact 10, this implies that the proposed integral solution in equation (8) is the only integral solution and hence Cholesky Decomposition of our SDP matrix returns the *signed indicator vector* and thus our planted set.

1.4.5 Low degree regimes

The discussion above about SDP holds only where $d = \gamma pk$ where $\gamma \geq 2/3$. Note that this covers our interesting regimes when $d \approx pk$ where the problem is non-trivial. The other case where $d \leq 2pk/3$, can actually be trivially solved for $k = \Omega_p\left(\sqrt{n \log n}\right)$ using a degree counting argument along the lines of [43] as we discuss below.

Now we consider the regimes when $d = \gamma pk$ with $\gamma \leq 2/3$. We show that a simple algorithm that collects the bottom k degrees of the graph will work in these regimes since the vertices in S will have smaller degrees compared to vertices in $V \setminus S$.

▶ **Lemma 16.** *For $k \geq 6\sqrt{6n \log n/p}$, Algorithm 1 returns the planted set S with high probability (over the randomness of the input).*

Algorithm 1 Kucera's algorithm for recovery in low degree regimes.

Input: $G = (V, E)$, sampled as per Definition 1 with adversary as per step 5.
Output: The set of vertices in planted bipartite graph (with high probability) S.
1: Sort the degrees of the vertices in G.
2: Return $\mathcal{S}$ be the set of bottom k degrees after sorting..

Proof. For a vertex $v \in S$ the expected degree is $d + p(n-k)$. We note that this is smaller than pn since $d \leq 2pk/3$. We can upper bound the degree of v (denoted $d(v)$), with high probability (over the randomness of the input) using Chernoff bounds (Lemma 4.5, [56]) as,

$$\mathbb{P}\left[d(v) \geq d + p(n-k) + \sqrt{6pn\log n}\right] \leq \exp\left(\frac{-pn(\sqrt{6\log n}/\sqrt{pn})^2}{3}\right) = \frac{1}{n^2}.$$

Using a union bound over all $v \in S$, we have that for an any $v \in S$,

$$\mathbb{P}\left[d(v) \geq d + p(n-k) + \sqrt{6pn\log n}\right] \leq \frac{1}{n}.$$

Therefore we have with high probability (over the randomness of the input) that $d(v) \leq d + p(n-k) + \sqrt{6pn\log n}$. Similarly for a vertex $v' \notin S$ the degree can be lower bounded with high probability (over the randomness of the input) using Chernoff bounds (Lemma 4.4, [56]) as,

$$\mathbb{P}\left[d(v') \leq pn - \sqrt{6pn\log n}\right] \leq \exp\left(\frac{-pn(\sqrt{6\log n}/\sqrt{pn})^2}{2}\right) = \exp\left(-3\log n\right) = \frac{1}{n^3}.$$

Now using a union bound over all $v' \notin S$, we have with high probability (over the randomness of the input) that $d(v') \geq pn - \sqrt{6pn\log n}$. Therefore, with high probability (over the randomness of the input), the degrees differ by,

$$d(v') - d(v) \geq pk - d - 2\sqrt{6pn\log n} \geq \frac{pk}{3} - 2\sqrt{6n\log n}\,. \tag{19}$$

where we have used $d \leq 2pk/3$. It is also evident from equation (19) that for $k \geq 6\sqrt{6n\log n/p}$, with high probability (over the randomness of the input), the degree for a vertex $v \in S$ is smaller than degree of any vertex $v' \notin S$. ◀

1.4.6 Action of Adversary

Finally, we discuss the action of adversary (allowed to add edges in $(V \setminus S) \times (V \setminus S)$ for $d \geq 2pk/3$). We show that the inductive argument given by [24] also works for our case. This argument also extends to the semi-random model in Definition 11. We leave these details to the full version of the paper.

For $d \leq 2pk/3$ regimes, Algorithm 1 continues to return the planted set, since the action of adversary only amplifies the difference of degree for a vertex $v \in S$ and vertices $v' \notin S$. This argument does not extend to the semi-random model in Definition 11.

References

1 Emmanuel Abbe, Afonso S. Bandeira, Annina Bracher, and Amit Singer. Decoding binary node labels from censored edge measurements: phase transition and efficient recovery. *IEEE Trans. Network Sci. Eng.*, 1(1):10–22, 2014. doi:10.1109/TNSE.2014.2368716.

2 Emmanuel Abbe, Afonso S. Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Trans. Inform. Theory*, 62(1):471–487, 2016. doi:10.1109/TIT.2015.2490670.

3 Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF deletion, and directed cut problems. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 573–581. ACM, New York, 2005. doi:10.1145/1060590.1060675.

4 A. Agrawal, P. Klein, S. Rao, and R. Ravi. Approximation through multicommodity flow. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 726–737 vol.2, Los Alamitos, CA, USA, October 1990. IEEE Computer Society. doi:10.1109/FSCS.1990.89595.

5 Noga Alon and Nabil Kahalé. A spectral technique for coloring random 3-colorable graphs. *SIAM J. Comput.*, 26(6):1733–1748, 1997.

6 Noga Alon, Michael Krivelevich, and Benny Sudakov. Finding a large hidden clique in a random graph. In *Proceedings of the Eighth International Conference "Random Structures and Algorithms" (Poznan, 1997)*, volume 13, pages 457–466, 1998. doi:10.1002/(SICI)1098-2418(199810/12)13:3/4<457::AID-RSA14>3.3.CO;2-K.

7 Claudio Arbib and Raffaele Mosca. Polynomial algorithms for special cases of the balanced complete bipartite subgraph problem. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 30, January 1999.

8 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science—FOCS 2010*, pages 563–572. IEEE Computer Soc., Los Alamitos, CA, 2010.

9 Sanjeev Arora and Rong Ge. New tools for graph coloring. In *Approximation, randomization, and combinatorial optimization*, volume 6845 of *Lecture Notes in Comput. Sci.*, pages 1–12. Springer, Heidelberg, 2011. doi:10.1007/978-3-642-22935-0_1.

10 Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2009*, pages 453–462. IEEE Computer Soc., Los Alamitos, CA, 2009. doi:10.1109/FOCS.2009.23.

11 Boaz Barak, Samuel B. Hopkins, Jonathan Kelner, Pravesh Kothari, Ankur Moitra, and Aaron Potechin. A nearly tight sum-of-squares lower bound for the planted clique problem. In *57th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2016*, pages 428–437. IEEE Computer Soc., Los Alamitos, CA, 2016.

12 Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science—FOCS 2011*, pages 472–481. IEEE Computer Soc., Los Alamitos, CA, 2011. doi:10.1109/FOCS.2011.95.

13 Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities—an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC'10—Proceedings of the 2010 ACM International Symposium on Theory of Computing*, pages 201–210. ACM, New York, 2010.

14 Avrim Blum and Joel Spencer. Coloring random and semi-random k-colorable graphs. *J. Algorithms*, 19(2):204–234, 1995.

15 Ravi B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 280–285, 1987. doi:10.1109/SFCS.1987.22.

16 T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987. doi:10.1007/BF02579448.

17 Ted Carson and Russell Impagliazzo. Hill-climbing finds random planted bisections. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (Washington, DC, 2001)*, pages 903–909. SIAM, Philadelphia, PA, 2001.

18 Yudong Chen and Jiaming Xu. Statistical-computational tradeoffs in planted problems and submatrix localization with a growing number of clusters and submatrices. *J. Mach. Learn. Res.*, 17:Paper No. 27, 57, 2016.

19 Yizong Cheng and George M. Church. Biclustering of expression data. In Philip E. Bourne, Michael Gribskov, Russ B. Altman, Nancy Jensen, Debra A. Hope, Thomas Lengauer, Julie C. Mitchell, Eric D. Scheeff, Chris Smith, Shawn Strande, and Helge Weissig, editors, *ISMB*, pages 93–103. AAAI, 2000.

20 Amin Coja-Oghlan. Colouring semirandom graphs. *Combin. Probab. Comput.*, 16(4):515–552, 2007. doi:10.1017/S0963548306007917.

21 Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures Algorithms*, 18(2):116–140, 2001. doi:10.1002/1098-2418(200103)18:2<116::AID-RSA1001>3.0.CO;2-2.

22 M. E. Dyer and A. M. Frieze. The solution of some random NP-hard problems in polynomial expected time. *J. Algorithms*, 10(4):451–489, 1989. doi:10.1016/0196-6774(89)90001-1.

23 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inform. Process. Lett.*, 51(4):207–211, 1994. doi:10.1016/0020-0190(94)90121-X.

24 Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *Journal of Computing and System Sciences*, 63:639–671, 2001.

25 Uriel Feige and Shimon Kogan. Hardness of approximation of the balanced complete bipartite subgraph problem. Technical report, Weizmann Institute, 2004.

26 Uriel Feige and Robert Krauthgamer. Finding and certifying a large hidden clique in a semirandom graph. *Random Structures Algorithms*, 16(2):195–208, 2000. doi:10.1002/(SICI)1098-2418(200003)16:2<195::AID-RSA5>3.3.CO;2-1.

27 Uriel Feige and Dorit Ron. Finding hidden cliques in linear time. In *21st International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms (AofA'10)*, Discrete Math. Theor. Comput. Sci. Proc., AM, pages 189–203. Assoc. Discrete Math. Theor. Comput. Sci., Nancy, 2010.

28 Vitaly Feldman, Elena Grigorescu, Lev Reyzin, Santosh S. Vempala, and Ying Xiao. Statistical algorithms and a lower bound for detecting planted cliques. In *STOC'13—Proceedings of the 2013 ACM Symposium on Theory of Computing*, pages 655–664. ACM, New York, 2013. doi:10.1145/2488608.2488692.

29 Michael R. Garey and David S. Johnson. *Computers and intractability.* A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness.

30 Naveen Garg, Vijay Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25, January 1998. doi:10.1137/S0097539793243016.

31 Shayan Oveis Gharan and Luca Trevisan. Improved arv rounding in small-set expanders and graphs of bounded threshold rank, 2013. arXiv:1304.2060.

32 Suprovat Ghoshal and Anand Louis. Approximation algorithms and hardness for strong unique games. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 414–433. SIAM, 2021. doi:10.1137/1.9781611976465.26.

33 Suprovat Ghoshal, Anand Louis, and Rahul Raychaudhury. Approximation algorithms for partially colorable graphs. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, volume 145 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 28, 20. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019.

34 Bruce Hajek, Yihong Wu, and Jiaming Xu. Achieving exact cluster recovery threshold via semidefinite programming. *IEEE Trans. Inform. Theory*, 62(5):2788–2797, 2016. doi:10.1109/TIT.2016.2546280.

35 Bruce Hajek, Yihong Wu, and Jiaming Xu. Achieving exact cluster recovery threshold via semidefinite programming: extensions. *IEEE Trans. Inform. Theory*, 62(10):5918–5937, 2016. doi:10.1109/TIT.2016.2594812.

36 Bruce Hajek, Yihong Wu, and Jiaming Xu. Semidefinite programs for exact recovery of a hidden community, 2016. arXiv:1602.06410.

37 Mark Jerrum and Gregory B. Sorkin. The Metropolis algorithm for graph bisection. *Discrete Appl. Math.*, 82(1-3):155–175, 1998. doi:10.1016/S0166-218X(97)00133-9.

38 David S Johnson. The np-completeness column: An ongoing guide. *Journal of Algorithms*, 8(3):438–448, 1987. doi:10.1016/0196-6774(87)90021-6.

39 Yash Khanna and Anand Louis. Planted models for the densest k-subgraph problem. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 182 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. 27, 18. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2020.

40 Yash Khanna, Anand Louis, and Rameesh Paul. Independent sets in semi-random hypergraphs. In Anna Lubiw, Mohammad Salavatipour, and Meng He, editors, *Algorithms and Data Structures*, pages 528–542, Cham, 2021. Springer International Publishing.

41 Alexandra Kolla. Spectral algorithms for unique games. *Comput. Complexity*, 20(2):177–206, 2011. doi:10.1007/s00037-011-0011-7.

42 Alexandra Kolla and Madhur Tulsiani. Playing random and expanding unique games, 2007.

43 Ludek Kucera. Expected complexity of graph partitioning problems. *Discret. Appl. Math.*, 57(2-3):193–212, 1995. doi:10.1016/0166-218X(94)00103-K.

44 Akash Kumar, Anand Louis, and Madhur Tulsiani. Finding pseudorandom colorings of pseudorandom graphs. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 93 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 37, 12. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2017.

45 James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multi-way spectral partitioning and higher-order Cheeger inequalities. In *STOC'12—Proceedings of the 2012 ACM Symposium on Theory of Computing*, pages 1117–1130. ACM, New York, 2012. doi:10.1145/2213977.2214078.

46 Yevgeny Levanzov. On finding large cliques in random and semi-random graphs. Master's thesis, Weizmann Institute of Science, January 2018.

47 Anand Louis, Prasad Raghavendra, Prasad Tetali, and Santosh Vempala. Algorithmic extensions of Cheeger's inequality to higher eigenvalues and partitions. In *Approximation, randomization, and combinatorial optimization*, volume 6845 of *Lecture Notes in Comput. Sci.*, pages 315–326. Springer, Heidelberg, 2011. doi:10.1007/978-3-642-22935-0_27.

48 Anand Louis, Prasad Raghavendra, Prasad Tetali, and Santosh Vempala. Many sparse cuts via higher eigenvalues. In *STOC'12—Proceedings of the 2012 ACM Symposium on Theory of Computing*, pages 1131–1140. ACM, New York, 2012. doi:10.1145/2213977.2214079.

49 Anand Louis and Rakesh Venkat. Semi-random graphs with planted sparse vertex cuts: algorithms for exact and approximate recovery. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 101, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.

50 Anand Louis and Rakesh Venkat. Planted models for k-way edge and vertex expansion. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 150 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 23, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019.

51 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Approximation algorithms for semi-random partitioning problems. In *STOC'12—Proceedings of the 2012 ACM Symposium on Theory of Computing*, pages 367–384. ACM, New York, 2012. doi:10.1145/2213977.2214013.

52 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Constant factor approximation for balanced cut in the PIE model. In *STOC'14—Proceedings of the 2014 ACM Symposium on Theory of Computing*, pages 41–49. ACM, New York, 2014.

53 Pasin Manurangsi. Inapproximability of maximum edge biclique, maximum balanced biclique and minimum k-cut from the small set expansion hypothesis. In *44th International Colloquium on Automata, Languages, and Programming*, volume 80 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 79, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2017.

54 Theo McKenzie, Hermish Mehta, and Luca Trevisan. A new algorithm for the robust semi-random independent set problem. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 738–746. SIAM, 2020. doi:10.1137/1.9781611975994.45.

55 Frank McSherry. Spectral partitioning of random graphs. In *42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001)*, pages 529–537. IEEE Computer Soc., Los Alamitos, CA, 2001.

56 Michael Mitzenmacher and Eli Upfal. *Probability and computing*. Cambridge University Press, Cambridge, second edition, 2017. Randomization and probabilistic techniques in algorithms and data analysis.

57 Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 849–856. MIT Press, 2001. URL: https://proceedings.neurips.cc/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html.

58 René Peeters. The maximum edge biclique problem is NP-complete. *Discrete Appl. Math.*, 131(3):651–654, 2003. doi:10.1016/S0166-218X(03)00333-0.

59 Tim Roughgarden. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021. doi:10.1017/9781108637435.

60 A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18 Suppl 1:S136–44, 2002.

61 Roman Vershynin. *High-dimensional probability*, volume 47 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 2018. An introduction with applications in data science, With a foreword by Sara van de Geer. doi:10.1017/9781108231596.

62 Van Vu. A simple SVD algorithm for finding hidden partitions. *Combin. Probab. Comput.*, 27(1):124–140, 2018. doi:10.1017/S0963548317000463.

63 Van H. Vu. Spectral norm of random matrices. *Combinatorica*, 27(6):721–736, 2007. doi:10.1007/s00493-007-2190-z.

64 Mihalis Yannakakis. Node-and edge-deletion np-complete problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 253–264, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804355.

65 Yun Zhang. Elissa J. Chesler, Michael A. Langston: On finding bicliques in bipartite graphs: a novel algorithm with application to the integration of diverse biological data types. In *41st Hawaii International International Conference on Systems Science (HICSS-41 2008), Proceedings, 7-10 January 2008, Waikoloa, Big Island, HI, USA*, page 473. IEEE Computer Society, 2008. doi:10.1109/HICSS.2008.507.

Optimal Time-Backlog Tradeoffs for the Variable-Processor Cup Game

William Kuszmaul
Massachusetts Institute of Technology, Cambridge, MA, USA

Shyam Narayanan
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

The *p-processor cup game* is a classic and widely studied scheduling problem that captures the setting in which a p-processor machine must assign tasks to processors over time in order to ensure that no individual task ever falls too far behind. The problem is formalized as a multi-round game in which two players, a filler (who assigns work to tasks) and an emptier (who schedules tasks) compete. The emptier's goal is to minimize backlog, which is the maximum amount of outstanding work for any task.

Recently, Kuszmaul and Westover (ITCS, 2021) proposed the *variable-processor cup game*, which considers the same problem, except that the amount of resources available to the players (i.e., the number p of processors) fluctuates between rounds of the game. They showed that this seemingly small modification fundamentally changes the dynamics of the game: whereas the optimal backlog in the fixed p-processor game is $\Theta(\log n)$, independent of p, the optimal backlog in the variable-processor game is $\Theta(n)$. The latter result was only known to apply to games with *exponentially many* rounds, however, and it has remained an open question what the optimal tradeoff between time and backlog is for shorter games.

This paper establishes a tight trade-off curve between time and backlog in the variable-processor cup game. We show that, for a game consisting of t rounds, the optimal backlog is $\Theta(b(t))$ where

$$b(t) = \begin{cases} t & \text{if } t \leq \log n \\ t^{1/3} \log^{2/3}\left(\frac{n^3}{t} + 1\right) & \text{if } \log n < t \leq n^3 \\ n & \text{if } n^3 < t. \end{cases}$$

An important consequence is that the optimal backlog is $\Theta(n)$ if and only if $t \geq \Omega(n^3)$. Our techniques also allow for us to resolve several other open questions concerning how the variable-processor cup game behaves in beyond-worst-case-analysis settings.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Parallel algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases Cup Games, Potential Functions, Greedy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.85

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2205.01722

Funding *William Kuszmaul*: Funded by a Fannie & John Hertz Foundation Fellowship; by an NSF GRFP Fellowship; and by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.the United States Air Force Research Laboratory under Cooperative Agreement Number

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 85; pp. 85:1–85:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
Shyam Narayanan: Funded by an NSF GRFP Fellowship and a Simons Investigator Award.

1 Introduction

The classical p-processor cup game. The p processor cup game captures the general problem in which there are some number n of tasks competing for a smaller number p of processors [7, 21, 8, 33, 31, 37, 6, 24, 34, 35, 17, 10, 27, 1, 16, 32, 25]. A scheduler must assign tasks to processors over time in order to ensure that no individual task ever falls too far behind.

Formally, this is captured as a game with n cups, each capable of holding an arbitrarily large amount of water, and two competing players, a filler and an emptier. In each round of the game, the filler distributes p new units of water into the cups, placing at most 1 unit of water into any particular cup. The emptier then selects p distinct cups and removes up to 1 unit of water from each of them. Note that, whereas the filler may place their p units of water in fractional amounts across arbitrarily many cups, the emptier can only choose p cups per step to empty from. The emptier's goal is to minimize the backlog of the system, which is the amount of water in the fullest cup.

If one views the cup game as a scheduling problem, then the cups represent tasks, the water represents work, the filler represents an adversary that determines when work arrives, and the emptier represents a scheduler that can select p tasks to run on a given time step (we will use the terms "round" and "time step" interchangeably). Although we will primarily be interested in the cup game as a scheduling problem [7, 21, 8, 33, 31, 37, 6, 24, 34, 35, 1, 32, 17], it has also found applications to many other problems (e.g. deamortization of data structures [2, 17, 16, 3, 38, 23, 18, 26, 9], network-switch buffer management [22, 4, 39, 20], quality-of-service guarantees [7, 1, 32], etc.).

Beginning in the late 1960s, much of the early work on the p-processor cup game focused on the fixed-rate version of the game, in which the filler's behavior is the same at every round [7, 21, 8, 33, 31, 37, 6, 24, 34, 35]. In this version of the game, it is possible for the emptier to achieve a backlog of $O(1)$, both in the single-processor cup game (i.e., $p = 1$) [34, 35] and in the multi-processor cup game (i.e., $p > 1$) [7]. In recent decades, much of the research has shifted to focus on the non-fixed-rate version of the game, in which the filler is an adaptive adversary that can change their behavior from step to step [1, 16, 27, 32, 5, 19, 15]. In this setting, it is possible for the emptier to achieve backlog $O(\log n)$ [1, 16, 27], and this is known to be asymptotically optimal for all $p \leq n - \sqrt{n}$ [27]. There is also a long history of researchers applying techniques from beyond-worst-case analysis to cup games, e.g., resource augmentation [10, 27, 32, 17], smoothed analysis [27, 10], adversary restrictions [10, 27, 14, 28, 17], semi-clairvoyance [32], etc.

A repeating theme in these directions of work has been the relative difficulty of analyzing the multi-processor case in comparison to the single-processor case. As Liu discussed in his seminal 1969 paper [34], and as many later authors have subsequently reiterated [32, 7, 24, 10, 27], the difficulty of the multi-processor case stems from the fact that the emptier must remove water from p *distinct* cups, even if the vast majority of the water is in a smaller number of cups. For both the fixed-rate and the non-fixed-rate games, the optimal backlog in the multi-processor version of the game [7, 27] was proven decades after the corresponding result for the single-processor game was first shown [34, 16].

The variable-processor cup game. Recent work by Kuszmaul and Westover [29] has considered the question of what happens if the parameter p is permitted to change over time, with the filler adaptively determining both what value of p will be used at each round and where the p new units of water are placed. The resulting game, which is known as the variable-processor cup game, captures settings in which the amounts of resources available to the players fluctuate over time.[1]

The problem of what to do when computing resources fluctuate has received increasing attention in recent years due to the proliferation of shared-computing systems in which multiple users and virtual operating systems simultaneously run on a single physical multi-core machine; the fact that the machine is shared means that the amount of resources (eg., cache, processors, memory bandwidth, etc.) available to each user is constantly changing, depending on the current demands of other users. This phenomenon has led researchers to revisit problems in which computing resources have traditionally been viewed as static [36, 11, 12, 13, 30, 29].

Intuitively, the variable-processor cup game would seem to be relatively similar to its classical p-processor counterpart. Indeed, the backlog in the p-processor cup game is $O(\log n)$ regardless of the value of the parameter p, suggesting that the same should be true if p is permitted to vary. The central result of [29] is that this intuition turns out to be completely wrong: given sufficiently many time steps, the filler can actually force a backlog of $\Omega(n)$ in the variable-processor cup game, and this backlog is asymptotically optimal.

In order to achieve the backlog of $\Omega(n)$, the authors [29] construct a strategy for the filler in which the number of processors p follows a recursive "fractal-like" pattern. The recursive structure requires a relatively large number of time steps to complete – to achieve the full backlog of $\Omega(n)$, the construction requires exponentially many time steps.

The unexpectedly large backlog prompts several questions. The main open question is the problem of determining the optimal trade-off between backlog and time in the variable-processor cup game, and, in particular, what the optimal backlog is in games of polynomial lengths. In this setting, it is not even known whether the filler can achieve polynomial backlog – the best known filler strategy in this case [29] achieves backlog $2^{O(\sqrt{\log n})}$. Understanding the optimal trade-off between time and backlog in polynomial-length games is especially important since, for instance, one may have a bound on the number of rounds in a given scheduling application, which may allow for a much better guarantee on the backlog.

The authors of [29] also raise the question of whether smaller bounds on backlog can be achieved via beyond-worst-case analysis. Based on their results, they propose two directions, in particular, that seem promising. One is to place an additional restriction on the filler that p can only change at a certain rate; this would thwart the recursive structure of their lower bound construction which changes p dramatically between levels of recursion. The other is to consider the use of resource augmentation, meaning that the emptier is allowed to remove slightly more water in each time step than the filler is permitted to place into the cups. This direction seems promising due to the large amount of time required by the filling strategy of [29], since over such a large amount of time, the resource augmentation would potentially offer a large advantage to the emptier.

[1] As discussed in [29], there is no fundamental reason why the amount p of resources should fluctuate in the same way for the filler as it does for the emptier over time. However, by assuming that p is always the same for both players, one ensures that there is a fair playing field: neither player has an advantage over the other in terms of their resources.

Our results. The main result of this paper is a tight trade-off curve between time and backlog in the variable-processor cup game. We show that, for a game consisting of t rounds, the optimal backlog is $\Theta(b(t))$ where

$$b(t) = \begin{cases} t & \text{if } t \leq \log n \\ t^{1/3} \log^{2/3} \left(\frac{n^3}{t} + 1\right) & \text{if } \log n < t \leq n^3 \\ n & \text{if } n^3 < t. \end{cases} \tag{1}$$

By optimal, we mean that there exists an emptying strategy such that no filler can achieve backlog greater than $\Omega(b(t))$ after t rounds, and there exists a filling strategy such that no emptier can achieve backlog less than $O(b(t))$ after t rounds. In the case of the emptying strategy, we show that this tradeoff curve is achieved by the greedy emptying algorithm (i.e., always empty from the fullest cups).

Equation (1) comes with several interesting takeaways. The first is that in short games, of length $o(n^3)$, the emptier *can* achieve sub-linear backlog – prior to this result, it remained open whether the emptier could even ensure $o(n)$ backlog for only n rounds. The second is that in games of size $\Omega(n^3)$, the optimal backlog is $\Theta(n)$ – this resolves the open question of [29] as to whether linear backlog can be achieved in polynomial time. The third is that the optimal tradeoff between backlog and time has a somewhat unexpected polylogarithmic low-order term, which disappears only when t grows to be $\Omega(n^3)$.

By examining the inverse of the function $b(t)$, another way to think about (1) is that, for any quantity $b \leq n$, the amount of time $t(b)$ needed for an optimal filler to force a backlog of $\Omega(b)$ against an optimal emptier is

$$t(b) = \Theta\left(b + \frac{b^3}{\log^2 \frac{n}{b}}\right), \tag{2}$$

and that backlogs $b = \omega(n)$ are not achievable by the filler (the latter fact, of course, is already known due to [29]).

The second contribution of this paper is to analyze the variable-processor cup game under two forms of beyond-worst-case analysis, each of which resolves an open question posed by [29]. We begin by considering the setting in which the rate at which the filler can change p is severely limited: p is permitted to change by most ± 1 per time step, and the filler can only change p every $\text{poly}(n)$ time steps. Remarkably, this has no effect on the optimal backlog, and the filler can still force a backlog of $n/2$ in polynomial time. Next, we consider the setting in which the emptier has $\varepsilon > 0$ resource augmentation, meaning that, in each time step, the emptier is permitted to remove up to $1 + \varepsilon$ (rather than 1) units of water from each of p cups. This has a dramatic effect on the optimal backlog, reducing it to $O(\frac{1}{\varepsilon} \log n)$, which asymptotically matches the optimal backlog of the standard p-processor cup game when $\varepsilon = \Omega(1)$.

Techniques and paper outline. We formally describe the necessary preliminaries in Section 2, and provide a detailed overview of the technical ideas in the paper in Section 3.

In Section 4, we prove a result that ends up being useful in many of the later sections, namely that the greedy emptying algorithm is actually the *exact* optimal online algorithm, and that this holds no matter what the starting state of the cups is. Interestingly, our proof of greedy emptying being the optimal emptying strategy also applies to the fixed p-processor cup game, for which the result was also not previously known: in this setting, the greedy emptying algorithm was previously only known to be *asymptotically* optimal [27], and this

was only known for the starting state with all empty fills. We remark that, although the fact that greedy emptying is optimal is certainly intuitive, it actually isn't true for every variant of the cup game; notably, the greedy emptying algorithm isn't even asymptotically optimal for the fixed-rate version of the game [1].

In Section 5, we construct an asymptotically optimal strategy for the filler. The strategy achieves the bound in (1) no matter what strategy the emptier follows, and the strategy also easily generalizes the case where the filler is restricted to change p at a slow rate. This implies that the optimal backlog $b(t)$ is *at least* that in (1). While the filling algorithm works against any emptier, we focus on the filler working against the greedy emptier, which we know is optimal. We start with a warm-up (subsection 5.1) proving that one can establish $\Omega(n)$ backlog in $O(n^3)$ time, and we then generalize our techniques to apply to games of arbitrary lengths.

The remainder of the paper has been deferred to the extended version of this paper[2]. In Section 6 in the extended version, we turn our attention to proving a tight upper bound for the maximum backlog against a greedy emptier, that is, we show that $b(t)$ is *at most* that in (1). We begin by creating a variation of the variable-processor cup game, which we call the ***stone game***, in which the filler's behavior is limited in a certain combinatorially natural way. We analyze the maximum backlog of any filling strategy for the stone game by devising two potential functions and comparing their growth rates; this allows us to establish that $\Omega(n^3)$ time steps are needed to achieve backlog $\Omega(n)$ in the stone game. We then tighten the bound on backlog when there are fewer time steps by partitioning the cups into levels and arguing that a constant fraction of the levels interact especially nicely with the potential functions; this yields (1) for the stone game. Finally, we show that, if the emptier behaves greedily, then this stone game encapsulates the main problem, that is, is always advantageous to the filler in the variable-processor cup game to act as though they are in the stone game. Thus we can transfer out bounds on the stone game into bounds on the variable-processor cup game.

Section 6 is, in our opinion, the most technically involved section in our work. Perhaps the most interesting mathematical contribution in this section is to analyze the furthest backlog in the stone game by constructing two different potential functions and comparing their relative growth rates. While individual potential functions have been used to analyze cup games [10], this is the first time that comparing two potential functions has been applied to cup games.

Finally, in Section 7 in the extended version, we give a very simple analysis of the variable-processor cup game in the presence of resource augmentation. Our argument is non-constructive, employing the probabilistic method in order to show that there *exists* an emptying algorithm that achieves backlog $O(\frac{1}{\varepsilon}\log n)$ (interestingly, the same argument also gives a nontrivial bound of $O(\sqrt{t\log n})$ in the resource-augmentation-free setting). Since the greedy emptying algorithm is optimal, it must also achieve the same bound. To the best of our knowledge, this is the first example of the probabilistic method being used to analyze cup games.

2 Preliminaries

We first formally define the ***variable-processor cup game***. In this game, there are n cups of real-valued fills $x_1, \ldots, x_n$, all starting at 0, and two *adaptive* players, a filler and an emptier. At each round, the filler chooses an integer $1 \le p \le n$, chooses real numbers

[2] Available at `https://arxiv.org/abs/2205.01722`

$a_1, \ldots, a_n$ such that $0 \le a_i \le 1$ for all i and $\sum_{i=1}^{n} a_i = p$, and replaces x_i with $x_i' = x_i + a_i$ for all $1 \le i \le n$. The emptier then chooses a set $S \subset [n]$ of size p, and for each $i \in S$, replaces x_i' with $\max(0, x_i' - 1)$ but does not change x_i' for $i \notin S$. A single ***round*** (which we also call ***time step***) consists of both the filler's and emptier's moves. We define the ***state*** of the cups at a fixed round to be the current sequence $\{x_1, x_2, \ldots, x_n\}$ of the values of cups. Since the filler and emptier are adaptive, we note that two states are equivalent if the sequences are equal up to permutation. We will also say that the ***fills*** are $x_1, x_2, \ldots, x_n$ (we think of x_i as the fill of the i^{th} cup). Finally, for any state $X = \{x_1, \ldots, x_n\}$, we define the ***backlog*** of X as the maximum fill, or $\max_{1 \le i \le n} x_i$. The goal of the filler is to maximize the backlog after t rounds for some fixed t, whereas the goal of the emptier is to minimize it. We also define the ***variable-processor cup game with ε resource augmentation*** as the same as the variable-processor cup game, except that the emptier, for each $i \in S$, replaces x_i' with $\max(0, x_i' - (1 + \varepsilon))$.

Next, we define the ***negative-fill variable-processor cup game*** as the same as the variable-processor cup game, except that the emptier, for each $i \in S$, replaces x_i' with $x_i' - 1$. This may mean that some of the fills in a state become negative, which is allowed. We analogously define round, state, fills, and backlog (note that the backlog is $\max_{1 \le i \le n} x_i$, not $\max_{1 \le i \le n} |x_i|$).

Unless explicitly stated otherwise, the *standard* game refers to the variable-processor cup game, and the *negative-fill* game refers to the negative-fill variable-processor cup game. We will explicitly state whenever we talk about the fixed p-processor cup game (where $p = \sum_{i=1}^{n} a_i$ is fixed for every round), or the variable-processor cup game with $\varepsilon > 0$ resource augmentation.

We conclude this section by briefly commenting on the relationship between the standard game and the negative-fill game. It is easy to see that the optimal backlog in the standard game is at least as large as the optimal backlog in the negative-fill game. What is less obvious, but worth noting, is that the optimal backlogs in the two games are actually asymptotically *equal*. We provide a proof of this in Appendix A in the extended version. Thus, in general, either version of the game is equally valid (although the only place where we will take advantage of this in any nontrivial way will be Section 6 in the extended version).

3 Technical Overview

In this section, we provide a technical overview for both the lower and upper bound for the variable-processor cup game, as well as the upper bound for the variable-processor cup game with resource augmentation.

3.1 Overview of the Lower Bound on Backlog

In Section 5, we provide a lower bound on the backlog that the filler can achieve over t rounds. For now, let us assume that we are playing the negative-fill game and that the emptier is greedy, i.e., at each time step, if the filler fills p units of water, then the emptier empties 1 unit from the p fullest cups. We shall remove these assumptions at the end of the subsection.

Achieving backlog $\Omega(n)$ in n^3 steps. Why is this type of move good for the filler? Consider the potential function measuring the sum of squares of the fills, i.e., $\Phi(x_1, \ldots, x_n) := \sum_{i=1}^{n} x_i^2$. If we let $q = \frac{j-i+1}{2}$, so that q of the fills go up by $1/2$ and q of the fills go down by $1/2$, then it is not hard to show that Φ increases by $\frac{q}{2} \ge \frac{1}{2}$. If the filler can force this to happen for n^3 consecutive time steps, then Φ will increase to $\Omega(n^3)$, which means that at least one of the

$|x_i|$'s must be $\Omega(n)$. If the filler is careful, then it turns out they can further ensure that the cups are symmetric (i.e., for every cup with fill s there is another cup with fill $-s$). Thus if $|x_i| \geq \Omega(n)$ for some n, then a backlog of $\Omega(n)$ has been achieved.

The only way that the filler might be prevented from performing this type of move for n^3 consecutive time steps is if, at some time step, no two cups have the same fills as each other. Note, however, that the filler always adds half-integer values to cups and the emptier always subtracts integer values, which means that $x_1, \ldots, x_n$ are always half-integers. So, if the x_i's are all distinct, then $\max_{1 \leq i \leq n} |x_i| \geq \frac{n-1}{4} = \Omega(n)$. Recalling that the filler can ensure symmetry of the cups, it follows that in this case the filler has also achieved an $\Omega(n)$ backlog in only n^3 time steps.

Considering smaller backlogs. What if we only want to reach some backlog $o(n)$? We will now describe how the filler can achieve backlog $\Omega(t \log(n/t))$ in $O(t^3 \log(n/t))$ time steps. Combining this with some edge cases (which we defer to Section 5) results an optimal filling strategy for any backlog.

First, we claim that within t^3 steps, the filler can cause $\Theta(n)$ cups to have fills $\Omega(t)$. To see this, note that if at least half of the cups have fills less than t, then by the pigeonhole principle, there must exist some half-integer s with $|s| \leq t$ and $\Omega(n/t)$ cups of fill exactly s. If the filler causes half of these cups to go up in fill by $1/2$ and half of them to go down in fill by $1/2$, then the net effect on Φ will be that it increases by $\Omega(n/t)$. The filler can repeatedly force Φ to increase by $\Omega(n/t)$, while keeping the maximum backlog at t, until either t^3 steps have passed and $\Phi = \Omega(n \cdot t^2)$ or until at least half of the cups have reached backlog $\pm t$. Either way, at least $\Theta(n)$ of the cups must have fills more than $\Omega(t)$ or less than $-\Omega(t)$, and recalling again that the filler can also ensure a symmetry of the cups, it follows that a constant fraction of the cups have fills $\Omega(t)$.

Once we have cn cups of fill $\Omega(t)$, for some constant $c > 0$, the filler can focus on these cups, and force $c^2 n$ of these cups to fill $2 \cdot \Omega(t)$, then $c^3 \cdot n$ to fill $3 \cdot \Omega(t)$, and so on for a logarithmic number of phases. Overall, one can achieve backlog $\Omega(t \log(n/t))$ in $O(t^3 \log(n/t))$ time steps.[3]

The final piece: establishing that greedy emptying is optimal. To conclude our overview of the lowerbound, let us revisit the assumptions that (a) we are playing the negative-fill game, and (b) the emptier is playing greedily. The first assumption is trivial to remove, since the filler in the standard game is strictly better off than the filler in the negative-fill game. The second assumption, that the emptier is playing greedily, requires us to prove that the greedy emptying algorithm is always optimal (this ends up being useful to have for later results as well).

To prove the greedy emptying is optimal, in Section 4, we construct a specially designed poset on the possible states X of the system. Say that a state $X = \{x_1, \ldots, x_n\}$ ***weakly monopolizes*** a state $Y = \{y_1, \ldots, y_n\}$ if it is possible to order the cups such that either:

- $x_i \geq y_i$ for all i;
- or we have (a) that $x_i = y_i$ for all $i > 2$, that (b) $x_1 > y_2$, and that (b) we can get from X to Y by removing exactly 1 unit of water from cup 1 and placing some quantity $0 \leq c \leq 1$ of water into cup 2.

[3] Note that one can only do $\log(n/t)$ phases, rather than the more intuitive $\log n$ phases, since we need at least t cups to reach fill $a \cdot t$ in order for anything to reach fill $(a+1) \cdot t$.

The transitive closure of weak monopolization induces a partial ordering on the set of all possible system states, where $A \geq B$ if there is a sequence $A = A_1, A_2, \ldots, A_j = B$ such that each A_i weakly monopolizes each A_{i+1}.

We prove that, given the choice between two states A, B with $A \geq B$, the emptier should always prefer state B. Furthermore, starting from any state X, greedy emptying always results in a state B that satisfies $B \leq A$ for every other state A that the emptier could have reached from X. Thus the greedy emptying algorithm is optimal.

3.2 Overview of the Upper Bound on Backlog

In Section 6 in the extended version, we show that the greedy emptying algorithm achieves an upper bound on backlog matching the lower bound of Section 5. We first consider a filler that is restricted to only making moves of the form described in Subsection 3.1, that is, so that the net effect of each round is that some number $2q$ of cups at some height k are replaced with q cups at height $k + 1/2$ and q cups at height $k - 1/2$. We will later show that these types of moves are (essentially) always optimal for the filler, which means this restriction is actually without loss of generality. Considering this restricted filler along with a greedy emptier leads to the following simple combinatorial problem, which we call the ***stone game*** (we call it the ***stone-variant cup game*** in Section 6):

> *Suppose you have n stones on a number line, all starting at 0. At each time step t, you may pick any point k on the number line with 2 or more stones, choose any integer $q \geq 1$ such that there are at least $2 \cdot q$ stones at k, and move q of the stones at position k to position $k - 1$ and q of the stones at position k to position $k + 1$. If you repeat this for T time steps, what is the furthest that any of the n stones may be from the origin?*

Analyzing the time to get a stone to position $\Omega(n)$. To analyze the stone game, we start by considering how long it takes for some stone to reach $b := n/10$ in absolute value. Our goal is to show that one needs at least $T = \Omega(n^3)$ time steps. To highlight the relationship between the cup game and the stone game, we shall sometimes refer to the distance of the furthest stone from the origin simply as the backlog.

Let the positions of the stones be $x_1, \ldots, x_n$. We start by recalling the potential function $\Phi = \sum_{i=1}^{n} x_i^2$. One would naturally hope that Φ could help us establish a bound of $T = \Omega(n^3)$ (just as it helped us with the $T = O(n^3)$ bound in Subsection 3.1). For example, one way to prove the $\Omega(n^3)$ time bound would be to show that Φ increases by at most $O(1)$ in each step, and that Φ takes a value of at least $\Omega(n^3)$ after the final step (i.e., when backlog $b = n/10$ is achieved). Unfortunately, we run into two problems. The first problem is that even if $\max |x_i| = b = n/10$, we may still have that $\sum_{i=1}^{n} x_i^2$ is just $O(n^2)$ after the final step (as opposed to the desired $\Omega(n^3)$). The second, more difficult problem is that the number of stones we move in each direction could be large at each time step. If we move q stones right and q stones left, Φ increases by $2q$ (rather than the desired $O(1)$), which could be as large as n. Together, these two problems mean that, a priori, we can only get a trivial $T = \Omega(n)$ bound, since the change in the potential function Φ is up to n at each time step and the final potential may be as small as $\Theta(n^2)$.

The first problem can be resolved with a more careful analysis: in particular, one can show that a backlog of $\Omega(n)$ actually does imply $\Phi = \Omega(n^3)$. The key is to prove that there can never be large gaps between consecutive stones. Namely, one can show that if there exists a stone at some position $k > 0$, there must be at least one stone at position $k - 1$

or $k-2$, and likewise, if there exists a stone at $k<0$, there must be at least one stone at position $k+1$ or $k+2$. As a result, if there is a stone at $b=n/10$, there must be a stone at position either $b-1$ or $b-2$, at either $b-3$ or $b-4$, and so on, so one can show that there are $\Omega(b)$ stones at positions $b/2$ or greater. Thus, if there is a stone at position b, then $\Phi \geq \Omega(b)\cdot(b/2)^2 \geq \Omega(n^3)$, as desired.

The more difficult piece of the analysis is to show that, even though Φ can grow significantly in a single step, Φ only increases *on average* by $O(1)$ per step. An important insight here is to create a second potential function, Ψ, and compare the growth rates of Ψ and Φ. We define this new potential function $\Psi := \sum_{i<j} |x_i - x_j|$, where $x_1, \ldots, x_n$ are the locations of the n stones. In a given step of the stone game, if we move q stones up from k to $k+1$ and q stones down from k to $k-1$, the first potential function Φ increases by $2q$. However, one can show that Ψ must increase by at least $2q^2$ – the core reason for this is that the q stones that moved up now each have distance 2 from the q stones that moved down, whereas before they had distance 0. Now suppose for contradiction that Φ grows on average by some amount $\overline{q} = \omega(1)$ per step. This would mean that the average growth of Ψ per step is at least $\Omega(\overline{q}^2)$, so the final values of Ψ and Φ must satisfy $\Psi = \omega(\Phi)$. However, we know that $\Phi = \Omega(n^3)$ at the end of the game, and it is not hard to see that Ψ must be $O(n^3)$, since all of the x_i's are bounded in the range $[-O(n), O(n)]$. Thus we have a contradiction, and the average growth of Φ per step is actually $O(1)$. The fact that Φ is $\Omega(n^3)$ at the end of the game and grows by $O(1)$ on average per step is sufficient to show that the time T needed to achieve backlog $b=n/10$ in the stone game is at least $\Omega(n^3)$ time steps.

Considering smaller backlogs in the stone game. But what happens if we wish to analyze the time needed to achieve backlog k in the stone game, for $k \ll n$? If, at the end of the game, we knew that at least a constant fraction of the stones were in positions $\Omega(k)$, then we could use the same argument as the one above to show that the average growth rate of Φ is $O(1)$. Unfortunately, the number of stones at positions $\Omega(k)$ might be as small as $O(k)$. In this case, Φ could be as small as $O(k^3)$, whereas Ψ must be at least $\Omega(k^2 n)$. If the average growth of Φ were $\Theta(n/k)$ and the average growth of Ψ were $\Theta((n/k)^2)$, these values of Φ and Ψ could be obtained in $O(k^4/n)$ rounds, which is much smaller than our desired bound of $\Omega(k^3/\log^2(n/k))$.

To obtain the optimal bound of $\Omega(k^3/\log^2(n/k))$, we introduce another variant of the stone game that has what we call checkpoints: namely, we choose an integer ℓ and play the same game but now, once a stone has reached a position $a\cdot\ell$ for any $a \geq 0$, it can never go below it. In other words, if we move q stones from $a\cdot\ell$ to $a\cdot\ell+1$, rather than moving q stones from $a\cdot\ell$ down to $a\cdot\ell-1$, we keep them as is. A key insight is that, if a player wishes to get a stone to position $10\log n\cdot\ell$, the player must have the property that, for the majority of the checkpoints, the player gets at least half of the stones that reach checkpoint $a\cdot\ell$ all the way up to checkpoint $(a+1)\cdot\ell$.

We can think of the steps of the stone game (with checkpoints) as being split into subgames, where each subgame takes place between two checkpoints $a\cdot\ell$ and $(a+1)\cdot\ell$. We analyze each subgame individually by creating potential functions Φ_a, Ψ_a between each pair of checkpoints at $a\cdot\ell$ and $(a+1)\cdot\ell$. At least half of the subgames have the property that at least half of their stones make it to the next checkpoint, and this property makes each such subgame amenable to being analyzed using Φ_a and Ψ_a. By analyzing the subgames individually, we can show that the total length of the full stone game (which is the sum of the lengths of the subgames) is at least $\Omega(k^3/\log^2(n/k))$.

Finally, one can show that adding the checkpoints to the original stone game only makes it easier for the player to achieve a large backlog. This involves proving that if states X and Y have $x_i \geq y_i$ for all i, then given any stone-game operation on Y to obtain a state Y', we can generate a corresponding (but perhaps different) stone-game operation on X to obtain X', and preserve the ordering $x'_i \geq y'_i$ for all i. Thus, the $T = \Omega(k^3/\log^2(n/k))$ bound also applies to the original stone game.

Transferring bounds from the stone game to the cup game. Overall, the potential-function arguments above get optimal bounds for the *stone game* but they do not directly give bounds for the more general variable-processor cup game. The main issue is that in certain time steps in the cup game, it might be possible for the filler to make "backward" moves where both Φ and Ψ decrease considerably: in the worst case, Ψ can even decrease by up to n^2. Our analysis of the stone game relied heavily on the fact that the change in Ψ was comparable to the square of the change in Φ (in a given step), but if Ψ can decrease significantly in a single step, then our comparison of growth rates no longer works. Our fix for this is not to modify the potential functions, but rather to show that these backward moves, or any other "non-stone game" moves, are never advantageous to the filler, even in the long run.

To show that non-stone-game moves are never advantageous for the filler, we analyze the relationship between the stone game and the variable-processor cup game. Say that a sequence $X = \{x_1, \ldots, x_n\}$ of real numbers ***majorizes*** another sequence $Y = \{y_1, \ldots, y_n\}$ (where $x_1 \geq \cdots \geq x_n$ and $y_1 \geq \cdots \geq y_n$) if $x_1 + \cdots + x_i \geq y_1 + \cdots + y_i$ for all $1 \leq i \leq n$, and if $\sum_i x_i = \sum_i y_i$. We prove that, for any sequence of cup game rounds, one can create a corresponding sequence of stone game rounds such that after every round, the sequence of stone positions majorizes the sequence of cup fills. Since X majorizing Y implies that $x_1 \geq y_1$, we get that the maximum backlog after T steps of the cup game is at most the maximum backlog after T steps of the stone game. Therefore, even if we cannot utilize the potential functions on the general cup game, it suffices to look at the stone game as it will have a greater maximum backlog.

The main technical claim needed to show this majorization result is to show that if a sequence X majorizes a sequence Y, and if Y can be converted to a sequence Y' in one round of filling/greedy-emptying in the variable-processor cup game, then it is possible to convert X into some X' (also with a single round of filling/greedy emptying) such that X' majorizes Y'. This claim is quite casework-heavy and crucially uses the fact that we are in the *variable-processor* cup game. Indeed, the choice of p may have to differ between the round performed on X and the round performed on Y, and perhaps surprisingly, the claim is actually *false* in the fixed p-processor cup game.

3.3 Overview of Resource-Augmentation Analysis

In Section 7 in the extended version, we analyze the variable-processor cup game with ε resource augmentation (and non-negative fills), meaning that in each time step, the emptier is permitted to remove up to $1 + \varepsilon$ units of water from each of p cups (rather than just 1 unit of water).

We prove that even a very small amount of resource augmentation significantly decreases backlog of the game: the greedy emptying algorithm achieves backlog $O(\varepsilon^{-1} \log n)$.

The proof uses the probabilistic method. Rather than analyzing the greedy emptying algorithm directly, we instead construct a randomized emptying algorithm that, at any given moment, achieves backlog $O(\varepsilon^{-1} \log n)$ with *non-zero* probability. (Importantly, the randomized algorithm is against an *adaptive* filler, not an oblivious one.) The existence

of such a randomized algorithm non-constructively implies the existence of a deterministic emptying algorithm with the same guarantee. But we already know that the best deterministic emptying algorithm is the greedy one. Thus greedy emptying achieves backlog $O(\varepsilon^{-1} \log n)$ (deterministically).

To simplify our discussion here, let us consider only games of polynomial length (in Section 7 in the extended version we will consider arbitrary game lengths). In this case, our randomized emptying algorithm can simply take an approach that we call ***proportional emptying***: in each time step of the game, if the filler places some amount q_j of water into cup j, then the emptier empties from cup j with probability exactly q_j.

To analyze proportional emptying, we show that, at any given moment, each cup has fill $O(\varepsilon^{-1} \log n)$. Roughly speaking, the amount of water in each cup can then be modeled as a biased random walk: in each time step, the expected amount of water that the emptier removes is a factor of $1 + \varepsilon$ larger than the amount of water that the filler inserts. The bias in the random walk prevents it from ever reaching a large fill. The result is that a simple Chernoff-style analysis (modified using a variation on Azuma's martingale inequality to handle the fact that the filler is an adaptive adversary) can be used to bound the fill in each cup by $O(\varepsilon^{-1} \log n)$ (with high probability).

Interestingly, the above argument also immediately yields a nontrivial bound in the resource-augmentation-free setting. Now the amount of water in each cup follows an *unbiased* random walk. At any given time step t, one can bound the height of such random walk by $O(\sqrt{t \log n})$ with high probability. Using the fact that greedy emptying is as good as any randomized emptying strategy, it follows that greedy emptying achieves backlog $O(\sqrt{t \log n})$ in a t time step game.

4 The Greedy Emptier is Always Optimal

Intuitively, the greedy emptying algorithm (i.e., always empty from the fullest cups) should be the optimal deterministic emptying algorithm (for both the p-processor cup game and the variable-processor cup game). This intuition is known to be true for the single-processor cup game starting in a state with all empty cups (in particular, the lower and upper bounds on backlog match in this case [1]), but whether or not the intuition is correct in general has remained an open question. (We remark that there are variants of the game, for example the fixed-rate cup game, where greedy emptying is *not* optimal, even asymptotically [1].) In this section, we prove that greedy emptying is, in fact, optimal for both of the p-processor cup game and the variable-processor cup game.. That is, for any starting state of the cups, and for any game length, greedy emptying is the best possible algorithm for minimizing backlog against an adaptive filler.

For any state S and any length t, define $\mathrm{OPT}(S, t)$ to be the supremum backlog that a filler can achieve at the end of a t-step game starting at state S assuming the emptier plays optimally. That is, $\mathrm{OPT}(S, 0)$ is just the amount of water in the fullest cup of S, and $\mathrm{OPT}(S, t)$ for $t > 0$ is defined by induction as

$$\sup_{S' \text{ reachable from } S \text{ by filler}} \quad \min_{S'' \text{ reachable from } S' \text{ by emptier}} \mathrm{OPT}(S'', t-1).$$

Note that this also allows for us to talk about the ***optimal emptier***, which is the emptier that achieves backlog at most $\mathrm{OPT}(S, t)$ in any t-step game starting at any state S.

Define GREEDY(S, t) to be the supremum backlog that a filler can achieve at the end of a t-step game starting at state S, assuming the emptier plays greedily. We wish to prove that OPT$(S, t) =$ GREEDY(S, t) for all S, t. Throughout the section we shall prove all of our results for both the versions of the games with non-negative fills and the versions of the games with negative fills.

We say that a state A ***monopolizes*** a state B if it is possible to assign labels $1, 2, \ldots, n$ to the cups in A and B such that:

- cups $3, 4, \ldots, n$ contain the same amounts of water in both states;
- cup 1 in A contains more water than cup 2 in B;
- cup 1 in A contains one more unit of water than cup 1 in B;
- cup 2 in B contains c more units of water than cup 2 in A for some $0 \leq c \leq 1$.

In other words, you can get from A to B by removing 1 unit of water from cup 1 and and placing $c \leq 1$ units into cup 2, and cup 1 in A contains more water than cup 2 in B.

We say that A ***dominates*** B if it is possible to label the cups such that each cup i in A contains at least as much water as cup i in B. Finally, we say that A ***weakly monopolizes*** B if either A monopolizes B, or A dominates B.

We now prove several properties of weak monopolization in the p-processor cup game (for any p). Our first lemma says that, if A weakly monopolizes B, then for any filler move on B, there is some filler move on A that preserves the weak monopolization of A over B.

▶ **Lemma 1.** *Suppose A weakly monopolizes B. Suppose that there is a p-processor filler move that transforms B into some B'. Then there exists a p-processor filler move that transforms A into some A' such that A' weakly monopolizes B'.*

Proof. If A dominates B, then the lemma is trivial. Thus we can assume that A monopolizes B. Let q be the amount by which cup 1 in A contains more water than cup 2 in B.

Define X to be a state that we reach from A if we perform the same filler move that transforms B into B'. If cup 1 in X contains more water than cup 2 in B' then we can set $A' = X$ and be done.

Suppose cup 1 in X does not contain more water than cup 2 in B'. Then, in the transformation from B to B', the filler must have placed at least q more water into cup 2 than into cup 1. Let r be the amount of water that the filler placed into cup 1, and let $r + q + s$ (for some $s \geq 0$) the amount of water that the filler placed into cup 2.

Now define A' to be the state that one would reach from A by performing the following p-processor filler move: place $r + s$ units of water into cup 1 and $q + r$ units of water into cup 2 (and then place water into cups $3, 4, \ldots, n$ in the same way as to transform B to B'). Cup 1 in A' contains the same amount of water as cup 2 in B', and cup 2 and A' contains the same amount of water as cup 1 in B'. Thus A' and B' are equivalent states, meaning that A' weakly monopolizes B'. ◀

Our next lemma says that, if A weakly monopolizes B, and if A is then greedily emptied from, then it is possible to empty from B in such a way that the monopolization relationship is preserved.

▶ **Lemma 2.** *Suppose A weakly monopolizes B. Let A' be the state reached if a p-processor emptier greedily empties from A. Then there exists a valid p-processor emptier move on B that achieves some state B' such that A' weakly monopolizes B'.*

Proof. If A dominates B, then the lemma is trivial. Thus we can assume that A monopolizes B.

Let a_1, a_2 denote the fills of cups 1 and 2 in A, and let b_1, b_2 denote the fills of cups 1 and 2 in B. So $a_1 = b_1 + 1$, $b_2 = a_2 + c$ for some $0 \leq c \leq 1$, and $a_1 > b_2$. Furthermore, let a_1', a_2' the fills of cups 1 and 2 in A', and let b_1', b_2' denote the fills of cups 1 and 2 in B' (once we've defined it).

If the greedy emptier on A empties from neither cup 1 nor cup 2, then the same set of empties on B results in a B' that is weakly monopolized by A'.

Next consider the case where the greedy emptier on A empties from both cups 1 and 2. Then we empty from the same set of cups in B to arrive at a state B'. In the negative-fill game, is immediate to see that A' monopolizes B'. In the standard game, we must be careful about the fact that one of b_1 or a_2 might be less than 1. If $b_1 < 1$, then $b_1' = 0 \leq a_2'$, and since $a_1 > b_2$, we also have $a_1' \geq b_2'$; thus A' dominates B' in this case. If $a_2 < 1$ but $b_1 \geq 1$, then we have $a_1' = b_1' + 1$ and $b_2' = a_2' + c'$ for some $0 \leq c \leq 1$; we also have $a_1' \geq b_2'$ (since $a_1 \geq b_2$), so A' monopolizes B' in this case.

Finally, consider the case where the greedy emptier on A empties from only one of the cups 1 or 2. Since $a_1 > b_2 \geq a_2$, the emptier must empty from cup 1. Suppose we construct B' by performing the same set of empties on B as were performed on A, but we remove water from cup 2 instead of cup 1. Then $a_1' = b_1'$ and $a_2' \geq b_2'$, so A' dominates B'. This completes the proof. ◀

By combining the previous two lemmas, we can arrive at the following.

▶ **Lemma 3.** *Consider either the p-processor cup game, for some p, or the variable processor cup game (and consider either the negative-fill case or the non-negative-fill case). Suppose A weakly monopolizes B. Then for any t,*

$$\text{GREEDY}(A, t) \geq \text{OPT}(B, t).$$

Proof. We prove the lemma by induction on t. In the base case of $t = 0$, the lemma reduces to showing that A has backlog at least as large as B; this follows from the fact that A weakly monopolizes B.

Now consider some $t > 0$, and suppose the result holds for $t - 1$. For each state B' that the filler can reach from B, Lemma 1 tells us that there must exist a state A' that the filler can reach from A in such that A' weakly monopolizes B'. Moreover, Lemma 2 tells us that, if greedy emptying from A' results in some state A'', then there must be an emptying move on B' that results in a state B'' such that A'' weakly monopolizes B''.

Note that, in the previous paragraph, the filler's move from B to B' (i.e., the choice of B') fully determines A' (by the construction in Lemma 1), A'' (by greedy emptying on A'), and B'' (by the construction in Lemma 2). Thus, we will think of A', A'', B'' as being functions of B'.

Expanding out $\text{GREEDY}(A, t)$, we have

$$\text{GREEDY}(A, t) \geq \sup_{B'} \text{GREEDY}(A'', t - 1).$$

By the inductive hypothesis, since each A'' weakly monopolizes B'', we have $\text{GREEDY}(A'', t - 1) \geq \text{OPT}(B'', t - 1)$. Thus

$$\text{GREEDY}(A, t) \geq \sup_{B'} \text{OPT}(B'', t - 1).$$

Since $\sup_{B'} \text{OPT}(B'', t - 1) \geq \text{OPT}(B, t)$, the lemma follows. ◀

Recall that our goal is to upperbound $\text{GREEDY}(A,t)$ by $\text{OPT}(A,t)$ for all A,t. Thus Lemma 3 may at first seem to be backward progress, since the lemma establishes a *lower bound* on $\text{GREEDY}(A,t)$. The trick to using Lemma 3 is that we will only ever apply the lemma to states A and game lengths t for which we have already proven by induction that $\text{OPT}(A,t) = \text{GREEDY}(A,t)$; in this setting, the lemma establishes that $\text{OPT}(A,t) \geq \text{OPT}(B,t)$ which, as we shall see, ends up being critical to the proof.

▶ **Theorem 4.** *Consider either the p-processor cup game, for some p, or the variable-processor cup game (and consider either the negative-fill case or the non-negative-fill case). For all states A and game lengths t,*

$$\text{GREEDY}(A,t) = \text{OPT}(A,t).$$

Proof. We prove the theorem by induction on t with a trivial base case of $t = 0$. Consider $t > 0$, and suppose the theorem holds for $t-1$.

Consider any state A' that the filler can reach from A. Let A'' be the state reached from A' if the emptier empties greedily, and let B be the state reached from A' if the emptier empties according to OPT. Since the transformation from A' to A'' empties from cups with at least as much water as the transformation from A' to B, there must be a sequence of states $X_0, X_1, \ldots, X_k$, with $X_0 = B$ and $X_k = A''$, such that each X_i weakly monopolizes X_{i-1}. In particular, one can define the X_i's such the only difference between each X_i and X_{i+1} is that, to get from A' to X_{i+1} instead of from A' to X_i, the emptier removes water from a cup j instead of a cup k, where cup j contains less water than cup k in state A'. It is straightforward to verify that this results in each X_{i+1} weakly monopolizing each X_i: if cup k (in state A') has fill at least 1, then X_{i+1} monopolizes X_i; and otherwise, both cups k and j (in state A') have fills less than 1, and thus X_{i+1} dominates X_i.

By the inductive hypothesis, we know that $\text{OPT}(X_i, t-1) = \text{GREEDY}(X_i, t-1)$ for each X_i. Thus Lemma 3 tells us that $\text{OPT}(X_i, t-1) \geq \text{OPT}(X_{i-1}, t-1)$. By transitivity, it follows that $\text{OPT}(X_k, t-1) \geq \text{OPT}(X_0, t-1)$, and thus $\text{OPT}(B, t-1) \geq \text{OPT}(A'', t-1)$. Again applying the inductive hypothesis to deduce that $\text{OPT}(A'', t-1) = \text{GREEDY}(A'', t-1)$, we have that

$$\text{OPT}(B, t-1) \geq \text{GREEDY}(A'', t-1).$$

Thus

$$\text{GREEDY}(A,t) = \sup_{A''} \text{GREEDY}(A'', t-1) \leq \sup_{B} \text{OPT}(B, t-1) = \text{OPT}(A,t).$$

Finally, as $\text{GREEDY}(A,t) \geq \text{OPT}(A,t)$ trivially, we have $\text{GREEDY}(A,t) = \text{OPT}(A,t)$. ◀

▶ **Corollary 5.** *Theorem 4 continues to hold for the game with non-negative fills even if the emptier is given $1+\varepsilon$ resource augmentation.*

Proof. The proof of Theorem 4 continues to hold without modification. The only difference is that, now, we say that a state A monopolizes a state B if there is a ordering of the cups for which two properties hold: (1) we can take $1+\varepsilon$ unit of water from some cup 1 in A, place some amount $0 \leq c \leq 1+\varepsilon$ of water into some other cup 2 in A, and in doing so arrive at B; and (2) cup 1 in A contains more water than cup 2 in B. ◀

5 Lower Bounding Backlog

In this section, we prove the optimal lower bound on the backlog. In other words, we show that for any integer $t \geq 1$, there exists a filling strategy that can guarantee a backlog of $\Omega(b(t))$ against *any* emptying strategy, after t rounds of the variable-processor cup game, where $b(t)$ is defined as in Equation (1) in the introduction (Section 1). This result can be captured compactly in the following theorem:

▶ **Theorem 6.** *For some absolute constant $c > 0$ and any $k \leq c \cdot n$, the filler in the variable-processor cup game can force a maximum backlog of $\Omega(k)$ using only $O\left(k + \frac{k^3}{\log^2(n/k)}\right)$ time steps.*

We remark that due to the optimality of greedy emptying, which we proved in the previous section, we focus in this section on designing filling strategies that are effective against the greedy emptier.

In Subsection 5.1, we give a simple proof for obtaining $\Omega(n)$ backlog in $O(n^3)$ rounds. In Subsection 5.2, we generalize Subsection 5.1 and develop the full proof of Theorem 6.

5.1 Warmup: Achieving a $\Theta(n)$ Backlog Quickly Against a Greedy Emptier

In this subsection, we show that in only n^3 rounds, a filler can achieve a backlog of $\frac{n-1}{2}$ against a greedy emptier. While the following argument is simpler than the more refined lower bound in Subsection 5.2, it conveys much of the intuition behind the general lower bound. Here, we only consider the standard variable-processor cup game, though in subsection 5.2 we will consider both the standard and negative-fill games.

▶ **Theorem 7.** *Against a greedy emptier in the standard variable-processor cup game, the filler can achieve a backlog of $\frac{n-1}{2}$ in only n^3 rounds.*

Proof. The filler follows the following simple strategy, which will guarantee that the fills at the end of each round are half integers (i.e., integer multiples of $\frac{1}{2}$). Suppose the cups after some round, in sorted order, are $x_1 \geq x_2 \geq \cdots \geq x_n$, which are all half-integers.

If $x_i > x_{i+1}$ for all $1 \leq i \leq n-1$, then we must have that $x_i \geq x_{i+1} + \frac{1}{2}$ for all $i \leq n-1$, so $x_1 \geq \frac{n-1}{2}$. In this case, we have already achieved a backlog of $\frac{n-1}{2}$.

Otherwise, we consider the potential function $\Phi = \sum_{i=1}^{n} x_i^2$, i.e., the sum of the squares of the backlogs. Consider the smallest p such that $x_p = x_{p+1}$. We set the number of processors for the round to be p, we place 1 unit of water into each of cups $x_1, \ldots, x_{p-1}$ and we place $1/2$ unit of water into each of cups x_p, x_{p+1}. Then, the greedy emptier will empty 1 unit from each of the first $p-1$ cups and will choose to empty 1 unit from either the p^{th} cup or the $(p+1)^{\text{th}}$ cup (WLOG, they choose the $(p+1)$-th cup). This means that, over the course of the entire round, the first $p-1$ cups are unchanged, the p^{th} cup goes up by $\frac{1}{2}$, and $(p+1)^{\text{th}}$ cup goes down by $\frac{1}{2}$. The only exception is if $x_p = x_{p+1} = 0$ at the beginning, in which case one of these two cups will go up to $1/2$ and the other remains at 0. Clearly, all of the fills remain half-integers at the end of each round. If we replace x_p and $x_{p+1} = x_p$ with $x_p + \frac{1}{2}$ and $x_p - \frac{1}{2}$, then the sum of the squares of the backlogs increases by $1/2$. Otherwise, if we replace $x_p = x_{p+1} = 0$ with 0 and $1/2$ in some order, then the sum of the squares of the backlogs increases by $1/4$. Thus, at the end of each round, unless one of the backlogs was already $\frac{n-1}{2}$ or greater, Φ increases by at least $\frac{1}{4}$.

Therefore, after n^3 rounds, either at some point we will have achieved $x_1 > x_2 > \cdots > x_n$ and thus a backlog of $\frac{n-1}{2}$, or we have that $\sum_{i=1}^{n} x_i^2 \geq \frac{n^3}{4}$, which would mean that $\max x_i \geq \frac{n}{2}$, since all of the x_i's are nonnegative. As a result, there exists a filling strategy that can guarantee a $\frac{n-1}{2}$ backlog against a greedy emptier in at most n^3 rounds. ◀

5.2 The General Lower Bound

In this subsection, we prove the lower bound on the backlog for the variable-processor cup game after t rounds, showing that the optimal backlog is $\Omega(b(t))$ for $b(t)$ defined as in Equation (1). Equivalently, we show that for any integer $k \leq n$, a filler can achieve backlog $\Omega(k)$ in $O\left(k + \frac{k^3}{\log^2(n/k)}\right)$ rounds against any emptying strategy. We remark that if we are playing t rounds of the game and the filler has achieved backlog b by round $t' < t$, the filler can ensure the backlog stays at b by setting $p = n$ and filling every cup for steps $t'+1, \ldots, t$. Thus, we just need to show the filler can obtain this backlog *within* this many rounds.

Throughout this subsection, we assume that we are playing the negative-fill game, and that the emptier is always greedy. We remove both of these assumptions at the end in Theorem 6.

▶ **Lemma 8.** *Assume that the emptier is always greedy, and that at the beginning of some round, the fills are all (possibly negative) half-integers. For some integer $k \in \mathbb{Z}$ and positive integer $q \in \mathbb{N}$, suppose there are at least $2q$ cups having fill exactly $k/2$. Then, the filler can ensure that at the end of the round (i.e., after both the filler and emptier move), exactly q of the cups of fill exactly $k/2$ have increased to $(k+1)/2$, exactly q of the cups of fill exactly $k/2$ have decreased to $(k-1)/2$, and all remaining cups are unchanged.*

Proof. Suppose the cups are sorted in order $x_1 \geq x_2 \geq \cdots \geq x_n$, such that $x_i = x_{i+1} = \cdots = x_{i+2q-1} = k/2$. Also, suppose i is the smallest integer such that $x_i = k/2$, so either $x_{i-1} > k/2$ or $i = 1$. Then, the filler will set $p = i - 1 + q$ and fill the first $i-1$ cups with 1 unit of water and the cups $x_i, \ldots, x_{i+2q-1}$ each with $1/2$ unit of water. Then, if $i > 1$, the emptier is forced to empty 1 unit of water from each of the first $i-1$ cups. In addition, we have that the cups $i, i+1, \ldots, i+2q-1$ all have fills $(k+1)/2$, which is at least $1/2$ unit of water more than all later cups, which means that the emptier will remove 1 unit of water from exactly q of the cups $i, i+1, \ldots, i+2q-1$. Therefore, all cups $1, \ldots, i-1$ are unchanged (since 1 unit of water is added and then removed) and all cups $i+2q, \ldots, n$ are unchanged (since water is never added nor removed). Finally, among the cups $i, i+1, \ldots, i+2q-1$, exactly q of them will end up at $(k-1)/2$ and exactly q of them will end up at $(k+1)/2$. ◀

▶ **Lemma 9.** *Suppose that k, m are positive integers such that $4k|m$, and suppose there are at least m cups that currently have fill 0 (where $n \geq m$ is the total number of cups). Then, against a greedy emptier, the filler can ensure that at least $m/4$ cups will have fill exactly $k/2$ after $O(k^3)$ rounds.*

Proof. WLOG assume that the first m cups have fills $x_1, x_2, \ldots, x_m$, and at the start, $x_1 = x_2 = \cdots = x_m = 0$. (We do not assume the cups are sorted in order of fill.) We only ever modify the first m cups, and after each step, we will maintain the following invariants:

1. All of the fills are half-integers.
2. For each integer j, the number of cups of fill $j/2$ equals the number of cups of fill $-j/2$.
3. For each integer j, the number of cups of fill $j/2$ is always a multiple of $\frac{m}{4k}$.
4. No cup has fill greater than $k/2$ or less than $-k/2$.

Trivially, all 4 of these invariants are true at the beginning, since all the fills are 0 and $\frac{m}{4k}|m$. Our procedure is the following. Suppose that there exists some integer j such that $-k < j < k$ and the number of cups of fill $j/2$ is at least $\frac{m}{2k}$. Then, we use Lemma 8 to move $\frac{m}{4k}$ of these cups to fill $(j+1)/2$ and move $\frac{m}{4k}$ of these cups to fill $(j-1)/2$. In addition, if $j \neq 0$, then we know the number of cups of fill $-j/2$ is at least $\frac{m}{2k}$. So, in the next step, we move $\frac{m}{4k}$ of these cups to fill $(-j+1)/2$ and move $\frac{m}{4k}$ of these cups to fill $(-j-1)/2$.

When $j \neq 0$, we perform these two steps consecutively as a pair. We keep repeating these types of steps (and pairs of steps) until we can no longer do so. It is clear that the first three invariants are preserved, and the last is preserved since we only modify cups that have fill between $-(k-1)/2$ and $(k-1)/2$, and we change their fills by at most $1/2$ per round.

Now, we note that at each step, the potential function $\Phi = \sum_{i=1}^{m} x_i^2$ increases by $\frac{m}{8k}$, since

$$\frac{m}{4k} \cdot \left(\frac{j+1}{2}\right)^2 + \frac{m}{4k} \cdot \left(\frac{j-1}{2}\right)^2 - \frac{m}{2k} \cdot \left(\frac{j}{2}\right)^2 = \frac{m}{8k}.$$

This also means that if we do a pair of steps (i.e., modifying cups of fill $j/2$ and $-j/2$), the potential function Φ increases by $\frac{m}{4k}$.

Now, after $4k^3$ steps (and pairs of steps), Φ is at least $4k^3 \cdot \frac{m}{8k} = \frac{1}{2} \cdot mk^2$. But this is impossible, since all of the fills of cups 1 through m are between $k/2$ and $-k/2$, so the maximum possible value of Φ is $\frac{1}{4} \cdot mk^2$. This means that before reaching $4k^3$ of steps or pairs of steps, we must not be able to do any more steps. That is, for all j such that $-k < j < k$, there are less than $\frac{m}{2k}$ cups of fill exactly $j/2$, and because of our third invariant, this means there are at most $\frac{m}{4k}$ cups of fill exactly $j/2$. Therefore, the number of cups of fill between $-k+1$ and $k-1$ is at most $(2k-1) \cdot \frac{m}{4k} \leq \frac{m}{2}$. So, at least $\frac{m}{2}$ cups must have fill $\pm\frac{k}{2}$, and by the second invariant, this means at least $\frac{m}{4}$ cups must have fill $\frac{k}{2}$. Reaching this stage required at most $4k^3$ steps and pairs of steps, which means the filler can succeed in at most $8k^3$ steps. ◀

▶ **Corollary 10.** *Suppose k, m are positive integers such that $4k|m$, and suppose there are at least m cups that currently have fill exactly t for some real number t. Then, after $O(k^3)$ time steps, the filler can force at least $m/4$ of these cups to have fill exactly $t + \frac{k}{2}$, against a greedy emptier.*

Proof. We can apply same argument as in Lemma 9, where all of the fills are shifted up by the same number t. So, after at most $8k^3$ time steps, at least $m/4$ of the cups will have fill exactly $t + \frac{k}{2}$. ◀

From here, we are able to conclude with our main result of this section.

▶ **Theorem 6.** *For some absolute constant $c > 0$ and any $k \leq c \cdot n$, the filler in the variable-processor cup game can force a maximum backlog of $\Omega(k)$ using only $O\left(k + \frac{k^3}{\log^2(n/k)}\right)$ time steps.*

Proof. For now we will assume that the emptier is greedy, and that we are playing the negative-fill cup game. We will remove both assumptions at the end of the proof.

First, suppose $k \geq c \log n$. In this case, let $k' := \left\lceil \frac{k}{c \log(n/k)} \right\rceil \geq 2$. Also, let n' be such that $n \geq n' > \frac{n}{4}$ and $n' = k' \cdot 4^r$ for some integer $r \geq 1$.

Since there are at least $n' = k' \cdot 4^r$ cups of fill 0 at the beginning of the game, we can apply Corollary 10 with $t = 0$ and $m = n'$ to make sure there are at least $k' \cdot 4^{r-1}$ cups of fill k' after $O(k'^3)$ time steps, since $4k'|n'$. Then, we can again apply Corollary 10 with $t = k'$ and $m = n'/4$ to make sure there are at least $k' \cdot 4^{r-2}$ cups of fill $2k'$ after an additional $O(k'^3)$ time steps. We can repeat this r times until we have at least k' cups of fill $r \cdot k'$ after a total of $O(r \cdot k'^3)$ time steps. But since $r = \log_4(n'/k') = \Theta(\log(n/k \cdot \log(n/k))) = \Theta(\log(n/k))$, this means that the filler can achieve a maximum backlog of $r \cdot k' = \Theta(k)$ using only $O(r \cdot k'^3) = O(k^3/\log^2(n/k))$ time steps.

Next, suppose $k < c \log n$. In this case, set $n' = 2^k$: if $c < 1$ then $n' < n$. By Lemma 8, we can send 2^{k-1} cups to fill 1 during the first round, then 2^{k-2} of those cups to fill 2 during the second round, and so on until we have 1 cup at fill k after k rounds.

Up until now we have assumed that the filler was operating against a greedy emptying algorithm in the negative-fill cup game. However, by Theorem 4, if the emptier uses a different emptying algorithm, the filler could always modify their strategy to get equal or greater backlog in the same amount of time. So, regardless of the emptying strategy, the filler can force a maximum backlog of $\Theta(k)$ using $O\left(k + \frac{k^3}{\log^2(n/k)}\right)$ time steps, assuming we are playing the negative-fill cup game. Finally, recall that the maximum backlog that the filler can ensure in the negative-fill game is at most the maximum backlog that the filler can ensure in the standard game for any fixed number of time steps. Thus, if the filler can force a maximum backlog of $\Theta(k)$ using $O\left(k + \frac{k^3}{\log^2(n/k)}\right)$ time steps in the negative-fill game, then the filler can force the same backlog using the same number of time steps in the standard game, which completes the theorem. ◀

Finally, to conclude the section, we use the optimality of greedy emptying to resolve an open question of [29] concerning whether a filler who is limited in the speed at which they can change p can still force a large backlog.

▶ **Proposition 11.** *Consider the variable-processor cup game, starting with all empty cups (and with negative fills either allowed for disallowed). Suppose the filler is restricted to only change p once every n^c steps, for some constant $c \geq 0$, and to only change p by ± 1 at a time. Then the filler can still force a backlog of $\Omega(n)$ in polynomial time.*

Proof. Call this version of the game the *change-limited variable-processor cup game*. Notice that the proof of optimality for greedy emptying never changes the value of p used in any given step. Thus the same proof implies that greedy emptying is optimal in the change-limited variable-processor cup game. Henceforth we will assume that the emptier is greedy.

Next, observe that, no matter the value of p, the filler can always "skip" a step in the following way: the filler simply places 1 unit of water into each of the p fullest cups, forcing the greedy emptier to remove water from those cups, so that the step has no net effect.

We have already proven that, in the standard variable-processor cup game, the filler can cause backlog $\Omega(n)$ in polynomial time. The filler can use the same strategy here, but with the following modification: between any two consecutive steps that use p and q processors, respectively, the filler spends $|p - q|n^c$ steps changing the number of processors from p to q. That is, for each $i \in [p, q]$, the filler spends n^c steps with i processors, and renders each of those steps to have no net effect. The result is that, in polynomial time, the filler can force backlog $\Omega(n)$, as desired. ◀

References

1 Micah Adler, Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul Goldberg, and Mike Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 101–108, 2003. doi:10.1145/777412.777430.

2 Amihood Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. Improved dynamic dictionary matching. *Inf. Comput.*, 119(2):258–282, 1995. doi:10.1006/inco.1995.1090.

3 Amihood Amir, Gianni Franceschini, Roberto Grossi, Tsvi Kopelowitz, Moshe Lewenstein, and Noa Lewenstein. Managing unbounded-length keys in comparison-driven data structures with applications to online indexing. *SIAM Journal on Computing*, 43(4):1396–1416, 2014.

4 Yossi Azar and Arik Litichevskey. Maximizing throughput in multi-queue switches. *Algorithmica*, 45(1):69–90, 2006.

5 Amotz Bar-Noy, Ari Freund, Shimon Landa, and Joseph (Seffi) Naor. Competitive on-line switching policies. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 525–534, 2002. URL: `http://dl.acm.org/citation.cfm?id=545381.545452`.

6 Amotz Bar-Noy, Aviv Nisgav, and Boaz Patt-Shamir. Nearly optimal perfectly periodic schedules. *Distributed Computing*, 15(4):207–220, 2002.

7 S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996. `doi:10.1007/BF01940883`.

8 Sanjoy K Baruah, Johannes E Gehrke, and C Greg Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 280–288, 1995.

9 Michael Bender, Rathish Das, Martín Farach-Colton, Rob Johnson, and William Kuszmaul. Flushing without cascades. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

10 Michael Bender, Martín Farach-Colton, and William Kuszmaul. Achieving optimal backlog in multi-processor cup games. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.

11 Michael A Bender, Rezaul A Chowdhury, Rathish Das, Rob Johnson, William Kuszmaul, Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Helen Xu. Closing the gap between cache-oblivious and cache-adaptive analysis. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 63–73, 2020.

12 Michael A Bender, Rezaul A Chowdhury, Rathish Das, Rob Johnson, William Kuszmaul, Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Helen Xu. Closing the gap between cache-oblivious and cache-adaptive analysis. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 63–73, 2020.

13 Michael A Bender, Roozbeh Ebrahimi, Jeremy T Fineman, Golnaz Ghasemiesfeh, Rob Johnson, and Samuel McCauley. Cache-adaptive algorithms. In *Proceedings of the twenty-fifth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 958–971. SIAM, 2014.

14 Michael A Bender and William Kuszmaul. Randomized cup game algorithms against strong adversaries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2059–2077. SIAM, 2021.

15 Peter Damaschke and Zhen Zhou. On queuing lengths in on-line switching. *Theoretical computer science*, 339(2-3):333–343, 2005.

16 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372, 1987. `doi:10.1145/28395.28434`.

17 Paul F. Dietz and Rajeev Raman. Persistence, amortization and randomization. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 78–88, 1991. URL: `http://dl.acm.org/citation.cfm?id=127787.127809`.

18 Johannes Fischer and Paweł Gawrychowski. Alphabet-dependent string searching with wexponential search trees. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 160–171, 2015.

19 Rudolf Fleischer and Hisashi Koga. Balanced scheduling toward loss-free packet queuing and delay fairness. *Algorithmica*, 38(2):363–376, February 2004. `doi:10.1007/s00453-003-1064-z`.

20 H Richard Gail, G Grover, Roch Guérin, Sidney L Hantler, Zvi Rosberg, and Moshe Sidi. Buffer size requirements under longest queue first. *Performance Evaluation*, 18(2):133–140, 1993.

21 Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 229–240. Springer, 2017.

22 Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010. doi:10.1145/1753171.1753195.

23 Michael T Goodrich and Paweł Pszona. Streamed graph drawing and the file maintenance problem. In *International Symposium on Graph Drawing*, pages 256–267. Springer, 2013.

24 Nan Guan and Wang Yi. Fixed-priority multiprocessor scheduling: Critical instant, response time and utilization bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 2470–2473. IEEE, 2012.

25 Sungjin Im, Benjamin Moseley, and Rudy Zhou. The matroid cup game. *Operations Research Letters*, 49(3):405–411, 2021.

26 Tsvi Kopelowitz. On-line indexing for general alphabets via predecessor queries on subsets of an ordered list. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 283–292, 2012.

27 William Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

28 William Kuszmaul. How asymmetry helps buffer management: achieving optimal tail size in cup games. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1248–1261, 2021.

29 William Kuszmaul and Alek Westover. The variable-processor cup game. In *12th Innovations in Theoretical Computer Science Conference (ITCS)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

30 Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Helen Xu. Cache-adaptive exploration: Experimental results and scan-hiding for adaptivity. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 213–222, 2018.

31 Ami Litman and Shiri Moran-Schein. On distributed smooth scheduling. In *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 76–85, 2005.

32 Ami Litman and Shiri Moran-Schein. Smooth scheduling under variable rates or the analog-digital confinement game. *Theor. Comp. Sys.*, 45(2):325–354, June 2009. doi:10.1007/s00224-008-9134-x.

33 Ami Litman and Shiri Moran-Schein. On centralized smooth scheduling. *Algorithmica*, 60(2):464–480, 2011.

34 Chung Laung Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary, 1969*, 1969.

35 Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

36 Richard T Mills, Andreas Stathopoulos, and Dimitrios S Nikolopoulos. Adapting to memory pressure from within scientific applications on multiprogrammed cows. In *Proc. 8th International Parallel and Distributed Processing Symposium (IPDPS)*, page 71, 2004.

37 Mark Moir and Srikanth Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 294–303, 1999.

38 Christian Worm Mortensen. Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 618–627, 2003.

39 Michael Rosenblum, Michel X Goemans, and Vahid Tarokh. Universal bounds on buffer size for packetizing fluid policies in input queued, crossbar switches. In *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 1126–1134, 2004.

Near-Optimal Decremental Hopsets with Applications

Jakub Łącki
Google Research, New York, NY, USA

Yasamin Nazari
Universität Salzburg, Austria

Abstract

Given a weighted undirected graph $G = (V, E, w)$, a hopset H of *hopbound* β and *stretch* $(1+\epsilon)$ is a set of edges such that for any pair of nodes $u, v \in V$, there is a path in $G \cup H$ of at most β hops, whose length is within a $(1+\epsilon)$ factor from the distance between u and v in G. We show the first efficient decremental algorithm for maintaining hopsets with a *polylogarithmic* hopbound. The update time of our algorithm matches the best known static algorithm up to polylogarithmic factors. All the previous decremental hopset constructions had a *superpolylogarithmic* (but subpolynomial) hopbound of $2^{\log^{\Omega(1)} n}$ [Bernstein, FOCS'09; HKN, FOCS'14; Chechik, FOCS'18].

By applying our decremental hopset construction, we get improved or near optimal bounds for several distance problems. Most importantly, we show how to decrementally maintain $(2k-1)(1+\epsilon)$-approximate all-pairs shortest paths (for any constant $k \geq 2$), in $\tilde{O}(n^{1/k})$ amortized update time[1] and $O(k)$ query time. This improves (by a polynomial factor) over the update-time of the best previously known decremental algorithm in the *constant* query time regime. Moreover, it improves over the result of [Chechik, FOCS'18] that has a query time of $O(\log\log(nW))$, where W is the aspect ratio, and the amortized update time is $n^{1/k} \cdot (\frac{1}{\epsilon})^{\tilde{O}(\sqrt{\log n})}$. For sparse graphs our construction nearly matches the best known static running time / query time tradeoff.

We also obtain near-optimal bounds for maintaining approximate multi-source shortest paths and distance sketches, and get improved bounds for approximate single-source shortest paths. Our algorithms are randomized and our bounds hold with high probability against an *oblivious* adversary.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic Algorithms, Data Structures, Shortest Paths, Hopsets

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.86

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2009.08416

Funding *Yasamin Nazari*: This work was conducted in part while the author was an intern at Google and a PhD student at Johns Hopkins University. Supported in part by NSF award CCF-1909111 and by Austrian Science Fund (FWF) grant P 32863-N.

Acknowledgements The authors would like to thank Michael Dinitz and Sebastian Forster for the helpful discussions.

1 Introduction

Given a weighted undirected graph $G = (V, E, w)$, a hopset H of *hopbound* β and *stretch* $(1+\epsilon)$ (or, a $(\beta, 1+\epsilon)$-hopset) is a set of edges such that for any pair of nodes $u, v \in V$, there is a path in $G \cup H$ of at most β hops, whose length is within a $(1+\epsilon)$ factor from the distance between u and v in G (see Definition 5 for a formal statement).

[1] Throughout this paper we use the notation $\tilde{O}(f(n))$ to hide factors of $O(\text{polylog}\,(f(n)))$.

Hopsets, originally defined by [13], are widely used in distance related problems in various settings, such as parallel shortest path computation [13, 15, 18, 28], distributed shortest path computation [10, 17, 29], routing tables [16], and distance sketches [14, 16]. In addition to their direct applications, hopsets have recently gained more attention as a fundamental object (e.g. [1, 4, 17, 23]), and are known to be closely related to several other fundamental objects such as additive (or near-additive) spanners and emulators [19].

A key parameter of a hopset is its hopbound. In many settings, after constructing a hopset, we can approximate distances in a time that is proportional to the hopbound. For instance, in parallel or distributed settings a hopset with a hopbound of β allows us to compute approximate single-source shortest path in β parallel rounds (e.g. by using Bellman-Ford). For many applications, such as approximate APSP (all-pairs shortest paths), MSSP (multi-source shortest paths), computing distance sketches, and diameter approximation, where we require computing distances from *many* sources, we are interested in the regime where the hopbound is polylogairthmic. Indeed, we obtain improved (and in some cases near-optimal) bounds for several of these problems in decremental settings.

In this paper, we study the maintenance of hopsets in a dynamic setting. Namely, we give an algorithm that given a weighted undirected graph G maintains a hopset of G under edge deletions. Our algorithm covers a wide range of hopbound/update time/hopset size tradeoffs. Importantly, we get the first efficient algorithm for decrementally maintaining a hopset with a *polylogarithmic* hopbound. In this case, assuming G initially has m edges and n vertices, our algorithm takes $O(mn^{\rho})$ time, given any constant $\rho > 0$, and maintains a hopset of polylogarithmic hopbound and $1+\epsilon$ stretch. This matches (up to polylogarithmic factors) the running time of the best known static algorithm [17, 18] for computing a hopset with polylogarithimic hopbound and $(1+\epsilon)$ stretch.

▶ **Theorem 1.** *Given an undirected graph $G = (V, E)$ with polynomial weights[2], subject to edge deletions, we can maintain a $(\beta, 1+\epsilon)$-hopset of size $\tilde{O}(n^{1+\frac{1}{2^k-1}})$ in total expected update time $\tilde{O}(\frac{\beta}{\epsilon} \cdot (m + n^{1+\frac{1}{2^k-1}})n^{\rho})$, where $\beta = (O(\frac{\log n}{\epsilon} \cdot (k + 1/\rho)))^{k+1/\rho+1}$, $k \geq 1$ is an integer, $0 < \epsilon < 1$ and $\frac{2}{2^k-1} < \rho < 1$.*

In the decremental setting, to the best of our knowledge, the previous state-of-the art hopset constructions have a hopbound of $2^{\tilde{O}(\log^{3/4} n)}$ [21], or $(1/\epsilon)^{\tilde{O}(\sqrt{\log n})}$ [5, 12]. As a special case, by setting $\rho = (2^k - 1)^{-1} = \frac{\log\log n}{\sqrt{\log n}}$, we can maintain a hopset with hopbound $2^{\tilde{O}(\sqrt{\log n})}$ in $2^{\tilde{O}(\sqrt{\log n})}$ amortized time. More importantly, by setting ρ and k to a constant, we can maintain a hopset of *polylogarithmic* hopbound.

While hopsets are extensively studied in other models of computation (e.g. distributed and parallel settings), their applicability in dynamic settings is less understood. Examples of results utilizing hopsets include the state-of-the art decremental SSSP algorithm for undirected graphs by Henzinger, Krinninger and Nanongkai [21], and implicit hopsets considered in [5,12]. As stated, these decremental hopset algorithms as stated only provide a *superpolylogarithmic* hopbound. It may be possible (while not discussed) to use the hop-reduction techniques of [21] (inspired by a similar technique in [5]) to obtain a wider-range of tradeoffs, however to the best of our knowledge these techniques do not lead to near-optimal size/hopbound tradeoffs[3]. Hence our result constitutes the first near-optimal decremental algorithm for maintaining hopsets with in a wide-range of settings including polylogarithmic hopbound.

---

[2] If weights are not polynomial the $\log n$ factor will be replaced with $\log W$ in the hopbound, and a factor of $\log^2 W$ will be added to the update time, where W is the aspect ratio.

[3] In particular, in all regimes the algorithm of [21] gives a hopset with size that is super-linear in number

Discussion on hopset limitations and alternative techniques. In [1] it was shown that for a $(\beta, 1+\epsilon)$-hopset with size $n^{1+\frac{1}{2^k-1}-\delta}$ for any fixed k, ϵ and $\delta > 0$ we must have[4] $\beta = \Omega_k(\frac{1}{\epsilon})^k$. Their lower bound suggests that we cannot construct a $(\beta, 1+\epsilon)$-hopset of size $\tilde{O}(n)$ with $\beta = \operatorname{poly}\log(n)$ hopbound, implying that hopsets cannot be used for obtaining optimal time (i.e. polylog amortized time) for *sparse* graphs and *very small* ϵ. However when the graph is slightly denser ($|E| = n^{1+\Omega(1)}$), the approximation factor is slightly larger (see e.g. [4,15]), or we aim to compute distances from many sources (in APSP or MSSP), using hopsets may still lead to optimal algorithms. Indeed, we show that our decremental hopsets allow us to obtain a running time matching the best static algorithm (up to polylogarithmic factors) both in $(2k-1)$-APSP and $(1+\epsilon)$-MSSP. We leave it as an open problem if hopsets can be used to obtain linear time algorithms for SSSP with *larger* approximation factors (e.g. $\epsilon \geq 1$), since as stated, the lower bound of [1] does not apply in this case.

It is worth noting that in Theorem 12 we first give a decremental algorithm that maintains static hopsets of [18] that matches the size/hopbound tradeoff in the lower bound of [1]. However, as we will see, this algorithm has a large update time, and thus we propose a new hopset with slightly worse size/hopbound tradeoff that can be maintained much more efficiently. This efficient variant has additional polylogarithmic (in aspect ratio) factors in the hopbound relative to the existentially optimal construction.

Finally, for *single source* shortest path computation in other models recently algorithms based on continuous optimization techniques are proposed (e.g. [2,3,25]) that outperform algorithms based only on combinatorial objects such as hopsets/emulators. These optimization techniques lead to much better dependence on ϵ, but are less suitable when there are many sources, as the running time scales with the number of sources. Interestingly, the authors of [2] use low-hop combinatorial structures with *larger (polylogrithmic)* stretch as a subroutine in their continuous optimization framework. Hence understanding both combinatorial and optimization directions seems crucial for distance computation in general.

1.1 Applications of Our Decremental Hopsets

To illustrate applicability of our decremental hopset algorithm, we show how it yields improved algorithms for decremetanlly maintaining shortest paths from a fixed set S of sources. We consider different variants of the problem which differ in the size of S: the single-source shortest paths (SSSP) problem ($|S| = 1$), all-pairs shortest paths (APSP) problem ($S = n$, where n is the number of vertices of the input graph), as well as the multi-source shortest paths (MSSP) problem (S is of arbitrary size), which is a generalization of the previous two.

Near-Optimal approximate APSP. We give a new decremental algorithm for maintaining approximate all-pairs shortest paths (APSP) with constant query time.

▶ **Theorem 2** (Approximate APSP). *For any constant integer*[5] $k \geq 2$, *there is a data structure that can answer* $(2k-1)(1+\epsilon)$*-approximate distance queries in a given a weighted undirected graph* $G = (V, E, w)$ *subject to edge deletions. The total expected update time*

of edges m (e.g. m^{1+p} for a parameter p), while our hopset size is $O(n^{1+p})$ for some (other but similar) parameter p, which is a constant when the hopbound is polylogarithmic. Moreover, our techniques lead to near-optimal approximate APSP, whereas it is unclear how to get comparable bounds using techniques in [21], as they do not maintain Thorup Zwick-based clusters.

[4] Ω_k hides exponential a factor of roughly $1/(k2^k)$. As written in [1] they assume k is constant (and hence the sparse hopset regime is not covered), but they also indicate that a tighter analysis could change the exact relationship between k and ϵ and hence allow a better k dependence and covering the sparse case (see Theorem 4.6 and Remark 4.7 in [1]).

[5] The k here should not be confused with the parameter k in the hopset size.

over any sequence of edge deletions is $\tilde{O}(mn^{1/k})$ and the expected size of the data structure is $\tilde{O}(m + n^{1+1/k})$. Each query for the distance between two vertices is answered in $O(k)$ worst-case time.

Our result improves upon a decremental APSP algorithm by Chechik [12] in a twofold way. First, for constant k, our update time bound is better by a $(1/\epsilon)^{O(\sqrt{\log n})}$ factor. Second, we bring down the query time from $O(\log\log(nW))$ to constant. We note that in the area of distance oracles a major goal is to preprocess a data structure that can return a distance estimate in *constant* time [11, 27, 30, 36][6].

Our results match the best known static algorithm with the same tradeoff (up to $(1+\epsilon)$ in the stretch and polylog in time) by Thorup-Zwick [34] for sparse graphs. For dense graphs there have been improvements by [36] in static settings.

Prior to [12], Roditty and Zwick [31] gave an algorithm for maintaining Thorup-Zwick distance oracles in total time $\tilde{O}(mn)$, stretch $(2k-1)(1+\epsilon)$ and $O(k)$ query time for *unweighted graphs*. Later on, Bernstein and Roditty [9] gave a decremental algorithm for maintaining Thorup-Zwick distance oracles in $O(n^{2+1/k+o(1)})$ time using emulators also only for *unweighted graphs*.

Distance Sketches. Another application of our hopsets with polylogarithmic hopbound is a near-optimal decremental algorithm for maintaining distance sketches (or distance labeling); an important tool in the context of distance computation. The goal is to store a small amount of information, a sketch, for each node, such that the distance between any pair of nodes can be approximated only using their sketches (without accessing the rest of the graph). Distance sketches are particularly important in networks, and distributed systems [16, 32], and large-scale graph processing [14]. Their significance is that at query time we only need to access/communicate the small sketches rather than having to access the whole graph. This is specially useful for processing large data when queries happen more frequently than updates.

The Thorup-Zwick [34] algorithm can be used to obtain distance sketches of expected size $O(kn^{1/k})$ (for each node) that supports $(2k-1)$-approximate queries in $O(k)$ time (in static settings), and this is known to be tight assuming a well-known girth conjecture. Our approximate APSP data structure has the additional property that the information stored for each node is a distance sketch of expected size $O(kn^{1/k})$ that supports $(2k-1)(1+\epsilon)$-approximate queries. Hence we can maintain distance sketches that almost match the guarantees of the best static algorithm. More specifically, for a fixed size our algorithm matches the best known static construction up to a $(1+\epsilon)$-factor in the stretch and polyloagrithmic factors in the update time. In decremental settings, distance oracles of [34], and hence distance sketches with the guarantees described are studied by [9, 31], but our total update time of $\tilde{O}(mn^{1/k})$ (for constant $k \geq 2$) significantly improves over these results. In particular [31] maintains these distance sketches in a total update time of $\Omega(mn)$, and [9] requires total update time of $O(n^{2+1/k+o(1)})$.

Near-Optimal $(1+\epsilon)$-MSSP. Our next result is a near-optimal algorithm for multi-source shortest paths.

[6] We need to store the original graph in addition to the distance oracle in order to update the distances and maintain correctness, however we do *not* need the whole graph for *querying distances* as we will also point out in describing the applications in maintaining distance sketches.

▶ **Theorem 3** (MSSP). *There is a data structure which given a weighted undirected graph $G = (V, E)$* explicitly *maintains $(1+\epsilon)$-approximate distances from a set of s sources in G under edge deletions, where $0 < \epsilon < \frac{1}{2}$ is a constant. Assuming that $|E| = n^{1+\Omega(1)}$ and $s = n^{\Omega(1)}$, the total expected update time is $\tilde{O}(sm)$. The data structure is randomized and works against an oblivious adversary.*

We note that total update time matches (up to polylogarithmic factors) the running time of the best known *static* algorithm for computing $(1+\epsilon)$-approximate distances from s sources for a wide range of graph densities. While for very dense graphs, using algorithms based on fast matrix multiplication is faster, the running time of our decremental algorithm matches the best known results in the static settings (up to polylogarithmic factors) whenever $ms = n^{\delta}$, for a constant $\delta \in (1, 2.37)$.

In the dynamic setting, our algorithm improves upon algorithms obtained by using hopsets of Henzinger, Krinninger and Nanongkai [21], or emulators of Chechik [12], both of which give a total update time of $O(sm \cdot 2^{\tilde{O}(\log^{\gamma} n)}), 0 < \gamma < 1$ (for a constant γ). In particular, by maintaining a hopset with polylogarithmic hopbound in $\tilde{O}(sm)$ time, we can maintain approximate SSSP from each source in $\tilde{O}(m)$ time. In contrast, in [12, 21] with hopset of hopbound $2^{\tilde{O}(\log^{\gamma} n)}$ is maintained, which if one simply applies existing techniques, results in a total update time of $m2^{\tilde{O}(\log^{\gamma} n)}$. In the general case, i.e., for very sparse graphs, the update bound of our algorithm is $sm2^{\tilde{O}(\sqrt{\log n})}$, which is similar but slightly better than the bound obtained by [21], and slightly improves over dependence on ϵ over [12].

Improved bounds for $(1+\epsilon)$-SSSP. Finally, in order to better demonstrate how our techniques compare to previous work, we show that we can obtain a slightly improved bound for decremental single-source shortest paths.

▶ **Theorem 4.** *Given an undirected and weighted graph $G = (V, E)$, there is data structure for maintaining $(1+\epsilon)$-approximate distances from a source $s_0 \in V$ under edge deletions, where $0 < \epsilon < 1$ is a constant and $|E| = n \cdot 2^{\tilde{\Omega}(\sqrt{\log n})}$. The total expected update time of the data structure is $m \cdot 2^{\tilde{O}(\sqrt{\log n})}$. There is an additional factor of $O(\frac{1}{\epsilon})^{\frac{\sqrt{\log n}}{\log\log n}}$ in the running time for non-constant ϵ.*

The amortized update time of our algorithm over all m deletions is $2^{\tilde{O}(\sqrt{\log n})}$. This improves upon the state-of-the art algorithm of [21], whose amortized update time is $2^{\tilde{O}(\log^{3/4} n)}$. We note that the techniques of [12] can also be used to obtain $(1+\epsilon)$-SSSP in amortized update time $\tilde{O}(1/\epsilon)\sqrt{\log n}$. This is close to our update time, but we get a better bound with respect to the dependence on ϵ.

Recent developments on decremental shortest paths. Recently and after a preprint of this paper was published, a decremental *deterministic* $(1+\epsilon)$-SSSP also with amortized update time of $n^{o(1)}$ was proposed by [8]. Several other recent results have also focused on deterministic dynamic shortest path algorithms or algorithms that work against an *adaptive adversary* (e.g. [6–8, 20]) most of which also use hopsets or related objects such as emulators. Our work leaves an open problem on whether hopsets with small hopbound can also be maintained and utilized deterministically[7]. This could have applications in

[7] One possible direction is considering derandomization of Throup-Zwick based clustering in static settings [34] combined with our techniques.

deterministic approximate all-pairs shortest paths, which could in turn have implications in using decremental shortest path algorithms for obtaining faster algorithms in classic/static settings (e.g. see [26]).

Hopsets vs. emulators. A majority of the previous work on dynamic distance computation are based on sparse *emulators* (e.g. [5, 9, 12]). For a graph $G = (V, E)$, an emulator $H' = (V, E')$ is a graph such that for any pair of nodes $x, y \in V$, there is a path in H' that approximates the distance between x and y on G (possibly with both multiplicative and additive factors). While there are some similarities in algorithms for constructing these objects, their analysis is different. More importantly, their maintenance and utilization for dynamic shortest paths have significant differences. An emulator approximates distances without using the original graph edges and hence we can restrict the computation to a sparser graph, whereas for using hopsets we also need the edges in the original graph. On the other hand, hopsets allow one to only consider paths with few hops.

1.2 Preliminaries and Notation

Given a weighted undirected graph $G = (V, E, w)$, and a pair $u, v \in V$ we denote the (weighted) shortest path distance by $d_G(u, v)$. We denote by $d_G^{(h)}(u, v)$ the length of the shortest path between u and v among the paths that use at most h-hops, and call this the h-hop limited distance between u and v.

In this paper, we are interested in designing *decremental* algorithms for distance problems in weighted graphs. In the decremental setting, the updates are only edge deletions or weight increases. This is as opposed to an *incremental* setting in which edges can be inserted, or a *fully dynamic* setting, in which we have both insertions and deletions. Specifically, given a weighted graph $G = (V, E, w)$, we want to support the following operations: DELETE$((u, v))$, where $(u, v) \in E$, which removes the edge (u, v), DISTANCE(s, u), which returns an (approximate) distance between a source s and any $u \in V$, and INCREASE$((u, v), \delta)$, which increases the weight of the edge (u, v) by $\delta > 0$. While our results also allow handling weight increases, in stating our theorems for simplicity we use the term *total update time* to refer to a sequence of up to m deletions.

▶ **Definition 5.** *Let $G = (V, E, w)$ be a weighted undirected graph. Fix $d, \epsilon > 0$ and an integer $\beta \geq 1$. A $(d, \beta, 1+\epsilon)$-*hopset *is a graph $H = (V, E(H), w_H)$ such that for each $u, v \in V$, where $d_G(u, v) \leq d$, we have $d_G(u, v) \leq d_{G \cup H}^{(\beta)}(u, v) \leq (1+\epsilon) d_G(u, v)$. We say that β is the* hopbound *of the hopset and $1+\epsilon$ is the* stretch *of the hopset. We also use $(\beta, 1+\epsilon)$-hopset to denote a $(\infty, \beta, 1+\epsilon)$-hopset.*

We sometimes call a $(d, \beta, 1+\epsilon)$-hopset a *d-restricted hopset*, when the other parameters are clear. We also sometimes consider hopset edges added for a specific distance range $(2^j, 2^{j+1}]$, which we call a hopset for a single distance *scale*.

In analyzing dynamic algorithms we sometimes also use a time subscript t to denote a distance (or a weight) after the first t updates. In particular we use $d_{t,G}(u, v)$ to denote the distance between u and v after t updates, and similarly use $d_{t,G}^{(h)}(u, v)$ to denote h-hop limited distance between u and v at time t.

2 Overview of Our Algorithms

The starting point of our algorithm is a known static hopset construction [18, 23]. We first review this construction. As we shall see, maintaining this data structure dynamically directly would require update time of up to $O(mn)$. Our first technical contribution is another

hopset construction that captures some of the properties of the hopsets of [18, 23], but can be maintained efficiently in a decremental setting. We then explain how by hierarchically maintaining a sequence of data structures we can obtain a near-optimal time and stretch tradeoff.

2.1 Static Hopset of [16]

In this section we outline the (static) hopset construction of Elkin and Neiman [18][8] (which is similar to [23]). We will later give a new (static) hopset algorithm that utilizes some of the properties of this construction but with modifications that allows us to maintain a *similar* hopset dynamically.

▶ **Definition 6** (Bunches and clusters). *Let $G = (V, E, w)$ be a weighted, n-vertex graph, k be an integer such that $1 \leq k \leq \log\log n$ and $\rho > 0$. We define sets $V = A_0 \supseteq A_1 \supseteq ... \supseteq A_{k+1/\rho+1} = \emptyset$. Let $\nu = \frac{1}{2^k-1}$. Each set A_{i+1} is obtained by sampling each element from A_i with probability $q_i = \max(n^{-2^i \cdot \nu}, n^{-\rho})$.*

Fix $0 \leq i \leq k + 1/\rho + 1$ and for every vertex $u \in A_i \setminus A_{i+1}$, let $p(u) \in A_{i+1}$ be the node of A_{i+1}, which is closest to u, and let $d(u, A_{i+1}) := d(u, p(u))$ (assume $d(u, \emptyset) = \infty$). We call $p(u)$ the pivot *of u. We define a* bunch *of u to be a set $B(u) := \{v \in A_i : d(u, v) < d(u, A_{i+1})\}$. Also, we define a set $C(v)$, called the* cluster *of $v \in A_i \setminus A_{i+1}$, defined as $C(v) = \{u \in V : d(u, v) < d(u, A_{i+1})\}$.*

Note that if $v \in B(u)$ then $u \in C(v)$, but the converse does not necessarily hold. The way we define the bunches and clusters here follows [18], but differs slightly from the definitions in [31, 34], where each vertex has a separate bunch and cluster defined for each level i (and stores the union of these for all levels).

The clusters are *connected* in a sense that if a node $u \in C(v)$ then any node z on the shortest path between v and u is also in $C(v)$. This property is important for bounding the running time (as also noted in [31, 34]):

▷ Claim 7. Let $u \in C(v)$, and let $z \in V$ be on a shortest path between v and u. Then $z \in C(v)$.

Proof. Let $v \in A_i$. If $z \notin C(v)$ then by definition $d(z, A_{i+1}) \leq d(v, z)$. On the other hand, since z is on the shortest path between u and v: $d(u, A_{i+1}) \leq d(z, u) + d(z, A_{i+1}) \leq d(u, z) + d(z, v) = d(u, v)$, which contradicts the fact that $u \in C(v)$. ◁

The hopset is then obtained by adding an edge (u, v) for each $u \in A_i \setminus A_{i+1}$ and $v \in B(u) \cup \{p(u)\}$, and setting the weight of this edge to be $d(u, v)$. These distances can be computed by maintaining the clusters.

▶ **Lemma 8** ([18, 23]). *Let $G = (V, E, w)$ be a weighted, n-vertex graph, k be an integer such that $1 \leq k \leq \log\log n$ and $0 < \rho, 0 < \epsilon < 1$. Assume the sets A_i and bunches are defined as in Definition 6. Define a graph $H = (V, E_H, w_H)$, such that for each $u \in A_i \setminus A_{i+1}$ and $v \in B(u) \cup \{p(u)\}$, we have an edge $(u, w) \in E_H$ with weight $d_G(u, v)$. Then H is a $(\beta, 1 + \epsilon)$-hopset of size $O(n^{1+\frac{1}{2^k-1}})$, where $\beta = (O(\frac{k+1/\rho}{\epsilon})^{k+1/\rho+1}$.*

[8] In [18] two algorithms with different sampling probabilities are given, where one removes a factor of k in the size. This factor does not impact our overall running time, so we will use the simpler version.

For reference we sketch a proof of the hopset properties in the full version. Our main result is based on a new construction consisted of a hierarchy of hopsets. Our dynamic hopset requires a new *stretch* analysis as estimates on the shortest paths are obtained from different data structures, but the *size* analysis is basically the same.

While we are generally interested in a hopset that is not much denser than the input, as we will see the running time (both in static and dynamic settings) is mainly determined by the number of clusters a node belongs to, rather than the size of the hopset. Moreover, unlike an emulator, for computing the distances using a hopset, we also need to consider the edges in G, and a small hopbound is the key to efficiency rather than the sparsity.

The hopset of [18] has some structural similarities to the emulators of [35]. One main difference is that the sampling probabilities are adjusted (lower-bounded by $n^{-\rho}$) to allow for efficient construction of these hopsets in various models, at the cost of slightly weaker size/hopbound tradeoffs. This adjustment is also crucial for our efficient decremental algorithms. Inspired by the construction described, in Section 2.2 we describe a *new static* hopset algorithm, and later in Section 2.3 we adapt it to decremental settings.

2.2 New static hopset based on path doubling and scaling

As a warm-up, before moving to our new dynamic hopset construction, we provide a simple static hopset and explain why we expect to maintain such a structure more efficiently than the structure in Section 2.1 in *dynamic settings*. Our main contribution is to maintain a dynamic hopset *efficiently* using ideas in the simple algorithm described in this section.

At a high level, computing one of the main components of the hopset of Lemma 8 involves multiple single-source shortest paths computations. Maintaining single-source shortest paths is easy in the decremental setting, if we limit ourselves to paths of low length (or allow approximation). Namely, assuming integer edge weights, one can maintain single source shortest paths up to length d under edge deletions in total $O(md)$ time.

If we simply modified the construction of the hopset of Lemma 8, and computed shortest paths up to length d instead of shortest paths of unbounded length, we would obtain a d-restricted hopset. We describe this idea in more detail in Section 3, where we show that an adaptation of the techniques by [31] allows us to maintain a d-restricted hopset in decremental settings, in total time $O(dmn^{\rho})$, for a parameter $0 < \rho < \frac{1}{2}$. However, for large d such a running time is prohibitive. In order to address this challenge, in this section we describe a static hopset, which can be computed using shortest path explorations up to only a polylogarithmic depth, yet can be used to approximate arbitrarily large distances. In the next sections we leverage this property to maintain a similar hopset in the decremental setting efficiently. This will require overcoming other obstacles, notably the fact that the dynamic shortest path problems that we need to solve are not decremental.

Path doubling. Assume that we are given a procedure $\textsc{Hopset}(G, \beta, d, \epsilon)$ that constructs a $(d, \beta, 1+\epsilon)$ hopset. In Section 3, we provide such an algorithm that uses only shortest path computation up to polylogarithmic depth. We argue that by applying the $\textsc{Hopset}(G, \beta, d, \epsilon)$ procedure *repeatedly* we can compute a full hopset, and in this process by utilizing the previously added hopset edges we can restrict our attention to short-hop paths only.

More formally, we construct a sequence of graphs $H_0, \ldots, H_{\log W}$, such that H_j is a hopset that handles pairs of nodes with distance in range $[2^{j-1}, 2^j)$, for $0 \leq j \leq \log W$. This implies that $\bigcup_{r=0}^{j} H_r$ is a $(2^j, \beta, (1+\epsilon)^j)$-hopset of G. Note that for $0 \leq j \leq \log \beta$ we can set $H_j = \emptyset$, since G covers these scales. We would like to use $G \cup_{r=0}^{j-1} H_r$ to construct H_j based on the following observation that has been previously used in other (static) models (e.g. parallel hopsets of Cohen [13]).

Consider $u, v, d_G(u,v) \in [2^{j-1}, 2^j)$, and let π be the shortest path between u and v in G. Then π can be divided into three segments π_1, π_2 and π_3, where π_1 and π_3 have length at most 2^{j-1} and π_2 consists of a single edge. But we know there is a path in $G \cup_{r=0}^{j-1} H_r$ with at most β-hops that approximates each of π_1 and π_2. Hence for constructing H_j we can compute *approximate* shortest paths by restricting our attention to paths of consisting of at most $2\beta+1$ *hops* in $G \cup_{r=0}^{j-1} H_r$.

This idea, which we call path doubling, has been previously used in hopset constructions in distributed/parallel models (e.g. [13,17,18]), but to the best of our knowledge this is the first use of this approach in a dynamic setting. Applying this idea in parallel/distributed settings is relatively straight-forward, since having bounded hop paths already leads to efficient parallel shortest path explorations (e.g. by using Bellman-Ford). However, utilizing it efficiently in dynamic settings is more involved for several reasons: we have to simultaneously maintain the clusters (including their connectivity property), apply a scaling idea on the whole structure, and handle insertions in our hopset algorithm and its analysis.

But first we describe a *scaling idea* that at a high-level allows to go from *h-hop-bounded* explorations to h-depth bounded explorations on a scaled graph.

Scaling. We review a scaling algorithm that allows us to utilize the path doubling idea. Similar scaling techniques are used in dynamic settings [5,9,21] for single-source shortest paths, but as we will see, using the scaling idea in our setting is more involved since it has to be carefully combined with other components of our construction.

This idea can summarized in the following scaling scheme due to Klein and Subramanian [24], which, roughly speaking, says that finding shortest paths of length $\in [2^{j-1}, 2^j)$ and at most ℓ hops, can be (approximately) reduced to finding paths of length at most $O(\ell)$ in a graph with in integral weights. This is done by a rounding procedure that adds a small *additive* term of roughly $\frac{\epsilon_0 w(e)}{\ell}$ to each edge e. Then for a path of ℓ hops the overall stretch will be $(1+\epsilon_0)$.

▶ **Lemma 9** ([24]). *Let $G=(V,E,w)$ be a weighted undirected graph. Let $R \geq 0$ and $\ell \geq 1$ be integers and $\epsilon_0 > 0$. We define the* scaled graph *to be a graph $\textsc{Scale}(G, R, \epsilon_0, \ell) := (V, E, \hat{w})$, such that $\hat{w}(e) = \lceil \frac{w(e)}{\eta(R,\epsilon_0)} \rceil$, where $\eta(R,\epsilon_0) = \frac{\epsilon_0 R}{\ell}$. Then for any path π in G such that π has at most ℓ hops and weight $R \leq w(\pi) \leq 2R$ we have,*

- $\hat{w}(\pi) \leq \lceil 2\ell/\epsilon_0 \rceil$,
- $w(\pi) \leq \eta(R,\epsilon_0) \cdot \hat{w}(\pi) \leq (1+\epsilon_0) w(\pi)$.

Similar scaling ideas have been used in the h-SSSP algorithm for maintaining approximate shortest paths [5]. The algorithm maintains a collection of trees and to return a distance estimate, it finds the tree that best approximates a given distance. But we note that in utilizing the scaling techniques in our final dynamic hopset construction we cannot simply maintain a *disjoint set of* bounded hop shortest path trees. We need to maintain the whole structure of the hopset on the scaled graphs together: firstly, based on definition of bunches in Lemma 8, nodes keep on leaving and joining clusters, so we cannot simply maintain a set of shortest trees from a fixed set of roots. We need to maintain the *connectivity* of clusters as described in Section 2.1 at the same time as maintaining the shortest path trees. Additionally, while we are maintaining distances over the set of clusters we also need to handle insertions introduced by smaller scales.

To maintain these efficiently, we need to apply the scaling to the whole structure, including the hopset edges added so far. But when we utilize the smaller scale hopset edges (for applying path doubling) insertions or distance decreases are introduced. As we will see, handling insertions at the same time the clusters (and the corresponding distances) are updated makes the stretch/hopbound analysis more involved.

Next we combine the scaling with the path doubling techniques. The path doubling property states that we can restrict our attention to $(2\beta+1)$-hop limited shortest path computation, and the scaling idea ensures that such $(2\beta+1)$-*hop* bounded paths in $G \cup \bigcup_{r=0}^{j-1} H_r$ correspond to paths bounded *in depth* by $d = \lceil \frac{2\ell}{\epsilon_0} \rceil = O(\frac{\beta}{\epsilon_0})$ in the scaled graph $G_{scaled} = \textsc{Scale}(G \cup \bigcup_{r=0}^{j-1} H_r, 2^j, \epsilon_0, 2\beta+1)$. Informally, this mean it is enough to construct shortest path trees up to depth ℓ on the scaled graphs in our hopset construction.

We can now summarize our new static hopset construction in Algorithm 1. Similarly to Scale, for a graph $G = (V, E, w)$ we define $\textsc{Unscale}(G, R, \epsilon, \ell)$ to be a graph (V, E, w'), where for each $e \in E$, $w'(e) = \eta(R, \epsilon) \cdot w(e)$. In static settings, the procedure $\textsc{Hopset}(G, \beta, d, \epsilon)$ for constructing a d-restricted hopset can be performed by running a $(\beta, 1+\epsilon)$-hopset construction algorithm in which the shortest path explorations are restricted to depth ℓ. In Section 3 we describe a decremental algorithm for this procedure, and describe how it leads to a $(d, \beta, 1+\epsilon)$-restricted hopset. Note that we can set $\beta = \text{poly} \log n$, and so the shortest path explorations can be bounded by a polylogairthmic value.

Algorithm 1 Simple static hopset.

for $j = 1$ *to* $\lceil \log W \rceil$ **do**
- $G_{scaled} := \textsc{Scale}(G \cup \bigcup_{r=0}^{j-1} H_r, 2^j, \epsilon_0, 2\beta+1)$
- $\hat{H} := \textsc{Hopset}(G_{scaled}, \beta, \lceil \frac{2(2\beta+1)}{\epsilon_0} \rceil, \epsilon)$
- $H_j := \textsc{Unscale}(\hat{H}, 2^j, \epsilon_0, 2\beta+1))$
- $H := H \cup H_j$

It is not hard to see that in such a static construction, three different approximation factors are combined in each scale: a $(1+\epsilon)$-stretch due to the Hopset procedure, a $(1+\epsilon_1)$-factor from the restricted hopset, and a $(1+\epsilon_0)$-factor due to scaling. This is summarized in the following lemma.

▶ **Lemma 10.** *Let G be a graph and H be a $(d, \beta, 1+\epsilon_1)$ hopset of G. Let $G_{scaled} = \textsc{Scale}(G \cup H, d, \epsilon_0, 2\beta+1)$ and let $H' = \textsc{Unscale}(\textsc{Hopset}(G_{scaled}, d, \lceil \frac{2(2\beta+1)}{\epsilon_0} \rceil, \epsilon_2))$. Then $H \cup H'$ is a $(2d, \beta, (1+\epsilon)(1+\epsilon_1)(1+\epsilon_0))$ hopset of G.*

Obtaining such a guarantee in dynamic settings is going to be more involved, since we also need to handle insertions, and at the same time ensure that the update time remains small. Moreover the stretch analysis will require combining estimates obtained by different procedures.

2.3 Near-Optimal Decremental Hopsets

In this section we describe how we can overcome the challenges of the dynamic settings in order to maintain a decremental hopset in near-optimal update-time.

The first step of our algorithm is constructing a d-restricted version of the hopset described in Section 2.1. As discussed, for this we can use the techniques by [31] to maintain a $(d, \beta, 1+\epsilon)$-hopset in $\tilde{O}(dmn^{\rho})$ total update time, where $0 < \rho < \frac{1}{2}$. Now in order to remove the time dependence on d, we use the path doubling and scaling ideas described as follows: we maintain this data structure on a sequence of scaled graphs simultaneously, and argue that this data structure gives us a hopset on G after *unscaling* the edge weights.

Sequence of restricted hopsets. Similiar to Section 3.1, our decremtnal algorithm maintains the sequence of graphs $H_0, \ldots, H_{\log W}$, where for each $0 \leq j \leq \log W$, $\bigcup_{r=0}^{j} H_r$ is a $(2^j, \beta, (1+\epsilon)^j)$-hopset of G. For *each scale* we show the following:

▶ **Lemma 11.** *Consider a graph $G = (V, E, w)$ subject to edge deletions, and parameters $0 < \epsilon < 1, \rho < \frac{1}{2}$. Assume that we have maintained $\bar{H}_j := H_1, ..., H_j$, which is a $(2^j, \beta, (1+\epsilon)^j)$-hopset of G. Then given the sequence of changes to G and $\bar{H}_j$, we can maintain a graph H_{j+1}, such that $\bar{H}_j \cup H_{j+1}$ is a $(2^{j+1}, \beta, (1+\epsilon)^{j+1})$-hopset of G. This restricted hopset can be maintained in $\tilde{O}((m+\Delta)n^\rho \cdot \frac{\beta}{\epsilon})$ total time, where m is the initial size of G, and Δ is the number of edges inserted to $\bar{H}_j$ over all updates, $\beta = (\frac{1}{\epsilon \cdot \rho})^{O(1/\rho)}$.*

Note that the lemma does not hold for *any* restricted hopset, and in dynamic settings we need to use special properties of our construction to prove this.

To prove this lemma we use the techniques of [31] to maintain the clusters. For obtaining near-optimal update time, we combine this algorithm with the path doubling and scaling ideas described earlier. However, in order to utilize these ideas, we need to deal with the fact that inserting hopset edges from smaller scales introduces *insertions*.

Handling insertions. In addition to maintaining clusters and distances decrementally, in our final construction we need to handle edge *insertions*. This is because we run it on a graph $G \cup \bigcup_{r=0}^{j-1} H_r$ (after applying scaling of Lemma 9). While edges of G can only be deleted, new edges are added to the H that we need to take into account for obtaining faster algorithms.

At a high-level, the algorithm of Roditty and Zwick [31] decrementally maintains a collection of single-source shortest path trees (up to a bounded depth) using the Even-Shiloach algorithm (ES-tree) [33] at the same time as maintaining a clustering. To handle edge insertions, we modify the algorithm to use the *monotone* ES-tree idea proposed by [21,22].

The goal of a monotone ES-tree is to support edge insertions in a limited way. Namely, whenever an edge (u, v) is inserted and the insertion of the edge causes a distance decrease in the tree, we do *not* update the currently maintained distance estimates. Still the inserted edge may impact the distance estimates in later stages by preventing some estimates from increasing after further deletions.

While it is easy to see that this change keeps the running time roughly the same as in the decremental setting, analyzing the correctness is a nontrivial challenge. This is because the existing analyses of a monotone ES-tree work under specific structural assumptions and do not generalize to any construction. Specifically, while [21] analyzed the stretch incurred by running monotone ES-trees on a hopset, the proof relied on the properties of the specific hopset used in their algorithm. Since the hopset we use is quite different, we need a different analysis, which combines the static hopset analysis, with the ideas used in [21], and also take into account the stretch incurred due to the fact that the restricted hopsets are maintained on the scaled graphs. Note also that our main hopset is not a simply a decremental maintenance of hopsets of [16], as our estimates are obtained from a *sequence of hopsets* and insertions in one scale introduce insertions in the next scale. This is why we need a new argument and cannot simply rely on arguments in [21] and [16].

Putting it together. We now go back to the setting of Lemma 11, and use a procedure similar to Algorithm 1. Given a 2^j-restricted hopset $\bar{H}_j = H_1 \cup ... \cup H_j$ for distances up to 2^j, we can now construct a graph G^j by applying the scaling of Lemma 9 to $G \cup \bar{H}_j$ and setting $R = 2^j$, $\ell = 2\beta + 1$. Then we can efficiently maintain an ℓ-restricted hopset on G^j. Then by Lemma 11 we can use this to update H_{j+1}. Importantly, ℓ is independent of R, and

thus we can eliminate the factor R to get $\tilde{O}(\beta mn^{\rho})$ total update time. Our final algorithm is a hierarchical construction that maintains the restricted hopsets on scaled graphs and the original graph simultaneously.

Stretch and hopbound analysis. As discussed, applying the path-doubling idea to the hopset analysis is straightforward in static settings (and can be to some extent separated from the rest of the analysis) as is the case in [18]. However in our adapted decremental hopset algorithm this idea needs to be combined with the properties of the monotone ES tree idea and the fact that distance estimate are obtained from a sequence of hopsets on the scaled graph. In particular, in our stretch analysis we need to divide paths into smaller segments, such that the length of some segments is obtained from smaller iterations i, and the length of some segments are obtained from this combination of monotone ES tree estimates based on path doubling and scaling. We need a careful analysis to show that the stretch obtained from these different techniques combine nicely, which is based on a threefold inductive analysis:

1. An induction on i, the iterations of the base hopset, which controls the sampling rate and the resulting size and hopbound tradeoffs.
2. An induction on the scale j, which allows us to cover all ranges of distances $[2^j, 2^{j+1}]$ by maintaining distances in the appropriate scaled graphs.
3. An induction on time t that allows us to handle insertions by using the estimates from previous updates in order to keep the distances monotone.

The overall stretch argument needs to deal with several error factors in *addition to* the base hopset stretch. First, the error incurred by using hopsets for smaller scales, which we deal with by maintaining our hopsets by setting $\epsilon' = \frac{\epsilon}{\log n}$. This introduces polylogarithmic factors in the hopbound. The second type of error comes from the fact that the restricted hopsets are maintained for scaled graphs, which implies the clusters are only approximately maintained on the original graph. This can also be resolved by further adjusting ϵ'. Finally, since we use an idea similar to the monotone ES tree of [21, 22], we may set the level of nodes in each tree is to be larger than what it would be in a static hopset. But we argue that the specific types of insertions in our algorithm will still preserve the stretch. At a high-level this is because in case of a decrease we use an estimate from time $t-1$, which we can show inductively has the desired stretch. We note that while the monotone ES tree is widely used, we always need a different construction-specific analysis to prove the correctness.

Technical differences with previous decremental hopsets. We note that while the use of monotone ES tree and the structure of the clusters in our construction are similar to [21], our algorithm has several important technical differences. First, our hopset algorithm is based on different base hopset with a polylogarithmic hopbound, which as noted is crucial for obtaining near-optimal bounds in most of our applications. Additionally, we use a different approach to maintain the hopset efficiently by using path doubling and maintaining restricted hopsets on a sequence of *scaled graphs*. Among other things, in [21] a notion of *approximate ball* is used that is rather more lossy with respect to the hopbound/stretch tradeoffs. By maintaining restricted hopsets on scaled graphs, we are also effectively preserving approximate clusters/bunches with respect to the original graph, but as explained earlier, the error accumulation combines nicely with the path-doubling idea. Moreover, [21] uses an edge sampling idea to bound the update time, which we can avoid by utilizing the sampling probability adjustments in [18], and the ideas in [31]. Finally, our algorithm is based on maintaining the clusters up to a low hop, whereas they directly maintain bunches/balls.

2.4 Applications in Decremental Shortest Paths

Our algorithms for maintaining approximate distances under edge deletions are as follows. First, we maintain a $(\beta, 1+\epsilon)$-hopset. Then, we use the hopset and Lemma 9 to reduce the problem to the problem of approximately maintaining short distances from a single source. For our application in MSSP and APSP the best update time is obtained by setting the hopbound to be polylgarithmic whereas for SSSP the best choice is $\beta = 2^{\tilde{O}(\sqrt{\log n})}$. Using this idea for SSSP and MSSP mainly involves using the monotone ES tree ideas described earlier. Maintaining the APSP distance oracle is slightly more involved but uses the same techniques as in our restricted hopset algorithm. This algorithm is based on maintaining Thorup-Zwick distance oracle [34] more efficiently. At a high-level, we maintain *both* a $(\beta, 1+\epsilon)$-hopset and Thorup-Zwick distance oracle simultaneously, and balance out the time required for these two algorithms. The hopset is used to improve the time required for maintaining the distance oracle from $O(mn)$ (as shown in [31]) to $O(\beta mn^{1/k})$, but with a slightly weaker stretch of $(2k-1)(1+\epsilon)$. Querying distances is then the same as in the static algorithm of [34], and takes $O(k)$ time. In the full version of this paper, we explain how our hopset can be used for applications in approximate shortest paths and distance sketches.

3 Decremental Hopset

In this section we describe two decremental hopset algorithms with different tradeoffs. The starting-point of our constructions are the static hopsets described in Section 2.1. But in order to get an efficient dynamic algorithm, we need to modify this construction in several ways. First we explain how we can adapt ideas by Roditty-Zwick [31] to obtain an algorithm for computing a d-restricted hopset. The total running time of this algorithm is $O(dmn^{\rho})$ (where $\rho < 1$ is a constant). While existentially this construction matches the state-of-the-art static hopsets with respect to size and hopbound tradeoffs, the update time is undesirable for large values of d, and thus in Section 3.1 we explain how we can remove this dependence from the running time at the cost of a slightly worst hopbound guarantee.

Maintaining a restricted hopset. We start by adapting the decremental algorithm by [31] that maintains the Thorup-Zwick distance oracles [34] with stretch $(2k-1)$ for pairs of nodes within distance d in $\tilde{O}(dmn^{1/k})$ total time, but we use it to obtain a d-restricted hopset. In particular, using ideas in [31], and by restricting the shortest path trees up to depth d, we can maintain a variant of the hopset defined in Lemma 8 in which the hopset guarantee only holds for nodes within distance d. In the full version, we describe how we adapt the algorithm of [31] to our settings to prove the following theorem.

▶ **Theorem 12.** *Fix $\epsilon > 0, k \geq 2$ and $\rho \leq 1$. Given a graph $G = (V, E, w)$ with integer and polynomial weights, subject to edge deletions we can maintain a $(d, \beta, 1+\epsilon)$-hopset, with $\beta = O\left(\left(\frac{1}{\epsilon} \cdot (k + 1/\rho)\right)^{k+1/\rho+1}\right)$ in $O(d(m + n^{1+\frac{1}{2^k-1}})n^{\rho})$ total time. The size/hopbound guarantee holds with high probability against an oblivious adversary.*

This algorithm has a large update time for d-restricted hopsets, when d is large. Next we show how we can eliminate this update time dependence on d, which is the main technical component of this work.

3.1 Decremental hopsets with improved update time

Next we provide a new hopset algorithm that is based on maintaining these restricted hopsets on a sequence of scaled graphs, and show how this improves the update time, in exchange for a small (polylogarithmic) loss in the hopbound.

Recall that our algorithm maintains a sequence of graphs $H_0, \ldots, H_{\log W}$, where for each $1 \leq j \leq \log W$, $H_0 \cup \ldots \cup H_j$ is a 2^j-restricted hopset of G. Instead of computing each H_j separately, we use $G \cup \bigcup_{r=0}^{j-1} H_r$ to construct H_j. The first technical challenge is making the running time independent of the distance bound 2^j, which is what we would get by directly using the algorithm of [31]. We observe that at the cost of some small approximation errors, any path of length $\in [2^{j-1}, 2^j)$ in G can be approximated by a path of at most $2\beta+1$ hops in $G \cup \bigcup_{r=0}^{j-1} H_r$. This relies on having the 2^j-restricted hopset $\bar{H}_j$, which allows us to maintain the hopset $\bar{H}_{j+1}$.

Second, while G is undergoing deletions, H_j may be undergoing edge *insertions* incurred by restricted hopset edges added for smaller scales, which we discuss next. All the missing details in this section can be found in the full-version of this paper.

Handling edge insertions. We handle edge insertions by combining of the monotone ES-tree algorithm [21] (and further used in the hopset construction of [22]). We summarize this idea and the relevant properties in the full version. As stated earlier, the algorithm itself is a simple extension of the Even-Schiloach tree. At a high-level we maintain an ES tree for each cluster and when an insertion causes the level of a node in an ES tree to decrease, we ignore the insertion and keep the same level. The more challenging aspect of using the monotone ES tree idea is proving the correctness (stretch), as this does not extend to all types of insertions but only for insertion with certain inductive structural properties. That is why even though this idea is widely used, we always need a construction-specific proof of correctness. In Theorem 14 we prove that specifically for the insertions in our final hopset algorithm the use of monotone ES tree does not violate our stretch argument.

Path doubling and scaling. We first state the path doubling idea more formally for a *static hopset* in the following lemma. However for utilizing this idea dynamically we need to combine it with other structural properties of our hopsets and the two algorithms described above.

▶ **Lemma 13.** *Given a graph $G = (V, E)$, $0 < \epsilon_1 < 1$, the set of $(\beta, 1+\epsilon_1)$-hopsets $H_r, 0 \leq r < j$ for each distance scale $(2^r, 2^{r+1}]$, provides a $(1+\epsilon_1)$-approximate distance for any pair $x, y \in V$, where $d(x, y) \leq 2^{j+1}$ using paths with at most $2\beta+1$ hops.*

This implies that it is enough to compute $(2\beta+1)$-hop limited distances in restricted hopsets for *each* scale. For using this idea in dynamic settings we have to deal with some technicalities. We should show that we can combine the rounding with the modification needed for handling insertions.

A hierarchy of restricted hopsets. We define a scaled graph using Lemma 9 as follows: $G^j := \textsc{Scale}(G \cup \bigcup_{r=0}^{j} H_r, 2^j, \epsilon_2, 2\beta+1)$. Here we set $R = 2^j, \ell = 2\beta+1$, and ϵ_2 is a parameter that we tune later. We first describe the operations performed on this scaled graph. We then explain how we can put things together for all scales to get the desired guarantees. The key insight for scaling $G \cup \bigcup_{r=0}^{j} H_r, 2^j$ is that we can obtain H_{j+1} by computing an $O(\ell)$-restricted hopset of G^j (using the algorithm of Lemma 11) and scaling back the weights of the hopset edges.

In addition to the graph G undergoing deletions, our decremental algorithm maintains the following data structures for each $1 \leq j \leq \log W$:

- The set $\bar{H}_j = \bigcup_{r=0}^{j} H_r$, union of all hopset edges for distance scales up to $[2^j, 2^{j+1}]$.
- The scaled graphs $G^1, ..., G^j$.
- Data structure obtained by constructing an $O(\beta/\epsilon_2)$-restricted hopset on G^j using Theorem 12 for the appropriate parameter $\epsilon_2 < 1$. We denote this data structure by D_j.

We update the data structures described as follows: we maintain d-restricted hopsets for $d = \lceil \frac{2(2\beta+1)}{\epsilon_2} \rceil$ starting on $j = 0, ..., \log W$ in increasing order of j to compute hopset edges H_j. After processing all the changes in scaled graph G^j, we add the inserted edges to G^{j+1}. Then we process the changes in G^{j+1} by computing a d-restricted hopset again and repeat until all distance scales of covered. As described, when the distances increase a node may join a new cluster which will lead to a set of insertions in H and in turn insertions in a sequence of graphs G^j. Note that we need to update both the restricted hopsets on the scaled graphs (denoted by D_j) and the hopset H_j for G obtained by scaling back the distances using Lemma 9. A pseudocode of this algorithm and its running time analysis can be found in the full version.

Hopset stretch and hopbound. We next show the stretch and hopbound of the hopset algorithm described for a single-scale by combining properties of the monotone ES-tree algorithm with the static hopset argument and the rounding framework.

▶ **Theorem 14.** *Given a graph $G = (V, E)$, and $0 \leq \epsilon_2 < 1$, assume that we have maintained a $(2^j, \beta, (1+\epsilon_j))$-restricted hopset $\bar{H}_j$, and let H_{j+1} be the hopset obtained by running the above algorithm on $G \cup \bar{H}_j$. Fix $0 < \delta \leq \frac{1}{8(k+1/\rho+1)}$, and consider a pair $x, y \in V$ where $d_{t,G}(x,y) \in [2^j, 2^{j+1}]$. Then for $0 \leq i \leq k + 1/\rho + 1$, either of the following conditions holds:*

1. $d^{((3/\delta)^i)}_{G \cup \bar{H}_{j+1}}(x,y) \leq (1+8\delta i)(1+\epsilon_j)(1+\epsilon_2) d_{t,G}(x,y)$ *or,*
2. *There exists $z \in A_{i+1}$ such that,*

$$d^{((3/\delta)^i)}_{G \cup \bar{H}_{j+1}}(x,z) \leq 2(1+\epsilon_j)(1+\epsilon_2) d_{t,G}(x,y).$$

Moreover, by maintaining a monotone ES tree on G^{j+1} up to depth $\lceil \frac{2(2\beta+1)}{\epsilon_2} \rceil$, and applying the rounding in Lemma 9, we can maintain $(1+\epsilon_{j+1})$-approximate single-source distances up to distance 2^{j+2} from a fixed source s on G, where $1+\epsilon_{j+1} = (1+\epsilon_j)(1+\epsilon_2)^2(1+\epsilon)$ and $\beta = (3/\delta)^{k+1/\rho+1}$.

Proof. The stretch argument is based on a threefold induction on i, j-th scale, and time t. By fixing i, j, t, and a source node s, we show that there is a $(1+\epsilon_j)$-stretch path between s and any other node with β hops (or if we are using previous scale $2\beta+1$-hops) such that each segment of this path has the desired stretch based on the inductive claim on one of these three factors. At a high level induction on i and j follows from static properties of our hopset. To show that bounded depth monotone ES tree maintains the approximate distances, we note that any segment of the path undergoing an insertion consistent of a single shortcut and the weight on such an edge is a distance estimate between its endpoints.

It is easy to see that we never underestimate distances. Roughly speaking, we either obtain an estimate from rounding estimates obtained from smaller scales, which is an upper bound on the original estimate, or we ignore a distance decrease.

We use a double induction on i and time t, and also rely on distance computed up to scaled graph G^j. First, using these distance estimates for smaller scales, we argue that when we add an edge to $\bar{H}_{j+1}$ it has the desired stretch. Let $L_{t,j}(u,v)$ denote the level of node v

in the tree rooted at u after running the monotone ES tree up to depth $D = \lceil \frac{2(2\beta+1)}{\epsilon} \rceil$ on graph G^j. This proof is based on a cyclic argument: assuming we have correctly maintained distances up to a given scale using our hopset, we show how we can compute the distances for the next scale. In particular, we first assume that based on the theorem conditions we are given $\bar{H}_j$ and have maintained all the clusters and the corresponding distances in $G^1, ..., G^j$ with stretch $1 + \epsilon_j$. This lets us analyze H_{j+1}. Then to complete the argument, we show how given the hopsets of scale $[2^j, 2^{j+1}]$, we can compute approximate SSSP distances for the next scale based on the monotone ES tree on G^{j+1}.

First, in the following claim, we observe that the edge weights inserted in the latest scale have the desired stretch by using our inductive assumption that all the shortest path trees on each cluster on $G^1, ..., G^j$ are approximately maintained. We use such distance to add edges in each cluster to construct H_{j+1}, and we observe the following about the weights on these edges:

▶ **Observation 15.** *Let $v \in B(u)$ such that $d_{t,G}(u, v) \leq 2^{j+1}$. Consider an edge (u, v) added to H_{j+1} after running the algorithm on $G^1, ..., G^j$ for $D = \lceil \frac{2(2\beta+1)}{\epsilon_2} \rceil$ rooted at node v. Let $w_{j+1}(u, v) := \min_{r=1}^{j} \eta(2^r, \epsilon_2) L_r(u, v)$, that is the unscaled edge weight. Then we have $d_{t,G}(u, v) \leq w_{j+1}(u, v) \leq (1 + \epsilon_j)(1 + \epsilon_2) d_{t,G}(u, v)$.*

This claim implies that the weights of hopset edges assigned by the algorithm correspond to approximate distance of their endpoints. Let $d_{t,j}(x, y) := \min_{r=1}^{j} \eta(2^r, \epsilon_2) L_{t,j}(x, y)$ which would be the estimate we obtain by for distance between x and y after scaling back distances on G^j. In other words this is the hop-bounded distance after running monotone ES tree on G^j and scaling up the weights.

For any time t and the base case of $i = 0$, we have three cases. If $y \in B(x)$ then edge (x, y) is in the hopset H_{j+1}, and by Observation 15 the weight assigned to this edge is at most $(1 + \epsilon_j)(1 + \epsilon_2) d_{t,G}(x, y)$. In this case the first condition of the theorem holds. Otherwise if $x \in A_1$, then $z = x$ trivially satisfies the second condition. Otherwise we have $x \in A_0 / A_1$, and by setting $z = p(x)$ we know that there is an edge $(x, z) \in \bar{H}_j$ such that $d_{t,j}(x, z) \leq (1 + \epsilon_2) d_{G \cup \bar{H}_j}(x, y)$ (by definition of $p(x)$ and using the same argument as above). Hence the second condition holds.

By inductive hypothesis assume the claim holds for i. Consider the shortest path $\pi(x, y)$ between x and y. We divide this path into $1/\delta$ segments of length at most $\delta d_{t,G}(x, y)$ and denote the a-th segment by $[u_a, v_a]$, where u_a is the node closest to x (first node of distance at least $a\delta d_{t,G}(x, y)$) and v_a is the node furthest to x on this segment (of distance at most $(a + 1)\delta d_{t,G}(x, y)$).

We then use the induction hypothesis on each segment. First consider the case where for all the segments the first condition holds for i, then there is a path of $(3/\delta)^i (1/\delta) \leq (3/\delta)^{i+1}$ hops consisted of the hopbounded path on each segment. We can show that this path satisfies the first condition for $i + 1$. In other words,

$$d^{((3/\delta)^{i+1})}_{t,G \cup \bar{H}_{j+1}}(x, y) \leq \sum_{a=1}^{1/\delta} d^{((3/\delta)^i)}_{t,G \cup \bar{H}_{j+1}}(u_a, v_a) + d^{(1)}_{t,G}(v_a, u_{a+1}) \leq (1 + 8\delta i)(1 + \epsilon_j)(1 + \epsilon_2) d_{t,G}(x, y)$$

Next, assume that there are at least two segments for which the first condition does not hold for i. Otherwise, if there is only one such segment a similar but simpler argument can be used. Let $[u_l, v_l]$ be the first such segment (i.e. the segment closest to x, where u_l is the first and v_l is the last node on the segment), and let $[u_r, v_r]$ be the last such segment.

First by inductive hypothesis and since we are in the case that the second condition holds for segments $[u_l, z_l]$ and $[u_r, v_r]$, we have,

- $d^{((3/\delta)^i)}_{t,G\cup\bar{H}_{j+1}}(u_l, z_l) \leq 2(1+\epsilon_2)(1+\epsilon_j)d_{t,G}(u_l, v_l)$, *and,*
- $d^{((3/\delta)^i)}_{t,G\cup\bar{H}_{j+1}}(v_r, z_r) \leq 2(1+\epsilon_2)(1+\epsilon_j)d_{t,G}(u_r, v_r)$

Again, we consider two cases. First, in case $z_r \in B(z_l)$ (or $z_l \in C(z_r)$), we have added a single hopset edge $(z_r, z_l) \in \bar{H}_{j+1}$. Note that $d_{t,G}(z_r, z_l) \leq 2^{j+1}$, since $d_{t,G}(z_r, z_l) \leq d_{t,G}(x, y) \leq 2^{j+1}$. Hence by Observation 15 the weight we assign to (z_r, z_l) is at most $(1+\epsilon_2)(1+\epsilon_j)d_{t,G}(z_r, z_l)$.

On the other hand, by triangle inequality, and the above inequalities (which are based on the induction hypothesis) we get,

$$d^{(1)}_{\bar{H}_{j+1}}(z_l, z_r) \leq (1+\epsilon_2)(1+\epsilon_j)d_G(z_l, z_r) \tag{1}$$

$$\leq (1+\epsilon_2)(1+\epsilon_j)[d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_l, z_l) + d_G(u_l, v_r) + d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(z_r, v_r)] \tag{2}$$

By applying the inductive hypothesis on segments before $[u_l, v_l]$, and after $[u_r, v_r]$, we have a path with at most $(3/\delta)^i$ for each of these segments, satisfying the first condition for the endpoints of the segment. Also, we have a $2(3/\delta)^i + 1$-hop path going through u_l, z_l, z_r, v_r that satisfies the first condition for u_l, v_r.

Putting all of these together, we argue that there is a path of hopbound $(3/\delta)^{i+1}$ satisfying the first condition. In particular, we have (the subscript t is dropped in the following),

$$d^{(3/\delta)^{(i+1)}}_{G\cup\bar{H}_{j+1}}(x, y) \leq \sum_{a=1}^{l-1}[d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_a, v_a) + d^{(1)}_G(v_a, u_{a+1})] + d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_l, z_l) \tag{3}$$

$$+ d^{(1)}_{\bar{H}_{j+1}}(z_l, z_r) + d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(z_r, v_r) + d^{(1)}_G(v_r, u_{r+1}) \tag{4}$$

$$+ \sum_{a=r+1}^{1/\delta}[d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_a, v_a) + d^{(1)}_G(v_a, u_{a+1})] \tag{5}$$

$$\leq (1+8\delta i)(1+\epsilon_j)(1+\epsilon_2)[d_G(x, u_l) + d_G(v_r, y)] + d_G(u_l, v_r) \tag{6}$$

$$+ (1+\epsilon_2)(1+\epsilon_j)[2d_G(u_l, z_l) + 2d_G(z_r, v_r)] \tag{7}$$

$$\leq (1+\epsilon_2)(1+\epsilon_j)[8\delta d_G(x, y) + (1+8\delta i)d_G(x, y)] \tag{8}$$

$$\leq (1+8\delta(i+1))(1+\epsilon_2)(1+\epsilon_j)d_G(x, y) \tag{9}$$

In the first inequality we used the induction on i for each segment, and triangle inequality. In the second inequality we are using the fact that nodes u_j, v_j for all j are on the shortest path between x and y in G, and we are replacing $d^{(1)}_{\bar{H}_{j+1}}(z_l, z_r)$ with inequality 2. In line 8 we used the fact that the length of each segment is at most $\delta \cdot d_G(x, y)$. Hence we have shown that the first condition in the lemma statement holds.

Finally, consider the case where $z_r \notin B(z_l)$. If $z_l \notin A_{i+2}$, we consider $z = p(z_l)$, where $z_l \in A_{i+2}$. We now claim that this choice of z satisfies the second lemma condition.

We have added the edge (z_l, z) to the hopset. Since $z_r \notin B(z_l)$, we have $d_{t-1,G}(z_l, p(z_l)) \leq d_{t-1,G}(z_l, z_r) \leq d_{t,G}(x, y) \leq 2^{j+1}$. Therefore we can use Observation 15 on the edge $(z_l, p(z_l))$.

$$d^{(3/\delta)^{(i+1)}}_{G\cup\bar{H}_{j+1}}(x,y) \leq \sum_{a=1}^{l-1}[d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_a,v_a) + d^{(1)}_{G}(v_a,u_{a+1})] \tag{10}$$

$$+ d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_l,z_l) + (1+\epsilon_2)(1+\epsilon_j)d^{(1)}_{\bar{H}_{j+1}}(z_l,z) \tag{11}$$

$$\leq (1+8\delta i)(1+\epsilon_2)(1+\epsilon_j)d_G(x,u_l) + d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_l,z_l) \tag{12}$$

$$+ (1+\epsilon_2)(1+\epsilon_j)d_{\bar{H}_{j+1}}(z_l,z_r) \tag{13}$$

$$\leq (1+8\delta i)(1+\epsilon_2)(1+\epsilon_j)d_G(x,u_l) + d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(u_l,z_l) \tag{14}$$

$$+ (1+\epsilon_2)(1+\epsilon_j)[2d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(z_l,u_l) + d_G(u_l,v_r) + d^{(3/\delta)^i}_{G\cup\bar{H}_{j+1}}(v_r,z_r)] \tag{15}$$

$$\leq (1+8\delta i)(1+\epsilon_2)(1+\epsilon_j)d^{((3/\delta)^i)}_{G\cup\bar{H}_{j+1}}(x,v_r) + 6\delta(1+\epsilon_j)d_G(x,y) \tag{16}$$

$$\leq 2(1+\epsilon_2)(1+\epsilon_j)d_G(x,y) \tag{17}$$

In the last inequality we used the fact that we set $\delta < \frac{1}{8(k+1/\rho+1)}$ and thus $8\delta i < 1$. The only remaining case is when $z_\ell \in A_{i+2}$, in which case a similar reasoning follows by setting $z = z_l$.

Next, we need to prove that after adding hopset edges H_{j+1} we can maintain approximate single-source shortest path distances from a given source s to conclude the proof of this theorem. For this we need to use scaling again, and by Lemma 9 an additional $(1+\epsilon_2)$-factor will be added to the stretch. This enables us to show that Observation 15 can be used for the next scale, i.e. that we can set the weights for the next scale by maintaining the clusters and $(1+\epsilon_{j+1})$ approximate distance rooted at a source s when we have $d(s,v) \in [2^{j+1}, 2^{j+2}]$, and hence close the inductive cycle in the argument. This argument uses a similar type of case-by-case analysis as the above argument combined with path-doubling. We omit this argument here due to space limitations. The complete proof can be found in the full version of this paper. ◀

Theorem 14 allows us to hierarchically use the restricted hopsets for smaller scales to compute the distance for larger scales, that is in turn used to update the hopset edges in the larger scales. Finally, for getting the final stretch and hopbound we set the parameters $\epsilon' = \frac{\epsilon}{6\log W}, \epsilon_2 = \epsilon'$ (error incurred by rounding), and $\delta = \frac{\epsilon}{8(k+1/\rho+1)}$ (details can be found in the full version). Putting it all together we get the following theorem:

▶ **Theorem 16.** *The total update time in each scaled graph G^j, $1 \leq j \leq \log W$, over all deletions is $\tilde{O}((\beta/\epsilon')(n^{1+\frac{1}{2^k-1}} + m)n^\rho)$, and hence the total update time for maintaining $(\beta, 1+\epsilon)$-hopset with hopbound $\beta = O(\frac{\log W}{\epsilon} \cdot (k+1/\rho))^{k+1/\rho+1}$ is $\tilde{O}(\frac{\beta}{\epsilon} \cdot mn^\rho \cdot \log W)$.*

4 Applications in Decremental Approximate Shortest Path

In the full version of this paper, we use our hopsets to maintain approximate shortest paths and distance sketches. At a high level, we maintain a $(\beta, 1+\epsilon)$-hopset using the appropriate parameter settings in Theorem 16. The applications in $(1+\epsilon)$-SSSP and $(1+\epsilon)$-MSSP are straightforward extensions of Theorem 14.

In our approximate APSP data structure we simultaneously maintain a $(\beta, 1+\epsilon)$-hopset for by setting β to be polylogarithmic and a Thorup-Zwick distance oracle [34]. Our algorithm for maintaining distance oracles of [34] is similar to the restricted hopset algorithm, combined

with the rounding procedure in Lemma 9 that allows us to maintain clusters up to $O(\beta/\epsilon)$ hops. One main difference between these algorithms is in the information/distances stored and the fact that the sampling probabilities stay fixed in case of distance oracles. In order to maintain $(2k-1)(1+\epsilon)$-approximate APSP, we set the parameters ρ and k in such a way that updating the hopset and the distance oracle are roughly the same. By maintaining the distance oracle, querying distances is the same as in the static algorithm of [34], and takes $O(k)$ time, which is constant when k is constant.

References

1. Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.
2. Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 322–335, 2020.
3. Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *SIAM Journal on Computing*, 50(3):815–856, 2021.
4. Uri Ben-Levy and Merav Parter. New (α, β) spanners and hopsets. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1695–1714. SIAM, 2020.
5. Aaron Bernstein. Fully dynamic (2+ ε) approximate all-pairs shortest paths with fast query and close to linear update time. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 693–702. IEEE, 2009.
6. Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, page 44. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2017.
7. Aaron Bernstein and Shiri Chechik. Deterministic partially dynamic single source shortest paths for sparse graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 453–469. SIAM, 2017.
8. Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *62 Annual IEEE Symposium on Foundatios of Computer Science (FOCS 2022)*, 2021.
9. Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1355–1365. SIAM, 2011.
10. Keren Censor-Hillel, Michal Dory, Janne H Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 74–83, 2019.
11. Shiri Chechik. Approximate distance oracles with constant query time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 654–663, 2014.
12. Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 170–181. IEEE, 2018.
13. Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *Journal of the ACM (JACM)*, 2000.
14. Michael Dinitz and Yasamin Nazari. Massively parallel approximate distance sketches. *OPODIS*, 2019.
15. Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Almost shortest paths and pram distance oracles in weighted graphs. *arXiv preprint*, 2019. `arXiv:1907.11422`.

16 Michael Elkin and Ofer Neiman. Near-optimal distributed routing with low memory. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2018.

17 Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *SIAM Journal on Computing*, 2019.

18 Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in rnc. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 333–341, 2019.

19 Michael Elkin and Ofer Neiman. Near-additive spanners and near-exact hopsets, a unified view. *arXiv preprint*, 2020. `arXiv:2001.07477`.

20 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2522–2541. SIAM, 2020.

21 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 146–155. IEEE, 2014.

22 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. *SIAM Journal on Computing*, 45(3):947–1006, 2016.

23 Shang-En Huang and Seth Pettie. Thorup–zwick emulators are universally optimal hopsets. *Information Processing Letters*, 142:9–13, 2019.

24 Philip N Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 1997.

25 Jason Li. Faster parallel algorithm for approximate shortest path. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 308–321, 2020.

26 Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 121–130, 2010.

27 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. *Journal of the European Mathematical Society*, 9(2):253–275, 2007.

28 Gary L Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures*. ACM, 2015.

29 Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the ACM Symposium on Theory of computing*. ACM, 2014.

30 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages, and Programming*, pages 261–272. Springer, 2005.

31 Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 499–508. IEEE, 2004.

32 Atish Das Sarma, Michael Dinitz, and Gopal Pandurangan. Efficient distributed computation of distance sketches in networks. *Distributed Computing*, 28(5):309–320, 2015.

33 Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

34 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

35 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 802–809, 2006.

36 Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 202–208. SIAM, 2012.

Tight Vector Bin Packing with Few Small Items via Fast Exact Matching in Multigraphs

Alexandra Lassota
EPFL, Lausanne, Switzerland

Aleksander Łukasiewicz
University of Wrocław, Poland

Adam Polak
EPFL, Lausanne, Switzerland

Abstract

We solve the Bin Packing problem in $O^*(2^k)$ time, where k is the number of items less or equal to one third of the bin capacity. This parameter measures the distance from the polynomially solvable case of only large (i.e., greater than one third) items. Our algorithm is actually designed to work for a more general Vector Bin Packing problem, in which items are multidimensional vectors. We improve over the previous fastest $O^*(k! \cdot 4^k)$ time algorithm.

Our algorithm works by reducing the problem to finding an exact weight perfect matching in a (multi-)graph with $O^*(2^k)$ edges, whose weights are integers of the order of $O^*(2^k)$. To solve the matching problem in the desired time, we give a variant of the classic Mulmuley-Vazirani-Vazirani algorithm with only a linear dependence on the edge weights and the number of edges – which may be of independent interest.

Moreover, we give a tight lower bound, under the Strong Exponential Time Hypothesis (SETH), showing that the constant 2 in the base of the exponent cannot be further improved for Vector Bin Packing.

Our techniques also lead to improved algorithms for Vector Multiple Knapsack, Vector Bin Covering, and Perfect Matching with Hitting Constraints.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Bin Packing, Vector Bin Packing, Parameterized Complexity, Matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.87

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2203.10077`

Funding *Alexandra Lassota*: Swiss National Science Foundation project *Lattice Algorithms and Integer Programming* (185030).
Aleksander Łukasiewicz: Polish National Science Centre grant 2019/33/B/ST6/00298.
Adam Polak: Swiss National Science Foundation project *Lattice Algorithms and Integer Programming* (185030).

1 Introduction

NP-hard problems often have special cases that can be solved in polynomial time, e.g., Vertex Cover is tractable in graphs with the Kőnig property, Dominating Set is tractable in trees, and Longest Common Subsequence is tractable in permutations. Many of these problems remain (fixed-parameter) tractable with a distance from the polynomially solvable case taken as a parameter, e.g., Vertex Cover Above Matching [16], Dominating Set in bounded treewidth graphs [1], or Longest Common Subsequence parameterized by the maximum occurrence number [6]. In parameterized complexity, this concept is sometimes dubbed *distance from triviality* [6].

In the Bin Packing problem, we are given n *items* from $\mathbb{Q}_{\geqslant 0}$, and we have to pack them into the smallest possible number of unit-sized *bins*. It is a classic strongly NP-hard problem. When all items are *large*, i.e., greater than $1/3$, then no three items can fit into a single bin and the problem reduces to the Maximum Matching problem, and hence it can be solved in polynomial time [5].

Bannach et al. [2] are the first to study Bin Packing parameterized by the number k of *small* (i.e., $\leqslant 1/3$) items – which is the distance from the above tractable case. They give algorithms running in randomized $O^*(k! \cdot 4^k)$ time, and deterministic $O^*\big((k!)^2 \cdot 2^k\big)$ time.[1] Their randomized algorithm works even for a more general Vector Bin Packing problem, in which items are d-dimensional vectors from $\mathbb{Q}^d_{\geqslant 0}$, and a set of items fits into a bin if their coordinate-wise sum does not exceed 1 in any coordinate. (The notion of a small item is more complex in the multidimensional case; see Section 1.1 for the definition.)

We improve upon their result by giving an $O^*(2^k)$ randomized time algorithm for Vector Bin Packing. We complement it with a matching conditional lower bound, showing that the constant 2 in the base of the exponent cannot be further improved, unless the Strong Exponential Time Hypothesis (SETH) fails.

Our algorithm works by reducing the problem to finding a perfect matching of a given total weight in an edge-weighted (multi-)graph. The graph has only $O(n)$ nodes, but can have up to $O(2^k n^2)$ edges, whose weights are integers of the order of $2^k \cdot k$. To solve the matching problem in the desired $O^*(2^k)$ time, we give a variant of the classic Mulmuley-Vazirani-Vazirani algorithm [12] with only a linear dependence on the edge weights and the number of edges – which may be of independent interest.

Our techniques also lead to improved algorithms for the two other problems studied by Bannach et al. [2], i.e., the Vector Multiple Knapsack and Vector Bin Covering problems, as well as for the Perfect Matching with Hitting Constraints problem, studied by Marx and Pilipczuk [11].

1.1 Vector Bin Packing with Few Small Items

First, let us formally define Vector Bin Packing as a decision problem. We remark that (Vector) Bin Packing is also often studied as an optimization problem – especially in the context of approximation algorithm – but one can always switch between the two variants via binary search, loosing at most of a factor of $O(\log n)$ in the running time.

Vector Bin Packing

Given: a set of n items $V = \{v_1, \ldots, v_n\} \subseteq \mathbb{Q}^d_{\geqslant 0}$,
and an integer $\ell \in \mathbb{Z}_+$ denoting the number of unit-sized bins.

Decide: if the items can be partitioned into ℓ bins $B_1 \cup \cdots \cup B_\ell = V$ such that $\sum_{v \in B_i} v[j] \leqslant 1$ for every bin $i \in [\ell]$ and every dimension $j \in [d]$.[2]

Note that the assumption that bins are unit-sized is without loss of generality, as one can always independently scale each dimension in order to meet that constraint. We can also safely assume that V is a set, as one can handle multiple occurrences of the same item by introducing one extra dimensions with negligibly small but unique coordinates.

[1] We use $O^*(\cdot)$ notation to suppress factors polynomial in the input size n, i.e., $O^*(f(k)) = f(k) \cdot n^{O(1)}$.
[2] We use $[n]$ to denote the set of integers $\{1, 2, \ldots, n\}$.

Unlike in the one-dimensional Bin Packing problem, where a small item can be defined simply as smaller or equal to $1/3$, we use a more complex definition, introduced by Bannach et al. [2]. Let $V \subseteq \mathbb{Q}^d_{\geqslant 0}$ be a set of d-dimensional items. We say that a subset $V' \subseteq V$ is **3-incompatible** if no three distinct items from V' fit into a unit-sized bin, i.e., for every distinct $u, v, w \in V'$ there exists a dimension $i \in [d]$ such that $u[i] + v[i] + w[i] > 1$. Now we can define the parameterized problem that we study.

Vector Bin Packing with Few Small Items

Parameter:	the number of *small* items k.
Given:	a set of n items $V = \{v_1, \ldots, v_n\} \subseteq \mathbb{Q}^d_{\geqslant 0}$, a subset of k items $V_S \subseteq V$ such that $V_L = V \setminus V_S$ is **3-incompatible**, and an integer $\ell \in \mathbb{Z}_+$ denoting the number of unit-sized bins.
Decide:	if the items can be partitioned into ℓ bins $B_1 \cup \cdots \cup B_\ell = V$ such that $\sum_{v \in B_i} v[j] \leqslant 1$ for every bin $i \in [\ell]$ and every dimension $j \in [d]$.

We say that items in V_S are *small*, and the remaining items in $V_L = V \setminus V_S$ are *large*. Note that we assume that a subset of small items is specified in the input. This way we can study the complexity of the packing problem independently of the complexity of finding a (smallest) subset of small items. This is similar, e.g., to the standard practice for treewidth parameterization, where one assumes that a suitable tree decomposition is given in the input (see, e.g., [4]). We remark that if only the set of all items V is given, a smallest possible subset of small items can be found in $O^*(2.27^k)$ time [14] via a reduction to the 3-Hitting Set problem [2].

1.2 Our results

Our main result is an $O^*(2^k)$ time randomized algorithm for Vector Bin Packing with Few Small Items. The algorithm consists of two parts: reducing the packing problem to a matching problem, and solving the matching problem. More formally, we first prove the following.

▶ **Lemma 1.** *An n-item instance of Vector Bin Packing with k small items can be reduced, in deterministic time $O(2^k n^2 kd)$, to the problem of finding an exact-weight perfect matching in a (multi-)graph. The graph has $O(n)$ vertices, $O(2^k n^2)$ edges, and non-negative integer edge weights that do not exceed $O(2^k k)$. The target exact total weight of a matching is $O(2^k k)$.*

The above matching problem is dubbed Exact Matching, and is known to be in randomized[3] (pseudo-)polynomial time since the Mulmuley-Vazirani-Vazirani algorithm [12]. The algorithm directly solves the 0/1 weights variant of Exact Matching in simple graphs. A prior reduction of Papadimitriou and Yannakakis [15] handles arbitrary non-negative integer edge weights and multiple parallel edges. The reduction replaces each edge of weight w by a path of length $2w + 1$ with alternating 0/1 edge weights.

The reduction multiplies the number of vertices by the edge weights and by the number of edges. Further, Mulmuley-Vazirani-Vazirani is not a linear time algorithm. Hence, this would give us only a $2^{O(k)} n^{O(1)}$ time algorithm for Vector Bin Packing. This is already an improvement over the previous factorial time algorithm, but still not our desired $2^k n^{O(1)}$ running time.

[3] It is a big open problem to derandomize the algorithm, see, e.g., [18].

There are more direct and faster ways to solve the general Exact Matching problem than going through the Papadimitriou-Yannakakis reduction. It seems folklore to handle arbitrary edge weights by replacing a monomial x, corresponding to a weight-one edge in the Mulmuley-Vazirani-Vazirani algorithm, with x^w, where w is the edge weight. It remains to handle multiple parallel edges. A crucial part of the algorithm is the so-called *isolation lemma*. It assigns random *costs* to edges, ensuring that the minimum cost perfect matching of the target weight is unique, and hence it cannot cancel out in the algebraic computations. The range of costs, required to ensures that property, on one hand depends on the number of edges, and on the other hand, determines the bitsize of the costs, on which the algorithm later needs to do arithmetic.

Due to the number of edges in Lemma 1, a direct application of isolation lemma would lead to an $O^*(4^k)$ time algorithm. To mitigate this issue, we carefully apply isolation lemma to pairs of vertices, and hence the number of edges appears in the running time only as a linear additive factor.

▶ **Theorem 2.** *Given an edge-weighted multigraph with n nodes and m edges, and an integer t, a randomized Monte-Carlo algorithm can decide whether there is a perfect matching of total weight exactly t in $\widetilde{O}(t \cdot n^8 + m)$ time.*

Lemma 1 and Theorem 2 together imply our main result.

▶ **Theorem 3.** *There is a randomized Monte Carlo algorithm solving Vector Bin Packing with Few Small Items in $O^*(2^k)$ time.*

Lower bound

We show that the above result is tight, via a matching conditional lower bound, under the Strong Exponential Time Hypothesis (SETH) [7]. The hypothesis states that deciding k-CNF-SAT with n variables requires time $2^{s_k n}$ for $\lim_{k\to\infty} s_k = 1$. In particular, it implies that deciding CNF-SAT requires $2^{(1-\varepsilon)n}$ time, for every $\varepsilon > 0$. SETH is a standard hardness assumption for conditional lower bounds in fine-grained and parameterized complexity [4, 20]. We prove the following lower bound for the (non-parameterized) Vector Bin Packing problem.

▶ **Theorem 4.** *Unless SETH fails, Vector Bin Packing cannot be solved in $O^*(2^{(1-\varepsilon)n})$ time, for any $\varepsilon > 0$. This holds even restricted to instances with only two bins and dimension $d = O(n)$.*

Since $k \leqslant n$, the corollary for the parameterized version of the problem follows immediately, proving that the algorithm of Theorem 3 is tight.

▶ **Corollary 5.** *Unless SETH fails, Vector Bin Packing with Few Small Items cannot be solved in $O^*(2^{(1-\varepsilon)k})$ time, for any $\varepsilon > 0$. This holds even restricted to instances with only two bins and dimension $d = O(n)$.*

We remark that our lower bound crucially relies on multiple dimensions. The best known hardness result for the (one-dimensional) Bin Packing problem rules out only $2^{o(n)}$ time algorithms [9], assuming the Exponential Time Hypothesis (ETH) [8]. It is a big open problem whether an $O(1.99^n)$ time algorithm for Bin Packing exists. Recently, Nederlof et al. [13] gave such an algorithm for any constant number of bins. This is in contrast to Vector Bin Packing, which, as we show, requires $2^{(1-\varepsilon)n}$ time already for two bins.

Other applications

Bannach et al. [2] studied two further problems closely related to the Vector Bin Packing problem – namely, Vector Multiple Knapsack and Vector Bin Covering – under similar parameterizations.

In the Vector Multiple Knapsack problem, each item comes with a *profit*, and instead of having to pack all the items, we aim to pack a subset of the items into a fixed number of bins while maximizing the overall profit of the packed items. In the few small items regime, the fastest known algorithm so far has a running time of $O^*(k! \cdot 4^k)$, where k is the number of small items [2]. Adapting our algorithm to handle the profits and the obstacle that only a subset of items might be packed, we obtain the following theorem.

▶ **Theorem 6.** *There is a randomized Monte Carlo algorithm solving Vector Multiple Knapsack with Few Small Items in $O^*(2^k)$ time when item profits are bounded by* $\mathrm{poly}(n)$.

In the Vector Bin Covering problem, we aim to *cover* bins. Intuitively speaking, instead of packing the items into as few bins as possible, we want to partition them into as many bins as possible while satisfying a *covering constraint* for each bin. This new desired property of a solution leads to a slightly different definition of the set of small items: instead of any three large items not fitting together into a bin, now they cover a bin. So far, the fastest algorithm solving this problem parameterized by the number k of small items[4] runs in time $O^*(k! \cdot 4^k)$ [2]. We give the following improvement.

▶ **Theorem 7.** *There is a randomized Monte Carlo algorithm solving Vector Bin Covering with Few Small Items in $O^*(2^k)$ time.*

Further, our results directly imply an improved running time for the Perfect Matching with Hitting Constraints problem. This problem asks whether we can find a perfect matching in a graph using at least one edge from each of given subsets of edges. It was studied by Marx and Pilipczuk [11] as a tool for solving a subgraph isomorphism problem in forests. They gave an algorithm (for the matching problem) running in time $2^{O(k)}n^{O(1)}$, where k is the number of edge subsets. Their algorithm shares certain similarities with our Vector Bin Packing algorithm. They use, however, a less efficient encoding of subsets into edge weights (using $2k$ bits, compared to $k \log k$ bits we achieve in Lemma 9), and they only coarsely analyze the polynomial dependence on the weights when solving Exact Matching. Avoiding these two inefficiencies, we prove the following theorem.

▶ **Theorem 8.** *There is a randomized Monte Carlo algorithm solving Perfect Matching with Hitting Constraints in $O^*(2^k)$ time.*

2 From Vector Bin Packing to Exact Matching

▶ **Lemma 1.** *An n-item instance of Vector Bin Packing with k small items can be reduced, in deterministic time $O(2^k n^2 kd)$, to the problem of finding an exact-weight perfect matching in a (multi-)graph. The graph has $O(n)$ vertices, $O(2^k n^2)$ edges, and non-negative integer edge weights that do not exceed $O(2^k k)$. The target exact total weight of a matching is $O(2^k k)$.*

[4] Even though Bannach et al. do not explicitly adapt their definition of a small item to this problem, they indeed work with the same definition as we do. In the full version on arXiv [3, page 11] they write: "The large vectors have the property that every subset of three vectors cover a container."

Proof. We interpret the problem of finding a packing as the problem of finding a perfect matching with a certain total weight in an edge-weighted (multi-)graph. Intuitively, each large item is represented by a vertex, and an edge connects two large items if they fit together into a bin. The edge weight indicates a set of small items which can be packed together with the endpoints (i.e., the corresponding large items). The goal is to match (pack) all large items while achieving the total weight that corresponds to all small items being assigned to some pairing of large items.

Formally, we first add $2\ell - |V_L|$ dummy items $\langle 0, \ldots, 0\rangle \in \mathbb{Q}^d_{\geqslant 0}$ to the set V_L so that each bin will contain exactly two large items. A dummy item can be paired with another dummy item (no original large item is in that bin), or with an original large item (only one original large item is in that bin). For each large item $v \in V_L$ (including the dummy items), create a vertex u_v. For each pair of large items $v_1, v_2 \in V_L$, $v_1 \neq v_2$, and for each subset $V'_S \subseteq V_S$ of small items, introduce an edge between u_{v_1} and u_{v_2} if $v_1[\ell] + v_2[\ell] + \sum_{v \in V'_S} v[\ell] \leqslant 1$ for all $\ell \in [d]$, i.e., the small items fit together with the two large ones into a bin.[5] The weight of the edge will depend on V'_S (but not on v_1 and v_2).

We need to design the edge weights such that each collection of edges of a certain total weight corresponds to a collection of subsets of small items that form a partition of the set of all small items V_s, and vice versa. A naive, but incorrect, solution would be to label the small items with integers $1, 2, \ldots, k$, and assign to a subset $X \subseteq [k]$ the integer whose binary representation corresponds to the indicator vector of X, i.e., $\sum_{x \in X} 2^{x-1}$. It is true that, with such weights, any collection of edges whose associated subsets form a partition of V_s has the total weight $1 \ldots 1_2 = 2^k - 1$. However, the reverse statement is not true: it is possible to obtain the total weight $2^k - 1$ by, e.g., taking $2^k - 1$ edges that each allow small item 1 but no other small items.

As we will show in Lemma 9, in order to prevent such false positives, it suffices to concatenate the indicator vectors with $(\log k)$-bit counters denoting the number of elements in a set.[6] More formally, we assign to a subset $X \subseteq [k]$ the weight $|X| \cdot 2^k + \sum_{x \in X} 2^{x-1}$, i.e., the $(k + \log k)$-bit integer whose k least significant bits correspond to the indicator vector of X and the $\log k$ most significant bits form the integer equal to the cardinality of X. The target total weight $k \cdot 2^k + (2^k - 1)$ can only be achieved by summing weights given to subsets forming a partition of V_s, i.e., by assigning each small item to (exactly) one matching edge. ◀

▶ **Lemma 9.** *Fix the universe size $k \in \mathbb{N}$, and let $f : 2^{[k]} \to \mathbb{N}$ be given by*

$$f(X) = |X| \cdot 2^k + \sum_{x \in X} 2^{x-1}.$$

Then, a family $X_1, \ldots, X_n \subseteq [k]$ is a partition[7] of $[k]$ if and only if

$$f(X_1) + \cdots + f(X_n) = k \cdot 2^k + (2^k - 1).$$

---

[5] Note that it is important to add an edge for each fitting subset, and not, e.g., only for inclusion-wise maximal fitting subsets. That is because we design the edge weights so that an exact matching corresponds to a partition (and not to a cover) of the set V_s.

[6] Marx and Pilipczuk [11] solve a similar issue by concatenating the indicator vector with its reverse, i.e., they assign to X weight $\sum_{x \in X}(2^{2k-x} + 2^{x-1})$. Their approach results in weights of the order of 4^k, which is prohibitively large for achieving $O^*(2^k)$ running time.

[7] That is, $X_1 \cup \cdots \cup X_n = [k]$, and $X_i \cap X_j = \emptyset$ for every $i \neq j$.

Proof. The "partition ⇒ sum" direction follows from a simple calculation. Let us prove the "sum ⇒ partition" direction. For $i \in [k]$, let c_i denote the number of sets containing element i. We want to show that $c_i = 1$, for every i. We have

$$f(X_1) + \cdots + f(X_n) = \left(\sum_{i=1}^{k} c_i\right) \cdot 2^k + \sum_{i=1}^{k} c_i 2^{i-1}.$$

Note that the k least significant bits of the sum $f(X_1) + \cdots + f(X_n)$ are lower bounding the term $\sum_{i=1}^{k} c_i 2^{i-1}$, and the remaining bits are upper bounding the term $\sum_{i=1}^{k} c_i$, that is,

$$\sum_{i=1}^{k} c_i 2^{i-1} \geqslant 2^k - 1 = \overbrace{1\ldots1}^{k \text{ ones}}_2, \quad \text{and} \quad \sum_{i=1}^{k} c_i \leqslant k.$$

For $i = 0, 1, \ldots, k$, let $p_i = c_1 + \cdots + c_i$, with $p_0 = 0$. Observe that $p_i \geqslant i$, for every i, as otherwise there are not enough bits to set the one in every position among the i least significant bits of the sum $f(X_1) + \cdots + f(X_n)$.[8] Moreover, $p_k = \sum_{i=1}^{k} c_i \leqslant k$, and thus $p_k = k$. Last but not least, by definition, $c_i = p_i - p_{i-1}$. We have

$$\begin{aligned}
2^k - 1 &\leqslant \sum_{i=1}^{k} 2^{i-1} c_i = \sum_{i=1}^{k} 2^{i-1}(p_i - p_{i-1}) = \sum_{i=1}^{k} 2^{i-1} p_i - \sum_{i=1}^{k} 2^{i-1} p_{i-1} \\
&= \sum_{i=1}^{k} 2^{i-1} p_i - \sum_{i=0}^{k-1} 2^i p_i = 2^k p_k + \sum_{i=1}^{k} (2^{i-1} - 2^i) p_i - 2^0 p_0 = 2^k p_k - \sum_{i=1}^{k} 2^{i-1} p_i \\
&= 2^k \cdot k - \sum_{i=1}^{k} 2^{i-1} p_i \\
&\leqslant 2^k \cdot k - \sum_{i=1}^{k} 2^{i-1} i = 2^k \cdot k - \big((k-1) \cdot 2^k + 1\big) = 2^k - 1.
\end{aligned}$$

Hence, all the inequalities must be tight. In particular, $p_i = i$ for every i, and thus $c_i = 1$, i.e., each element of the universe is contained in exactly one set of the family. ◀

3 Fast Exact Weight Matching in Multigraphs

In this section we give our variant of the Mulmuley-Vazirani-Vazirani algorithm, with only a linear dependence on the edge weights and a linear additive dependence on the number of edges, proving Theorem 2.

3.1 The Pfaffian

At the heart of the matching algorithm lies a computation of the Pfaffian of a skew-symmetric matrix of certain polynomials. In order to introduce the notion of a Pfaffian properly, let us fix some definitions and notation first.

[8] It follows from the fact that the number of one-bits in the sum is less or equal to the total number of one-bits in the summands, and that this holds even if we look only at the i least significant bits.

For an $n \times n$ matrix A, we denote by $A[i,j]$ the value in the i-th row and j-th column. We say that A is skew-symmetric if and only if $A[i,j] = -A[j,i]$ for every $i,j \in [n]$. Let $\mathcal{M}$ be a perfect matching in the complete graph K_n. We can look at $\mathcal{M}$ as a sequence of edges in some arbitrary order, i.e.,

$$\mathcal{M} = (i_1, j_1), (i_2, j_2), \ldots, (i_{n/2}, j_{n/2}),$$

where, by convention, $i_k \leqslant j_k$ for any k. Now, we define the sign of $\mathcal{M}$ as follows:

$$\operatorname{sgn} \mathcal{M} = \operatorname{sgn} \begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & n-1 & n \\ i_1 & j_1 & i_2 & j_2 & \cdots & i_{n/2} & j_{n/2} \end{pmatrix},$$

where the right-hand side is the sign of a permutation. One can easily show that this definition does not depend on the chosen order of the edges.

Now, we are ready to give the definition of a Pfaffian.

▶ **Definition 10** (Pfaffian). *Let A be an $n \times n$ skew-symmetric matrix. The Pfaffian of A is denoted by* $\operatorname{pf}(A)$ *and is defined as follows*

$$\operatorname{pf}(A) = \sum \left\{ \operatorname{sgn} \mathcal{M} \cdot \prod_{(i_k, j_k) \in \mathcal{M}} A[i_k, j_k] \;\middle|\; \mathcal{M} \text{ perfect matching in } K_n \right\}.$$

We note that since A is skew-symmetric, our convention that $i_k \leqslant j_k$ does not affect the definition of the Pfaffian at all – if we were to switch i_k and j_k, the sign of the matching changes, but so does the sign of the product of the weights.

Several equivalent definitions of a Pfaffian exist in the literature. However, we have chosen this one, as it immediately illustrates the connection between the Pfaffian and perfect matchings.

The Pfaffian of a matrix over an arbitrary field can be computed by, e.g., a variant of the Gaussian elimination. However, since we are dealing with polynomial matrices, we would like to avoid divisions. Fortunately, several division-free polynomial time algorithms for computing Pfaffian exist [10, 17, 19].

Incidentally, Mahajan, Subramanya, and Vinay [10] give a dynamic programming algorithm computing the Pfaffian of a matrix with entries from an arbitrary ring that makes $O(n^4)$ additions and multiplications (see also survey [17] for an alternative exposition)[9]. By analysing the structure of their algorithm, we get the following result for matrices with polynomial entries.

▶ **Theorem 11** (cf. [10], Section 4). *Given an $n \times n$ matrix A of univariate polynomials of degree at most d and integer coefficients bounded by M, the Pfaffian* $\operatorname{pf}(A)$ *can be computed in $\widetilde{O}(n^6 d \log M)$ time.*

Proof. The algorithm in [10], Section 4, is described as a weighted DAG H_A with each vertex corresponding to a state of the dynamic program. The weights on the edges are signed entries of the matrix A. There is an auxiliary starting state $s \in H_A$ and the dynamic programming value for a state $v \in H_A$ is a sum of products of weights along all the paths from s to v.

Moreover, H_A has $O(n^3)$ vertices, depth equal to $O(n)$ and indegree of each vertex equal to $O(n)$. Therefore, if the entries of A are polynomials of degree d and coefficients bounded by M, then the values of the dynamic programming states are polynomials with

[9] Urbańska's algorithm [19] runs even faster, in $O(n^{3.03})$ time. But since we care more about getting linear dependence on the target weight in our matching algorithm, rather than optimizing the polynomial dependence on n, we have chosen to use a slightly slower, yet simpler algorithm for the sake of clarity.

a degree bounded by $O(nd)$ and coefficients bounded by $O(n^n M^n)$. Hence, by using FFT, we can perform each arithmetic operation in $\widetilde{O}(n^2 d \log M)$ time. The number of arithmetic operations needed is proportional to the number of edges in H_A, which is $O(n^4)$. This yields the desired time bound. ◀

Since we do not need to compute the whole Pfaffian in the Exact Matching problem, but are only interested in the coefficient of the monomial x^t (which conveys the information about matchings of the target weight t), we can speed up the computation by a factor of n.

▶ **Corollary 12.** *Given an integer t and an $n \times n$ matrix A of univariate polynomials with integer coefficients bounded by M, a coefficient of the monomial x^t in $\mathrm{pf}(A)$ can be computed in $\widetilde{O}(n^5 t \log M)$ time.*

Proof. In the algorithm from Theorem 11, we can perform all the arithmetic operations modulo x^{t+1}. Then, the degree of the polynomials is bounded by $O(t)$ instead of $O(nd)$, and a similar analysis follows. ◀

3.2 The algorithm

We first recall the central lemma of the Mulmuley-Vazirani-Vazirani algorithm, used to deal with possible cancellations caused by varying signs in the Pfaffian definition.

▶ **Lemma 13** (Isolation Lemma, cf. [12])**.** *Let S be a finite set, and let $F \subseteq 2^S$ be a family of subsets of S. To each element $x \in S$, we assign an integer cost $c(x)$ chosen uniformly and independently at random from $\{1, \ldots, 2|S|\}$. For a subset $S' \subseteq S$, we define a total cost of S' to be $c(S') = \sum_{x \in S'} c(x)$. Then,*

$$\mathbb{P}(\textit{there is a unique minimum total cost set in } F) \geqslant \frac{1}{2}.$$

Now we are ready to present the matching algorithm.

▶ **Theorem 2.** *Given an edge-weighted multigraph with n nodes and m edges, and an integer t, a randomized Monte-Carlo algorithm can decide whether there is a perfect matching of total weight exactly t in $\widetilde{O}(t \cdot n^8 + m)$ time.*

Proof. We first present the algorithm. Then we argue its correctness and analyse the running time.

Algorithm. For every $\{u, v\} \in \binom{V}{2}$, let $E_{\{u,v\}} = \{e \in E : e \text{ connects } u \text{ and } v\}$ denote the set of (parallel) edges between nodes u and v. For an edge $e \in E$, we use $w(e) \in \mathbb{Z}_{\geqslant 0}$ to denote the weight of e. Moreover, we assume w.l.o.g. that $V = [n]$.

The algorithm works as follows.

1. Set $\lambda = 2m^n$.
2. For every $\{i, j\} \in \binom{V}{2}$, assign a cost $c(\{i, j\})$ uniformly at random from $\{1, \ldots, 2\binom{n}{2}\}$.
3. Set up an $n \times n$ matrix A of univariate polynomials: For each $i, j \in [n]$, $i \leqslant j$, put

$$A[i, j] = \lambda^{c(\{i,j\})} \sum_{e \in E_{\{i,j\}}} x^{w(e)}, \quad \text{and} \quad A[j, i] = -A[i, j].$$

4. Compute the coefficient of x^t in $\mathrm{pf}(A)$ using the algorithm from Corollary 12.
5. If the coefficient of x^t in $\mathrm{pf}(A)$ is nonzero return YES, otherwise return NO.

Correctness. We use $\mathrm{coef}_t(\mathrm{pf}(A))$ to denote the coefficient of x^t in $\mathrm{pf}(A)$. For every perfect matching $\mathcal{M}$ in the complete graph K_n, let

$$f(\mathcal{M}) = \mathrm{sgn}\ \mathcal{M} \cdot \lambda^{c(\mathcal{M})} \cdot \#\text{perfect matchings in } G \text{ of weight } t \text{ contained}^{10} \text{ in } \mathcal{M}$$

denote the contribution of matching $\mathcal{M}$ to the coefficient $\mathrm{coef}_t(\mathrm{pf}(A))$. Now, we have

$$\mathrm{coef}_t(\mathrm{pf}(A)) = \sum \{ f(\mathcal{M}) \mid \mathcal{M} \text{ perfect matching in } K_n \}. \tag{1}$$

Let F be the family of all perfect matchings in K_n that contain a perfect matching in G of weight exactly t. If $F = \emptyset$, then every summand in (1) is zero. Hence, $\mathrm{coef}_t(\mathrm{pf}(A)) = 0$ and our algorithm answers correctly.

If $F \neq \emptyset$, then by Isolation Lemma, with probability at least $1/2$, there is only one minimum cost perfect matching $\mathcal{N} \in F$.

Let $c = c(\mathcal{N})$. Observe that the number of perfect matchings in G of weight t that are contained in $\mathcal{N}$ is trivially bounded by $m^n < \lambda$. This means that $|f(\mathcal{N})| < \lambda^{c+1}$. In other words, $f(\mathcal{N})$ is divisible by λ^c, but not by λ^{c+1}. On the other hand, every other summand in (1) is divisible by λ^{c+1}, as $\mathcal{N}$ is the unique minimum cost matching. Therefore, $\mathrm{coef}_t(\mathrm{pf}(A))$ is divisible by λ^c, but not by λ^{c+1} – so it cannot be zero.

If we want to amplify the probability of giving the correct answer to $1 - 1/n^C$, for some constant $C > 0$, we repeat the algorithm $C \log n$ times.

Time cost analysis. The time needed to complete steps 1–3 is $O(n^2 + m)$. Since the coefficients of the polynomial entries of A are bounded by $2m^{n \cdot 2\binom{n}{2}} = 2^{O(n^3 \log m)}$, we get that invoking the algorithm from Corollary 12 takes $\widetilde{O}(t \cdot n^8 \log m)$ time. In total, that yields $\widetilde{O}(t \cdot n^8 + m)$ time complexity. ◀

4 Lower bound

▶ **Theorem 4.** *Unless SETH fails, Vector Bin Packing cannot be solved in $O^*(2^{(1-\varepsilon)n})$ time, for any $\varepsilon > 0$. This holds even restricted to instances with only two bins and dimension $d = O(n)$.*

Proof. Given a CNF formula with n variables and m clauses,[11] we will construct $n + 1$ instances of Vector Bin Packing such that the formula is satisfiable if and only if at least one of them is a yes-instance. Intuitively, this corresponds to guessing the number of variables set to true in a satisfying assignment. Formally, for $t \in \{0, \ldots, n\}$, the t-th Vector Bin Packing instance is a yes-instance if and only if the formula has a satisfying assignment with exactly t variables set to true.

Let us fix t. The t-th instance consists of $n + 2$ items $V = \{v_1, \ldots, v_n, T, F\} \subseteq \mathbb{Q}^{m+2}_{\geqslant 0}$. The first n items correspond to the n variables; the remaining two items T and F are used to break the symmetry – in any feasible solution they are necessarily in two different bins, which we call the T-bin and the F-bin, respectively. Packing item v_i to the T-bin corresponds to setting variable i to true, and packing it to the F-bin corresponds to setting the variable to false.

[10] We say that a matching (in multigraph G) is contained in another matching (in the complete graph K_n) if the set of $n/2$ pairs of endpoints is the same for the two matchings.

[11] Note that, thanks to the *sparsification lemma* [8], we can assume that $m = O(n)$.

The items are $(m+2)$-dimensional. The first m dimensions correspond to clauses, and we will discuss them in a moment. Dimension $m+1$ ensures that T and F go to different bins; we have $T[m+1] = F[m+1] = 1$, and $v_i[m+1] = 0$ for every $i \in [n]$. Dimension $m+2$ ensures that (at most) t items go to the T-bin and (at most) $n-t$ items go to the F-bin; we have $T[m+2] = (n-t)/n$, $F[m+2] = t/n$, and $v_i[m+2] = 1/n$ for every $i \in [n]$.

Now, fix a clause $j \in [m]$. We set

$$v_i[j] = \begin{cases} {}^{0}\!/_{2n}, & \text{if variable } i \text{ appears in a positive literal in clause } j, \\ {}^{1}\!/_{2n}, & \text{if variable } i \text{ does not appear in clause } j, \\ {}^{2}\!/_{2n}, & \text{if variable } i \text{ appears in a negative literal in clause } j. \end{cases}$$

Let n_j denote the number of variables that appear negated in clause j. We set

$$T[j] = 1 - \frac{t + n_j - 1}{2n}, \quad \text{and} \quad F[j] = 0.$$

This ends the description of the instance. To finish the proof, it remains to show that the above items can be packed into two bins if and only if the formula has a satisfying assignment with exactly t variables set to true.

Note that there is a natural one-to-one correspondence between (not necessarily satisfying) assignments that set exactly t variables to true and (not necessarily feasible) Vector Bin Packing solutions that are feasible in the last two dimensions. We now show that, for $j \in [m]$, such an assignment satisfies clause j if and only if the corresponding solution is feasible in dimension j. The F-bin is never overfull in dimension j. To analyse the T-bin, let α, β, γ denote the numbers of variables set to true that, in clause j, appear in a positive literal, do not appear, and appear in a negative literal, respectively. Let δ denote the number of variables set to false that appear in clause j in a negative literal. Note that $t = \alpha + \beta + \gamma$, and $n_j = \gamma + \delta$. Consider the following chain of equivalent inequalities, starting with the condition saying that the T-bin is not overfull in dimension j.

$$\begin{aligned} \alpha \cdot {}^{0}\!/_{2n} + \beta \cdot {}^{1}\!/_{2n} + \gamma \cdot {}^{2}\!/_{2n} &\leqslant 1 - T[j] \\ \beta + 2\gamma &\leqslant t + n_j - 1 \\ \beta + 2\gamma &\leqslant (\alpha + \beta + \gamma) + (\gamma + \delta) - 1 \\ 1 &\leqslant \alpha + \delta \end{aligned}$$

The last inequality states that clause j is satisfied. ◀

5 Other applications

In this section we explain how the techniques presented in our paper can be adapted to also solve Vector Multiple Knapsack and Vector Bin Covering, two closely related problems to the Vector Bin Packing problem. The main difference lies in the reduction to the Exact Matching problem, which has to integrate profits of the items, or the new covering property, respectively. Further, we show that our techniques directly apply to the Perfect Matching with Hitting Constraints problem, leading to an improved running time.

Vector Multiple Knapsack

In Vector Multiple Knapsack, instead of packing all items into the smallest number of bins, we aim to place a subset of items with profits into a fixed number of bins while maximizing the profit of the packed items. Like in Vector Bin Packing, small items hinder us from solving

the problem using a polynomial time algorithm for the maximum weight perfect matching. Hence, following Bannach et al. [2], we study the problem parameterized by the number k of small items.

Vector Multiple Knapsack with Few Small Items

Parameter:	the number of *small* items k.
Given:	a set of n items $V = \{v_1, \dots, v_n\} \subseteq \mathbb{Q}^d_{\geqslant 0}$, **item profits** $p(v_1), \dots, p(v_n) \in \mathbb{Z}_+$, a subset of k items $V_S \subseteq V$ such that $V_L = V \setminus V_S$ is **3-incompatible**, an integer $\ell \in \mathbb{Z}_+$ denoting the number of unit-sized bins, and an integer $P \in \mathbb{Z}_+$, denoting the **goal profit**.
Decide:	if a subset V' of the items can be partitioned into ℓ bins $B_1 \cup \dots \cup B_\ell = V'$ such that $\sum_{v \in B_i} v[j] \leqslant 1$ for every bin $i \in [\ell]$ and every dimension $j \in [d]$, and $\sum_{v \in V'} p(v) \geqslant P$.

To solve the problem, we reduce the instance to the Exact Matching problem as in Section 2. It remains to handle the fact that only a subset of items has to be packed, and that we need to integrate the profits. We do so in the following manner: With each edge between v_1 and v_2 and the weight corresponding to $V'_S \subseteq V_S$, we associate the *cost* of $p(v_1) + p(v_2) + \sum_{v \in V'_s} p(v)$. Further, we introduce $g = n - 2 \cdot \ell$ new vertices $b_1, b_2 \dots, b_g$, called *blocker* vertices. These vertices serve as "garbage collectors" for the items which are not packed in any of the ℓ bins, i.e., they match g unpacked items, and by that block them. To do so, for each $V'_S \subseteq V_S$, each large vector v_i, and each blocker vertex b_j, introduce an edge between v_i and b_j with weight dependent on V'_S as before, and cost 0. Note that, because of the dummy items introduced in the reduction in Section 2, we can assume that each bin in an optimal solution contains exactly two large items (some original, some dummy), so we know that exactly $g = n - 2 \cdot \ell$ large items has to be handled by blockers.

Using Lemma 9, clearly, each yes-instance of the Vector Multiple Knapsack problem has a prefect matching of weight exactly $k \cdot 2^k + (2^k - 1)$ and cost at least P in the above graph, and vice versa. This is due to the equivalence of choosing ℓ edges with non-zero costs and the packing of the ℓ bins. The remaining items can be matched with the blocker vertices, and all small items are covered due to the weights.

We are left with solving the following matching problem: Given a (multi-)graph with edge weight and edge costs, find a perfect matching with a given total weight and the maximum possible total cost. This can be done with a slight modification of the algorithm of Theorem 2. Indeed, note that the algorithm already looks for a perfect matching minimizing the sum of edge costs coming from Isolation Lemma. All we have to do is to (1) combine input costs with Isolation Lemma costs, and (2) turn minimization into maximization. For (1), it suffices to put the input cost into the most significant bits, and the Isolation Lemma cost into the least significant bits of the combined edge cost. For (2), to find out what the maximum (instead of the minimum) possible total cost is, it suffices to look at the most (instead of the least) significant digit in the λ-ary representation of the coefficient $\operatorname{coef}_t(\operatorname{pf}(A))$. Last but not least, we remark that Isolation Lemma is symmetric with respect to minimization/maximization, i.e., it also ensures that the maximum total cost set is unique with probability at least $1/2$.

To analyze the running time, let $p_{\max} = \max_{v \in V} p(v)$ denote the maximum item profit. The maximum input cost of an edge is $(k+2)p_{\max}$. Hence, the coefficients of the polynomial entries of matrix A are now bounded by $2m^{(k+2)p_{\max} n \cdot 2\binom{n}{2}} = 2^{O(p_{\max} n^4 \log m)}$, and the matching algorithm takes $\widetilde{O}(t \cdot p_{\max} n^9 \log m)$ time. This leads to the following theorem.

▶ **Theorem 6.** *There is a randomized Monte Carlo algorithm solving Vector Multiple Knapsack with Few Small Items in $O^*(2^k)$ time when item profits are bounded by* $\mathrm{poly}(n)$.

Vector Bin Covering

Another set of problems asks to *cover* the largest number of bins possible. In the one-dimensional setting, covering typically refers to the bin capacity being exceeded by the set of items packed into it. This property can be extended in multiple ways to a d-dimensional case, for example by requiring that at least one dimension is exceeded. However, other properties, such as "all dimension have to be exceeded", "certain set combinations of dimensions have to be exceeded", et cetera, are possible as well. Our algorithm works for all such definitions of covering. Thus, in the following, we refer to the one chosen as the covering property $\mathcal{P}$.

Following our story line to study a parameter capturing a distance to triviality, we consider the problem variant parameterized by the number k of small items. However, the property of being a small item depends on $\mathcal{P}$, so we introduce a new definition for the covering problems: We say that a subset $V' \subseteq V$ is **3-covering** if every three distinct items from V' cover a unit-sized bin w.r.t. $\mathcal{P}$.

Vector Bin Covering with Few Small Items

Parameter:	the number of *small* items k.
Given:	a set of n items $V = \{v_1, \ldots, v_n\} \subseteq \mathbb{Q}^d_{\geqslant 0}$, a subset of k items $V_S \subseteq V$ such that $V_L = V \setminus V_S$ is **3-covering** w.r.t. $\mathcal{P}$, and an integer $\ell \in \mathbb{Z}_+$ denoting the number of unit-sized bins.
Decide:	if the items can be partitioned into ℓ bins $B_1 \cup \cdots \cup B_\ell = V$ such that $\sum_{v \in B_i} v$ satisfies $\mathcal{P}$ for every bin $i \in [\ell]$.

The algorithm proceeds similarly to the one for Vector Bin Packing. However, we have to handle the fact that a bin can contain more than two large items in this case. Thus, we first guess the number of bins ℓ_i admitting i large items for $i \in \{0, 1, 2\}$. This yields $O(\ell^3) = O(n^3)$ guesses. The remaining bins will be covered by triples of the unassigned large items. Hence, the guess has to satisfy that $\ell_0 + \ell_1 + \ell_2 + \lfloor (n - k - \ell_1 - 2\ell_2)/3 \rfloor \geqslant \ell$.

Now we construct the graph as in Section 2 with $2\ell_0 + \ell_1$ dummy items. For each $V'_S \subseteq V_S$, an edge is introduced between v_1 and v_2 if $v_1 + v_2 + \sum_{v \in V'_S} v$ covers the bin w.r.t. $\mathcal{P}$. The weight of the edge is defined by V'_S as before. Additionally, we introduce $(n - k - \ell_1 - 2\ell_2)$ *blocker* vertices, and introduce an edge of weight 0 between each blocker vertex and each large item. The blocker vertices collect all large items not being placed into bins with 0, 1, or 2 large items.

With Lemma 9 being proven, clearly, each yes-instance of the Vector Bin Covering problem has a perfect matching of weight $k \cdot 2^k + (2^k - 1)$ in the above graph, and vice versa. Indeed, a matching has to choose $(n - k - \ell_1 - 2\ell_2)$ edges between blocker vertices and large items. These are the ones greedily packed as triples. Note that this might leave up to two large items unpacked, which will be assigned to an arbitrary, already covered bin. The remaining packing is defined by the remaining matching edges as previously.

This together with Theorem 2 leads to the following result.

▶ **Theorem 7.** *There is a randomized Monte Carlo algorithm solving Vector Bin Covering with Few Small Items in $O^*(2^k)$ time.*

Perfect Matching with Hitting Constraints

The Perfect Matching with Hitting Constraints problem asks whether there exists a perfect matching in a graph using at least one edge from each given set of edges. Formally, the problem is defined as follows.

Perfect Matching with Hitting Constraints

Parameter:	the number of edge subsets k.
Given:	a graph $G = \langle V, E \rangle$, and k (not necessarily disjoint) edge subsets $E_1, \ldots, E_k \subseteq E$.
Decide:	if there is a perfect matching M in G such that there exists k *distinct* edges $e_1, \ldots, e_k \in M$ such that $e_i \in E_i$ for every $i \in [k]$.

We again reduce this problem to finding an exact weight perfect matching in a multigraph. Our approach is similar to the one of Marx and Pilipczuk [11]. However, in their reduction, they introduce larger edge weights, and, by that, obtain a larger running time. We can circumvent this using edge weights as defined in Lemma 9.

In detail, we create a copy of each edge $e \in E_i$, for each $i \in [k]$, and assign weight $1 \cdot 2^k + 2^{i-1}$ to it – i.e., we concatenate the indicator vector of the singleton $\{i\}$ with the counter set to 1, as previously. The original edge gets weight 0. The target weight is $t = k \cdot 2^k + (2^k - 1)$. Clearly, there exists a perfect matching with hitting constraints in G if and only if there is a perfect matching in the transformed graph with edge weights summing up to the correct target value t, see Lemma 9.

This together with Theorem 2 leads to the following result.

▶ **Theorem 8.** *There is a randomized Monte Carlo algorithm solving Perfect Matching with Hitting Constraints in $O^*(2^k)$ time.*

References

1 Jochen Alber, Hans L. Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. Fixed parameter algorithms for DOMINATING SET and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002. doi:10.1007/s00453-001-0116-5.

2 Max Bannach, Sebastian Berndt, Marten Maack, Matthias Mnich, Alexandra Lassota, Malin Rau, and Malte Skambath. Solving Packing Problems with Few Small Items Using Rainbow Matchings. In *MFCS*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.11.

3 Max Bannach, Sebastian Berndt, Marten Maack, Matthias Mnich, Alexandra Lassota, Malin Rau, and Malte Skambath. Solving Packing Problems with Few Small Items Using Rainbow Matchings. *CoRR*, abs/2007.02660, 2020. arXiv:2007.02660.

4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

5 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

6 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *IWPEC*, volume 3162 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2004. doi:10.1007/978-3-540-28639-4_15.

7 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

8 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *FOCS*, pages 653–663. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743516.

9 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM Journal on Discrete Mathematics*, 30(1):343–366, 2016. doi:10.1137/140952636.

10 Meena Mahajan, P. R. Subramanya, and V. Vinay. A combinatorial algorithm for Pfaffians. In *COCOON*, COCOON'99, pages 134–143, Berlin, Heidelberg, 1999. Springer-Verlag. doi:10.1007/3-540-48686-0_13.

11 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *STACS*, volume 25 of *LIPIcs*, pages 542–553. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.542.

12 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, March 1987. doi:10.1007/BF02579206.

13 Jesper Nederlof, Jakub Pawlewicz, Céline M. F. Swennenhuis, and Karol Wegrzycki. A faster exponential time algorithm for bin packing with a constant number of bins via additive combinatorics. In *SODA*, pages 1682–1701. SIAM, 2021. doi:10.1137/1.9781611976465.102.

14 Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003. doi:10.1016/S1570-8667(03)00009-1.

15 Christos H Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM*, 29(2):285–309, 1982. doi:10.1145/322307.322309.

16 Igor Razgon and Barry O'Sullivan. Almost 2-SAT is fixed-parameter tractable (extended abstract). In *ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2008. doi:10.1007/978-3-540-70575-8_45.

17 Günter Rote. Division-free algorithms for the determinant and the Pfaffian: algebraic and combinatorial approaches. In *Computational Discrete Mathematics: advanced lectures*, pages 119–135. Springer-Verlag, 2001. doi:10.1007/3-540-45506-X_9.

18 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-NC. In *FOCS*, pages 696–707. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.70.

19 Anna Urbańska. Faster combinatorial algorithms for determinant and Pfaffian. In *Algorithms and Computation, 18th International Symposium, ISAAC 2007*, pages 599–608. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-77120-3_52.

20 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *ICM*, pages 3447–3487. World Scientific, 2018. doi:10.1142/9789813272880_0188.

Parameterized Complexity of Untangling Knots

Clément Legrand-Duchesne
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France

Ashutosh Rai
Department of Mathematics, IIT Delhi, Hauz Khas, New Delhi, India

Martin Tancer
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

Deciding whether a diagram of a knot can be untangled with a given number of moves (as a part of the input) is known to be NP-complete. In this paper we determine the parameterized complexity of this problem with respect to a natural parameter called defect. Roughly speaking, it measures the efficiency of the moves used in the shortest untangling sequence of Reidemeister moves.

We show that the II^- moves in a shortest untangling sequence can be essentially performed greedily. Using that, we show that this problem belongs to W[P] when parameterized by the defect. We also show that this problem is W[P]-hard by a reduction from MINIMUM AXIOM SET.

2012 ACM Subject Classification Mathematics of computing → Geometric topology; Theory of computation → Problems, reductions and completeness

Keywords and phrases unknot recognition, parameterized complexity, Reidemeister moves, W[P]-complete

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.88

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2111.05001

Funding *Clément Legrand-Duchesne*: Part of this work has been done when C. L.-D. visited Charles University, whose stay was partially supported by the GAČR grant 19-04113Y.
Ashutosh Rai: Supported by SERB grant SRG/2021/002412.
Martin Tancer: Supported by the GAČR grant 19-04113Y.

1 Introduction

A classical and extensively studied question in algorithmic knot theory is to determine whether a given diagram of a knot is actually a diagram of an unknot. This question is known as the *unknot recognition problem*. The first algorithm for this problem was given by Haken [12]. Currently, it is known that the unknot recognition problem belongs to NP ∩ co-NP but no polynomial time algorithm is known. See [13] for the NP-membership and [19] for co-NP-membership (co-NP-membership modulo Generalized Riemann Hypothesis was previously established in [16]). In addition, a quasi-polynomial time algorithm for unknot recognition has been recently announced by Lackenby [20].

One possible path for attacking the unknot recognition problem is via Reidemeister moves (see Figure 2): if D is a diagram of an unknot, then D can be untangled to a diagram U with no crossing by a finite number of Reidemeister moves. In addition, Lackenby [17] provided a polynomial bound (in the number of crossings of D) on the required number of Reidemeister moves. This is an alternative way to show that the unknot recognition problem belongs to NP, because it is sufficient to guess the required Reidemeister moves for unknotting.

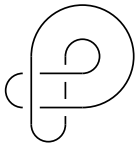


Figure 1 An example of a diagram.

However, if we slightly change our viewpoint, de Mesmay, Rieck, Sedgwick, and Tancer [23] showed that it is NP-hard to count the number of required Reidemeister moves exactly. (An anologous result for links has been shown to be NP-hard slightly earlier by Koenig and Tsvietkova [15].) More precisely, it is shown in [23] that given a digram D and a parameter k as input, it is NP-hard to decide whether D can be untangled using at most k Reidemeiser moves. For more background on unknotting and unlinking problems, we also refer to Lackenby's survey [18].

The main aim of this paper is to extend the line of research started in [23] by determining the parameterized complexity of untangling knots via Reidemeister moves. On the one hand, it is easy to see that if we consider parameterization by the number of moves, then the problem is in FPT (class of *fixed parameter tractable* problems). This happens due to a somewhat trivial reason: if a diagram D can be untangled with at most k moves, then D contains at most $2k$ crossings, thus we can assume that the size of D is (polynomially) bounded by k. In notions of parameterized complexity, this gives a kernel of size bounded by k which immediately gives the FPT membership. In the full version, we provide a bit more details; see Observation 1 in the full version.

On the other hand, we also consider parameterization with an arguably much more natural parameter called the defect (used in [23]). This parameterization is also relevant from the point of view of *above guarantee parameterization* introduced by Mahajan and Raman [21]. Here we show that the problem is W[P]-complete with respect to the defect. This is the core of the paper.

In order to state our main result more precisely, we need a few preliminaries on diagrams and Reidemeister moves. For purposes of this part of the introduction, we also assume that the reader is at least briefly familiar with complexity classes FPT and W[P]. Otherwise we refer to the end of the introduction, where we briefly overview these classes.

Diagrams and Reidemeister moves. A *diagram* of a knot is a piecewise linear map $D\colon S^1 \to \mathbb{R}^2$ in general position; for such a map, every point in $\mathbb{R}^2$ has at most two preimages, and there are finitely many points in $\mathbb{R}^2$ with exactly two preimages (called *crossings*). Locally at a crossing two arcs cross each other transversely, and the diagram contains the information of which arc passes "over" and which "under". This we usually depict by interrupting the arc that passes under. Diagrams are always considered up-to isotopy. The unique diagram without crossings is denoted U (untangled). See Figure 1 for an example of a diagram.

Let D be a diagram of a knot. *Reidemeister moves* are local modifications of the diagram depicted in Figure 2. We distinguish Reidemeister moves of types I, II, and III as depicted in the figure. In addition, for types I and II, we distinguish whether the moves remove crossings (types I^- and II^-) or whether they introduce new crossings (types I^+ and II^+).

A diagram D is a *diagram of an unknot* if it can be transformed to the untangled diagram U by a finite sequence of Reidemeister moves. (This is well known to be equivalent to stating that the lift of the diagram to $\mathbb{R}^3$, keeping the underpasses/overpasses, is ambient isotopic to

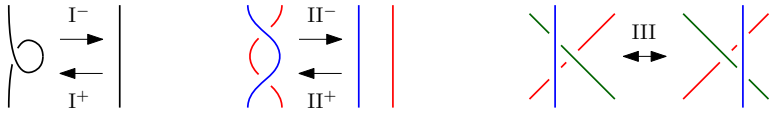


Figure 2 Reidemeister moves.

the unknot, that is standardly embedded S^1 in $\mathbb{R}^3$.) The diagram on Figure 1 is a diagram of an unknot. Diagrams may be encoded purely combinatorially as 4-regular plane graphs (with some additional combinatorial information) and the size of this encoding is comparable to the number of crossings. Similarly, the Reidemeister moves can be encoded by purely combinatorial data. For more details we refer to Sections 2 and 3 of the full version.

Parameterization via defect. As we discussed earlier, parameterization in the number of Reidemeister moves has the obvious disadvantage that once we fix k, the problem becomes trivial for arbitrarily large inputs (they are obviously a NO instance). We also see that if we have a diagram D with n crossings and want to minimize the number of Reidemeister moves to untangle D, presumably the most efficient way is to remove two crossings in each step, thus requiring at least $n/2$ steps. This motivates the following definition of the notion of defect.

Given a diagram D, by an *untangling* of D we mean a sequence $\mathcal{D} = (D_0, \dots, D_\ell)$ such that $D = D_0$; D_{i+1} is obtained from D_i by a Reidemeister move; and $D_\ell = U$ is the diagram with no crossings. Then we define the *defect* of an untangling $\mathcal{D}$ as above as

$$\operatorname{def}(\mathcal{D}) := 2\ell - n$$

where n is the number of crossings in D. Note that ℓ is just the number of Reidemeister moves in the untangling. It is easy to see that $\operatorname{def}(\mathcal{D}) \geq 0$ and $\operatorname{def}(\mathcal{D}) = 0$ if and only if all moves in the untangling are II^- moves. Therefore, $\operatorname{def}(\mathcal{D})$ in some sense measures the number of "extra" moves in the untangling beyond the trivial bound. (Perhaps, a more accurate expression for this interpretation would be $\ell - n/2 = \frac{1}{2}\operatorname{def}(\mathcal{D})$ but this is a minor detail and it is more convenient to work with integers.) In addition, it is possible to get diagrams with arbitrarily large number of crossings but with defect bounded by a constant (even for defect 0 this is possible). The defect also plays a key role in the reduction in [23] which suggests that the hardness of the untangling really depends on the defect.

As we have seen above, asking the question whether a diagram can be untangled with defect at most k is same as asking if it can be untangled in $k/2$ moves above the trivial, but tight lower bound of $n/2$. This fits perfectly in the framework of above guarantee parameterization, which was introduced by Mahajan and Raman [21] for Max-Sat and Max-Cut problems. In this framework, when there is a trivial lower bound for the solution in terms of the size of the input, parameterizing by solution size trivially gives an FPT algorithm by either giving a trivial answer if the input is large, or bounding size of the input by a function of the solution size. Hence, for those problems, it makes more sense to parameterize above a tight lower bound. The paradigm of above guarantee parameterization has been very successful in the field of parameterized complexity and many results have been obtained [1, 4, 5, 9, 10, 11, 22].

For these reasons, we find the defect to be a more natural parameter than the number of Reidemeister moves. Therefore, we consider the following problem.

▶ **Problem** (UNKNOTTING VIA DEFECT).

INPUT	A diagram D of a knot.
PARAMETER	k.
QUESTION	Can D be untangled with defect at most k?

▶ **Theorem 1.** *The problem* UNKNOTTING VIA DEFECT *is W[P]-complete.*

The proof of Theorem 1 consists of two main steps: W[P]-membership and W[P]-hardness. Both of them are non-trivial.

For W[P]-membership, roughly speaking, the idea is to guess a small enough set of *special* crossings on which we perform all possible Reidemeister moves, while we remove other crossings in a greedy fashion. In order to succeed with such an approach we will need some powerful and flexible enough lemmas on changing the ordering of Reidemeister moves in some untangling by swapping them. In Section 2 we provide an algorithm for W[P]-membership but we only sketch why it works correctly. For more details, we refer to Sections 4 and 5 of the full version.

For W[P]-hardness, we combine some techniques that were quite recently used in showing parameterized hardness of problems in computational topology [2, 3], along with the tools in [23] for lower bounding the defect. Namely, we use a reduction from the MINIMUM AXIOM SET problem, which proved to be useful in [2, 3]. Roughly speaking, from an instance I of the MINIMUM AXIOM SET problem, we need to build a diagram which has a small defect if and only if I admits a small set of axioms. For the "if" part, we use properties of Brunnian rings to achieve our goal. For the "only if" part, we need to lower bound the defect of our construction. We use the tools from [23] to show that the defect (of some subinstances) is at least 1. Then we use the very simple but powerful boosting lemma (Lemma 9) that shows that the defect is actually high. We describe the reduction in Section 4 and we sketch in a bit more detail why it may work. We refer to the Section 6 of the full version for a proof of correctness.

We conclude this part of introduction by proving a lemma on the properties of the defect which we will use soon after. Given a Reidemeister move m, let us define the *weight* of this move $w(m)$ via the following table:

Type of the move	II^-	I^-	III	I^+	II^+
$w(m)$	0	1	2	3	4

▶ **Lemma 2** (Lemma 3 in full version). *Let $\mathcal{D}$ be an untangling of a diagram D. Then* $\mathrm{def}(\mathcal{D})$ *equals to the sum of the weights of the Reidemeister moves in $\mathcal{D}$.*

The lemma is proved in the full version by a simple induction.

A brief overview of the parameterized complexity classes. Here we briefly overview the notions from parameterized complexity needed for this paper. For further background, we refer the reader to monographs [6, 7, 8]. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is the set of strings over a finite alphabet Σ and the input strings are of the form (x, k). Here the integer k is called *the parameter*. In the rest of the paper, given an input for a parameterized problem, n denotes the size of the input and k denotes the value of the parameter on this input.

A parameterized problem belongs to the class FPT (fixed parameter tractable) if it can be solved by a deterministic Turing machine in time $f(k) \cdot n^c$ where $c > 0$ is some constant and $f(k)$ is some computable function of k. In other words, if we fix k, then the problem

can be solved in polynomial time while the degree of the polynomial does not depend on k. This is, of course, sometimes not achievable and there is a wider class XP of problems, that can be solved in time $O(n^{f(k)})$ by a deterministic Turing machine. The problems in XP are still polynomial time solvable for fixed k, however, at the cost that the degree of the polynomial depends on k.

Somewhere in between FPT and XP there is an interesting class W[P]. A parameterized problem belongs to the class W[P] if it can be solved by a nondeterministic Turing machine in time $h(k) \cdot n^c$ provided that this machine makes only $O(f(k) \log n)$ non-deterministic choices where $f(k), h(k)$ are computable functions and $c > 0$ is some constant. Given an algorithm for some computational problem Π, we say that this algorithm is a *W[P]-algorithm* if it is represented by a Turing machine satisfying the conditions above.

Given two parameterized problems Π and Π', we say that Π reduces to Π' via an *FPT-reduction* if there exist computable functions $f\colon \Sigma^* \times \mathbb{N} \to \Sigma^*$ and $g\colon \Sigma^* \times \mathbb{N} \to \mathbb{N}$ such that (i) $(x,k) \in \Pi$ if and only if $(f(x,k), g(x,k)) \in \Pi'$ for every $(x,k) \in \Sigma^* \times \mathbb{N}$; (ii) $g(x,k) \leq g'(k)$ for some computable function g'; and (iii) there exists computable function h and a fixed constant $c > 0$ such that for all input string, $f(x,k)$ can be computed by a deterministic Turing-machine in $O(h(k)n^c)$ steps. Our definition of reduction is consistent with [8, Definition 2.1] or [6, Definition 13.1], though some authors, e.g., [7, Definition 20.2.1] require $g(x,k) = g'(k)$ in (ii).

The classes FPT, W[P] and XP are closed under FPT-reductions. A problem Π is said to be *C-hard* where C is a parameterized complexity class, if all problems in C can be FPT-reduced to Π. Moreover, if $\Pi \in$ C, we say that Π is *C-complete*.

2 An algorithm for W[P]-membership

In this section we provide the algorithm used to prove W[P]-membership in Theorem 1.

Brute force algorithm. First, let us however look at a brute force algorithm for Unknotting via defect which does not give W[P]-membership. Spelling it out will be useful for explaining the next steps. We will exhibit this algorithm as a non-deterministic algorithm, which will be useful for comparison later on. In the algorithm, D is a diagram, and k is an integer, not necessarily positive. Also, given a diagram D and a feasible Reidemeister move m, then $D(m)$ denotes the diagram obtained from D after performing m.

BruteForce(D,k):

1. If $k < 0$, then output No. If $D = U$ is a diagram without crossings and $k \geq 0$, then output Yes. In all other cases continue to the next step.
2. (Non-deterministic step.) Enumerate all possible Reidemeister moves $m_1, \ldots, m_t$ in D up to isotopy. Make a "guess" which move m_i is the first to perform. Then, for such m_i, run BruteForce$(D(m_i), k - w(m_i))$.

Therefore, altogether, the algorithm outputs Yes, if there is a sequence of guesses in step 2 which eventually yields Yes in step 1.

It can be easily shown that the algorithm terminates because whenever step 2 is performed, either $k - w(m_i) < k$, or m_i is a Π^- move, $k - w(m_i) = k$, but $D(m_i)$ has fewer crossings than D.

It can be also easily shown by induction that this algorithm provides a correct answer due to Lemma 2. Indeed, step 1 clearly provides a correct answer, and regarding step 2, if $D(m_i)$ can be untangled with defect at most $k - w(m_i)$, then Lemma 2 shows that D

can be untangled with defect at most k. Because, this way we try all possible sequences of Reidemeister moves, the algorithm outputs `Yes` if and only if D can be untangled with defect at most k.

On the other hand, this algorithm (unsurprisingly) does not provide W[P]-membership as there are at least $n/2$ Reidemeister moves, which is unbounded in k. Thus not all moves can be guessed non-deterministically.

Naive greedy algorithm. In order to fix the problem with the previous algorithm, we want to reduce the number of non-deteministic steps. It turns out that the problematic non-deterministic steps in the previous algorithm are those where k does not decrease. (Because the other steps appear at most k times.) Therefore, we want to avoid non-deterministic steps where we perform a II^- move. The naive way is to perform such steps greedily and hope that if D untangles with defect at most k, there is also such a "greedy" untangling (and therefore, we do not have to search through all possible sequences of Reidemeister moves). This is close to be true but it does not really work in this naive way. Anyway, we spell this naive algorithm explicitly, so that we can easily upgrade it in the next step, though it does not always provide the correct answer to `Unknotting via defect`.

`NaiveGreedy`(D, k):

1. If $k < 0$, then output `No`. If $D = U$ is a diagram without crossings and $k \geq 0$, then output `Yes`. In all other cases continue to the next step.
2. If there is a feasible Reidemeister II^- move m, run `NaiveGreedy`$(D(m), k)$ otherwise continue to the next step.
3. (Non-deterministic step.) If there is no feasible Reidemeister II^- move, enumerate all possible Reidemeister moves $m_1, \ldots, m_t$ in D up to isotopy. Make a "guess" which m_i is the first move to perform and run `NaiveGreedy`$(D(m_i), k - w(m_i))$.

The algorithm must terminate for the same reason why `BruteForce` terminates. It can be shown that this is a W[P]-algorithm. However, we do not do this here in detail as this is not our final algorithm. The key is that the step 3 is performed at most $(k+1)$-times.

If the algorithm outputs `Yes`, then this is a correct answer from the same reason as in the case of `BruteForce`. However, as we hinted earlier, outputting `No` need not be a correct answer to `Unknotting via defect`. Indeed, there are known examples of diagrams of an unknot when untangling requires performing a II^+ move, see for example [14]. With `NaiveGreedy`, we would presumably undo such II^+ move immediately in the next step, thus we would not find any untangling using the II^+ move. We have to upgrade the algorithm a little bit to avoid this problem (and a few other similar problems).

Special greedy algorithm. The way to fix the problem above will be to guess in advance a certain subset S of so-called special crossings. This set will be updated in each non-deterministic step described above so that the newly introduced crossings will become special as well. Then, in the greedy steps we will allow to perform only those II^- moves which avoid S. (This also means that a II^+ move cannot be undone by a II^- move avoiding S in the next step.) It will turn out that we may also need to perform II^- moves on S but there will not be too many of them, thus such moves can be considered in the non-deterministic steps.

For the description of the algorithm, we introduce the following notation. Let D be a diagram of a knot and S be a subset of the crossings of D; we will refer to crossings in S as *special* crossings. Then we say that a feasible Reidemeister II^- move m is *greedy* (with respect to S), if it avoids S; that is, the crossings removed by m do not belong to S. On the other hand, a feasible Reidemeister move m is *special* (with respect to S) if it is

- a I^- or a II^- move removing only crossings in S; or
- a III move such that all three crossings affected by m are special; or
- a I^+ or a II^+ move performed on the edges with all their endpoints in S.

Given a move m in D, special or greedy with respect to S, by $S(m)$ we denote the following set of crossings in $D(m)$.

- If m is a greedy II^- move or if m is a III move, then $S(m) = S$ (under the convention that the three crossings affected by m persist in $D(m)$).
- If m is a special II^- move or a I^- move, then $S(m)$ is obtained from S by removing the crossings removed by m.
- If m is a I^+ or a II^+ move, then $S(m)$ is obtained from S by adding the crossings introduced by m.

Now, we can describe the algorithm.

SpecialGreedy(D, k):

0. (Non-deterministic step.) Guess a set S of at most $3k$ crossings in D. Then run SpecialGreedy(D, S, k).

SpecialGreedy(D, S, k):

1. If $k < 0$, then output No. If $D = U$ is a diagram without crossings and $k \geq 0$, then output Yes. In all other cases continue to the next step.

2. If there is a feasible greedy Reidemeister II^- move m with respect to S, run SpecialGreedy$(D(m), S(m), k)$ otherwise continue to the next step.

3. (Non-deterministic step.) If there is no feasible greedy Reidemeister II^- move with respect to S, enumerate all possible special Reidemeister moves $m_1, \ldots, m_t$ in D with respect to S up to isotopy. If there is no such move, that is, if $t = 0$, then output No. Otherwise, make a guess which m_i is performed first and run SpecialGreedy$(D(m_i), S(m_i), k - w(m_i))$.

The bulk of the proof of W[P]-membership in Theorem 1 will be to show that the algorithm SpecialGreedy(D, k) provides a correct answer to Unknotting via defect. Of course, if the algorithm outpus Yes, then this is the correct answer by similar arguments for the previous two algorithms. Indeed, Yes answer corresponds to a sequence of Reidemeister moves performed in step 2 or guessed in step 3 (no matter how we guessed S in step 0 and the role of S in the intermediate steps of the run of the algorithm is not important if we arrived at Yes). The defect of the untangling given by this sequence of moves is at most k by Lemma 2. On the other hand, we also need to show that if D untanlges with defect at most k, then we can guess some such untangling while running the algorithm. This is based on the following two auxiliary results.

For stating the results, we inductively define the notation $D(m_1, \ldots, m_{k-1}, m_k) := D(m_1, \ldots, m_{k-1})(m_k)$ provided that m_k is a feasible Reidemeister move in $D(m_1, \ldots, m_{k-1})$. We also say that a sequence $(m_1, \ldots, m_k)$ is a *feasible* sequence of Reidemeister moves for D if m_i is a feasible Redemeister move in $D(m_1, \ldots, m_{i-1})$ for every $i \in [k]$. Similarly, we inductively define $S(m_1, \ldots, m_{k-1}, m_k) := S(m_1, \ldots, m_{k-1})(m_k)$, provided that the move m_k is special or greedy in $D(m_1, \ldots, m_{k-1})$ with respect to $S(m_1, \ldots, m_{k-1})$. Then $(m_1, \ldots, m_k)$ is a *feasible* sequence of Reidemeister moves for the pair (D, S) if $S(m_1, \ldots, m_k)$ is well defined this way; that is, each move m_i in the sequence is special or greedy (with respect to the intermediate set $S(m_1, \ldots, m_{i-1})$ of special crossings).

▶ **Lemma 3** (Lemma 8 in full version). *Let D be a diagram of a knot and let $(m_1, \ldots, m_\ell)$ be a feasible sequence of Reidemeister moves in D. Then there is a set of crossings S in D such that $(m_1, \ldots, m_\ell)$ is feasible for (D, S).*

In addition, if this sequence induces an untangling with defect at most k, then $|S| \leq 3k$.

▶ **Theorem 4** (Theorem 9 in full version). *Let D be a diagram of a knot and S be a set of crossings in D. Let $(m_1, \dots, m_\ell)$ be a feasible sequence of Reidemeister moves for (D, S), inducing an untangling of D with defect k. Assume that there is a greedy move $\widetilde{m}$ in D with respect to S. Then there is a feasible sequence of Reidemeister moves for (D, S) starting with $\widetilde{m}$ and inducing an untangling of D with defect k.*

Roughly speaking, the idea of the proof of Lemma 3 is to add into S all crossings that will eventually become affected by some of the moves $m_1, \dots, m_\ell$ which are not II^- moves. These moves must be special. However, if we have a II^- move removing a crossing that we have already added to S, then this move also has to be special and we add the other crossing to S as well. This does not propagate further, and we can bound the number of special II^- moves which also provides a bound on S. Details are given in the full version. Proof of Theorem 4 requires more work and we sketch it in Section 3.

Correctness of the algorithm. Here we explain that the algorithm $\textsc{SpecialGreedy}(D, k)$ provides the answer YES whenever there is an untangling with defect at most k modulo Lemma 3 and Theorem 4. We also need to know that this is a W[P]-algorithm (mainly) by bounding the number of non-deterministic steps. This part is relatively straightforward and we refer to Section 4 of full version for details.

Given an input (D, k) such that D admits an untangling with defect k, by Lemma 3, there is a set of crossings S in D of size at most $3k$ such that there is a sequence $\mathcal{M}$ of Reidemeister moves feasible for (D, S) inducing an untangling of D with defect at most k. Thus it is sufficient to show that $\textsc{SpecialGreedy}(D, S, k)$ outputs YES if such a sequence for a given S exists. (In fact this is if and only if but we only need the if case.) We will show this by a double induction on k and $\mathrm{cr}(D)$, where $\mathrm{cr}(D)$ is the number of crossings in D. The outer induction is on k, the inner one is on $\mathrm{cr}(D)$. It would be sufficient to start our induction with the pair $(k, \mathrm{cr}(D)) = (0, 0)$; however, whenever $\mathrm{cr}(D) = 0$, then the algorithm outputs YES in step 1. Thus we may assume that $\mathrm{cr}(D) > 0$.

If (D, S) admits any greedy move, then we are in step 2. For every greedy move m there is a sequence of Reidemeister moves starting with m feasible for (D, S) inducing an untangling of D with defect at most k by Theorem 4. This move has weight 0. Thus by Lemma 2 there is a sequence of Reidemeister moves feasible for $(D(m), S(m))$ inducing an untangling of $D(m)$ with defect at most k. In addition $\mathrm{cr}(D(m)) < \mathrm{cr}(D)$, thus $\textsc{SpecialGreedy}(D(m), S(m), k)$ outputs YES by induction.

If (D, S) does not admit any greedy move, then we are in step 3. We in particular know that the first move in the sequence $\mathcal{M}$ must be special. We guess this move as we are in a non-deterministic step and denote it m_i (in consistence with the notation in step 3). Now, if we remove m_i from $\mathcal{M}$, this is a feasible sequence for $(D(m_i), S(m_i))$ inducing an untangling of $D(m_i)$ with defect at most $k - w(m_i)$ by Lemma 2. Thus $\textsc{SpecialGreedy}(D(m_i), S(m_i), k - w(m_i))$ outputs YES by induction, as we need.

3 Sketch for a proof of Theorem 4

Here we sketch a proof of Theorem 4. In general, the Reidemeister moves are not commutative: if (m, m') is a feasible sequence of Reidemeister moves in a diagram D, we cannot even expect that m' is feasible in D. A fortiori we cannot expect that (m', m) is feasible in D and that the resulting diagram would be the same when applying (m, m'). For the proof of Theorem 4, we need commutativity of the greedy moves under certain circumstances. In particular, we need to be able to swap the greedy moves with the neighbors so that we

can shift them both forward and backward through the sequence. Hint that this might be sometimes possible is the following: if m and m' can be performed inside disjoint balls, then they commute. In this section, we only state the lemmas on rearranging Reidemeister moves. The proofs of these lemmas (and their corollaries) are in the full version. Then we deduce Theorem 4 from the lemmas up to minor details explained in the full version.

First, we extend our earlier notation: Let D be a diagram of a knot, S be a set of crossings and $(m_1, \dots, m_k)$ be a sequence of Reidemeister moves feasible for a pair (D, S). Then by $(D, S)(m_1, \dots, m_k)$ we denote the pair (D', S') obtained by applying moves $(m_1, \dots, m_k)$ to (D, S). In our earlier notation, this means that $D' = D(m_1, \dots, m_k)$ and $S' = S(m_1, \dots, m_k)$. We also denote a II^- move removing crossings x and y by $\{x, y\}_{\mathrm{II}^-}$.

The first lemma allows to perform a greedy move m earlier in a sequence (m', m).

▶ **Lemma 5** (Lemma 10 in full version). *Let D be a diagram of a knot and S be a set of crossings in D. Let $m = \{x, y\}_{\mathrm{II}^-}$ be a II^- move in D, greedy with respect to S. Let m' be a feasible move for the pair (D, S) which avoids $\{x, y\}$. Then there is a move $\widehat{m}'$ feasible in $D(m)$ of same type as m' and the following conditions hold:*

- **(i)** *m is greedy in $D(m')$ with respect to $S(m')$, in particular m is feasible in $(D, S)(m')$*
- **(ii)** *$\widehat{m}'$ is feasible in $(D, S)(m)$; and*
- **(iii)** *$(D, S)(m, \widehat{m}') = (D, S)(m', m)$.*

Lemma 5 is proved by a careful local analysis depending on the type of m'. By a suitable induction, Lemma 5 implies the following corollary for longer sequences of moves.

▶ **Corollary 6** (Corollary 11 in full version). *Let D be a diagram, S be a set of crossings in D, $\ell \geq 2$, and $(m_1, \dots, m_\ell)$ be a feasible sequence of Reidemeister moves for the pair (D, S). Assume that m_ℓ is a II^- move which is also feasible and greedy in D with respect to S. Then there is a sequence of Reidemester moves $(m_\ell, \widehat{m}_1, \dots, \widehat{m}_{\ell-1})$ feasible for (D, S) such that*

- **(i)** *$(D, S)(m_1, \dots, m_\ell) = (D, S)(m_\ell, \widehat{m}_1, \dots, \widehat{m}_{\ell-1})$; and*
- **(ii)** *$w(m_i) = w(\widehat{m}_i)$ for $i \in [\ell - 1]$.*

We will also need a variant of Corollary 6 which allows to postpone a greedy move. The following corollary allows us to do that. (this follows from a lemma analogous to Lemma 5 which we skip here).

▶ **Corollary 7** (Corollary 13 in full version). *Let D be a diagram, S be a set of crossings in D, $\ell \geq 2$ and $(m_1, \dots, m_\ell)$ be a feasible sequence of moves for (D, S). Assume that $m_1 = \{x_1, y_1\}_{\mathrm{II}^-}$ and $m_\ell = \{x_\ell, y_\ell\}_{\mathrm{II}^-}$ are II^- moves where $x_1, y_1, x_\ell, y_\ell \notin S$ (in particular m_1 is greedy in D with respect to S). Finally, assume also that $\{x_1, x_\ell\}_{\mathrm{II}^-}$ is a feasible Reidemeister move for (D, S) (again, it must be greedy). Then, there is a feasible sequence $(\widehat{m}_2, \widehat{m}_3, \dots, \widehat{m}_{\ell-1}, m_1)$ of moves for (D, S) such that*

- **(i)** *$(D, S)(m_1, \dots, m_{\ell-1}) = (D, S)(\widehat{m}_2, \widehat{m}_3, \dots, \widehat{m}_{\ell-1}, m_1)$; and*
- **(ii)** *$w(m_i) = w(\widehat{m}_i)$ for $i \in \{2, \dots, \ell - 1\}$.*

Last, but not the least, both the corollaries above essentially only allow swapping the moves. In the corollaries above, m_i is essentially the same move as $\hat{m}_i$ up to a combinatorial description (not discussed in this extended abstract) which is a reason why we use a different notation. Such corollaries cannot be sufficient for a proof of Theorem 4 in the case that $\tilde{m}$ (from the statement of the theorem) is not in the sequence $(m_1, \dots, m_\ell)$. We also need to be able to replace some greedy moves with different ones which is the content of the following lemma.

▶ **Lemma 8** (Lemma 14 in full version). *Let D be a diagram of a knot and let $m = \{x, y\}_{\mathrm{II}^-}$, $\widetilde{m} = \{x, z\}_{\mathrm{II}^-}$ be two feasible Reidemeister moves in D where $y \neq z$. Assume also that $m' = \{w, z\}_{\mathrm{II}^-}$ is feasible in $D(m)$ with x, y, z, w mutually distinct. Then*

(i) *$\widetilde{m}' = \{w, y\}_{\mathrm{II}^-}$ is feasible in $D(\widetilde{m})$; and*

(ii) *$D(m, m') = D(\widetilde{m}, \widetilde{m}')$.*

Lemma 8 is again proved by local analysis in the full version.

Proof of Theorem 4 (sketch). First, let us assume that $\widetilde{m} = m_j$ for some $j \in [\ell]$. If $j = 1$, there is nothing to prove, thus we may assume $j \geq 2$. Then, by using Corollary 6 on the sequence $(m_1, \ldots, m_j)$, we get a sequence of moves $(m_j, \widehat{m}_1, \ldots, \widehat{m}_{j-1})$ feasible for (D, S). By item (i) of Corollary 6, the sequence $(m_j, \widehat{m}_1, \ldots, \widehat{m}_{j-1}, m_{j+1}, \ldots, m_\ell)$ is also feasible for (D, S) and induces an untangling of D. By item (ii) of Corollary 6 and Lemma 2, the defect of this sequence equals k, and that is what we need.

Thus it remains to consider the case where $\widetilde{m} \neq m_j$ for every $j \in [\ell]$. Because $\widetilde{m}$ is greedy, it is a II^- move; say $\widetilde{m} = \{x, z\}_{\mathrm{II}^-}$. Because the final diagram $D(m_1, \ldots, m_\ell)$ has no crossings, the crossings x and z have to be removed by some moves in the sequence $(m_1, \ldots, m_\ell)$. Say that a move m_i removes x and m_j removes z. Since $\widetilde{m}$ is greedy with respect to S, we get $x, z \notin S$. This also implies that $x \notin S(m_1, \ldots, m_{i-1})$ (see the full version for details). Thus m_i has to be greedy move in $D(m_1, \ldots, m_{i-1})$ with respect to $S(m_1, \ldots, m_{i-1})$. Similarly, m_j is greedy in $D(m_1, \ldots, m_{j-1})$ with respect to $S(m_1, \ldots, m_{j-1})$. Because we assume that $\widetilde{m}$ is not in the sequence $(m_1, \ldots, m_\ell)$, we get $i \neq j$; without loss of generality $i < j$. Let y and w be such that $m_i = \{x, y\}_{\mathrm{II}^-}$ and $m_j = \{w, z\}_{\mathrm{II}^-}$. As these moves are greedy, we get $y, w \notin S$ (we again refer to the full version for more detail).

Let $D' := D(m_1, \ldots, m_{i-1})$ and $S' := S(m_1, \ldots, m_{i-1})$. By Corollary 7, applied to D', S' and the sequence $(m_i, \ldots, m_j)$ feasible for (D', S'), we get another sequence $(\widehat{m}_{i+1}, \ldots, \widehat{m}_{j-1}, m_i)$ feasible for (D', S'). (For verifying the assumptions of the corollary note that $\widetilde{m}$ is feasible for (D', S') as all the moves $m_1, \ldots, m_{i-1}$ are special or greedy, thus they cannot remove x or z nor affect the arcs connecting them.) Then we get that $(\widehat{m}_{i+1}, \ldots, \widehat{m}_{j-1}, m_i, m_j, \ldots, m_\ell)$ is also feasible for (D', S') by item (i) of Corollary 7.

Next, let $D'' := D'(\widehat{m}_{i+1}, \ldots, \widehat{m}_{j-1})$ and $S'' := S'(\widehat{m}_{i+1}, \ldots, \widehat{m}_{j-1})$. By a similar argument as above, we get that $\widetilde{m}$ is feasible in (D'', S''). (Note that the moves m_i, m_j removing x and z have not been performed yet in order to get D''.) By Lemma 8 used in D'', we get that $\widetilde{m}' := \{w, y\}_{\mathrm{II}^-}$ is feasible in $(D'', S'')(\widetilde{m})$ and $(D'', S'')(m_i, m_j) = (D'', S'')(\widetilde{m}, \widetilde{m}')$. Altogether, by expanding D'' and D'

$$(m_1, \ldots, m_{i-1}, \widehat{m}_{i+1}, \ldots, \widehat{m}_{j-1}, \widetilde{m}, \widetilde{m}', m_{j+1}, \ldots, m_\ell) \tag{1}$$

is an untangling of D, feasible for (D, S). We also get that the defect of this untangling is equal to k by Lemma 2 and item (ii) of Corollary 7, when we used it. Note that all the moves m_i, m_j, $\widetilde{m}$ and $\widetilde{m}'$ are II^- moves, thus they do not contribute to the weight.

The sequence (1) is not the desired sequence yet, because it does not start with $\widetilde{m}$. However, it contains $\widetilde{m}$, thus we can further modify this sequence to the desired sequence starting with $\widetilde{m}$ as in the first paragraph of this proof. ◀

4 W[P]-hardness

In this section we sketch a proof of W[P]-hardness part of Theorem 1. This is done by an FPT-reduction from Minimum axiom set defined below.

4.1 Minimum axiom set

It is well known that the MINIMUM AXIOM SET problem is W[P]-hard; see [8, Exercise 3.20] or [7, Lemma 25.1.3] (however, let us recall that our definition of FPT-reduction is consistent with [8]).

▶ **Problem** (MINIMUM AXIOM SET).

INPUT	A finite set S, and a finite set $\mathcal{R}$ which consists of pairs of form (T, t) where $T \subseteq S$ and $t \in S$.
PARAMETER	k.
QUESTION	Does there exist a subset $S_0 \subseteq S$ of size k such that if we define inductively S_i to be the union of S_{i-1} and all $t \in S$ such that there is $T \subseteq S_{i-1}$ with $(T, t) \in \mathcal{R}$, then $\bigcup_{i=1}^{\infty} S_i = S$?

The problem above deserves a brief explanation. The elements of S are called *sentences* and the elements of $\mathcal{R}$ are relations. A relation (T, t) with $T = \{t_1, \ldots, t_m\}$ should be understood as an implication

$$t_1 \wedge t_2 \wedge \cdots \wedge t_m \Rightarrow t.$$

Given a set $S_0 \subseteq S$, let us define the *consequences* of S_0 as $c(S_0) := \bigcup_{i=1}^{\infty} S_i$ where S_i is defined as in the statement of the problem. Intuitively, $c(S_0)$ consists of all sentences that can be deduced from S_0 via the *relations* (implications) in $\mathcal{R}$. As we work with finite sets, $c(S_0) = S_i$ for some high enough i. A set A is a set of *axioms* if $c(A) = S$. Therefore, the goal of the minimum axiom set problem is to determine whether there is a set of axioms of size k. Note that the axiom sets are upward-closed: If A is an axiom set and $A \subseteq A' \subseteq S$, then A' is an axiom set as well.

The following boosting lemma is very useful in our reduction.

▶ **Lemma 9** (Boosting lemma; Lemma 15 in full version). *Let $(S, \mathcal{R})$ be an input of the minimum axiom set problem (ignoring the parameter for now). Let $\mu\colon S \to \mathbb{Z}$ be a non-negative function. Given $U \subseteq S$, let $\mu(U) = \sum_{s \in U} \mu(s)$. Assume that $\mu(U) \geq 1$ for all U such that $S \setminus U$ is not an axiom set. (Equivalently, U meets every axiom set.) Then $\mu(S) \geq k^*$ where k^* is the size of a minimum axiom set.*

Proof. Let $Z := \{s \in S\colon \mu(s) = 0\}$ be the zero set of μ. Then $\mu(Z) = 0$, thus $S \setminus Z$ is a set of axioms by the assumptions. This gives $|S \setminus Z| \geq k^*$ and, in addition, $\mu(S) = \mu(S \setminus Z) \geq |S \setminus Z| \geq k^*$ because $\mu(s) \geq 1$ for every $s \in S \setminus Z$. ◀

4.2 Construction of the reduction

Our aim is to show that there is an FPT-reduction from the MINIMUM AXIOM SET to UNKNOTTING VIA DEFECT. It is not hard to see (see the full version) that we can assume that the input $(S, \mathcal{R}, k)$ is *preprocessed*; i.e., (i) $\mathcal{R}$ does not contain relations of the form $(\emptyset, t)$; (ii) for every $s \in S$ there is a relation of a form (T, s) in $\mathcal{R}$; and (iii) $t \notin T$ for every $(T, t) \in \mathcal{R}$.

Doubling the instance. Now let $(S, \mathcal{R}, k)$ be a preprocessed instance of the MINIMUM AXIOM SET. We will need the following doubled instance: Let $\widehat{S} := \{\widehat{s}\colon s \in S\}$ be an auxiliary copy of S. Given $T = \{t_1, \ldots, t_m\} \subseteq S$, let $\widehat{T} := \{\widehat{t_1}, \ldots, \widehat{t_m}\}$. Then we define $\widehat{\mathcal{R}} := \{(\widehat{T}, \widehat{t}) : (T, t) \in \mathcal{R}\}$. Then $(S \cup \widehat{S}, \mathcal{R} \cup \widehat{\mathcal{R}}, 2k)$ is a *double* of the instance $(S, \mathcal{R}, k)$. The proof of the following observation is straightforward and it is given in the full version; see Observation 16 there.

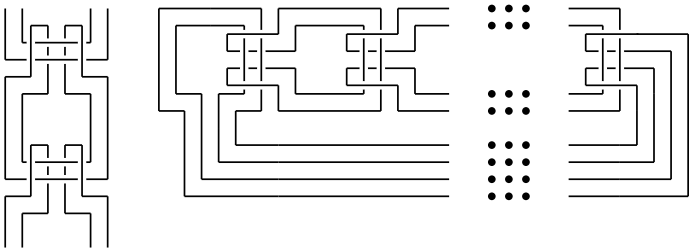


Figure 3 A Brunnian link.

▶ **Observation 10.** *The pair $(S, \mathcal{R})$ admits an axiom set of size k if and only if its double $(S \cup \widehat{S}, \mathcal{R} \cup \widehat{\mathcal{R}})$ admits an axiom set of size $2k$.*

Brunnians. A *Brunnian link* is a nontrivial link that becomes trivial whenever one of the link components is removed. We will use the following well known construction of a Brunnian link with $\ell \geq 2$ components. We take an untangled unknot and we interlace it with two "neighboring" unknots as in Figure 3, left. We repeat this ℓ-times and we get a Brunnian link with ℓ components as in Figure 3, right.

Gadgets. From now on let $(S, \mathcal{R}, k)$ be a preprocessed instance of the MINIMUM AXIOM SET and $(S \cup \widehat{S}, \mathcal{R} \cup \widehat{\mathcal{R}}, 2k)$ be its double. Our aim is to build a diagram $D(S, \mathcal{R})$ such that $D(S, \mathcal{R})$ untangles with defect $2k$ if and only if $(S, \mathcal{R})$ has an axiom set of size k. We will build $D(S, \mathcal{R})$ using several gadgets. Formally speaking, gadgets will be maps of a form $G\colon I_1 \sqcup \cdots \sqcup I_h \to \mathbb{R}^2$ where $I_1 \sqcup \cdots \sqcup I_h$ stands for a disjoint union of intervals $[0, 1]$. We will work with them in the same way as with diagrams. In particular, we assume the same transversality assumptions on crossings as for diagrams and we mark the underpasses and overpasses. We also extend the notion of arc to this setting: It is a set $G(A)$, where A is a closed subinterval of one of the intervals in $I_1 \sqcup \cdots \sqcup I_h$.

Sentence gadget. For each sentence $s \in S$ we define a sentence gadget $G(s)$ as follows. (We will also create an analogous gadget $G(\widehat{s})$ for $\widehat{s} \in \widehat{S}$ which we specify after describing $G(s)$.) We consider all relations $R \in \mathcal{R}$ of the form (T, s). Let $\ell = \ell(s)$ be the number of such relations and we order these relations as $R_1(s), \ldots, R_\ell(s)$, or simply as $R_1, \ldots, R_\ell$ if s is clear from the context (which is the case now). Note that $\ell \geq 1$ due to preprocessing.

Now we take our Brunnian link with $\ell+1$ components, which we denote by $C_0(s), \ldots, C_\ell(s)$, in the order along the Brunnians. We disconnect each component of this Brunnian link in an arc touching the outer face and we double each such disconnected component. See Figure 4, left. For the further description of the construction, we assume that our construction is rotated exactly as in the figure. (If $\ell \neq 2$, then we just modify the length of $C_0(s)$ and insert more or fewer components $C_i(s)$ in the same way as $C_1(s)$ and $C_2(s)$ are inserted for $\ell = 2$.) Now we essentially have a collection of interlacing arcs where each former component of Brunnians yields a pair of parallel arcs with four loose ends.

We merge each pair of arcs, coming from $C_i(s)$ into a single arc $\gamma_i(s)$ as follows; see Figure 4, right. For the pair coming from $C_i(s)$ for $i \neq 0$, we connect the bottom loose ends by a straight segment up to isotopy. In Figure 4, right, we have isotoped the figure a bit which will be useful in further steps of the construction, and we call this subarc the *head* of $\gamma_i(s)$. For the pair coming from $C_0(s)$, we first cross the arcs next to the top loose ends as well as next to the bottom loose ends as in the figure. Then we connect the bottom loose

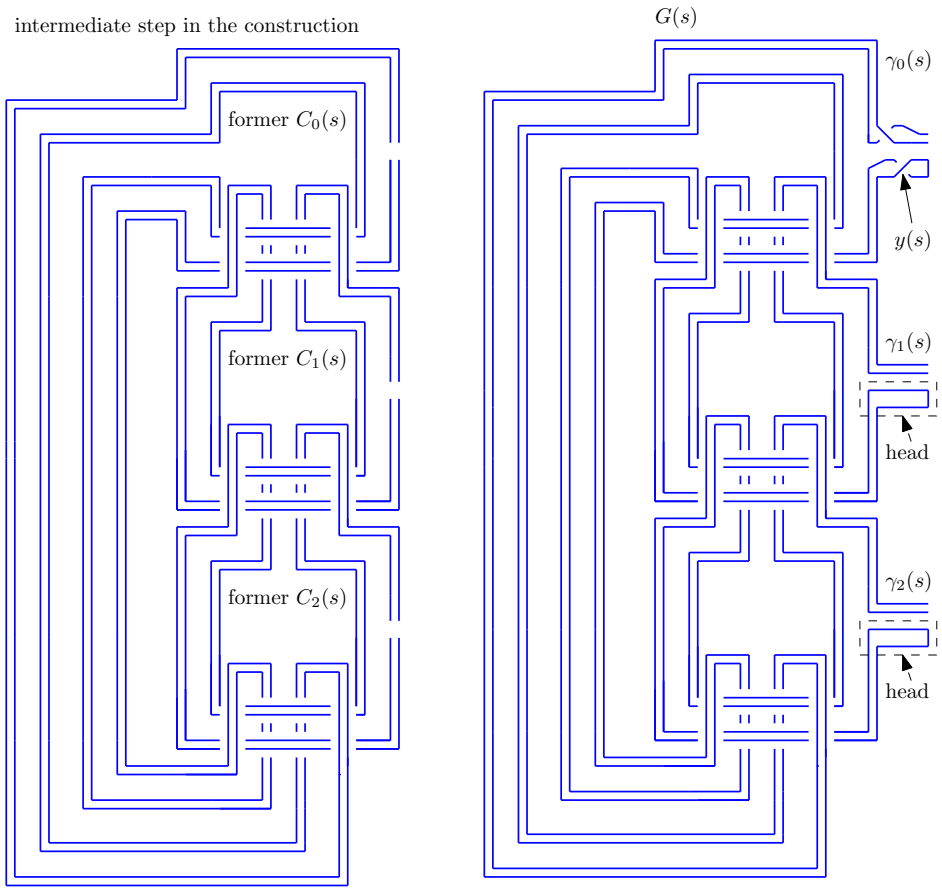


Figure 4 The sentence gadget with $\ell = 2$.

ends. Note that if we remove $\gamma_1(s), \ldots, \gamma_l(s)$ from the figure, then the crossings on $\gamma_0(s)$ can be removed by a II^- move. This finishes the construction of $G(s)$. The gadget $G(\widehat{s})$ is a mirror image of $G(s)$ along the vertical line (y-axis).

Merging gadget. We will also need a merging gadget depicted on Figure 5. We order all sentences as $s_1, \ldots, s_m$. The merging gadget consists of subgadgets $M(s_1), M(\widehat{s}_1), M(s_2), \ldots, M(s_m), M(\widehat{s}_m)$ separated by the dotted lines in the figure. Each subgadget has several loose ends. Two or four of them serve for connecting it to other subgadgets. The remaining ones come in pairs and the number of pairs equals to $\ell(s) + 1$. (Recall that this is the number of components of the sentence gadget $G(s)$.) These pairs of loose ends will be eventually connected to the loose ends of $\gamma_0(s), \ldots, \gamma_{\ell(s)}(s)$ in the top-down order.

Interconnecting the gadgets. Now, we describe how to interconnect the gadgets.

We place the sentence gadgets $G(s_1), \ldots, G(s_m)$ to the left of the merging gadget M in a top-down ordering. Similarly, we place $G(\widehat{s}_1), \ldots, G(\widehat{s}_m)$ to the right, again in a top-down ordering. First, for any $\bar{s} \in S \cup \widehat{S}$ and $i \in \{0, 2, 3, \ldots, \ell(\bar{s})\}$ (that is, $i \neq 1$), we connect $\gamma_i(\bar{s})$ with the $(i+1)$th pair of loose ends of $M(\bar{s})$ by a pair of parallel arcs as directly as possible without introducing new crossings; see Figure 7 (left).

Now we want to connect $\gamma_1(\bar{s})$ to the second pair of loose ends of $M(\bar{s})$. For simplicity, we describe this in the case $\bar{s} = s \in S$. The case $\bar{s} \in \widehat{S}$ is mirror symmetric. We pull a pair of parallel arcs from $\gamma_1(s)$ with the aim to reach $M(s)$ while obeying the following rules:

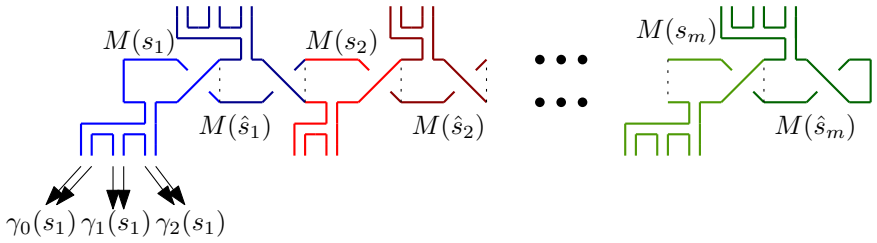


Figure 5 The merging gadget. The gadget is rotated. See also Figure 7 for correct orientation of the gadget when connecting it to other gadgets.

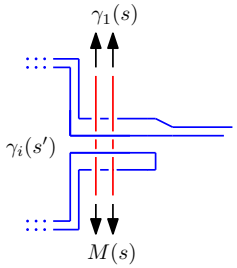


Figure 6 Interlacing the parallel arcs pulled out of $\gamma_1(s)$ with $\gamma_j(s')$.

(R1) We are not allowed to cross the merging gadget or the sentence gadgets (except the case described in the third rule below). We keep the newly introduced arcs on the left side from the merging gadget.

(R2) We are allowed to cross other pairs of parallel arcs introduced previously (when connecting $\gamma_i(s')$ to $M(s')$ for some i and s'). However, if we cross such a pair, we require that all four newly introduced crossings are resolved simultaneously (for example the new pair of parallel arcs is always above the older one). We even allow a self crossing of the newly introduced parallel arcs (but we again require that four newly introduced crossings are resolved simultaneously).

(R3) For every relation $R = (T, s')$ where $s \in T$, let $R = R_i(s')$. We interlace the newly introduced parallel arcs with $\gamma_i(s')$ as in Figure 6.

(R4) The total number of crossing is of polynomial size in the size of our instance $(S, \mathcal{R})$.

As we have quite some freedom how to perform the construction obeying the rules above, the resulting construction is not unique. An example how to get this construction systematically is sketched on Figure 7(right): The newly introduced pair of arcs is pulled little bit to the right, then we continue down towards the level of $G(s_m)$, then up towards the level of $G(s_1)$, and then back down to the original position. On this way we make a detour towards $\gamma_j(s')$ whenever we need to apply the rule (R3). Finally, we connect the parallel arcs to $M(s)$ without any further detour. This finishes the construction of $D(S, \mathcal{R})$.

A small set of axioms implies small defect. Now we very briefly sketch the easier implication that if $(S, \mathcal{R})$ admits an axiom set of size k, then $D(S, \mathcal{R})$ can be untangled with defect $2k$. For every sentence s in some fixed minimum axiom set, we unscrew the loop next to $y(s)$ (compare with Figure 4, right) and we also unscrew the loops next to $y(\widehat{s})$. This is altogether $2k$ I^- moves. If we show that the remaining crossings can be removed by II^- moves only, then we are done via Lemma 2. It is not hard to see that for s in the minimum axiom set, the initial unscrewing allows to simplify the gadget $G(s)$ essentially to $M(s)$ via II^- moves. This

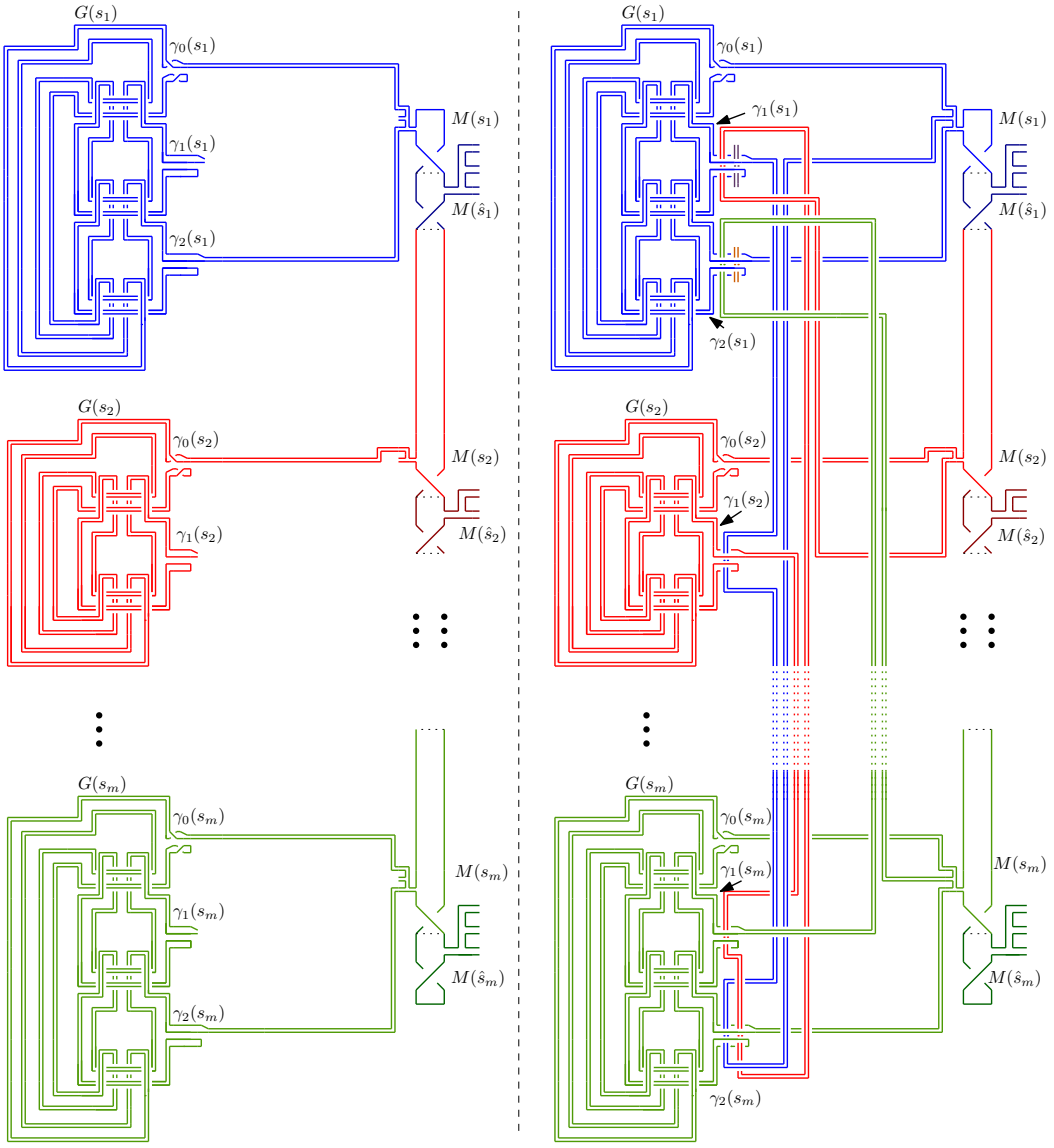


Figure 7 Left: The first step of interconnecting the gadgets – connecting $M(s)$ to the arcs $\gamma_i(s)$ for $i \neq 1$. The mirror symmetric part for $\widehat{s}$ is not displayed. Right: The second step of interconnecting the gadgets – connecting $M(s)$ to the arc $\gamma_1(s)$.

in particular releases at least one of the heads (see Figure 4, right) of the gadgets implied by the sentences in the minimum axiom set. Such a gadget can be then simplified again by II^- moves. By repeating this procedure we simplify all gadgets $G(s_1), \ldots, G(s_m)$ and analogously $G(\widehat{s}_1), \ldots, G(\widehat{s}_m)$. Then we remove remaining crossings by m more II^- moves.

Big minimum axiom sets implies big defect. It turns out that the core of the second implication in the reduction is to show the following: If the minimum axiom set for $(S, \mathcal{R})$ has size at least k, then any untangling of $D(S, \mathcal{R})$ has defect at least $2k$. For a proof of this claim we use the approach from [23] that allows to show by a local analysis that for some diagrams the defect of any untangling is strictly positive. On the other hand, this local

analysis does not show much more than that the defect is at least 1 but we need $2k$. In order to circumvent this problem we use the boosting lemma (Lemma 9) and we verify, roughly speaking, that the defect is at least 1 also for certain subdiagrams corresponding to sets of sentences U such that $S \setminus U$ is not an axiom set (as in the assumptions of Lemma 9). Then Lemma 9 implies that the defect is as high as we need.

References

1 N. Alon, G. Z. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving MAX-r-sat above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.

2 U. Bauer, A. Rathod, and J. Spreer. Parametrized Complexity of Expansion Height. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

3 B. A. Burton, T. Lewiner, J. Paixão, and J. Spreer. Parameterized complexity of discrete Morse theory. *ACM Trans. Math. Software*, 42(1):Art. 6, 24, 2016.

4 R. Crowston, M. R. Fellows, G. Z. Gutin, M. Jones, E. J. Kim, F. Rosamond, I. Z. Ruzsa, S. Thomassé, and A. Yeo. Satisfying more than half of a system of linear equations over GF(2): A multivariate approach. *J. Comput. Syst. Sci.*, 80(4):687–696, 2014.

5 R. Crowston, M. Jones, G. Muciaccia, G. Philip, A. Rai, and S. Saurabh. Polynomial kernels for lambda-extendible properties parameterized above the Poljak-Turzik bound. In A. Seth and N. K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24 of *LIPIcs*, pages 43–54. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

6 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, Ma. Pilipczuk, Mi. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, Cham, 2015.

7 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

8 J. Flum and M. Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

9 G. Z. Gutin, L. van Iersel, M. Mnich, and A. Yeo. Every ternary permutation constraint satisfaction problem parameterized above average has a kernel with a quadratic number of variables. *J. Comput. Syst. Sci.*, 78(1):151–163, 2012.

10 G. Z. Gutin, E. J. Kim, M. Lampis, and V. Mitsou. Vertex cover problem parameterized above and below tight bounds. *Theory Comput. Syst.*, 48(2):402–410, 2011.

11 G. Z. Gutin and V. Patel. Parameterized traveling salesman problem: Beating the average. *SIAM J. Discret. Math.*, 30(1):220–238, 2016.

12 W. Haken. Theorie der Normalflächen. *Acta Math.*, 105:245–375, 1961.

13 J. Hass, J. C. Lagarias, and N. Pippenger. The computational complexity of knot and link problems. *J. ACM*, 46(2):185–211, 1999.

14 L. H. Kauffman and S. Lambropoulou. Hard unknots and collapsing tangles. In *Introductory lectures on knot theory*, volume 46 of *Ser. Knots Everything*, pages 187–247. World Sci. Publ., Hackensack, NJ, 2012.

15 D. Koenig and A. Tsvietkova. NP–hard problems naturally arising in knot theory. *Trans. Amer. Math. Soc. Ser. B*, 8:420–441, 2021.

16 G. Kuperberg. Knottedness is in **NP**, modulo GRH. *Adv. Math.*, 256:493–506, 2014.

17 M. Lackenby. A polynomial upper bound on Reidemeister moves. *Ann. of Math. (2)*, 182(2):491–564, 2015.

18 M. Lackenby. Elementary knot theory. In *Lectures on geometry*, Clay Lect. Notes, pages 29–64. Oxford Univ. Press, Oxford, 2017.

19 M. Lackenby. The efficient certification of knottedness and Thurston norm. *Adv. Math.*, 387, 2021.

20 M. Lackenby. Marc Lackenby announces a new unknot recognition algorithm that runs in quasi-polynomial time, 2021. URL: `https://www.maths.ox.ac.uk/node/38304`.

21 M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999.

22 M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *J. Comput. Syst. Sci.*, 75(2):137–153, 2009.

23 A. de Mesmay, Y. Rieck, E. Sedgwick, and M. Tancer. The unbearable hardness of unknotting. *Adv. Math.*, 381:107648, 36, 2021.

Almost Tight Approximation Hardness for Single-Source Directed k-Edge-Connectivity

Chao Liao ✉
Shanghai Jiao Tong University, China

Qingyun Chen ✉
University of California, Merced, CA, USA

Bundit Laekhanukit ✉
Shanghai University of Finance and Economics, China

Yuhao Zhang ✉
Shanghai Jiao Tong University, China

Abstract

In the k-outconnected directed Steiner tree problem (k-DST), we are given an n-vertex directed graph $G = (V, E)$ with edge costs, a connectivity requirement k, a root $r \in V$ and a set of terminals $T \subseteq V$. The goal is to find a minimum-cost subgraph $H \subseteq G$ that has k edge-disjoint paths from the root vertex r to every terminal $t \in T$. The problem is NP-hard, and inapproximability results are known in several parameters, e.g., hardness in terms of n: $\log^{2-\varepsilon} n$-hardness for $k = 1$ [Halperin and Krauthgamer, STOC'03], $2^{\log^{1-\varepsilon} n}$-hardness for general case [Cheriyan, Laekhanukit, Naves and Vetta, SODA'12], hardness in terms of k [Cheriyan et al., SODA'12; Laekhanukit, SODA'14; Manurangsi, IPL'19] and hardness in terms of $|T|$ [Laekhanukit, SODA'14].

In this paper, we show the approximation hardness of k-DST for various parameters.

- $\Omega\left(|T|/\log|T|\right)$-approximation hardness, which holds under the standard complexity assumption NP $\neq$ ZPP. The inapproximability ratio is tightened to $\Omega\left(|T|\right)$ under the Strongish Planted Clique Hypothesis [Manurangsi, Rubinstein and Schramm, ITCS 2021]. The latter hardness result matches the approximation ratio of $|T|$ obtained by a trivial approximation algorithm, thus closing the long-standing open problem.
- $\Omega\left(2^{k/2}/k\right)$-approximation hardness for the general case of k-DST under the assumption NP $\neq$ ZPP. This is the first hardness result known for survivable network design problems with an inapproximability ratio exponential in k.
- $\Omega\left((k/L)^{L/4}\right)$-approximation hardness for k-DST on L-layered graphs for $L \leq O\left(\log n\right)$. This almost matches the approximation ratio of $O(k^{L-1} \cdot L \cdot \log|T|)$ achieved in $O(n^L)$-time due to Laekhanukit [ICALP'16].

We further extend our hardness results in terms of $|T|$ to the undirected cases of k-DST, namely the single-source k-vertex-connected Steiner tree and the k-edge-connected group Steiner tree problems. Thus, we obtain $\Omega\left(|T|/\log|T|\right)$ and $\Omega\left(|T|\right)$ approximation hardness for both problems under the assumption NP $\neq$ ZPP and the Strongish Planted Clique Hypothesis, respectively. This again matches the upper bound obtained by trivial algorithms.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Mathematics of computing → Paths and connectivity problems

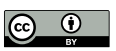
Keywords and phrases Directed Steiner Tree, Hardness of Approximation, Fault-Tolerant and Survivable Network Design

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.89

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2202.13088

Funding *Qingyun Chen*: Qingyun Chen is supported in part by NSF grants CCF-2121745 and CCF-1844939.



49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 89; pp. 89:1–89:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Bundit Laekhanukit: Bundit Laekhanukit is supported by the 1000-talent plan award by the Chinese government, supported by Science and Technology Innovation 2030 — "New Generation of Artificial Intelligence" Major Project No.(2018AAA0100903), NSFC grant 61932002, Program for Innovative Research Team of Shanghai University of Finance and Economics (IRTSHUFE) and the Fundamental Research Funds for the Central Universities.
Yuhao Zhang: Yuhao Zhang is supported by National Natural Science Foundation of China (NSFC) under Grant No. 62102251.

1 Introduction

Fault-Tolerant and Survivable Network Design have been an active area of research for decades as enterprises depend more on communication networks and distributed computing. The need to design a network that can operate without disruption when one or more components fail has been growing dramatically. Henceforth, network scientists have formulated many models to address these problems. Amongst them, the simplest and arguably most fundamental problem in the area is the *minimum-cost k-outconnected spanning subgraph* (k-OCSS) problem that captures the problem of designing a multi-casting network with survivability property. The k-OCSS problem is a generalization of the *minimum spanning tree* and the *minimum-cost arborescence* problems, where the goal is to design a network that can operate under failures of at most $k-1$ points. More formally, k-OCSS asks to find a minimum-cost subgraph such that the root vertex is k-connected to every other vertex.

In this paper, we study the analog of k-OCSS in the presence of Steiner vertices, namely the *k-outconnected directed Steiner tree* problem (k-DST): Given a directed graph $G=(V,E)$ with cost c_e on arcs, a root vertex r and a set of terminals T, the goal is to find a minimum-cost subgraph $H \subseteq G$ such that H has k edge-disjoint paths from the root r to every terminal $t \in T$, i.e., the root remains connected to every terminal even after the removal of $k-1$ arcs. The k-DST problem is a natural generalization of the classical *directed Steiner tree* problem (DST) to high connectivity settings.

The undirected counterpart of k-DST is the *minimum-cost single source k-edge-connected Steiner tree* problem, which admits a factor-two approximation algorithm [28], and the vertex-connectivity variant admits an $O(k \log k)$-approximation algorithm due to Nutov [36]. The k-DST problem, on the other hand, has no non-trivial approximation algorithm for $k \geq 3$, except for the special case of L-layered graph, which admits $O(k^L \cdot L \cdot \log |T|)$-approximation algorithm due to Laekhanukit [33]. The cases of $k=1$ and $k=2$ are also notorious problems themselves, as both admit polylogarithmic approximation algorithms that run in quasi-polynomial time, but no polynomial-time approximation algorithms with sub-polynomial approximation. It has been long-standing open problems whether such algorithms exist for DST and 2-DST.

In this paper, we obtain several inapproximability results for k-DST. First, we show an approximation hardness of $\Omega\left(|T|/\log|T|\right)$ for k-DST under $\mathsf{NP} \neq \mathsf{ZPP}$, which holds when k is larger than $|T|$, thus implying that a trivial $|T|$-approximation algorithm for the problem is tight up to the lower order term.

▶ **Theorem 1.** *For $k > |T|$, unless $\mathsf{NP} = \mathsf{ZPP}$, it is hard to approximate the k-DST problem to within a factor of $\Omega\left(|T|/\log|T|\right)$.*

Assuming the *Strongish Planted Clique Hypothesis* (SPCH) [35], our hardness result is tight up to a constant factor, and it, indeed, rules out $f(|T|) \cdot \mathrm{poly}(n)$-time $o(|T|)$-approximation algorithm for any function f depending only on $|T|$. See discussion in the appendices of the full version of this paper.

Table 1 Summary of the results for k-DST.

Parameter	Lower Bound (This paper)	Lower Bound (Previous)	Upper Bound
Connectivity k	$\boldsymbol{\Omega\left(2^{k/2}/k\right)}$ Theorem 4	$\Omega\left(k/\log k\right)$ [34]	unknown for general $k \geq 3$
Connectivity k, Depth L	$\boldsymbol{\Omega\left((k/L)^{(1-\epsilon)L/4-2}\right)}$ Theorem 3	$\Omega\left(k/\log k\right)$ [34]	$O\left(k^{L-1} \cdot L \cdot \log \lvert T\rvert\right)$ [33]
Terminals $\lvert T\rvert$	$\boldsymbol{\Omega(\lvert T\rvert/\log \lvert T\rvert)}$ Theorem 1	$\lvert T\rvert^{1/4-\epsilon}$ [32]	$\lvert T\rvert$ folklore

▶ **Theorem 2.** *Assuming the Strongish Planted Clique Hypothesis, there is no $f(|T|)\cdot\mathrm{poly}(n)$-time $o(|T|)$-approximation algorithm for the k-DST problem.*

Next, we show that the k-DST admits no $O\left((k/L)^{L/4}\right)$-approximation algorithm even on an L-layered graph, which consists of L parts, called *layers*, and every arc joins a vertex from the i-th layer to the $(i+1)$-th layer.

▶ **Theorem 3.** *It is hard to approximate the k-DST problem on L-layered graphs $G=(V,E)$ for $\Omega(1) \leq L \leq O\left(\log |V|\right)$ to within a factor of $\Omega\left((k/L)^{(1-\epsilon)L/4-2}\right)$ for any constant $\epsilon > 0$, unless* NP = ZPP.

In addition, we obtain an approximation hardness exponential in k by setting a different parameter in the reduction, which improves upon the previously known approximation hardness of $\Omega\left(k/\log k\right)$ due to Manurangsi [34] (which is in turn based on the two previous results [32, 11]), and is the first known approximation hardness for connectivity problems whose ratio is exponential in the connectivity requirement.

▶ **Theorem 4.** *For $k < |T|$, it is hard to approximate the k-DST problem to within a factor of $\Omega\left(2^{k/2}/k\right)$, unless* NP = ZPP.

Using the technique of Cheriyan, Laekhanukit, Naves and Vetta [11], which is based on the padding technique introduced by Kortsarz, Krauthgamer and Lee [31], we extend our hardness result to the undirected counterpart of k-DST, namely, the *single source k-vertex-connected Steiner tree* problem (k-ST) (a.k.a. *undirected rooted subset k-connectivity, shorty, rooted-k-VC*) and the special case of k-DST, namely *k-edge-connected group Steiner tree* problem (k-GST).

The latter problem is a natural fault-tolerant generalization of the classical group Steiner tree problem [19], which has been studied in [29, 24, 6, 3]. To the best of our knowledge, a non-trivial approximation algorithm for this problem is known only for $k = 1, 2$. For $k \geq 3$, only a bicriteria approximation algorithm, where the connectivity requirement can be dropped by a factor $O(\log n)$, is known in [6]. Nevertheless, a trivial $|T|$-approximation algorithm exists for all values of k and we also show its tightness (up to the lower order term) for sufficiently large k.

▶ **Theorem 5.** *For $k > |T|$, unless* NP = ZPP, *it is hard to approximate the k-ST problem to within a factor of $\Omega\left(|T|/\log |T|\right)$.*

▶ **Theorem 6.** *For $k > |\mathcal{T}|$, unless* NP = ZPP, *it is hard to approximate the k-GST problem to within a factor of $\Omega\left(|\mathcal{T}|/\log |\mathcal{T}|\right)$, where $|\mathcal{T}|$ is the number of groups.*

Related work

The k-DST is well-studied in the special case where all vertices are terminals. This problem is, as mentioned, known as the k-outconnected spanning subgraph problem (k-OCSS), which admits polynomial-time algorithms due to the seminal work of Frank and Tardos [17] (also, see [16]). However, while k-OCSS is polynomial-time solvable, its undirected counterpart is NP-hard. Nevertheless, Frank-Tardos's algorithm has been used as subroutines to derive a 2-approximation algorithm for the undirected variant of k-DST and its generalization [30].

In the presence of Steiner vertices, k-DST becomes much harder to approximate. For the case of $k = 1$, the best known polynomial-time approximation algorithms are $|T|^{\epsilon}$, for any constant $\epsilon > 0$, due to the work of Charikar et al. [8], and the same approximation ratio (with an additional log factor) applies for the case $k = 2$ due to the work of Grandoni and Laekhanukit [22]. These two special cases of k-DST, especially for the case $k = 1$, have been perplexing researchers for many decades as it admits polylogarithmic approximation algorithms in quasi-polynomial-time, whereas there is no known sub-polynomial-approximation algorithm for the problems; see, e.g., [8, 23, 20, 22]. It has been a long-standing open problem whether polylogarithmic or even sub-polynomial-approximation ratios can be achieved in polynomial time. Some special cases of k-DST have been studied in the literature. Laekhanukit [33] studied the k-DST instances on L-layered graphs, and its extensions to the *L-shallow instances*, and presented an $O(k^L \cdot L \cdot \log |T|)$-approximation algorithm that runs in $n^{O(L)}$ time. Polynomial-time polylogarithmic approximation algorithms for k-DST are known in quasi-bipartite graphs [7, 37] (also, see [18, 27] for the case $k = 1$, which matches the approximation lower bound of $(1 - \epsilon) \ln k$, assuming P $\neq$ NP, inherited from the *Set Cover* problem [14, 12]).

For the undirected case of $k = 1$, namely the Steiner tree problem, it admits a 1.39-approximation algorithm due to the breakthrough result of Byrka et al. [2] and admits a $\frac{73}{60}$-approximation algorithm on quasi-bipartite graphs due to the work of Goemans et al. [21]. For $k \geq 2$, the problems on undirected graphs are branched into edge and vertex connectivity variants. This is not the case for directed graphs as there is a simple approximation-preserving reduction from edge-connectivity to vertex-connectivity and vice versa. For the edge-connectivity problem, it admits a 2-approximation algorithm by using Frank-Tardos's algorithm as subroutines when there is no Steiner vertex [30], and a 2-approximation algorithm via *iterative rounding* due to the seminal result of Jain [28], which also applies for the more general case of the *edge-connectivity survivable network design* problem. For the vertex-connectivity problem, there is a 2-approximation algorithm for $k = 2$ due to Fleischer, Jain and Williamson [15], but the problem becomes hard polynomial in k, for sufficiently large k [11], assuming P $\neq$ NP. The best known approximation algorithm for the *single-source k-vertex-connectivity* problem on undirected graphs is $O(k \log k)$ due to Nutov [36].

The network design problem where the connectivity requirements are between pairs of vertices is sometimes called point-to-point network design. A natural generalization is to extend the requirements to be between subsets of vertices, called groups. The classical problem in this genre is the well-studied *group Steiner tree* problem (see, e.g., [19, 9, 10, 25, 26, 4]). The group Steiner tree problem admits an approximation ratio of $O(\log q \log n)$ [19], which requires a probabilistic metric-tree embedding [1, 13]. This approximation ratio is almost tight as it matches the lower bound of $O(\log^{2-\epsilon} n)$, for any constant $\epsilon > 0$, due to the hardness result of Halperin and Krauthgamer [26] assuming NP $\not\subseteq$ ZPTIME$(n^{\text{polylog}(n)})$; also, see the improved hardness result in [23]. The fault-tolerant variant of the group Steiner tree problem is called the k-edge-connected group Steiner tree problem, studied in [29, 24, 6, 3]. As mentioned, true approximation algorithms for this problem are known only for $k = 1, 2$ [19, 29, 24]. For $k \geq 3$, only a bi-criteria approximation algorithm is known [6].

Organization

Section 2 is devoted to preliminary notations, definitions and facts. The reductions for k-DST are presented in Section 3, 4 and 5. We give some intuitions on the techniques in Section 3 and describe the reduction for hardness in terms of $|T|$ in Section 4 and for hardness in terms of k in Section 5. In Section 6 we briefly discuss the results for k-DST. Finally, we extend our techniques to undirected graphs and tighten the $\Omega(|T|/\log|T|)$ lower bound to $\Omega(|T|)$ under a stronger complexity hypothesis. These results are presented in the appendices of the full version of this paper.

2 Preliminaries

We use a standard graph terminology. Let $G = (V, E)$ be any graph, which can be either directed or undirected. For undirected graphs, we refer to the elements in E as the "edges" of G and denote by $\deg_G(v)$ the number of edges incident to a vertex $v \in V$. For directed graphs, we refer to the elements in E as the "arcs" of G and denote by $\text{indeg}_G(v)$ the number of arcs entering v. The notation for an edge/arc is (u, v), or sometimes $u \to v$ for an arc. For a path between vertex u and w, we call it a (u, w)-path and write it as $(u, v, \ldots, w)$ for both directed and undirected graphs, or $u \to v \to \cdots \to w$ for only directed graphs. The graphs may have multiple edges/arcs between two same vertices u and v, and both $\deg_G(v)$ and $\text{indeg}_G(v)$ count multiple ones. We drop G from the notations when it is clear from the context. When more than one graph is considered, we use $V(G)$ to clarify the vertex set of G, and $E(G)$ the edge/arc set.

k-(Edge)-Connected Directed Steiner Tree

The k-(edge)-connected directed Steiner tree problem (k-DST) is defined as follows. An input instance is of the form (k, G, r, T) where $k \in \mathbb{Z}_{\geq 1}$ is the connectivity requirement, $G = (V, E)$ is a directed graph with weight (or cost) on arcs $c : E \to \mathbb{Q}_{\geq 0}$, $r \in V$ is called root and $T \subseteq V$ is a set of terminals. A subgraph $H = (V, F)$ of G is k-connected if there exist k edge-disjoint paths in H from r to t for each terminal $t \in T$. Sometimes we also refer to a k-connected subgraph as a feasible solution to the k-DST problem. The problem is to find a k-connected subgraph $H = (V, F)$ of minimum cost $c(H) = \sum_{e \in F} c(e)$.

Here, we define the problem in terms of edge-connectivity. A vertex-connectivity variant is defined similarly except that it asks for (openly) vertex-disjoint paths instead of edge-disjoint paths. Both variants are equivalent in terms of approximability on directed graphs because there exist straightforward polynomial-time approximation-preserving reductions from any one to the other.

(Minimum) Label Cover

An instance of the label cover problem is given by an (undirected) bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$, a set of labels $\Sigma = [g]$ and (projection) constraints $\pi_{uv} : \Sigma \to \Sigma$ on each edge $(u, v) \in \mathcal{E}$. A multilabeling $\sigma : \mathcal{U} \cup \mathcal{V} \to 2^{\Sigma}$ is a subset of labels assigned to each vertex. We say that σ *covers* an edge $(u, v) \in \mathcal{E}$ if $\pi_{uv}(a) = b$ for some $a \in \sigma(u)$ and $b \in \sigma(v)$. A multilabeling is *feasible* if it covers all the edges of $\mathcal{E}$. The problem asks for a feasible multilabeling σ with minimum cost $c(\sigma) = \sum_{u \in \mathcal{U} \cup \mathcal{V}} |\sigma(u)|$.

Manurangsi [34] proved that the label cover problem has a hardness gap in terms of the maximum degree of $\mathcal{G}$.

▶ **Theorem 7** ([34]). *For every positive integer $g > 1$, unless* NP = ZPP*, it is hard to approximate a label cover instance of maximum degree $O(g \log g)$ and alphabet size $O(g^4 \log^2 g)$ within a factor of g.*

The following corollary can be deduced straightforwardly.

▶ **Corollary 8.** *It is hard to approximate a label cover instance of maximum degree Δ and alphabet size $O\left(\Delta^4/\text{polylog}(\Delta)\right)$ to within a factor of $\Omega\left(\Delta/\log\Delta\right)$, or to within a factor of $\Omega\left(\Delta^{1-\varepsilon}\right)$ for any constant $\varepsilon > 0$, unless* NP = ZPP*.*

To obtain the hardness results on k-DST (and related problems), we present reductions from the label cover problem on an instance $(\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E}), \Sigma = [g], \pi = \{\pi_{uv} : \Sigma \to \Sigma\}_{(u,v)\in\mathcal{E}})$ of maximum degree Δ. For the ease of presentation, let $\mathcal{U} = \{u_1, u_2, \ldots\}$ and $\mathcal{V} = \{v_1, v_2, \ldots\}$.

Finally, we prepare a technical lemma for future reference. We say that a subgraph I of G is an *induced matching* if 1) I is a matching, i.e., each vertex in G is the endpoint of at most one edge in $E(I)$ and 2) I is an induced subgraph, i.e., all edges with two endpoints both in $V(I)$ are included in $E(I)$.

▶ **Lemma 9** (Folklore). *Let $G = (U, V, E)$ be a bipartite graph of maximum degree Δ. There exist a* partition *of the edges $E = E_1 \cup E_2 \cup \cdots \cup E_\Delta$ such that each E_i is a matching of G, and a partition $E'_1, E'_2, \ldots, E'_\delta$ of E for some $\delta \leq 2\Delta^2$ such that each E'_i is an induced matching. Furthermore, such partitions can be found in polynomial time.*

3 Overview of the Reductions

To give some intuitions on how our reductions work, we dedicate this section to providing an overview. We have two main reductions, which are tailored for inapproximability results in different parameters, say $|T|$ and k.

Both of the reductions inherit approximation hardness from the same source – the label cover problem, denoted by $(\mathcal{G}, \Sigma, \pi)$. We design reductions that have a one-to-one correspondence between a feasible solution to the label cover problem and that to the k-DST problem, i.e.,

- **Completeness**: Given a feasible multilabeling σ of the label cover instance $(\mathcal{G}, \Sigma, \pi)$, there is a corresponding k-connected subgraph H of G such that $c(\sigma) = c(H)$.
- **Soundness**: Given a k-connected subgraph H of the k-DST instance, there is a corresponding feasible multilabeling σ of the label cover instance $(\mathcal{G}, \Sigma, \pi)$ such that $c(\sigma) = c(H)$.

We note that certain details in the ideas mentioned below are intentionally omitted, which can be found in later chapters.

Basic Construction

First, we present the basic construction for the case $k = 1$, which is sufficient to maintain the completeness property (but not the soundness). We start by adding a root vertex r and terminal vertices t_{ij}, one for each edge $(u_i, v_j) \in \mathcal{E}$. Observe that each terminal corresponds to an edge in the label cover instance. Thus, we wish to make an (r, t_{ij})-path in G to correspond to a labeling that covers an edge in $\mathcal{G}$, which we call a *cover path*. To be more precise, if the edge (u_i, v_j) is covered by a label pair (a, b), then the corresponding *cover path* in G is the path $r \to u_i^a \to v_j^b \to v_j \to t_{ij}$ where the vertices u_i^a and v_j^b are simply added to the graph G. The appearance of the vertex $u_i^a \in V$ along the cover path can be interpreted as assigning the label a to the vertex $u_i \in \mathcal{U}$, and similarly, v_j^b means assigning the label b to

the vertex $v_j \in \mathcal{V}$. See Figure 1 for illustration. Note that any feasible multilabeling covers all the edges of $\mathcal{G}$, so we can collect all cover paths (and the involved vertices) to form a subgraph, where the root is already 1-connected to the terminals. By setting the weight of all arcs $r \to u_i^a$ and $v_j^b \to v_j$ to be 1 while leaving other arcs zero-cost, the subgraph has the same cost as the multilabeling.

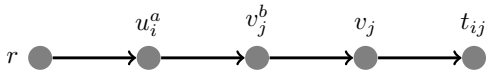


Figure 1 A cover path $(r, u_i^a, v_j^b, v_j, t_{ij})$.

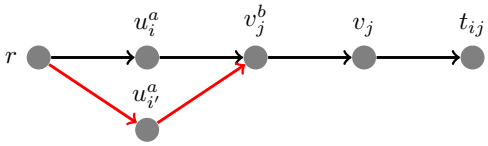


Figure 2 An illegal path $(r, u_{i'}^a, v_j^b, v_j, t_{ij})$.

However, the soundness property does not hold on the basic construction because it creates many *illegal* (r, t_{ij})-paths. Such a path goes from the root to a terminal t_{ij} by using the route that differs from $(u_i, v_j) \in \mathcal{E}$, e.g., $r \to u_{i'}^a \to v_j^b \to v_j \to t_{ij}$, where $i' \neq u_i$; see Figure 2. This means that, although a solution is feasible to the 1-DST instance, the subgraph may not have a cover path for every terminal. Thus, the corresponding multilabeling may leave some edges in the label cover instance uncovered. To ensure that at least one cover path exists for every terminal, we need to modify our instance using the *padding arc* technique.

Padding Arcs

Consider an illegal (r, t_{ij})-path in the basic construction. While we wish the (r, t_{ij})-path to visit vertices u_i^a and v_j^b, which corresponds to a satisfying labeling to the edge (u_i, v_j) in the label cover instance, the illegal path instead visits $u_{i'}^{a'}$ with $u_{i'} \neq u_i$. In particular, the illegal path exploits a cover path for some other edge $(u_{i'}, v_j)$ that share the same endpoint with (u_i, v_j).

To prevent this from happening, we construct a zero-cost *padding path* from the root to the terminal t_{ij} that shares some arcs with the illegal path. These two paths are mutually exclusive in contributing to the edge-connectivity between the root r and the terminal t_{ij}. As we set the connectivity requirement to be the same as the indegree of t_{ij}, it forces all the padding paths for t_{ij} to be used in any feasible solution. Once all the padding paths are used to form $k-1$ edge-disjoint (r, t_{ij}), the only path available is forced to be a cover path.

Size of $|T|$ and k

The construction as mentioned above yields a one-to-one correspondence between the feasible solutions to k-DST and that of the label cover problem. However, the size of the construction in terms of the parameter k (resp., $|T|$) is too large comparing to the inapproximation factor of $\Omega(\Delta/\log\Delta)$ inherited from the label cover problem. Specifically, the value of k can be as large as the number of illegal paths, and the value of $|T|$ can be as large as $|\mathcal{E}| = |\mathcal{U}| \cdot \Delta$. Therefore, we need to optimize the size of $|T|$ and k, where we use two different techniques, one for each parameter.

- To control the size of the terminal set, we partition the edge set $\mathcal{E}$ into Δ **matchings**, and only **one** terminal is constructed for **each matching** rather than having one terminal for each edge. This, thus, reduces the size of T to Δ.[1]

[1] Laekhanukit [32] applies a similar techniques using strong edge-coloring, which gives a worse factor of Δ^2.

- To control the connectivity requirement k, we partition the edge set $\mathcal{E}$ into $\delta \leq 2\Delta^2$ **induced matchings** and create a *d-ary* tree structure with δ leaves, where d is an adjustable parameter. Roughly speaking, by exploiting the tree structure, we need to add to each terminal only d *padding arcs* for each of the $O(\log_d \delta)$ layers of the tree. Thus, the connectivity requirement k is reduced to $O(d \log_d \delta)$.

Generalized to Undirected Settings

We also apply the techniques mentioned above to the undirected settings. For the undirected k-connected Steiner tree problem (k-ST), we migrate the reduction for k-DST with a hardness result in terms of $|T|$ to its vertex-connectivity version, with necessary adaptions, thus reproducing the same $|T|/\log|T|$ inapproximability. The k-connected group Steiner tree problem (k-GST) is a generalization of the k-ST problem that turns out to be one of the key components in designing approximation algorithms for k-DST [38, 22] for $k = 1, 2$. By incorporating ideas from [5], we achieved the same $q/\log q$ inapproximability for even the edge-disjoint version of k-GST where q is the number of groups. The main difficulty is that undirected edges allow new illegal paths to enter a terminal. Fortunately, by equalizing the connectivity requirement and the size of a group, it turns out we can handle the extra paths. See the appendices in the full version of this paper for a complete presentation.

4 Inapproximability in Terms of the Number of Terminals

In this section, we discuss the hardness reduction that is tailored for the parameter $|T|$. Our reduction takes as input a label cover instance $(\mathcal{G}, \Sigma, \pi)$ and then produces a k-DST instance $(k, G = (V, E), r, T)$ as an output. The reduction runs in polynomial-time, and there is a one-to-one mapping between the solutions to the two problems. Thus, the inapproximability result of label cover is mapped to the inapproximability of k-DST directly. The main focus in this section is in reducing the number of terminals by exploiting edge-disjoint paths.

Base Construction

The construction starts from a basic building block.

1. First, create a graph G with a single vertex, the root r.
2. (Refer to Figure 3) For each $u_i \in \mathcal{U}$, create in G each vertex u_i^a from $A_i = \{u_i^a : a \in \Sigma\}$; connect r to u_i^a by an arc (r, u_i^a) of cost one.
3. (Refer to Figure 4) For each $v_j \in \mathcal{V}$, create a counterpart of it (also named v_j) in G and create each vertex v_j^b from $B_j = \{v_j^b : b \in \Sigma\}$; connect v_j^b to v_j by an arc (v_j^b, v_j) of cost one.
4. (Refer to Figure 5) For each $(u_i, v_j) \in \mathcal{E}$ and $a, b \in \Sigma$, connect u_i^a to v_j^b by a zero-cost arc if $\pi_{u_i v_j}(a) = b$.

Final Construction

Finally, we have to add some zero-cost arcs so called *padding arcs* to the base construction, which are meant to enforce the constraints of the label cover problem into the k-DST instance. We first partition the edge set $\mathcal{E}$ of the graph $\mathcal{G}$ in the label cover instance into Δ matchings, denoted by $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_\Delta$. This step can be done in polynomial-time due to Lemma 9. We then create a set of Δ terminals corresponding to these matchings.

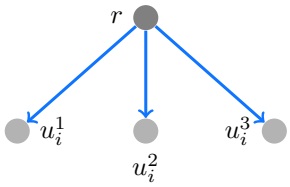


Figure 3 $\Sigma = \{1, 2, 3\}$.

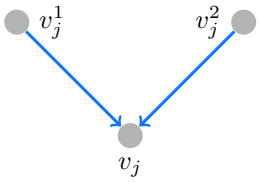


Figure 4 $\Sigma = \{1, 2\}$.

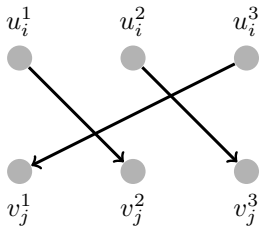


Figure 5 $\pi_{u_i v_j}(x) = x \bmod 3 + 1$.

1. For each matching $\mathcal{E}_m$, add a terminal t_m to G and connect the counterpart of each $v_j \in V(\mathcal{E}_m) \cap \mathcal{V}$ in G to t_m. (Refer to Figure 6).

Next, we add some padding arcs to form padding paths that "kills" illegal paths.

2. Instead of connecting u_i^a to v_j^b for the edges $(u_i, v_j) \in \mathcal{E}$ such that $\pi_{u_i v_j}(a) = b$ directly, we add an internal node w_{ij}^{ab} and replace the original arc (u_i^a, v_j^b) in G by arcs (u_i^a, w_{ij}^{ab}) and (w_{ij}^{ab}, v_j^b). (Here and thereafter refer to Figure 7.)
3. For each u_i^a, we connect r to u_i^a by $\deg_{\mathcal{G}}(u_i)$ copies of an arc (r, u_i^a).
4. For $1 \le m \le \Delta$, $(u_i, v_j) \in \mathcal{E}$ and $a, b \in \Sigma$ such that $\pi_{u_i v_j}(a) = b$, if $(u_i, v_j) \in \mathcal{E}_m$, then we add an arc (u_i^a, t_m); otherwise, we add an arc (w_{ij}^{ab}, t_m). Thus, we finally have $|\Sigma| \cdot |\mathcal{E}_m|$ arcs from the vertex set $\{u_i^a : u_i \in \mathcal{U}, a \in \Sigma\}$ to t_m, and $|\Sigma| \cdot (|\mathcal{E}| - |\mathcal{E}_m|)$ arcs from the internal vertex set $\left\{w_{ij}^{ab} : \pi_{u_i v_j}(a) = b\right\}$ to t_m.
5. We set $k = \max_{1 \le m \le \Delta} \text{indeg}_G(t_m)$. To make the connectivity requirement uniform, we add $k - \text{indeg}_G(t_m)$ copies of an arc (r, t_m) for each terminal t_m.

Please see Figure 7 for an illustration. Observe that the connectivity requirement k is exactly the indegree of each terminal. Thus, all of its incoming arcs are needed in any feasible solution. Now, consider the edge $(u_i, v_{j'})$ in the figure, which is not in $\mathcal{E}_m$. It is possible that a feasible solution includes the path $r \to u_i^a \to w_{ij'}^{ab'} \to v_{j'}^{b'} \to v_{j'} \to t_m$, which is an illegal path for the terminal t_m. However, if we wish to route k-edge-disjoint paths between the root r and t_m, then the arc $w_{ij'}^{ab'} \to t_m$ must be used, and the only way to use this arc is to traverse from $u_i^a \to w_{ij'}^{ab'}$. This prevents the illegal path from using this arc, meaning that it cannot be included in any k-edge-disjoint (r, t_m)-paths. Less formally, we may say that it gets killed by the padding path $r \to u_i^a \to w_{ij'}^{ab'} \to t_m$.

Why Do We Need A Matching?

Consider a terminal t_m. Our construction promises edge-disjoint cover paths from r to t_m for every edge in $\mathcal{E}_m$. However, if the edges in $\mathcal{E}_m$ do not form a matching, then two cover paths may share some edge. That is, the corresponding subgraph of a feasible multilabeling may have connectivity less than k, and the completeness property breaks.

Next, we prove the one-to-one correspondence between solutions to the two instances. Wlog., assume that any solution to the k-DST problem contains all of the zero-cost arcs.

Completeness

Given a feasible multilabeling σ of the label cover instane $(\mathcal{G}, \Sigma, \pi)$, we show that there is a corresponding feasible subgraph $H = (V, F)$ of $G = (V, E)$ such that $c(\sigma) = c(H)$. The set F consists of three types of arcs: 1) all zero-cost arcs in G; 2) the one-cost arcs (r, u_i^a) for each $u_i \in \mathcal{U}$ and $a \in \sigma(u_i)$; 3) the one-cost arcs (v_j^b, v_j) for each $v_j \in \mathcal{V}$ and $b \in \sigma(v_j)$. Clearly, $c(H) = c(\sigma)$ and the definition of F induces an injective mapping.

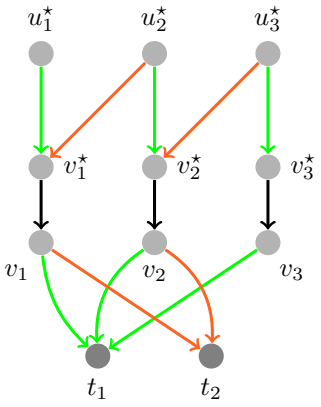


Figure 6 The terminal t_1 corresponds to the matching $\{(1,1),(2,2),(3,3)\}$ and t_2 corresponds to $\{(2,1),(3,2)\}$.

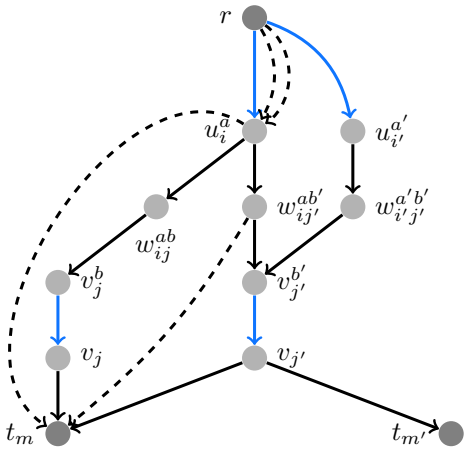


Figure 7 Padding arcs are dashed. The matching $\mathcal{E}_m$ contains the edges (u_i, v_j) and $(u_{i'}, v_{j'})$, and the matching $\mathcal{E}_{m'}$ contains the edge $(u_i, v_{j'})$.

Next we prove the feasibility of H. That is, we will show that, for any terminal t_m, there exist k edge-disjoint paths from r to t_m. We will construct a set of such paths, namely P. Note that every arc entering t_m must be contained in a distinct path in P because $\text{indeg}_H(t_m) = k$. This gives four types of paths.

- (r, t_m): The arc itself forms a path from r to t_m.
- (u_i^a, t_m): We choose one of the zero-cost arcs (r, u_i^a) to combine with (u_i^a, t_m) to constitute a path $r \to u_i^a \to t_m$. Since $\mathcal{E}_m$ is a matching, by construction we know that the arc (u_i^a, t_m) has multiplicity one in G and thus the paths in this category are edge-disjoint. After selecting paths in this way, there are still $\deg_{\mathcal{G}}(u_i) - 1$ unoccupied copies of the zero-cost arc (r, u_i^a) if $u_i \in V(\mathcal{E}_m)$; or $\deg_{\mathcal{G}}(u_i)$ unoccupied copies otherwise.
- (w_{ij}^{ab}, t_m): We choose one of the zero-cost arcs (r, u_i^a) and the arc (u_i^a, w_{ij}^{ab}) to constitute a path $r \to u_i^a \to w_{ij}^{ab} \to t_m$. If $u_i \in V(\mathcal{E}_m)$, such paths use $\deg_{\mathcal{G}}(u_i) - 1$ copies of the zero-cost arc (r, u_i^a), otherwise $\deg_{\mathcal{G}}(u_i)$ copies are used. In both cases it is valid to do so because the first two categories of path leave enough copies. It is clear that the paths up to this point are edge-disjoint.
- (v_j, t_m): In this case there is a unique $u_i \in \mathcal{U}$ such that $(u_i, v_j) \in \mathcal{E}_m$. It holds that $\pi_{u_i v_j}(a) = b$ for some $a \in \sigma(u_i), b \in \sigma(v_j)$ since σ covers the edge (u_i, v_j). We choose the one-cost arcs $(r, u_i^a) \in F$ and $(v_j^b, v_j) \in F$ to constitute a path $r \to u_i^a \to w_{ij}^{ab} \to v_j^b \to v_j \to t_m$. Since $\mathcal{E}_m$ is a matching, the paths here are edge-disjoint. Note that previous paths only use arcs added in the final construction except for arcs of the form (u_i^a, w_{ij}^{ab}), while here we only use arcs from the base construction. The construction of G guarantees (u_i^a, w_{ij}^{ab}) is not used by previous paths if $(u_i, v_j) \in \mathcal{E}_m$.

Therefore, we selected k edge-disjoint paths in H from r to t_m successfully.

Soundness

Given a k-connected subgraph $H = (V, F)$ (that contains all zero-cost arcs) of the k-DST instance (k, G, r, T), we show that there is a corresponding feasible multilabeling σ of the label cover instance $(\mathcal{G}, \Sigma, \pi)$ such that $c(\sigma) = c(H)$. The multilabeling σ is specified by checking the one-cost arcs in H, i.e., set $\sigma(u_i)$ as $\{a \in \Sigma : \text{the one-cost arc } (r, u_i^a) \text{ is in } F\}$ for $u_i \in \mathcal{U}$ and set $\sigma(v_j)$ as $\left\{b \in \Sigma : (v_j^b, v_j) \in F\right\}$ for $v_j \in \mathcal{V}$. Clearly, $c(\sigma) = c(H)$ and the definition of σ induces an injective mapping.

We prove that there exist $a_i \in \sigma(u_i)$ and $b_j \in \sigma(v_j)$ such that $\pi_{u_iv_j}(a_i) = b_j$ for each edge $(u_i, v_j) \in \mathcal{E}$. Recall that $\mathcal{E}$ is partitioned into Δ matchings $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_\Delta$. Let us fix an arbitrary matching $\mathcal{E}_m$ and discuss the edges inside $\mathcal{E}_m$. Consider the set S of arcs in G coming into the terminal t_m. By our construction, S contains k arcs of the following categories:

1. $|\mathcal{E}_m|$ arcs of type (v_j, t_m);
2. $|\Sigma| \cdot (|\mathcal{E}| - |\mathcal{E}_m|)$ arcs of type $(w^{a'b'}_{i'j'}, t_m)$ (one for each $(u_{i'}, w_{j'}) \notin \mathcal{E}_m$ and $a', b' \in \Sigma$ such that $\pi_{u_{i'}v_{j'}}(a') = b'$);
3. $|\Sigma| \cdot |\mathcal{E}_m|$ arcs of type (u^a_i, t_m) (one for each $(u_i, v_j) \in \mathcal{E}_m$ and $a \in \Sigma$);
4. $k - |\mathcal{E}_m| - |\Sigma| \cdot |\mathcal{E}|$ arcs of type (r, t_m).

Let P be the subgraph formed by the k edge-disjoint paths from r to t_m in H. The connectivity requirement forces that each arc in S must belong to some path in P, so we can also categorize the paths in P into the four types above. Let $P_{2\text{-}4}$ be the subgraph consisting of all type-2,3,4 paths, and P_i similarly. We prove two observations:

▷ Claim 10. We have the following facts for paths.
I: $\forall (u_{i'}, v_{j'}) \notin \mathcal{E}_m$, $\forall a', b' \in \Sigma$ such that $\pi_{u_{i'}v_{j'}}(a') = b'$, $(u^{a'}_{i'}, w^{a'b'}_{i'j'}) \in P_2 \notin P_1$.
II: $\forall (u_i, v_j) \in \mathcal{E}_m$ and $a, b \in \Sigma$ such that $\pi_{u_iv_j}(a) = b$, there are $\deg_{\mathcal{G}}(u_i)$ arcs (r, u^a_i) in $P_{2\text{-}4}$.

Proof. Note that we need a type-2 path from all $w^{a'b'}_{i'j'}$ such that $\pi_{u_{i'}v_{j'}}(a') = b'$, and using the arc $(u^{a'}_{i'}, w^{a'b'}_{i'j'})$ is the only way to enter $w^{a'b'}_{i'j'}$ to form a type-2 path. So $(u^{a'}_{i'}, w^{a'b'}_{i'j'})$ belongs to P_2 and not in P_1 (edge disjoint with P_2).

For the second claim, there are $\deg_{\mathcal{G}}(u_i) - 1$ edges $(u_i, v_{j'}) \in \mathcal{E} \setminus \mathcal{E}_m$. Plugging in Claim-I implies that there are $\deg_{\mathcal{G}}(u_i) - 1$ type-2 paths that use u^a_i. Moreover, because $(u_i, v_j) \in \mathcal{E}_m$, there is another type-3 path that use u^a_i. So, in total there are $\deg_{\mathcal{G}}(u_i)$ paths in $P_{2\text{-}4}$ that use u^a_i, and Claim-II follows. ◁

Then, fix an arbitrary $(u_i, v_j) \in \mathcal{E}_m$, we claim that the type-1 paths P_1 induce $a, b \in \Sigma$ such that $\pi_{u_iv_j}(a) = b$. Let p be the type-1 path that goes through v_j. For the path p to enter v_j, it must go through a one-cost arc (v^b_j, v_j) for some $b \in \Sigma$, and thus $b \in \sigma(v_j)$. Then, there are two ways to enter v^b_j:

1. from $w^{a'b}_{i'j}$ for some $(u_{i'}, v_j) \notin \mathcal{E}_m$ and $a' \in \pi^{-1}_{u_{i'}v_j}(b)$;
2. from w^{ab}_{ij} for $(u_i, v_j) \in \mathcal{E}_m$ and some $a \in \pi^{-1}_{u_iv_j}(b)$.

The first way is infeasible because $(u^{a'}_{i'}, w^{a'b}_{i'j}) \notin P_1$ due to Claim-I. Hence, the only way is the second and p must be exactly $r \to u^a_i \to w^{ab}_{ij} \to v^b_j \to v_j \to t_m$. Putting together Claim-II and the edge-disjointness of P_1 and $P_{2\text{-}4}$, there are $\deg_{\mathcal{G}}(u_i) + 1$ arcs (r, u^a_i) in total in P. Thus the one-cost arc (r, u^a_i) must be included in $P \subseteq H$ because there are totally $\deg_{\mathcal{G}}(u_i) + 1$ arcs (r, u^a_i) in G and thus $a \in \sigma(u_i)$. Therefore, we conclude that the arbitrarily fixed edge (u_i, v_j) is covered by $\pi_{u_iv_j}(a) = b$, where $a \in \sigma(u_i)$ and $b \in \sigma(v_j)$.

Hardness Gap

The one-to-one correspondence between solutions to the two problems is established by collecting the proofs for completeness and soundness. Furthermore, the reduction can be done in polynomial time in the size of the label cover instance and it guarantees that $|T| = \Delta$. Plugging in Corollary 8, the following inapproximability result for the k-DST problem is obtained.

▶ **Theorem 1.** *For $k > |T|$, unless* NP = ZPP*, it is hard to approximate the k-DST problem to within a factor of $\Omega\left(|T|/\log|T|\right)$.*

5 Inapproximability in Terms of the Connectivity Requirement

This section presents a hardness reduction, which is tailored for the approximation hardness in terms of the connectivity requirement k. Our reduction again takes a label cover instance $(\mathcal{G}, \Sigma, \pi)$ as an input and produces a k-DST instance $(k, G = (V, E), r, T)$. As we wish to obtain an inapproximability in terms of k, them main focus is in controlling the size of k.

Base Construction

Our reduction starts from a basic building block.

1. Let G be an empty graph. For each vertex $u_i \in \mathcal{U}$, create in G a counterpart of it (also named u_i) and a set of vertices $A_i = \{u_i^a : a \in \Sigma\}$; connect u_i to each $u_i^a \in A_i$ by an arc (u_i, u_i^a) of cost one.
2. For each vertex $v_j \in \mathcal{V}$, create in G a counterpart of it (also named v_j) and a set of vertices $B_j = \{v_j^b : b \in \Sigma\}$; connect each $v_j^b \in B_j$ to v_j by an arc (v_j^b, v_j) of cost one.
3. For each edge $(u_i, v_j) \in \mathcal{E}$, add to G a terminal t_{ij} and connect v_j to t_{ij} by a zero-cost arc (v_j, t_{ij}). For $a, b \in \Sigma$, connect $u_i^a \in A_i$ to $v_j^b \in B_j$ by a zero-cost arc if $\pi_{u_i v_j}(a) = b$.

Gadget of d-ary Arborescence

All arcs created hereafter have zero cost. By Lemma 9, $\mathcal{E}$ is partitioned into $\delta \leq 2\Delta^2$ induced matchings $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_\delta$. Let $d \geq 2$ be an integral parameter to be determined later. We add to G a complete d-ary out-rooted tree (arborescence) Q of height $h = \lceil \log_d \delta \rceil$ with a rooted vertex r; we set r as the root of the k-DST instance G. Choose an arbitrary order for the vertices in each layer of Q and use q_i^j to denote the j-th vertex of at the i-th layer. Note that q_0^1 is the root r. We then join each leaf q_h^j $(1 \leq j \leq \delta)$ of Q to the counterpart in G of each vertex $u_i \in V(\mathcal{E}_j) \cap \mathcal{U}$. See Figure 8 and Figure 9 for an illustration. Intuitively, if we add a padding path that passes q_i^j, all illegal paths that go through the subtree rooted at q_i^j will get killed.

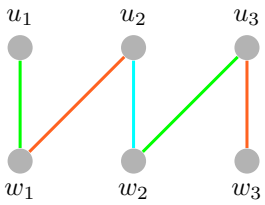


Figure 8 A bipartite graph $\mathcal{G}$ with three induced matchings: $\mathcal{E}_1 = \{(u_1, w_1), (u_3, w_2)\}$, $\mathcal{E}_2 = \{(u_2, w_1), (u_3, w_3)\}$, $\mathcal{E}_3 = \{(u_2, w_2)\}$.

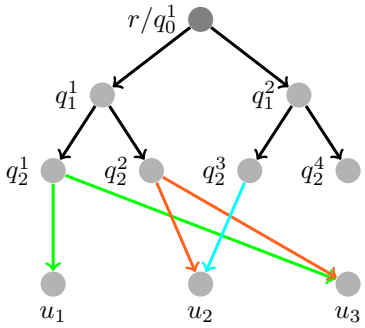


Figure 9 The gadget when $d = 2$ with induced matchings $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$.

Final Construction

In the final construction, we add some *padding arcs* to the based graph. These arcs form *padding paths*, which then kill all the *illegal paths* through the help of the d-ary arborescence.

1. For each arc $(q_i^j, q_{i+1}^{j'})$ of Q, we replace it by two arcs $(q_i^j, w_i^{jj'})$ and $(w_i^{jj'}, q_{i+1}^{j'})$.

We still use Q to denote the original arborescence for notational convenience.

2. For each $1 \le i \le h-1$ and j, we add $d-1$ copies of an arc (r, q_i^j) to G.
3. (Refer to Figure 10) For each terminal t_{ij}, suppose the edge (u_i, v_j) is in the group $\mathcal{E}_m$ (corresponding to q_h^m). There is a unique path in the d-ary arborescence from r to q_h^m: $(r = q_0^{j_0=1}) \to q_1^{j_1} \to q_2^{j_2} \to \cdots \to q_h^{j_h=m}$. At each level $1 \le \ell \le h$, there are $d-1$ siblings of $q_\ell^{j_\ell}$, and for each sibling $q_\ell^{j \neq j_\ell}$, we add a zero-cost arc from $w_{\ell-1}^{j_{\ell-1} j}$ to the terminal t_{ij}.

After the construction, the in-degree of each terminal t_{ij} is $h(d-1)+1$ and we set k as it.

Why Does it Work under Grouping by Induced Matchings?

For any edge $(u_i, v_j) \in \mathcal{E}_m$, now we can promise that there must be a path from r to t_{ij} going through the leaf q_h^m and the vertex $u_i \in V(G)$. Let us examine if illegal paths are bypassing either q_h^m or u_i. The padding arcs can kill illegal paths from r to t_{ij} that depart from the unique path $r \to q_1^{j_1} \to q_2^{j_2} \to \cdots \to q_h^{j_h=m} \to u_i$ at some vertex $q_\ell^{h_\ell}$ for $\ell < h$. It remains to check if there is an illegal path of the form $r \to q_1^{j_1} \to q_2^{j_2} \to \cdots \to q_h^{j_h=m} \to u_{i'} \to \cdots \to v_j \to t_{ij}$ for some $u_{i'} \neq u_i$. The existence of such $u_{i'}$ implies that $(u_{i'}, v_j) \in \mathcal{E}_m$, which contradicts that $\mathcal{E}_m$ is an induced matching.

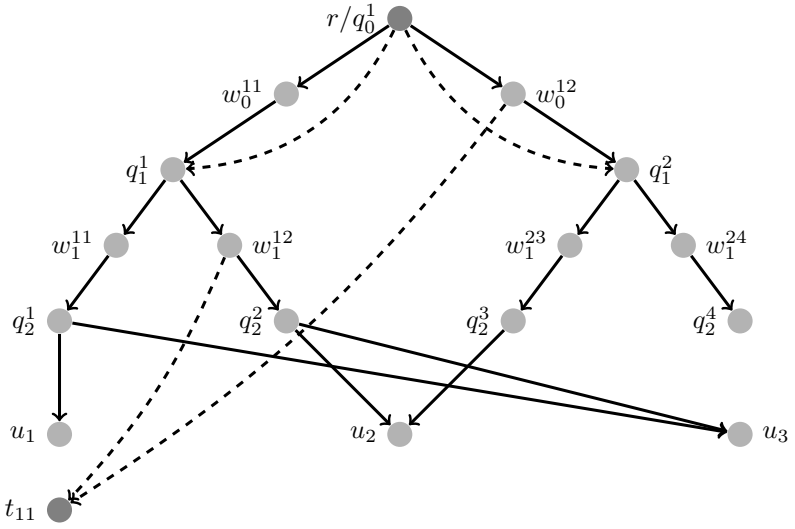


Figure 10 Padding arcs for the terminal t_{11} are dashed. Note that the vertices u_i have outgoing arcs, but not drawn here.

Completeness

Given a feasible multilabeling σ of the instance $(\mathcal{G}, \Sigma, \pi)$, we show that there is a corresponding k-connected subgraph $H = (V, F)$ of $G = (V, E)$ such that $c(\sigma) = c(H)$. The set F consists of three types of arcs: 1) all zero-cost arcs in G; 2) the one-cost arcs (u_i, u_i^a) for each $u_i \in \mathcal{U}$ and $a \in \sigma(u_i)$; 3) the one-cost arcs (v_j^b, v_j) for each $v_j \in \mathcal{V}$ and $b \in \sigma(v_j)$. Clearly, $c(H) = c(\sigma)$ and the definition of F induces an injective mapping.

Now we prove that for any terminal $t_{ij} \in T$ there are k edge-disjoint paths, denoted by P, in H from r to t_{ij}. Let $\mathcal{E}_m$ be the induced matching that contains t_{ij}. There is a unique path $p = (r = q_0^{j_0=1}) \to q_1^{j_1} \to \cdots \to q_h^{j_h=m}$ in the arborescence Q from r to q_h^m. Since $\text{indeg}_H(t_{ij}) = k$, each arc entering t_{ij} must be contained in P. We consider two cases:

1. (v_j, t_{ij}): The feasibility of σ induces $a \in \sigma(u_i)$ and $b \in \sigma(v_j)$ such that $\pi_{u_i v_j}(a) = b$, so that $(u_i, u_i^a), (u_i^a, v_j^b), (v_j^b, v_j) \in F$. Guided by the path p, we add to P the path $r \to w_0^{j_0 j_1} \to q_1^{j_1} \to w_1^{j_1 j_2} \to \cdots \to w_{h-1}^{j_{h-1} j_h = m} \to q_h^{j_h = m} \to u_i \to u_i^a \to v_j^b \to v_j \to t_{ij}$.
2. $(w_\ell^{j_\ell j'}, t_{ij})$: Here $j' \neq j_{\ell+1}$. We add to P the path $r \to q_\ell^{j_\ell} \to w_\ell^{j_\ell j'} \to t_{ij}$.

For $0 \leq \ell < h$, the type-1 path goes through $q_\ell^{j_\ell} \to w_\ell^{j_\ell j_{\ell+1}} \to q_{\ell+1}^{j_{\ell+1}}$, while each type-2 path goes through $r \to q_\ell^{j_\ell} \to w_\ell^{j_\ell j'} \to t_{ij}$ for some j'. Since $j' \neq j_{\ell+1}$, the type-1 path and the type-2 paths do not share any common arc. For the type-2 paths themselves, for each $0 \leq \ell < h$ they consume in total $d-1$ duplicates of the arc $(r, q_\ell^{j_\ell})$, which is prepared well in the second step of the final construction of G.

Soundness

Given a k-connected subgraph $H = (V, F)$ (that contains all zero-cost arcs) of the k-DST instance (k, G, r, T), we show that there is a corresponding feasible multilabeling σ of the label cover instance $(\mathcal{G}, \Sigma, \pi)$ such that $c(\sigma) = c(H)$. We define σ as follows: set $\sigma(u_i)$ as $\{a \in \Sigma : (u_i, u_i^a) \in F\}$ for $u_i \in \mathcal{U}$ and set $\sigma(v_j)$ as $\left\{b \in \Sigma : (v_j^b, v_j) \in F\right\}$ for $v_j \in \mathcal{V}$. Clearly, $c(\sigma) = c(H)$ and the definition of σ induces an injective mapping. Then we prove that σ covers all the edges in $\mathcal{E}$.

Consider an edge $(u_i, v_j) \in \mathcal{E}_m$ and its corresponding terminal t_{ij}. Let $(r = q_0^{j_0=1}) \to q_1^{j_1} \to \cdots \to q_h^{j_h=m}$ be the unique path in the arborescence Q from r to q_h^m. Let P be any set of k edge-disjoint paths from r to t_{ij} in H. The fact that $\text{indeg}_G(t_{ij}) = k = h(d-1)+1$ forces P to contain all arcs entering t_{ij}. These arcs are of two types:

1. One arc (v_j, t_{ij});
2. $h(d-1)$ arcs of $(w_\ell^{j_\ell j'}, t_{ij})$ for $0 \leq \ell < h$ and $j' \neq j_{\ell+1}$.

Let P_1 be the only path in P that uses the type-1 arc, and let P_2 be the union of paths in P that use type-2 arcs. By backtracking the paths in P_2 from t_{ij} for two steps, it holds, for $0 \leq \ell < h$ and $j' \neq j_{\ell+1}$, that $(q_\ell^{j_\ell}, w_\ell^{j_\ell j'}) \in P_2$. Thus, the path P_1 has to be $q_0^{j_0=1} \to w_0^{j_0 j_1} \to q_1^{j_1} \to \cdots \to q_h^m \to \cdots \to v_j \to t_{ij}$. Let us backtrack P_1 from v_j. The previous vertex must be v_j^b for some $b \in \Sigma$, and $u_{i'}^a$ for some $a \in \Sigma$, and $u_{i'}$, and then q_h^m. If $i' \neq i$, then by the construction of G, we know that $u_{i'} \in V(\mathcal{E}_m)$. However, it also holds that $v_j \in V(\mathcal{E}_m)$ because $(u_i, v_j) \in \mathcal{E}_m$. Thus, the induced subgraph on $V(\mathcal{E}_m)$ contains both (u_i, v_j) and $(u_{i'}, v_j)$, contradicting the matching property of $\mathcal{E}_m$. Therefore, $i' = i$, implying that $a \in \sigma(u_i), b \in \sigma(v_j)$ and $\pi_{u_i v_j}(a) = b$ because $(u_i, u_i^a), (u_i^a, v_j^b), (v_j^b, v_j) \in P_1 \subseteq P \subseteq H$.

Hardness Gap

In this subsection, we deduce approximation hardness for k-DST. Clearly, the reduction can be made in polynomial time in the size of the input label cover instance. By setting different values of the parameter d, we have the following two propositions.

▶ **Theorem 3.** *It is hard to approximate the k-DST problem on L-layered graphs $G = (V, E)$ for $\Omega(1) \leq L \leq O(\log |V|)$ to within a factor of $\Omega\left((k/L)^{(1-\epsilon)L/4-2}\right)$ for any constant $\epsilon > 0$, unless* NP = ZPP.

Proof. Let L be the height (the maximum length of paths) of the underlying graph in the k-DST instance. Recall that in the construction we have a modified d-ary arborescence and 5 base levels with vertices u_i, u_i^a, v_j^b, v_j and t_{ij}. So, $L = 2\lceil \log_d \delta \rceil + 5$. By Lemma 9, δ is at most $2\Delta^2$. Thus, $L \leq 2\lceil \log_d(2\Delta^2) \rceil + 5$. To complete the proof, we add some dummy vertices to the modified d-ary arborescence so that the height becomes exactly $2\lceil \log_d(2\Delta^2) \rceil + 5$.

We fix $d = \Delta^x \geq 2$ for some $0 < x \leq 1$. Then $k = \lceil \log_d \delta \rceil (d-1) + 1 \leq (d-1)(2\log\Delta/\log d + 1/\log d + 1) + 1 \leq (2/x + 2)\,\Delta^x$. We also have that $4/x \leq L = 2\lceil \log_d(2\Delta^2) \rceil + 5 \leq 4/x + 8$. Therefore, $(k/L)^{\frac{(1-\epsilon)L}{4} - 2} \leq \Delta^{1-\epsilon}$, and we have the claimed result on layered graphs by plugging in Corollary 8. The parameter x can be used to obtain specific values of L that we want. ◀

▶ **Theorem 4.** *For $k < |T|$, it is hard to approximate the k-DST problem to within a factor of $\Omega\left(2^{k/2}/k\right)$, unless* NP = ZPP*.*

Proof. Recall that $k = h(d-1) + 1 = \lceil \log_d \delta \rceil (d-1) + 1$ where $\delta \leq 2\Delta^2$. If we set $d = 2$, then $k = \lceil \log \delta \rceil + 1 \leq 2\log\Delta + 3$. If k is too small, then we can add dummy arcs from r to terminals to make $k \geq \log\Delta$. Plugging in Corollary 8, the claimed hardness gap follows. ◀

6 Discussion

In this paper we obtain improved inapproximability results for the k-DST problem in terms of parameters k and $|T|$. All of the results are derived from the same hardness source, the (minimum) label cover problem, which admits no $\Delta/\log\Delta$-approximation algorithm under standard complexity assumptions, where Δ is the maximum degree of the underlying bipartite graph in the label cover problem. The label cover problem also admits no $2^{\log^{1-\varepsilon} N}$-approximation algorithm for any constant $\varepsilon > 0$, where N is the underlying graph size. Note that $2^{\log^{1-\varepsilon} N} = N^{o(1)}$. If we apply the reduction in the $\Omega(|T|/\log|T|)$-hardness, then it transfers to $k^{o(1)}$ which is even worse than the previous $\Omega(k/\log k)$-hardness, not to say the factor of $\Omega(2^{k/2}/k)$ in this paper. If we apply the reduction in the $\Omega(2^{k/2}/k)$-hardness, then it transfers to $|T|^{o(1)}$ which is still much worse than $|T|^{1/4-\varepsilon}$ and $\Omega(|T|/\log|T|)$.

References

1. Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548477.
2. Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1), February 2013. doi:10.1145/2432622.2432628.
3. Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.8.
4. Parinya Chalermsook, Syamantak Das, Bundit Laekhanukit, and Daniel Vaz. Beyond metric embedding: Approximating group steiner trees on bounded treewidth graphs. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 737–751. SIAM, 2017. doi:10.1137/1.9781611974782.47.
5. Parinya Chalermsook, Fabrizio Grandoni, and Bundit Laekhanukit. On survivable set connectivity. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 25–36. SIAM, 2014.
6. Parinya Chalermsook, Fabrizio Grandoni, and Bundit Laekhanukit. On survivable set connectivity. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 25–36. SIAM, 2015. doi:10.1137/1.9781611973730.3.

7 Chun-Hsiang Chan, Bundit Laekhanukit, Hao-Ting Wei, and Yuhao Zhang. Polylogarithmic approximation algorithm for k-connected directed steiner tree on quasi-bipartite graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176, pages 63:1–63:20. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

8 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.

9 Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: Deterministic approximation algorithms for group steiner trees and k-median. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 114–123. ACM, 1998. doi:10.1145/276698.276719.

10 Chandra Chekuri, Guy Even, and Guy Kortsarz. A greedy approximation algorithm for the group steiner problem. *Discret. Appl. Math.*, 154(1):15–34, 2006. doi:10.1016/j.dam.2005.07.010.

11 Joseph Cheriyan, Bundit Laekhanukit, Guyslain Naves, and Adrian Vetta. Approximating rooted steiner networks. *ACM Transactions on Algorithms (TALG)*, 11(2):1–22, 2014.

12 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

13 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. doi:10.1016/j.jcss.2004.04.011.

14 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

15 Lisa Fleischer, Kamal Jain, and David P. Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006. Special Issue on FOCS 2001.

16 András Frank. Rooted k-connections in digraphs. *Discret. Appl. Math.*, 157(6):1242–1254, 2009. doi:10.1016/j.dam.2008.03.040.

17 András Frank and Éva Tardos. Generalized polymatroids and submodular flows. *Math. Program.*, 42(1-3):489–563, 1988. doi:10.1007/BF01589418.

18 Zachary Friggstad, Jochen Könemann, and Shadravan Mohammad. A Logarithmic Integrality Gap Bound for Directed Steiner Tree in Quasi-bipartite Graphs . In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

19 Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000. doi:10.1006/jagm.2000.1096.

20 Rohan Ghuge and Viswanath Nagarajan. Quasi-polynomial algorithms for submodular tree orienteering and other directed network design problems. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1039–1048. SIAM, 2020. doi:10.1137/1.9781611975994.63.

21 Michel X Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1161–1176. SIAM, 2012.

22 Fabrizio Grandoni and Bundit Laekhanukit. Surviving in directed graphs: a quasi-polynomial-time polylogarithmic approximation for two-connected directed steiner tree. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 420–428, 2017.

23 Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. O $(\log^2 k/\log\log k)$-approximation algorithm for directed Steiner tree: a tight quasi-polynomial-time algorithm. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–264, 2019.

24 Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Tree embeddings for two-edge-connected network design. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1521–1538. SIAM, 2010. doi:10.1137/1.9781611973075.124.

25 Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Aravind Srinivasan, and Nan Wang. Integrality ratio for group steiner trees and directed steiner trees. *SIAM J. Comput.*, 36(5):1494–1511, 2007. doi:10.1137/S0097539704445718.

26 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 585–594. ACM, 2003. doi:10.1145/780542.780628.

27 Tomoya Hibi and Toshihiro Fujito. Multi-rooted greedy approximation of directed steiner trees with applications. *Algorithmica*, 74(2):778–786, 2016.

28 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

29 Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. Approximating fault-tolerant group-steiner problems. *Theor. Comput. Sci.*, 416:55–64, 2012. doi:10.1016/j.tcs.2011.08.021.

30 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994. doi:10.1145/174652.174654.

31 Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM J. Comput.*, 33(3):704–720, 2004. doi:10.1137/S0097539702416736.

32 Bundit Laekhanukit. Parameters of two-prover-one-round game and the hardness of connectivity problems. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1626–1643. SIAM, 2014.

33 Bundit Laekhanukit. Approximating directed steiner problems via tree embedding. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

34 Pasin Manurangsi. A note on degree vs gap of min-rep label cover and improved inapproximability for connectivity problems. *Information Processing Letters*, 145:24–29, 2019.

35 Pasin Manurangsi, Aviad Rubinstein, and Tselil Schramm. The strongish planted clique hypothesis and its consequences. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 10:1–10:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.10.

36 Zeev Nutov. Approximating minimum-cost connectivity problems via uncrossable bifamilies. *ACM Transactions on Algorithms (TALG)*, 9(1):1–16, 2012.

37 Zeev Nutov. On rooted k-connectivity problems in quasi-bipartite digraphs. In *International Computer Science Symposium in Russia*, pages 339–348. Springer, 2021.

38 Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.

On Lower Bounds of Approximating Parameterized k-Clique

Bingkai Lin
Nanjing University, China

Xuandi Ren
Peking University, Beijing, China

Yican Sun
Peking University, Beijing, China

Xiuhan Wang
Tsinghua University, Beijing, China

Abstract

Given a simple graph G and an integer k, the goal of the k-Clique problem is to decide if G contains a complete subgraph of size k. We say an algorithm approximates k-Clique within a factor $g(k)$ if it can find a clique of size at least $k/g(k)$ when G is guaranteed to have a k-clique. Recently, it was shown that approximating k-Clique within a constant factor is W[1]-hard [20].

We study the approximation of k-Clique under the *Exponential Time Hypothesis* (ETH). The reduction of [20] already implies an $n^{\Omega(\sqrt[6]{\log k})}$-time lower bound under ETH. We improve this lower bound to $n^{\Omega(\log k)}$. Using the gap-amplification technique by expander graphs, we also prove that there is no $k^{o(1)}$ factor FPT-approximation algorithm for k-Clique under ETH.

We also suggest a new way to prove the *Parameterized Inapproximability Hypothesis* (PIH) under ETH. We show that if there is no $n^{O(\frac{k}{\log k})}$-time algorithm to approximate k-Clique within a constant factor, then PIH is true.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases parameterized complexity, k-clique, hardness of approximation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.90

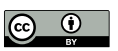
Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2111.14033

1 Introduction

In this paper, we study the k-Clique problem: given a simple graph G and an integer k, decide whether G contains a complete subgraph of size k. As shown in [18], k-Clique is one of the most classical NP-complete problems. Its inapproximability in the classical complexity regime has also been studied extensively [10, 4, 5, 14, 11, 15, 28]. Along a long line of research, it was proved that even approximating Clique into a ratio of $n^{1-\varepsilon}$ is NP-hard.

In recent years, the hardness of approximating k-Clique has received increased attention in the parameterized complexity regime. When guaranteed that the maximum clique is of size k, people wonder if there is an algorithm which runs in $f(k)n^{O(1)}$ time, and can find a clique of size at least $k/g(k)$, for some computable functions f and g. Such an algorithm is called a $g(k)$-*FPT-approximation* for the k-Clique problem.



49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 90; pp. 90:1–90:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Previously, [6] ruled out all $g(k)$-FPT-approximation algorithms of k-Clique for any $g(k) = o(k)$ under the *Gap Exponential Time Hypothesis* (Gap-ETH) [1]. They even showed that assuming Gap-ETH, it is impossible to find a clique of size $\varepsilon(k)$ in $f(k)n^{o(\varepsilon(k))}$ time. However, as Gap-ETH is such a strong hypothesis that it already gives a gap in hardness of approximation, it is still of great interest to prove the same lower bound under an assumption without an inherent gap. People may further wonder:

Assuming ETH*, does finding a clique of size* $\varepsilon(k)$ *in* k-Clique *require* $f(k)n^{\Omega(\varepsilon(k))}$ *time?*

In a recent work [20], Lin showed that k-Clique does not admit constant factor FPT-approximation algorithms unless W[1] = FPT. This was the first successful attempt to bypass Gap-ETH to prove the hardness of approximating k-Clique. Unfortunately, [20] reduces a k-Clique instance to a constant gap k'-Clique[2] instance with $k' = 2^{k^6}$. As there is no $f(k)n^{o(k)}$ time algorithm for k-Clique assuming ETH, [20] actually ruled out $f(k)n^{o(\sqrt[6]{\log k})}$ time constant approximation algorithms for k-Clique under ETH. Comparing to [6], such a lower bound is still far beyond satisfaction, and it remains open to avoid the huge parameter blow-up in the gap-producing reduction to obtain a better lower bound.

The main result of this paper is

▶ **Theorem 1.** *Assuming* ETH*, for any constant* $c > 1$ *and any computable function* f*, no algorithm can find a clique of size* k/c *in the* k-Clique *problem in* $f(k)n^{o(\log k)}$ *time.*

As an application, we combine our main result with the classical gap-amplification technique to rule out any $k^{o(1)}$-ratio FPT-approximation algorithms for k-Clique under ETH. Let us not fail to mention that [24] recently proved similar lower bound based on a weaker hypothesis W[1] $\neq$ FPT. Our result is formally stated as follows.

▶ **Corollary 2.** *Assuming* ETH*, for any* $g(k) = k^{o(1)}$*, the* k-Clique *problem has no* $g(k)$*-FPT-approximation algorithm.*

We also study the relationship between the constant gap k-Clique problem and the *parameterized inapproximablity hypothesis* (PIH) [21], a central conjecture in parameterized complexity. Roughly speaking, PIH states that it is impossible to approximate a 2-CSP instance over k variables with alphabet $[n]$ to a constant factor in FPT time. It is known in [10] that if PIH is true, then there is no FPT algorithm for constant gap k-Clique. However, the reverse direction is not known yet. Furthermore, although PIH can be deduced from Gap-ETH via standard reductions in [7, 8], proving PIH under gap-free hypotheses (e.g. ETH, W[1] $\neq$ FPT) is still quite open and is believed to require a PCP-like theorem in parameterized complexity. We show that an almost-tight running time lower bound of constant gap k-Clique could imply PIH. Our theorem suggests a new way to prove PIH under ETH, namely, by using constant gap k-Clique as an intermediate problem. It is formally stated as follows.

▶ **Theorem 3.** *If there is no* $f(k)n^{O(\frac{k}{\log k})}$ *time algorithm for constant gap* k-Clique*, then* PIH *is true.*

---

[1] Gap-ETH states that no subexponential time algorithm can distinguish whether a 3SAT formula is satisfiable or every assignment satisfies at most $1 - \varepsilon$ fraction of clauses for some $\varepsilon > 0$.

[2] Given $k \in \mathbb{N}$, $\varepsilon \in (0, 1)$ and a simple graph G, the constant gap k-Clique problem is to decide whether G contains a K_k subgraph or G contains no $K_{\varepsilon k}$ subgraph.

1.1 Our Techniques

From 3SAT **to gap** k-Clique**.** Recall that the reduction in [20] consists of two steps. First, it reduces k-Clique to k^2-VectorSum, while introducing a quadratic blow-up of the parameter. Next, it transforms k-VectorSum to CSP on $k' = 2^{O(k^3)}$ variables $\{x_{\vec{a}_1,\ldots,\vec{a}_k} : \vec{a}_1,\ldots,\vec{a}_k \in \mathbb{F}^k\}$, and then to constant gap k'-Clique. The two steps together cause the parameter to grow from k to $2^{O(k^6)}$.

To give a tighter lower bound of constant gap k-Clique under ETH, we deal with the above two steps separately. First, we show a reduction directly from 3SAT to k-VectorSum, resulting in a tighter lower bound of k-VectorSum under ETH. Then, we give a more succinct reduction from k-VectorSum to CSP on $k' = 2^{O(k)}$ variables $\{x_{a_1,\ldots,a_k} : a_1,\ldots,a_k \in \mathbb{F}\}$, and then to constant gap k'-Clique. In our new reduction, the parameter blow-up throughout is only $2^{O(k)}$, leading to an $n^{\Omega(\log k)}$ lower bound for constant gap k-Clique.

Since the second step is more complicated, we will briefly introduce the ideas here. Given an k-VectorSum instance $(V_1,\ldots,V_k,\vec{t})$, we build a CSP instance on variable set $X = \{x_{a_1,\ldots,a_k} : a_1,\ldots,a_k \in \mathbb{F}\}$. Each variable takes value in $\mathbb{F}^m$ where m is the dimension specified by the k-VectorSum problem. In the yes-case, let $\vec{v}_1 \in V_1,\ldots,\vec{v}_k \in V_k$ be a solution that sum up to $\vec{t}$, we expect $x_{a_1,\ldots,a_k}$ to take the value $\sum_{i\in[k]} a_i\vec{v}_i$. Similar to [20], we want to make the following three types of tests:

- $\forall(a_1,\ldots,a_k),(b_1,\ldots,b_k) \in \mathbb{F}^k$, test whether $x_{a_1,\ldots,a_k} + x_{b_1,\ldots,b_k} = x_{a_1+b_1,\ldots,a_k+b_k}$.
- $\forall(a_1,\ldots,a_k) \in \mathbb{F}^k, a \in \mathbb{F}$, test whether $x_{a_1,\ldots,a_i+a,\ldots,a_k} - x_{a_1,\ldots,a_k} \in aV_i$.
- $\forall(a_1,\ldots,a_k) \in \mathbb{F}^k, a \in \mathbb{F}$, test whether $x_{a_1+a,\ldots,a_k+a} - x_{a_1,\ldots,a_k} = a\vec{t}$.

If an assignment passes most of the linearity tests, then there must be vectors $\vec{u}_1,\ldots,\vec{u}_k \in \mathbb{F}^m$ such that $x_{a_1,\ldots,a_k} = \sum_{i\in[k]} a_i\vec{u}_i$ for most $(a_1,\ldots,a_k) \in \mathbb{F}^k$. The second step is meant to guarantee that the selected vectors indeed come from the input. Finally we need the third step to check whether they sum up to $\vec{t}$.

Note that in our reduction from 3SAT to k-VectorSum, we require the dimension m to be at least $\Omega(k\log n)$. Thus in the CSP instance, we cannot simply leave the alphabet to be $\mathbb{F}^m = n^{\Omega(k)}$, which is too large. To reduce the dimension, we pick $\ell = \Theta(k + \log n)$ matrices $A_1,\ldots,A_\ell \in \mathbb{F}^{k\times m}$ independently at random, and define a new CSP problem on variable set $Y = \{y_{\vec{\alpha},\vec{\beta}} : \vec{\alpha},\vec{\beta} \in \mathbb{F}^k\}$, where each $y_{\vec{\alpha},\vec{\beta}}$ is supposed to take the value

$$\begin{aligned} y_{\vec{\alpha},\vec{\beta}} &= (\vec{\alpha}A_1x_{\vec{\beta}},\ldots,\vec{\alpha}A_\ell x_{\vec{\beta}}) \\ &= (\vec{\alpha}A_1\sum_{i\in[k]}\beta_i\vec{v}_i,\ldots,\vec{\alpha}A_\ell\sum_{i\in[k]}\beta_i\vec{v}_i) \\ &= \sum_{i\in[k],j\in[k]}\beta_i\alpha_j(A_1[j]\vec{v}_i,\ldots,A_\ell[j]\vec{v}_i) \\ &\triangleq \sum_{i\in[k],j\in[k]}\beta_i\alpha_jC_{i,j}. \end{aligned} \tag{1}$$

Now the alphabet size is only $\mathbb{F}^\ell = 2^{O(k)}n^{O(1)}$. With this idea in mind, we add local constraints to enforce that the assignment to Y is of the above quadratic form (in terms of $\alpha_1,\ldots,\alpha_k$ and $\beta_1,\ldots,\beta_k$), and then use locally decodable properties of quadratic polynomials to extract information about vectors $\vec{v}_1,\ldots,\vec{v}_k$.

In a high level, our construction generalizes that of [20] by replacing the linear code with the Reed–Muller code based on quadratic polynomials.

Expander graph production. To obtain an FPT time lower bound for k-Clique with $k^{o(1)}$ gap, we apply the standard expander graph product technique. Starting from a constant gap k-Clique instance, we amplify the gap using an expander graph H on vertex set $[k]$. The new instance contains k^t groups of vertices. Each group corresponds to a unique path of length-t random walk on H, and forms an independent set of size n^t. A vertex in a group represents a length-t sequence of vertices from the original instance. Two (sequences of) vertices are linked if and only if the vertices contained in them form a clique in the original instance. By properties of expander graphs, we get a k^t-Clique instance with gap $(\varepsilon')^t$ for some constant ε'. Take $t = o(\log k)$, we can rule out $k^{o(1)}$-ratio FPT-approximation algorithms for k-Clique under ETH.

From gap k-Clique to PIH. The proof that strong lower bound of constant gap k-Clique implies PIH goes as follows. First, we reduce constant gap k-Clique to constant gap k-Biclique in the canonical way. Next, we use a combinatorial object called disperser to amplify the gap from a constant to $\frac{k}{\log k}$. The result then follows from the Kõvári-Sós-Turán Theorem which states that every $2k$-vertex graph without a $K_{\log k, \log k}$-subgraph has at most $O((2k)^{2-\frac{1}{\log k}})$ edges.

1.2 Organization of the Paper

The paper is organized as follows. In Section 2, we put some preliminaries, including the definitions of problems, hypotheses, and some algebraic and combinatorial tools used in our proofs. In Section 3, we prove the ETH lower bound of constant gap k-Clique. In Section 4, we show how to amplify the gap to rule out $k^{o(1)}$-ratio FPT-approximation algorithms for k-Clique under ETH. In Section 5, we show how an almost-tight running time lower bound of constant gap k-Clique implies PIH. Finally, in Section 6, we conclude with a few open questions.

2 Preliminaries

2.1 Problems

Here we list all the computational problems which are relevant to our paper.

- 3SAT. The input is a 3-CNF formula φ with m clauses on n variables. The goal is to decide whether there is a satisfying assignment for φ.
- CSP. The input of a constraint satisfaction problem is a set of variables $X = \{x_1, \ldots, x_n\}$ together with a family of constraints $\{C_1, C_2, \ldots, C_m\}$ and an alphabet Σ. For every $i \in [m]$, $C_i = (\vec{s}_i, R_i)$, where $\vec{s}_i = (x_{j_1}, \ldots, x_{j_{\ell_i}})$ is an ℓ_i-tuple of variables for some $\ell_i \in [n]$, and $R_i \subseteq \Sigma^{\ell_i}$ indicates a restriction on valid assignments for those ℓ_i variables. The goal is to find an assignment $\sigma : X \to \Sigma$ such that for all $i \in [m]$, $\sigma(\vec{s}_i) \in R_i$. We call n, m, q and $|\Sigma|$ respectively the number of vertices, the number of clauses, the arity ($= \max_{i\in[m]} \ell_i$), and the alphabet size of this CSP problem.
- k-Clique. The input is an undirected graph $G = (V_1 \dot\cup \ldots \dot\cup V_k, E)$ with n vertices divided into k disjoint groups. The goal is to decide whether we can pick one vertex from each group, such that they form a clique of size k.
- k-Biclique. The input is an undirected bipartite graph $G = (V_1 \dot\cup \ldots \dot\cup V_k, U_1 \dot\cup \ldots \dot\cup U_k, E)$, where n vertices are divided into $2k$ disjoint groups. The goal is to decide whether we can pick one vertex from each group, such that they form a biclique $K_{k,k}$.

- DENSEST k-SUBGRAPH. The input is an undirected graph $G = (V_1 \dot\cup \ldots \dot\cup V_k, E)$ with n vertices divided into k disjoint groups. The goal is to pick one vertex from each group, such that they induce maximum number of edges.
- k-VECTORSUM. The input consists of k groups of vectors $V_1, \ldots, V_k \subseteq \mathbb{F}^d$ together with a target vector $\vec{t} \in \mathbb{F}^d$, where $\mathbb{F}$ is a finite field of constant size. The goal is to decide whether there exists $\vec{v}_1 \in V_1, \ldots \vec{v}_k \in V_k$ such that $\sum_{i=1}^{k} \vec{v}_i = \vec{t}$. Throughout our paper we only need the version that $\vec{t}$ equals to $\vec{0}$, and will omit it afterwards.

2.2 Hypotheses

Now we list some computational complexity hypotheses which are related to our results.

▶ **Hypothesis 4** (Exponential Time Hypothesis (ETH) [16, 17, 25])**.** 3SAT *with n variables and $m = O(n)$ clauses cannot be solved deterministically in $2^{o(n)}$ time. Moreover, this holds even when restricted to formulae in which each variable appears in at most three clauses.*

Note that the original statement in [16] is does not enforce the requirement that each variable appears in at most three clauses. For the restricted version, we first apply the Sparsification Lemma in [17], which implies that without loss of generality we can assume the number of clauses $m = O(n)$. Then we apply Tovey's reduction [25], which produces a 3SAT instance with at most $3m + n = O(n)$ variables and each variable appears in at most three clauses. Thus the restricted version is equivalent to the original statement.

The next hypothesis is *Parameterized Inapproximability Hypothesis* (PIH), a central conjecture in parameterized complexity. We state it in terms of inapproximability of DENSEST k-SUBGRAPH as follows.

▶ **Hypothesis 5** (Parameterized Inapproximability Hypothesis (PIH) [21])**.** *There exists a constant $\varepsilon > 0$ such that DENSEST k-SUBGRAPH has no $(1+\varepsilon)$ factor FPT-approximation algorithm. In other words, no algorithm can distinguish the following two cases in $f(k) \cdot n^{O(1)}$ time, for any computable function f:*

- *(Completeness.) There exist $v_1 \in V_1, \ldots, v_k \in V_k$ such that they form a clique.*
- *(Soundness.) For any $v_1 \in V_1, \ldots, v_k \in V_k$, they induce only $\binom{k}{2}/(1+\varepsilon)$ edges.*

The factor $(1+\varepsilon)$ can be replaced by any constant larger than 1, and the conjecture remains equivalent. Note that the original statement of PIH in [21] says that DENSEST k-SUBGRAPH is W[1]-hard to approximate, but for our use, we choose a relaxed form which states that it has no constant ratio FPT-approximation algorithm, as in [12].

It is worth noting that the relationship between PIH and gap k-CLIQUE is not completely known yet. If for a graph the number of edges induced by k vertices is only $\approx \varepsilon^2 \binom{k}{2}$, it cannot have a clique of size $> \varepsilon k$. Thus, PIH implies k-CLIQUE does not admit constant ratio FPT-approximation algorithms. However, the other direction is not necessarily true (forbidding small clique does not imply low edge density), and it remains an important open problem that whether PIH holds if we assume k-CLIQUE is hard to approximate within any constant factor in FPT time [12].

2.3 Low Degree Test

Let $\mathbb{F}$ be a field of prime cardinality. We say a function f is δ-close to a function class $\mathcal{F}$ if it is possible to modify at most δ fraction of values of f such that the modified function lies in $\mathcal{F}$.

The canonical low degree test proposed in [23] can query a function f at $d+2$ points, and

- accepts with probability 1 whenever f is a degree-d polynomial,
- rejects with probability at least $\varepsilon > 0$ if f is not δ-close to degree-d polynomials, where ε, δ are two constants.

Throughout our paper we will consider the function class $\mathcal{F}$ to be vector-valued degree-d polynomials, namely,

$$\mathcal{F} = \{(f_1, f_2, \ldots, f_\ell) : \mathbb{F}^m \to \mathbb{F}^\ell \mid \forall i \in [\ell], f_i \text{ is a degree-}d \text{ multivariate polynomial}\}.$$

By slightly modifying the proof in [23], the low degree test can be easily generalized to vector-valued version, as formally stated below:

▶ **Lemma 6.** *Let $d < |\mathbb{F}|/2$ and $m \in \mathbb{N}$. There is an algorithm which, by querying the function $f = (f_1, f_2, \ldots, f_\ell) : \mathbb{F}^m \to \mathbb{F}^\ell$ at $d+2$ points,*

- *accepts with probability 1 whenever f lies in $\mathcal{F}$,*
- *rejects with probability at least $\min(\delta/2, cd^{-2})$ if f is not δ-close to $\mathcal{F}$, where c, δ are two constants.*

Moreover, the queries are generated by selecting $\vec{x}, \vec{h} \in \mathbb{F}^m$ uniformly at random, and f is queried at $\{\vec{x} + i\vec{h} | 0 \le i < d+2\}$.

The proof is implicit in literature. We omit it here due to the page limitation. Readers can refer to our arxiv version [3] for a full proof.

2.4 Expander Graphs

Given a d-regular undirected graph G on n vertices, define its *normalized adjacency matrix* to be a matrix A where A_{ij} equals to the number of edges between (i, j) divided by d. Define

$$\lambda(G) = \max_{\|\vec{v}\|=1, \langle \vec{v}, \vec{1} \rangle = 0} \|A\vec{v}\|_2 .$$

G is an (n, d, λ)-expander if and only if $\lambda(G) \le \lambda$, and we have the following two Lemmas.

▶ **Lemma 7** ([2])**.** *Let G be an (n, d, λ)-expander, $\mathcal{B} \subseteq [n]$ be a set of size $\le \varepsilon n$ for some $0 < \varepsilon < 1$ and $(X_1, X_2, \ldots, X_t)$ be a sequence of random variables denoting a length-t random walk where the starting vertex is also picked uniformly at random. Then,*

$$\Pr[\forall 1 \le i \le t, X_i \in \mathcal{B}] \le ((1-\lambda)\sqrt{\varepsilon} + \lambda)^{t-1}.$$

▶ **Lemma 8** ([22])**.** *For some constants $d \in \mathbb{N}$, $\lambda < 1$ and for sufficiently large n, an (n, λ, d)-expander can be constructed in $n^{O(1)}$ time.*

2.5 Disperser

▶ **Definition 9** (Disperser [9, 26, 27])**.** *For positive integers $m, k, \ell, r \in \mathbb{N}$ and constant $\varepsilon \in (0, 1)$, an $(m, k, \ell, r, \varepsilon)$-disperser is a collection $\mathcal{I}$ of k subsets $I_1, \ldots, I_k \subseteq [m]$, each of size ℓ, such that the union of any r different subsets from the collection has size at least $(1-\varepsilon)m$.*

[3] https://arxiv.org/abs/2111.14033

Dispersers could be constructed efficiently by probabilistic methods, as in the following Lemma.

▶ **Lemma 10.** *For positive integers $m, \ell, r \in \mathbb{N}$ and constant $\varepsilon \in (0,1)$, let $\ell = \lceil \frac{3m}{\varepsilon r} \rceil$ and let $I_1, \ldots, I_k$ be random ℓ-subsets of $[m]$. If $\ln k \leq \frac{m}{r}$ then $\mathcal{I} = \{I_1, \ldots, I_k\}$ is an $(m, k, \ell, r, \varepsilon)$-disperser with probability at least $1 - e^{-m}$.*

Due to the page limitation, we also put the proof into the Appendix of our arxiv version.

3 An Improved Lower Bound for Constant Gap k-Clique under ETH

3.1 Reduction from 3SAT to k-VectorSum

To prove Theorem 1, we first need an $f(k) \cdot n^{\Omega(k)}$-time lower bound for k-VectorSum under ETH. Previously, it is known that k-Clique has no $f(k) \cdot n^{o(k)}$-time algorithms assuming ETH [8]. Combining this with the reduction from k-Clique to $\Theta(k^2)$-VectorSum [1], we only have an $f(k) \cdot n^{\Omega(\sqrt{k})}$-time lower bound for k-VectorSum under ETH. It is an interesting question whether there is an FPT reduction from k-Clique to k'-VectorSum with $k' = O(k)$. In this section, we give a reduction directly from 3SAT to k-VectorSum, which suits our purpose. Recall that in the ETH statement we can assume without loss of generality that each variable appears in at most 3 clauses, which is a key ingredient in our proof.

▶ **Theorem 11.** *There is a reduction which, for every integer $k \in \mathbb{N}$, and every 3SAT formula φ with m clauses and n variables such that each variable appears in at most 3 clauses, outputs a k-VectorSum instance $\Gamma = (\mathbb{F}, d, V_1, \ldots, V_k)$ with the following properties in $2^{O(n/k)}$ time.*

- $\mathbb{F} = \mathbb{F}_5$.
- $d = O(n)$.
- *For any $i \in [k]$, distinct $\vec{u}, \vec{v} \in V_i$ and any $a \in \mathbb{F}_5 \setminus \{0\}$, $\vec{u} \neq a \cdot \vec{v}$.*
- *For any $i \in [k]$, distinct $\vec{u}, \vec{v}, \vec{w} \in V_i$ and any $a \in \mathbb{F}_5 \setminus \{0\}, \vec{u} - \vec{w} \neq a \cdot (\vec{w} - \vec{v})$.*
- *(Completeness.) If φ is satisfiable, then there exists $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ such that $\sum_{i=1}^{k} \vec{v}_i = \vec{0}$.*
- *(Soundness.) If φ is not satisfiable, then for any $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$, $\sum_{i=1}^{k} \vec{v}_i \neq \vec{0}$.*

▶ Remark 12. Note if the size of the produced k-VectorSum instance N appears to be only $2^{o(n/k)}$, we can use brute force to solve it in $N^k = 2^{o(n)}$ time, thus solve 3SAT in $2^{o(n)}$ time. Therefore, we only need to consider the case $N = 2^{\Theta(n/k)}$ without loss of generality, and in this case $d = O(n) = O(k \log N)$.

Proof of Theorem 11. Let $C = \mathcal{C}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{C}_k$ be a partition of the clauses into k approximately equal-sized parts. We will let vectors in V_i represent partial satisfying assignments for $\mathcal{C}_i$, and use entries of vectors to check consistency of those partial assignments.

Define X to be the set of variables appearing in exactly two different parts and define Y to be the set of variables appearing in three different parts. Let $d = |X| + 2|Y|$, we associate one entry of vector to each variable $x \in X$ and two entries to each variable $y \in Y$. In the following, we abuse notation a bit and use $\vec{v}[x]$ to denote the entry in a vector $\vec{v} \in \mathbb{F}^d$ associated to a variable $x \in X$, and use $\vec{v}[y, 1], \vec{v}[y, 2]$ to denote the two entries associated to a variable $y \in Y$.

The construction of vector set V_i proceeds as follows. Let Z_i be the set of variables appearing in $\mathcal{C}_i$. For an assignment $\tau : Z_i \to \{0, 1\}$ which satisfies all clauses in $\mathcal{C}_i$, we map it to a vector $\vec{v} \in \mathbb{F}^d$ in the following way.

Let $x \in Z_i \cap X$ be a variable appearing in $\mathcal{C}_{j_1}$ and $\mathcal{C}_{j_2}$ $(j_1 < j_2)$,

- in case that $\tau(x) = 0$, set $\vec{v}[x] = 0$.
- in case that $\tau(x) = 1$, set $\vec{v}[x] = 1$ if $i = j_1$, and set $\vec{v}[x] = -1$ if $i = j_2$.

Let $y \in Z_i \cap Y$ be a variable appearing in $\mathcal{C}_{j_1}, \mathcal{C}_{j_2}$ and $\mathcal{C}_{j_3}$ $(j_1 < j_2 < j_3)$,

- in case that $\tau(y) = 0$, set $\vec{v}[y, 1] = 0$ and $\vec{v}[y, 2] = 0$.
- in case that $\tau(y) = 1$, set $\vec{v}[y, 1] = 1$ and $\vec{v}[y, 2] = 1$ if $i = j_1$; set $\vec{v}[y, 1] = -1$ and $\vec{v}[y, 2] = 0$ if $i = j_2$; and set $\vec{v}[y, 1] = 0$ and $\vec{v}[y, 2] = -1$ if $i = j_3$.

For the remaining entries of $\vec{v}$ (which are associated to variables in $(X \cup Y) \setminus Z_i$), set them to be 0.

It's easy to see the whole reduction runs in $2^{O(n/k)}$ time, and the dimension $d = O(n)$.

Now we prove the third and the fourth properties.

For two distinct vectors $\vec{u}, \vec{v} \in V_i$, suppose $\vec{u}[j] \neq \vec{v}[j]$. It must be the case that one of them is 0 and the other is ± 1. Thus they still differ after being multiplied by any $a \in \mathbb{F}_5 \setminus \{0\}$.

For three distinct vectors $\vec{u}, \vec{v}, \vec{w} \in V_i$, suppose $\vec{u}[j] \neq \vec{w}[j]$, then either $\vec{v}[j] = \vec{w}[j]$ or $\vec{v}[j] = \vec{u}[j]$. In the former case, $\vec{u}[j] - \vec{w}[j] \neq 0 = \vec{w}[j] - \vec{v}[j]$, so they still differ after being multiplied by any $a \in \mathbb{F} \setminus \{0\}$. In the latter case, suppose $\vec{u} - \vec{w} = a \cdot (\vec{w} - \vec{v})$, then $\vec{u}[j] - \vec{w}[j] = a \cdot (\vec{w}[j] - \vec{v}[j])$ will lead to $a = -1$ and thus $\vec{u} = \vec{v}$, a contradiction. Therefore, $\vec{u} - \vec{w} \neq a \cdot (\vec{w} - \vec{v})$ for any $a \in \mathbb{F}_5 \setminus \{0\}$.

Next follows the proof of completeness and soundness.

Completeness. If the 3SAT formula φ has a satisfying assignment τ, we can pick one vector $\vec{v}_i$ from each V_i according to the restriction of τ on $Z_i \cap (X \cup Y)$. Let $\vec{v} = \sum_{i=1}^{k} \vec{v}_i$ be the sum of picked vectors.

For a variable $x \in X$, let $\mathcal{C}_{j_1}, \mathcal{C}_{j_2} (j_1 < j_2)$ be the two clause parts in which x appears,

- in case that $\tau(x) = 0$, $\vec{v}[x] = 0$ since this entry equals to 0 in all vectors.
- in case that $\tau(x) = 1$, $\vec{v}[x] = 1 + (-1) = 0$ where 1 comes from $\vec{v}_{j_1}[x]$ and -1 comes from $\vec{v}_{j_2}[x]$.

For a variable $x \in Y$, let $\mathcal{C}_{j_1}, \mathcal{C}_{j_2}, \mathcal{C}_{j_3} (j_1 < j_2 < j_3)$ be the three clause parts in which x appears.

- in case that $\tau(x) = 0$, $\vec{v}[x, 1] = \vec{v}[x, 2] = 0$ since these entries equal to 0 in all vectors.
- in case that $\tau(x) = 1$, $\vec{v}[x, 1] = 1 + (-1) = 0$ where 1 comes from $\vec{v}_{j_1}[x, 1]$ and -1 comes from $\vec{v}_{j_2}[x, 1]$, and $\vec{v}[x, 2] = 1 + (-1) = 0$ where 1 comes from $\vec{v}_{j_1}[x, 1]$ and -1 comes from $\vec{v}_{j_3}[x, 1]$.

Soundness. If the 3SAT formula φ has no satisfying assignments, any collection of partial assignments satisfying individual clause parts must be inconsistent on some variable in $X \cup Y$. For any $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$, let $\vec{v} = \sum_{i=1}^{k} \vec{v}_i$.

Suppose assignments for a variable $x \in X$ which appears in $\mathcal{C}_{j_1}$ and $\mathcal{C}_{j_2}$ are inconsistent, there must be one 0 and one ± 1 in $\vec{v}_{j_1}[x]$ and $\vec{v}_{j_2}[x]$. Since this entry equals to 0 in all other vectors, it results that $\vec{v}[x] \neq 0$.

Suppose assignments for a variable $x \in Y$ which appears in $\mathcal{C}_{j_1}, \mathcal{C}_{j_2}$ and $\mathcal{C}_{j_3}$ $(j_1 < j_2 < j_3)$ are inconsistent. If the values for x specified by $\vec{v}_{j_1}$ and $\vec{v}_{j_2}$ are inconsistent, there must be one 0 and one ± 1 in $\vec{v}_{j_1}[x, 1]$ and $\vec{v}_{j_2}[x, 1]$, while in all other vectors this entry equals to 0, thus $\vec{v}[x, 1] \neq 0$. Otherwise the value for x specified by $\vec{v}_{j_1}$ and $\vec{v}_{j_3}$ must be inconsistent, there must be one 0 and one ± 1 in $\vec{v}_{j_1}[x, 2]$ and $\vec{v}_{j_3}[x, 2]$, while in all other vectors this entry equals to 0, thus $\vec{v}[x, 2] \neq 0$.

Therefore, $\sum_{i=1}^{k} \vec{v}_i \neq \vec{0}$ as desired. ◀

3.2 Reduction from k-VectorSum to Constant Gap k-Clique

▶ **Theorem 13.** *There is an FPT reduction which, given as input a* k-VectorSum *instance* $\Gamma_0 = (\mathbb{F}, d, V_1, \ldots, V_k)$ *with the following properties:*

- $\mathbb{F} = \mathbb{F}_5$,
- $d = O(k \log n)$ *where* $n = \sum_{i=1}^{k} |V_i|$ *denotes instance size,*
- *for any* $i \in [k]$, *distinct* $\vec{u}, \vec{v} \in V_i$ *and any* $a \in \mathbb{F}_5 \setminus \{0\}$, $\vec{u} \neq a \cdot \vec{v}$,
- *for any* $i \in [k]$, *distinct* $\vec{u}, \vec{v}, \vec{w} \in V_i$ *and any* $a \in \mathbb{F}_5 \setminus \{0\}, \vec{u} - \vec{w} \neq a \cdot (\vec{w} - \vec{v})$.

outputs a k'-Clique *instance* $G = (V, E)$ *such that*

- $k' \leq c^k$ *for some constant* c,
- *(Completeness.) if* Γ_0 *is a yes-instance of* k-VectorSum, *then* G *contains a clique of size* k',
- *(Soundness.) if* Γ_0 *is a no-instance of* k-VectorSum, *then* G *doesn't contain a clique of size* $\varepsilon k'$ *for some constant* $\varepsilon < 1$.

The first step of the reduction involves $\ell = 2k + 4 \log n$ matrices $A_1, \ldots, A_\ell \in \mathbb{F}^{k \times d}$. For $\alpha \in \mathbb{F}^k, v \in \mathbb{F}^d$, define a bilinear function $f(\alpha, v) = (\langle \alpha, A_1 v \rangle, \ldots, \langle \alpha, A_\ell v \rangle) \in \mathbb{F}^\ell$, where $\langle \cdot, \cdot \rangle$ denotes inner product.

▶ **Lemma 14** ([20]). *We can find* $\ell = 2k + 4 \log n$ *matrices* $A_1, A_2, \cdots, A_\ell$ *in time polynomial in* n, k, *which satisfy the following properties:*

1. *for any nonzero vector* $\vec{v} \in \mathbb{F}^d$, *there exists* $i \in [\ell]$ *such that* $A_i \vec{v} \neq \vec{0}$,
2. *for any* $i \in [k]$, *distinct* $\vec{u}, \vec{v} \in V_i$ *and nonzero* $\alpha \in \mathbb{F}^k$, $f(\alpha, \vec{u}) \neq f(\alpha, \vec{v})$,
3. *for any* $i \in [k]$, *distinct* $\vec{u}, \vec{v}, \vec{w} \in V_i$ *and* $\alpha, \alpha' \in \mathbb{F}^k, f(\alpha, \vec{u}) + f(\alpha', \vec{v}) \neq f(\alpha + \alpha', \vec{w})$.

The reduction then goes as follows. For every $\alpha, \beta \in \mathbb{F}^k$, we introduce a variable $x_{\alpha,\beta}$ which takes value in $\mathbb{F}^\ell$. In the yes-case of k-VectorSum, there exists $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ such that $\sum_{i=1}^{k} \vec{v}_i = \vec{0}$, and we expect $x_{\alpha,\beta}$ to be $f(\alpha, \sum_{i=1}^{k} \beta_i \vec{v}_i)$, in other words, $\sum_{i=1}^{k} \sum_{j=1}^{k} \alpha_i \beta_j (A_1[i]\vec{v}_j, \ldots, A_\ell[i]\vec{v}_j)$ where $A_w[i]$ indicates the i-th row of the w-th matrix. Note that $f(\alpha, \sum_{i=1}^{k} \beta_i \vec{v}_i)$ is a degree-2 polynomial of α and β.

For simplicity of notation, we will use $e_i \in \mathbb{F}^k$ to denote the i-th unit vector, and use $\mathbf{1} \in \mathbb{F}^k$ to denote the all-one vector.

We want to apply four types of tests on those variables:

1. Check whether $x : \mathbb{F}^{2k} \to \mathbb{F}^\ell$ is a vector-valued degree-2 polynomial. This can be done by the low-degree test described in Lemma 6. For each $\vec{\alpha}, \vec{\beta}, \vec{t}_1, \vec{t}_2 \in \mathbb{F}^k$, check whether $\{x_{\vec{\alpha}+i\cdot\vec{t}_1, \vec{\beta}+i\cdot\vec{t}_2} | 0 \leq i \leq 3\}$ are point values of a degree-2 polynomial. Each test is applied on 4 variables, and we say the arity of each such test is 4 in shorthand.
2. Check whether x which maps (α, β) to $x_{\alpha,\beta}$ is linear in both α and β, i.e., whether $x_{\alpha+\alpha',\beta} = x_{\alpha,\beta} + x_{\alpha',\beta}, \forall \alpha, \alpha', \beta \in \mathbb{F}^k$, and $x_{\alpha,\beta+\beta'} = x_{\alpha,\beta} + x_{\alpha,\beta'}, \forall \alpha, \beta, \beta' \in \mathbb{F}^k$. The arity of each such test is 3.
3. For each $u \in [k], \alpha, \beta \in \mathbb{F}^k$, check whether $x_{\alpha,\beta+e_u} - x_{\alpha,\beta} = f(\alpha, \vec{v}) \in \mathbb{F}^\ell$ for some $\vec{v} \in V_u$. The arity of each such test is 2.
4. For each $\alpha, \beta \in \mathbb{F}^k$, check whether $x_{\alpha,\beta+\mathbf{1}} - x_{\alpha,\beta} = \vec{0}$. The arity of each such test is 2.

Construction of the Graph. The vertices are divided into three types. Vertices in each type are further partitioned into groups, and each group forms an independent set:

type-1 There are $(|\mathbb{F}|^{2k})^2$ groups, each of which indicates a test of type 1, and consists of $\leq |\mathbb{F}|^{4\ell}$ vertices corresponding to all satisfying assignments of the 4 variables in the test.

type-2 There are $2(|\mathbb{F}|^k)^3$ groups, each of which indicates a test of type 2, and consists of $\leq |\mathbb{F}|^{2\ell}$ vertices corresponding to all satisfying assignments of the 3 variables in the test.

type-3 There are $|\mathbb{F}|^{2k}$ groups indexed by $(\alpha, \beta) \in \mathbb{F}^{2k}$, each consisting of $\mathbb{F}^\ell$ vertices which correspond to $\mathbb{F}^\ell$ possible assignments for variable $x_{\alpha,\beta}$.

We make copies of vertices, so that the numbers of type-1 groups and type-2 groups are the same, and their sum equals to the number of type-3 groups. Specifically, the three types of vertices are made into $2, |\mathbb{F}|^k, 4\mathbb{F}^{2k}$ copies, respectively. The total number of groups is therefore $k' = 8|\mathbb{F}|^{4k}$, while the total number of vertices is at most $2|\mathbb{F}|^{4k+4\ell} + 2|\mathbb{F}|^{4k+2\ell} + 4|\mathbb{F}|^{4k+\ell} = |\mathbb{F}|^{O(k+\log n)}$.

The edges are specified as follows:

1. A variable (type-3) vertex and a test (type-1/2) vertex are linked if and only if they specify the same assignment for the variable, or the test is irrelevant of that variable.
2. Two test vertices are linked if and only if they are consistent in all variables which appear in both tests.
3. Two variable vertices are linked if and only if the assignments specified by them can pass the above-mentioned third and fourth tests, or there is no such a test between them.

Two different copies of a same vertex are always linked.

Proof of Completeness. If Γ_0 is a yes-instance of k-VectorSum, i.e., there exists $\vec{v}_1 \in V_1, \ldots, \vec{v}_k \in V_k$ such that $\sum_{i=1}^k \vec{v}_i = \vec{0}$, by letting $x_{\alpha,\beta}$ take value $f(\alpha, \sum_{i=1}^k \beta_i \vec{v}_i)$, it's easy to see that such an assignment can pass all tests. Therefore, by picking a vertex from each group accordingly, one can obtain a clique of size k'.

Proof of Soundness. If Γ_0 is a no-instance of k-VectorSum, we will prove that there is no clique of size $\geq (1-\varepsilon)k'$ in G for some small constant ε.

Prove by contradiction. If there is a clique of size $\geq (1-\varepsilon)k'$, it must contain vertices from $\geq (1-2\varepsilon)$ fraction of type-3 groups which represent variables, vertices from $\geq (1-4\varepsilon)$ fraction of type-1 groups which represent low-degree tests, vertices from $\geq (1-8\varepsilon)$ fraction of type-2 groups which represent linearity tests $x_{\alpha+\alpha',\beta} = x_{\alpha,\beta} + x_{\alpha',\beta}$ and vertices from $\geq (1-8\varepsilon)$ fraction of type-2 groups which represent linearity tests $x_{\alpha,\beta+\beta'} = x_{\alpha,\beta} + x_{\alpha,\beta'}$.

In the following, we will denote by $\overline{x}_{\alpha,\beta}$ the assignment for $x_{\alpha,\beta}$ specified by the clique. If no assignment for $x_{\alpha,\beta}$ is specified, set $\overline{x}_{\alpha,\beta}$ arbitrarily as long as it is consistent with all selected test vertices (it is always possible since the selected test vertices are themselves consistent). As almost all low-degree tests and linearity tests are passed, we have:

▶ **Lemma 15.** *If $\varepsilon < \frac{c}{16}$ where c is the constant in Lemma 6, and there is a clique of size $\geq (1-\varepsilon)k'$, then the function $\pi(\alpha,\beta) = \overline{x}_{\alpha,\beta}$ $(\alpha, \beta \in \mathbb{F}^k)$ is 9ε-close to a function on α, β of the form*

$$\sum_{i=1}^{k}\sum_{j=1}^{k} \alpha_i \beta_j C_{i,j}$$

where $C_{i,j} \in \mathbb{F}^l$ denotes the coefficient of the term $\alpha_i\beta_j$.

Proof. Plugging $\delta = 9\varepsilon$ into Theorem 6, if $\pi(\alpha,\beta)$ is not δ-close to any degree-2 polynomial, at least $\min(\delta/2, cd^{-2}) > 4\varepsilon$ fraction of the degree-2 polynomial tests will not be passed. However, when there is a clique of size $\geq (1-\varepsilon)k'$, only $\leq 4\varepsilon$ fraction of degree-2 polynomial tests may fail. Therefore, π must be 9ε-close to a function of the form

$$\sum_{i=1}^{k}\sum_{j=1}^{k} \alpha_i\alpha_j A_{i,j} + \sum_{i=1}^{k}\sum_{j=1}^{k} \beta_i\beta_j B_{i,j} + \sum_{i=1}^{k}\sum_{j=1}^{k} \alpha_i\beta_j C_{i,j} + \sum_{i=1}^{k} D_i\alpha_i + \sum_{i=1}^{k} E_i\beta_i + F \tag{2}$$

where each coefficient is in $\mathbb{F}^l$. We only need to prove that if they also pass most of the linearity tests on α and on β, all coefficients except $C_{i,j}$ must be zeros.

Suppose $A_{i,j} \neq 0$ for some $i, j \in [k]$. Regarding $x_{\alpha+\alpha',\beta} - x_{\alpha,\beta} - x_{\alpha',\beta}$ as a function on $3k$ variables $\alpha_1 \dots \alpha_k, \alpha'_1 \dots \alpha'_k, \beta_1 \dots \beta_k$ and expand it by (2). There is a term $A_{i,j}(\alpha_i\alpha'_j + \alpha'_i\alpha_j)$ which can never be canceled. The function $x_{\alpha+\alpha',\beta} - x_{\alpha,\beta} - x_{\alpha',\beta}$ is not a zero function and by Schwartz-Zippel Lemma, only $\frac{2}{|\mathbb{F}|}$ fraction of α, α', β can make it equal to zero. However, by union bound, there are at least $1 - 8\varepsilon - 3 \cdot (9\varepsilon) > \frac{2}{|\mathbb{F}|}$ fraction of (α, α', β) such that $\overline{x}_{\alpha,\beta}, \overline{x}_{\alpha',\beta}, \overline{x}_{\alpha+\alpha',\beta}$ are all specified value according to the clique, consistent with equation (2), and satisfying $\overline{x}_{\alpha,\beta} + \overline{x}_{\alpha',\beta} = \overline{x}_{\alpha+\alpha',\beta}$, a contradiction. Therefore, $\forall i, j \in [k], A_{i,j} = 0$.

Similar arguments can be applied for the other coefficients and are omitted here. The only term remaining is $\sum_{i=1}^{k}\sum_{j=1}^{k} \alpha_i\beta_j C_{i,j}$ as desired. ◀

We call a variable $x_{\alpha,\beta}$ *good* if the clique consists of a variable vertex of it and it satisfies $\overline{x}_{\alpha,\beta} = \sum_{i=1}^{k}\sum_{j=1}^{k} \alpha_i\beta_j C_{i,j}$. Let $\varepsilon' = 2\varepsilon + 9\varepsilon$, from the above we know at least $(1 - \varepsilon')$ fraction of variables are good. Recall that from the construction of our graph and the property of clique, all arity-2 constraints between good variables are satisfied.

We call an $\alpha \in \mathbb{F}^k$ *excellent* if $\Pr_{\beta \in_R \mathbb{F}^k}[x_{\alpha,\beta} \text{ is good}] \geq \frac{2}{3}$. By Markov's Inequality, at least $1 - 3\varepsilon'$ fraction of α's are excellent.

▶ **Lemma 16.** *For each excellent α and for each $u \in [k]$, $\sum_{i=1}^{k} \alpha_i C_{i,u} = f(\alpha, \vec{v})$ for some unique $\vec{v} \in V_u$.*

Proof. For any fixed $u \in [k]$ and α, the set of edges between the vertex of $x_{\alpha,\beta}$ and the vertex of $x_{\alpha,\beta+e_u}$ for all β can be partitioned into disjoint length-5 cycles $\{x_{\alpha,\beta}, x_{\alpha,\beta+e_u}, \dots, x_{\alpha,\beta+4e_u}\}$ since the characteristic of $\mathbb{F}$ is 5. Observe that if at most two variables in a 5-cycle are not good, there still exist two adjacent vertices that are good.

For an excellent α, at most $\frac{1}{3} \leq \frac{2}{5}$ fraction of variables $x_{\alpha,\beta}$ are not good by definition, so there must exists a $\beta \in \mathbb{F}^k$ such that $x_{\alpha,\beta}$ and $x_{\alpha,\beta+e_u}$ are both good variables. According to the definition of good variables, we have

$$\overline{x}_{\alpha,\beta+e_u} - \overline{x}_{\alpha,\beta} = \sum_{i=1}^{k} \alpha_i C_{i,u}$$

and from the third type of constraints between them we can infer that

$$\overline{x}_{\alpha,\beta+e_u} - \overline{x}_{\alpha,\beta} = f(\alpha, \vec{v})$$

for some $\vec{v} \in V_u$.

Additionally, since for $\vec{v} \in V_u$, $f(\alpha, \vec{v})$ are all different (the second property in Lemma 14), for an excellent α and for all $u \in [k]$, we can deduce $\sum_{i=1}^{k} \alpha_i C_{i,u} = f(\alpha, \vec{v})$ for some unique $\vec{v} \in V_u$. ◀

In the following when $u \in [k]$ is fixed and omitted, we use $\vec{v}_\alpha$ to denote the unique vector in V_u specified by an excellent α. For an α which is not excellent, we also assign a unique vector $\vec{v}_\alpha \in V_u$ to it arbitrarily so that v_α is defined for all $\alpha \in \mathbb{F}^k$.

▶ **Lemma 17.** *If there is a clique of size $\geq (1 - \varepsilon)k'$, then for each $u \in [k]$,*

$$C_{i,u} = (A_1[i]\vec{v}, \dots, A_\ell[i]\vec{v})$$

for some unique $\vec{v} \in V_u$, where $A_w[i]$ indicates the i-th row of the w-th matrix.

Proof. Fix any $u \in [k]$, we first argue that at least $\geq \frac{3}{10}$ fraction of α specify the same $\vec{v} \in V_u$. It suffices to prove

$$\Pr_{\alpha,\alpha' \in_R \mathbb{F}^k}[\vec{v}_\alpha = \vec{v}_{\alpha'}] \geq \frac{3}{10}.$$

If this is not true, by union bound $3 \cdot \frac{3}{10} + 3(3\varepsilon') < 1$, there must exist α, α' such that the following two conditions hold:

1. $\alpha, \alpha', \alpha + \alpha'$ are all excellent.
2. $\vec{v}_\alpha, \vec{v}_{\alpha'}, \vec{v}_{\alpha+\alpha'}$ are all different.

Now that they are all excellent, we have

$$\begin{aligned} f(\alpha + \alpha', \vec{v}_{\alpha+\alpha'}) &= \sum_{i=1}^{k} (\alpha_i + \alpha'_i) C_{i,u} \\ &= f(\alpha, \vec{v}_\alpha) + f(\alpha', \vec{v}_{\alpha'}), \end{aligned}$$

contradicting with the third property in Lemma 14.

Therefore, at least $\frac{3}{10} - 3\varepsilon' > \frac{1}{|\mathbb{F}|}$ fraction of α are all excellent and specify the same $\vec{v} \in V_u$. Now suppose $C_{i,u} \neq (A_1[i]\vec{v}, \ldots, A_\ell[i]\vec{v})$, then there are at most $\frac{1}{|\mathbb{F}|}$ fraction of $\alpha \in \mathbb{F}^k$ making $\sum_{i=1}^{k} \alpha_i C_{i,u} = f(\alpha, \vec{v})$ by Schwartz-Zippel Lemma. However, we have $> \frac{1}{|\mathbb{F}|}$ fraction of such α, a contradiction. ◀

▶ **Lemma 18.** *If there is a clique of size $\geq (1-\varepsilon)k'$, then there exists $\vec{v}_1 \in V_1, ..., \vec{v}_k \in V_k$ such that $\sum_{i=1}^{k} \vec{v}_i = \vec{0}$.*

Proof. From Lemma 17 we know that for each $u \in [k]$, $C_{i,u} = (A_1[i]\vec{v}, \ldots, A_\ell[i]\vec{v})$ for some unique $\vec{v} \in V_u$. Thus $\overline{x}_{\alpha,\beta}$ indeed equals to $f(\alpha, \sum_{i=1}^{k} \beta_i \vec{v}_i)$ for every good variable $x_{\alpha,\beta}$.

The remaining proof is very similar to the proof of Lemma 16. Edges between the vertex of $x_{\alpha,\beta}$ and the vertex of $x_{\alpha,\beta+\mathbf{1}}$ for all $\beta \in \mathbb{F}^k$ can be divided into disjoint length-5 cycles $\{x_{\alpha,\beta}, x_{\alpha,\beta+\mathbf{1}}, \ldots, x_{\alpha,\beta+4\cdot\mathbf{1}}\}$ since the characteristic of $\mathbb{F}$ is 5.

For an excellent α, at most $\frac{1}{3} \leq \frac{2}{5}$ fraction of variables $x_{\alpha,\beta}$ are not good, so there must be two variables $x_{\alpha,\beta}$ and $x_{\alpha,\beta+\mathbf{1}}$ which are both good. According to the definition of good variables, we have

$$\overline{x}_{\alpha,\beta+s} - \overline{x}_{\alpha,\beta} = f\left(\alpha, \sum_{i=1}^{k} \vec{v}_i\right)$$

and from the fourth type of constraints between them we can infer that

$$\overline{x}_{\alpha,\beta+\mathbf{1}} - \overline{x}_{\alpha,\beta} = \vec{0}.$$

Therefore, $f(\alpha, \sum_{i=1}^{k} \vec{v}_i) = \vec{0}$ for every excellent α.

Suppose $\sum_{i=1}^{k} \vec{v}_i \neq \vec{0}$, then there exists $i \in [\ell]$ such that $A_i(\sum_{i=1}^{k} \vec{v}_i) \neq \vec{0}$ by the first property in Lemma 14. There are at most $\frac{1}{|\mathbb{F}|}$ fraction of α such that $f(\alpha, \sum_{i=1}^{k} \vec{v}_i) = \vec{0}$ by Schwartz-Zippel Lemma, but we have $\geq 1 - 3\varepsilon'$ fraction of excellent α, a contradiction. ◀

3.3 Putting Things Together

Combing Theorem 11 with Theorem 13, we obtain an improved lower bound for constant gap k-Clique under ETH as follows.

▶ **Theorem 1.** *Assuming* ETH*, for any constant $c > 1$ and any computable function f, no algorithm can find a clique of size k/c in the k-Clique problem in $f(k)n^{o(\log k)}$ time.*

Proof. Without loss of generality assume f is non-decreasing and unbounded. Given a 3SAT formula φ with n variables, each appearing in at most 3 clauses, we first run the reduction in Theorem 11 to produce a k-VectorSum instance of size $2^{O(n/k)}$, then run the reduction

in Theorem 13 to produce a constant gap c^k-Clique instance of size at most $c^k 2^{O(n/k)}$. Using the graph product method, we can amplify the gap to any constant, while keeping the parameter $k' = c^{O(k)}$ and instance size $n' \leq c^{O(k)} 2^{O(n/k)}$. Therefore, an $f(k')n^{o(\log k')}$ time algorithm for constant gap k'-Clique would lead to an algorithm for 3SAT in

$$f(k')(n')^{o(\log k')} \leq f(c^{O(k)})(c^{O(k)}2^{O(n/k)})^{o(k)} \leq 2^{o(n)}$$

time, contradicting ETH. The last inequality holds because n can be sufficiently large compared to k. ◀

Below we present two remarks about possible extensions of our results on ETH lower bounds of gap k-Clique.

▶ Remark 19 (On higher degree Reed-Muller Codes). It is natural to extend our idea to obtain a reduction from k-VectorSum to a CSP problem with $< 2^{o(k)}$ variables using Reed–Muller code with larger degree polynomials. However, the reduction from CSP to k-Clique has such an important property: when there is a clique of size εk for some constant ε, the following two conditions hold:

1. A constant fraction of arity-d constraints ($d > 2$) are satisfied.
2. All arity-2 constraints between a constant fraction of variables are satisfied.

The second condition holds because an arity-2 constraint between two variables can be directly transformed into an edge between two vertices. If there is a large clique, it means all arity-2 constraints between the involved variables are simultaneously satisfied. However, if we use larger degree polynomials, the arity of constraints has to be larger, too. It cannot directly fit into the framework of k-Clique. If this barrier can be broken, it may be possible to obtain reductions with an even smaller parameter blow-up using larger degree polynomials.

▶ Remark 20 (On locally decodable codes). Our reduction implicitly depends on the property of 2-query locally decodable code, that we could decode $f(\alpha, \vec{v_i})$ for some fixed α by querying only 2 positions. As pointed out in [13], 2-query locally decodable code has at least an exponential blow up. Hence our method is optimal in this sense. We could also consider how to remove this dependence.

4 $k^{o(1)}$-Ratio FPT Inapproximability of k-Clique under ETH

In this section, we show how to use expander graphs to amplify the gap efficiently, and how it leads to an improved inapproximability ratio of k-Clique in FPT time under ETH. The idea comes from the classical technique used to amplify gap in the non-parameterized version of Clique problem, which was proposed by Alon et al. [3].

▶ **Theorem 21.** *For some constants $d \in \mathbb{N}, 0 < \lambda < 1$ and for any $t \in \mathbb{N}$, there is an algorithm which runs in $O(k^2 d^{2t} |V|^{2t})$ time, on input an instance $\Gamma = (V, E)$ of k-Clique problem, outputs an instance Γ' of k'-Clique problem such that*

- *$k' = kd^{t-1}$.*
- *(Completeness.) If Γ has a k-clique, then Γ' has a k'-clique.*
- *(Soundness.) If Γ has no εk-clique, then Γ' has no clique of size $k'((1-\lambda)\sqrt{\varepsilon} + \lambda)^{t-1}$.*

Proof. Let H be an (k, λ, d)-expander constructed from Lemma 8. We construct Γ' as follows. Each of the $k' = kd^{t-1}$ groups in Γ' is associated with a unique path of length-t random walk on H. We use $(c_1, \ldots, c_t)$ to name a group in Γ', where each $c_i \in [k]$ indicates a group in Γ.

A vertex in Γ' is a length-t sequence of vertices in Γ. Namely, there is a vertex $(u_1, \ldots, u_t)$ in the $(c_1, \ldots, c_t)$-th group in Γ' if and only if each u_i is belongs to group c_i in Γ. Therefore, the total number of vertices is at most $kd^{t-1}|V|^t$ in Γ'.

A vertex $(u_1, \ldots, u_t)$ in group $(c_1, \ldots, c_t)$ is linked to a vertex $(v_1, \ldots, v_t)$ in group $(d_1, \ldots, d_t)$ if and only if

- $(c_1, \ldots, c_t) \neq (d_1, \ldots, d_t)$,
- and the vertices $\{v_1, \ldots, v_t, u_1, \ldots, u_t\}$ form a clique in Γ.

The reduction runs in $O(k^2 d^{2t}|V|^{2t})$ time by simply enumerating every pair of vertices in Γ' and checking if there is an edge between them.

Completeness. Let $\{v_1, \ldots, v_k\}$ be an k-clique in Γ. Then in group $(c_1, \ldots, c_t)$ in Γ', we can pick the vertex $(v_{c_1}, \ldots, v_{c_t})$. It's easy to see those vertices form an kd^{t-1}-clique.

Soundness. For any clique V in Γ', let U be the collection of vertices in Γ which appear as part of the name of a vertex in V. Since V is a clique in Γ', it follows by construction that U is also a clique in Γ and thus $|U| \leq \varepsilon k$. Recall that each $(c_1, \ldots, c_t)$ represents a length-t random walk on H, and all those c_i's lie in a set of size $\leq \varepsilon k$ (which corresponds to the collection of groups that vertices in U belong to). By plugging $n = k, |\mathcal{B}| \leq \varepsilon k$ into Lemma 7, the number of different groups that vertices in V belong to is bounded by $kd^{t-1}((1-\lambda)\sqrt{\varepsilon} + \lambda)^{t-1}$, and so is $|V|$. ◀

For any function $\delta(k) = o(1)$, by setting t to be as large as some $o(\log k)$, we can make $\varepsilon' = ((1-\lambda)\sqrt{\varepsilon} + \lambda)^{t-1}$ smaller than $k^{-\delta(k)}$ while keeping $k' = kd^{t-1} \leq k^{O(1)}$. Thus, by combining Theorem 1 and Theorem 21, we have the following corollary.

▶ **Corollary 2.** *Assuming* ETH*, for any $g(k) = k^{o(1)}$, the k-*Clique *problem has no $g(k)$-FPT-approximation algorithm.*

5 From Constant Gap k-Clique to PIH

In this section we will show that strong lower bound of constant gap k-Clique implies PIH. For simplicity of notation, we additionally define problems Gap-clique(k, ℓ) and Gap-biclique(k, ℓ) $(k > \ell)$, whose definitions are almost the same as k-Clique and k-Biclique, except that the soundness parameter is ℓ. We have the following theorem:

▶ **Theorem 3.** *If* Gap-clique$(k, \varepsilon k)$ *does not admit $f(k) \cdot n^{O(\frac{k}{\log k})}$-time algorithms for some $0 < \varepsilon < 1$, then* PIH *is true.*

The proof is relatively elementary and consists of three steps: first reduce constant gap k-Clique to constant gap k-Biclique, then use a *disperser* to compress the soundness parameter to $\frac{\log k}{k}$, finally use the Kővári-Sós-Turán Theorem to show that in the soundness case, the density of every $2k$-vertex bipartite subgraph is low.

▶ **Lemma 22.** *If* Gap-clique$(k, \varepsilon k)$ *does not admit $f(k) \cdot n^{O(\frac{k}{\log k})}$-time algorithms for some $0 < \varepsilon < 1$, then neither does* Gap-biclique$(k, \frac{1+\varepsilon}{2}k)$.

Proof. We reduce a Gap-clique$(k, \varepsilon k)$ instance $G = (V_1 \dot\cup \ldots \dot\cup V_k, E)$ to a Gap-biclique$(k, \frac{1+\varepsilon}{2}k)$ instance $G' = (U_1 \dot\cup \ldots \dot\cup U_k, W_1 \dot\cup \ldots \dot\cup W_k, E)$ as follows.

The vertex sets in each side are just copies of V, i.e., $U_i = W_i = V_i, \forall i \in [k]$. Two vertices $u \in U_i, w \in W_j$ where $i \neq j$ are linked if and only if their corresponding vertices are linked in G, while two vertices $u \in U_i, w \in W_i$ are linked iff they correspond to the same vertex in G.

The completeness case is obvious. In the soundness case, suppose we can pick $\frac{1+\varepsilon}{2}k$ vertices from different parts of U and $\frac{1+\varepsilon}{2}k$ vertices from different parts of W such that they form a biclique. Let the collection of picked vertices be S. Then there must be an index set $\mathcal{I}$ of size εk such that $\forall i \in \mathcal{I}$, $(S \cap U_i \neq \emptyset) \wedge (S \cap W_i \neq \emptyset)$. Moreover, $|S \cap U_i| = |S \cap W_i| = 1$ by our construction of edges between U_i and W_i. Then consider the set $\bigcup_{i \in \mathcal{I}}(S \cap U_i)$ which is of size at least εk. The vertices in it must form a clique of size $|\mathcal{I}|$ in the original graph G. ◀

▶ **Lemma 23.** *If* Gap-Biclique$(k, \varepsilon k)$ *does not admit* $f(k) \cdot n^{O(\frac{k}{\log k})}$*-time algorithms for some constant* $0 < \varepsilon < 1$*, then for any constant* $0 < c < 1$*, no algorithm can solve* Gap-Biclique$(k, c \log k)$ *in* $f(k) \cdot n^{O(1)}$ *time.*

Proof. Given a Gap-Biclique$(k, \varepsilon k)$ instance $G = (U_1 \dot\cup \ldots \dot\cup U_k, W_1 \dot\cup \ldots \dot\cup W_k, E)$, let $\ell = \lceil \frac{3k}{\varepsilon c \log k} \rceil$ and let $\mathcal{I} = (I_1, \ldots, I_k)$ be a $(k, k, \ell, c \log k, 1-\varepsilon)$-disperser. Since the size of $\mathcal{I}$ is independent of n, we can deterministically enumerate all possible $\mathcal{I}$ to find a valid one in $f(k)$ time. The existence of such a disperser is guaranteed by Lemma 10. We construct a new Gap-Biclique$(k, c \log k)$ instance $G' = (U'_1 \dot\cup \ldots \dot\cup U'_k, W'_1 \dot\cup \ldots \dot\cup W'_k, E)$ as follows.

The groups of vertices in G' correspond to the combination of groups in G according to the disperser. Specifically, for $1 \leq i \leq k$, let $I_i = \{i_1, \ldots, i_\ell\}$, then each vertex in U'_i will correspond to a tuple of vertices $(u_{i_1}, \ldots, u_{i_\ell})$ where u_{i_j} comes from U_{i_j} in G for all $1 \leq j \leq \ell$. The construction of right vertices $W'_1, \ldots W'_k$ is similar. The size of the new instance is therefore at most $n^\ell = n^{O(\frac{k}{\log k})}$.

An edge between a left vertex $(u_{i_1}, \ldots, u_{i_\ell})$ and a right vertex $(w_{j_1}, \ldots, w_{j_\ell})$ exists if and only if the vertices $\{u_{i_1}, \ldots, u_{i_\ell}, w_{j_1}, \ldots, w_{j_\ell}\}$ form a biclique $K_{\ell,\ell}$ in G.

The completeness case is still obvious, and we focus on the soundness case. Prove by contradiction, if there exists $c \log k$ vertices from different groups of U' and $c \log k$ vertices from different groups of W' which form a biclique $K_{c \log k, c \log k}$, let S be the collection of vertices which appear as part of one of the $2c \log k$ tuples. For $1 \leq i \leq k$, arbitrarily pick one vertex from each $S \cap U_i$, $S \cap W_i$ if not empty, then we claim that the resulting collection must be a biclique of size $\geq \varepsilon k$ on both sides. The promise of biclique is from our construction, while the size is guaranteed by properties of the disperser.

Therefore, an $f(k) \cdot n^{O(1)}$ time algorithm for the Gap-Biclique$(k, c \log k)$ problem would lead to an $f(k) \cdot \left(n^{O(\frac{k}{\log k})}\right)^{O(1)} = f(k) \cdot n^{O(\frac{k}{\log k})}$ time algorithm for Gap-Biclique$(k, \varepsilon k)$ problem. This completes the proof. ◀

▶ **Theorem 24** (Kővári-Sós-Turán, [19])**.** *For any graph* G *on* n *vertices, if* G *does not contain* $K_{a,a}$ *as a subgraph, then* G *has at most* $O(n^{2-1/a})$ *edges.*

Proof of Theorem 3. By plugging in $a = c \log k$ for some sufficiently small constant $0 < c < 1$ such that the $O(k^{2-1/(c \log k)})$ in Theorem 24 is no more than $\varepsilon' k^2$ for some constant $0 < \varepsilon' < 1$, Theorem 24 and Lemma 23 imply that if Gap-Biclique$(k, \varepsilon k)$ does not admit $f(k) \cdot n^{O(\frac{k}{\log k})}$-time algorithms, then no FPT algorithm can distinguish the following cases for a k-Biclique instance $G = (U, W, E)$ with

- *(Completeness.)* there exists $u_1 \in U_1, \ldots, u_k \in U_k, w_1 \in W_1, \ldots w_k \in W_k$ such that they form a biclique $K_{k,k}$.
- *(Soundness.)* for all $u_1 \in U_1, \ldots, u_k \in U_k, w_1 \in W_1, \ldots w_k \in W_k$, the vertex set $\{u_1, \ldots, u_k, v_1, \ldots, v_k\}$ induce a subgraph with at most $O(k^{2-1/(c \log k)}) \leq \varepsilon' k^2$ edges for some constant $0 < \varepsilon' < 1$.

We link all pairs of vertices which are on the same side but not in the same group. In the completeness case, we can find $2k$ vertices from distinct groups such that they form a clique and thus the number of edges induced is $\binom{2k}{2}$. In the soundness case, the number of edges induced by $2k$ vertices from distinct groups is at most $\varepsilon' k^2 + 2\binom{k}{2} < \varepsilon''\binom{2k}{2}$ for some $0 < \varepsilon'' < 1$.

At last, by Lemma 22, if Gap-clique$(k,\ \varepsilon k)$ does not admit $f(k) \cdot n^{O(\frac{k}{\log k})}$-time algorithms for some $0 < \varepsilon < 1$, then neither does Gap-biclique$(k,\ \frac{1+\varepsilon}{2}k)$, hence no FPT-algorithm can approximate Densest k-Subgraph to an ε'' factor. ◀

6 Conclusion

In this paper, we provide a tighter ETH-lower bound for constant gap k-Clique by replacing the Hardamard code used in [20] by the Reed-Muller Code with degree-2 polynomials. We use gap amplification techniques by expander graphs to rule out $k^{o(1)}$-ratio FPT-approximation algorithms for k-Clique under ETH. We also study the relationship between the constant gap k-Clique problem and PIH. We show that almost tight lower bounds for constant gap k-Clique can imply PIH.

A natural open question is whether we can derive such a lower bound for constant gap k-Clique under ETH. Formally, it is stated as follows:

Question 1. Assuming ETH, does constant gap k-Clique admit an algorithm in $f(k) \cdot n^{O(\frac{k}{\log k})}$ time?

It is also worth noting that assuming ETH, there is no $2^{o(n)}$-time algorithm for non-parameterized Max-Clique problem on n-vertex graphs. Hence, it is natural to ask whether our technique can be analogously applied to non-parameterized Max-Clique to obtain tight lower bounds:

Question 2. Assuming ETH, does constant gap Max-Clique admit an algorithm in $2^{o(n)}$ time?

References

1 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *European Symposium on Algorithms*, pages 1–12. Springer, 2014.

2 Miklós Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in LOGSPACE. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 132–140. ACM, 1987. doi:10.1145/28395.28410.

3 Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. Derandomized graph products. *Computational Complexity*, 5(1):60–75, 1995.

4 Mihir Bellare, Shafi Goldwasser, Carsten Lund, and Alexander Russell. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 294–304, 1993.

5 Mihir Bellare and Madhu Sudan. Improved non-approximability results. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 184–193, 1994.

6 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 743–754. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.74.

7 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Linear fpt reductions and computational lower bounds. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 212–221, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007352.1007391.

8 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

9 Aviad Cohen and Avi Wigderson. Dispersers, deterministic amplification, and weak random sources. In *30th Annual Symposium on Foundations of Computer Science*, pages 14–19. IEEE Computer Society, 1989.

10 Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*, 43(2):268–292, 1996.

11 Uriel Feige and Joe Kilian. Two-prover protocols—low error at affordable rates. *SIAM Journal on Computing*, 30(1):324–346, 2000.

12 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

13 Oded Goldreich, Howard J. Karloff, Leonard J. Schulman, and Luca Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. *Comput. Complex.*, 15(3):263–296, 2006. doi:10.1007/s00037-006-0216-3.

14 Shafi Goldwasser. Introduction to special section on probabilistic proof systems. *SIAM Journal on Computing*, 27(3):737, 1998.

15 Johan Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 627–636. IEEE, 1996.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62:367–375, 2001.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

18 R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.

19 Tamás Kõvári, Vera T. Sós, and Paul Turán. On a problem of k. zarankiewicz. *Colloquium Mathematicum*, 3:50–57, 1954.

20 Bingkai Lin. Constant approximating k-clique is w[1]-hard. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1749–1756. ACM, 2021. doi:10.1145/3406325.3451016.

21 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200. SIAM, 2020.

22 Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 3–13. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892006.

23 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

24 Karthik C. S. and Subhash Khot. Almost polynomial factor inapproximability for parameterized k-clique. *CoRR*, abs/2112.03983, 2021. arXiv:2112.03983.

25 C. Tovey. A simplified np-complete satisfiability problem. *Discret. Appl. Math.*, 8:85–89, 1984.

26 David Zuckerman. On unapproximable versions of np-complete problems. *SIAM J. Comput.*, 25(6):1293–1304, 1996. doi:10.1137/S0097539794266407.

27 David Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16(4/5):367–391, 1996. doi:10.1007/BF01940870.

28 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.

Backdoor Sets on Nowhere Dense SAT

Daniel Lokshtanov
University of California, Santa Barbara, CA, USA

Fahad Panolan
Indian Institute of Technology Hyderabad, India

M. S. Ramanujan
University of Warwick, Coventry, UK

Abstract

For a satisfiable CNF formula ϕ and an integer t, a *weak backdoor set* to treewidth-t is a set of variables such that there is an assignment to this set that reduces ϕ to a *satisfiable* formula that has an incidence graph of treewidth at most t. A natural research program in the work on fixed-parameter algorithms (FPT algorithms) for SAT is to delineate the tractability borders for the problem of detecting a small weak backdoor set to treewidth-t formulas. In this line of research, Gaspers and Szeider (ICALP 2012) showed that detecting a weak backdoor set of size at most k to treewidth-1 is W[2]-hard parameterized by k if the input is an arbitrary CNF formula. Fomin, Lokshtanov, Misra, Ramanujan and Saurabh (SODA 2015), showed that if the input is d-CNF, then detecting a weak backdoor set of size at most k to treewidth-t is fixed-parameter tractable (parameterized by k, t, d). These two results indicate that sparsity of the input plays a role in determining the parameterized complexity of detecting weak backdoor sets to treewidth-t.

In this work, we take a major step towards characterizing the precise impact of sparsity on the parameterized complexity of this problem by obtaining algorithmic results for detecting small weak backdoor sets to treewidth-t for input formulas whose incidence graphs belong to a *nowhere-dense graph class*. Nowhere density provides a robust and well-understood notion of sparsity that is at the heart of several advances on model checking and structural graph theory. Moreover, nowhere-dense graph classes contain many well-studied graph classes such as *bounded treewidth graphs, graphs that exclude a fixed (topological) minor and graphs of bounded expansion*.

Our main contribution is an algorithm that, given a formula ϕ whose incidence graph belongs to a fixed nowhere-dense graph class and an integer k, in time $f(t,k)|\phi|^{\mathcal{O}(1)}$, either finds a satisfying assignment of ϕ, or concludes correctly that ϕ has no weak backdoor set of size at most k to treewidth-t.

To obtain this algorithm, we develop a strategy that only relies on the fact that nowhere-dense graph classes are biclique-free. That is, for every nowhere-dense graph class, there is a p such that it is contained in the class of graphs that exclude $K_{p,p}$ as a subgraph. This is a significant feature of our techniques since the class of biclique-free graphs also generalizes the class of graphs of bounded degeneracy, which are *incomparable* with nowhere-dense graph classes. As a result, our algorithm also generalizes the results of Fomin, Lokshtanov, Misra, Ramanujan and Saurabh (SODA 2015) for the special case of d-CNF formulas as input when d is fixed. This is because the incidence graphs of such formulas exclude $K_{d+1,d+1}$ as a subgraph.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Fixed-parameter Tractability, Satisfiability, Backdoors, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.91

Category Track A: Algorithms, Complexity and Games

Funding *Daniel Lokshtanov*: Supported by United States-Israel Binational Science Foundation (BSF) grant no. 2018302 and National Science Foundation (NSF) award CCF-2008838.
Fahad Panolan: Supported by Seed grant, IIT Hyderabad (SG/IITH/F224/2020-21/SG-79).
M. S. Ramanujan: Supported by Engineering and Physical Sciences Research Council (EPSRC) grants EP/V007793/1 and EP/V044621/1.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 91; pp. 91:1–91:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A central concept in the design of fixed-parameter algorithms for SAT is the notion of *backdoor sets* introduced by Williams et al. [41], which was also introduced under a different name by Crama et al. [9]. These are sets of variables in the input formula whose instantiation can lead to a dramatic simplification of the formula, moving the formula into a class where SAT is easy to solve, such as *Horn formulas*. Backdoor sets provide a natural way to express the deviation of a SAT instance from well-studied "islands of tractability" (i.e., tractable cases of SAT) and allow us to study the question -"could we easily solve SAT instances that do not necessarily fall neatly within a known tractable fragment, yet are 'close' to them?"

To formalize this idea, fix a class of CNF formulas, call it $\mathcal{C}$, which is also called the *base* class. Let ϕ be the input instance of SAT. If ϕ is satisfiable, then a nonempty subset B of the variables in ϕ is a *weak backdoor* of ϕ to the class $\mathcal{C}$ (or a weak $\mathcal{C}$-backdoor set) if there exists an assignment τ to the variables in B such that the reduced instance $\phi[\tau]$ is a satisfiable formula in the class $\mathcal{C}$. We say that B is a *strong backdoor* of ϕ to the class $\mathcal{C}$ (or strong $\mathcal{C}$-backdoor set) if, for every assignment τ to the variables in B, the reduced instance $\phi[\tau]$ is in $\mathcal{C}$. Clearly, if $\mathcal{C}$ is a class of formulas on which SAT can be efficiently solved and one knows a weak/strong $\mathcal{C}$-backdoor set of the input instance, then a straightforward way of checking satisfiability and computing a satisfying assignment of the input instance involves exploring all instantiations of the variables in the backdoor set and efficiently solving the simplified instances.

Over the last two decades, there has been an active theoretical research program on SAT – the design of algorithms for SAT by exploring the *parameterized complexity* of computing and utilizing small backdoor sets to an efficiently-solvable class of CNF formulas. The study of the parameterized complexity of the problem of finding small backdoor sets was formally initiated by Nishimura et al. [33] and over the past decade and a half, there has been an extensive study of backdoors for SAT in the realm of parameterized complexity (see [37, 39, 35, 26, 13, 21] for a partial list). We refer to the survey of Gaspers and Szeider [25] for an overview of many of these works. In recent years, the success brought to the parameterized complexity of SAT by the study of backdoor detection has been replicated for CSP [2, 4, 20, 17, 18] and Mixed-ILP [16]. We refer to the survey [22] for a more comprehensive picture of the state of the art on backdoors to CSPs.

A basic part of this research program exploring the parameterized complexity of SAT is to answer the question:

For which input formulas is the detection of small backdoors to the class of bounded-treewidth formulas fixed-parameter tractable?

Bounded-treewidth formulas are CNF formulas whose incidence graphs have bounded treewidth and it is well-known that SAT is efficiently solvable on these [10, 15, 38]. The study of computing small backdoors to bounded-treewidth formulas from the parameterized complexity perspective was initiated by Gaspers and Szeider in [23] and further studied in [24, 26]. Gaspers and Szeider [23] gave an FPT-approximation algorithm for the problem of computing a strong backdoor set to Acyclic formulas, which was later extended to bounded-treewidth formulas [26]. Specifically, they gave an algorithm that runs in time $f(k)|\phi|^{\mathcal{O}(1)}$, for some computable function f, and either detects a strong backdoor in ϕ of size at most 2^k to bounded-treewidth formulas or reports that there is no such strong backdoor of size at most k. Since approximate backdoor sets are also sufficient to solve SAT, they were able to conclude that SAT parameterized by the size of a smallest strong backdoor set to

bounded-treewidth formulas is fixed-parameter tractable *with no restrictions on the input*. On the other hand, the picture for weak backdoor sets is much less comprehensive and is the main motivation behind our work.

State of the art for detecting weak backdoors

In [23], Gaspers and Szeider studied the problem of detecting a weak backdoor of size at most k to Acyclic formulas, i.e, formulas with incidence graphs of treewidth at most 1. They showed that this problem is W[2]-hard in general but fixed-parameter tractable (FPT) on d-CNF formulas for every fixed d. This implies that d-SAT parameterized by the size of a smallest weak backdoor set to Acyclic formulas is fixed-parameter tractable. This result was subsequently improved by Fomin et al. [13], who showed that d-SAT parameterized by the size of a smallest weak backdoor set to bounded-treewidth formulas is fixed-parameter tractable. The lack of progress in this direction since then, means that the state of the art on detecting weak backdoors to bounded-treewidth formulas continues to experience a huge gap between instances for which we know parameterized hardness results (i.e., general CNF formulas) and instances for which we have FPT algorithms (i.e., d-CNF). Where exactly does the tractability border lie? This brings us to the overarching goal of identifying maximal classes of CNF formulas for which SAT is FPT when parameterized by the size of a smallest weak backdoor set to the class of bounded-treewidth formulas.

Our work

Since the incidence graphs of d-CNF formulas only have linearly-many edges in the number of clauses, a natural step towards the overarching goal outlined above is to first understand the effect that the "sparsity" of the input incidence graph has on the tractability of the problem. In particular, could one show that as long as the input has bounded degeneracy or is nowhere dense [32], then SAT is FPT parameterized by the size of a smallest weak backdoor set to bounded-treewidth formulas? Nowhere density yields a robust notion of a maximal sparse graph class and nowhere-dense graph classes have been at the heart of major developments in graph theory and model checking [31, 27, 29] in the last decade. Therefore, one could reasonably hope to harness the techniques developed in these efforts, towards our purpose. Starting from this point, in this paper, we obtain tractability results that take us simultaneously beyond the class of nowhere-dense graphs and graphs of bounded degeneracy. Towards this, we initiate the study of weak backdoor detection on *biclique-free formulas*. More precisely, we consider $K_{d,d}$-free formulas, which are CNF formulas where no set of d clauses contain literals from d variables in their common intersection. In graph-theoretic terms, the incidence graphs of $K_{d,d}$-free formulas exclude the complete bipartite graph $K_{d,d}$ as a subgraph. $K_{d,d}$-free graphs are very wide class of sparse graphs and include many well-studied graph classes such as *bounded treewidth graphs, graphs that exclude a fixed minor, graphs of bounded expansion, nowhere-dense graphs and graphs of bounded degeneracy*. That is, for any of the classes – bounded treewidth graphs, graphs that exclude a fixed minor, graphs of bounded expansion, nowhere-dense graphs and graphs of bounded degeneracy, there is a p such that the class is contained in the class of $K_{p,p}$-free graphs (see Figure 1 for an illustration of the inclusion relation between these classes).

1.1 Our results

In order to present our results in the most general terms possible, we consider weak backdoor sets not just to bounded treewidth formulas, but what we call $\mathcal{F}$-minor-free formulas. Let $\mathcal{F}$ be a finite set of graphs. For a CNF formula ϕ, a set S of variables is called *a weak*

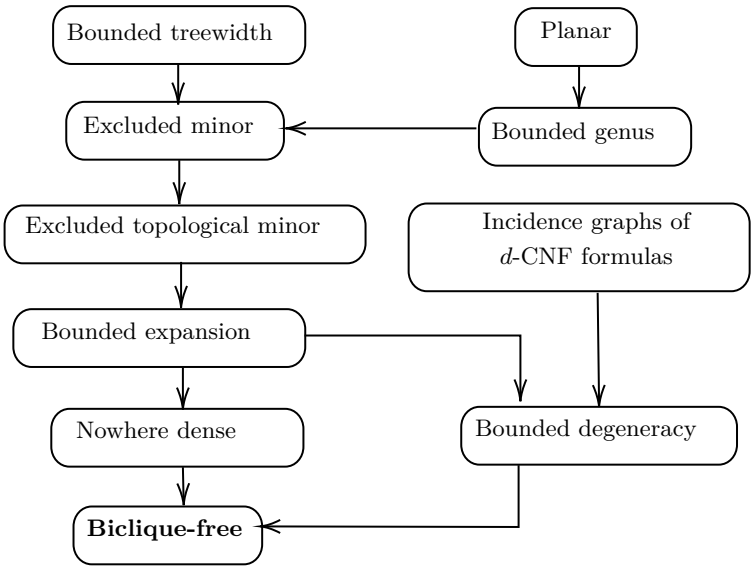


Figure 1 Inclusion relation between the class of biclique-free graphs considered in this paper and well-studied graph classes in the literature (figure based on [40]). If there in arrow of the form $A \to B$, then class A is a subclass of B.

$\mathcal{F}$-minor-free backdoor set of ϕ if it is a weak backdoor set of ϕ into the class of CNF formulas whose incidence graphs exclude every graph in $\mathcal{F}$ as a minor. Notice that a weak backdoor set of a formula into treewidth-t formulas is also a weak $\mathcal{F}$-minor-free backdoor set for some $\mathcal{F}$ – take $\mathcal{F}$ to be precisely the set of forbidden minors for graphs of treewidth at most t. Note that the size of $\mathcal{F}$ depends only on t. The reason for studying the problem in this more general setting is that it allows us to also obtain an additional result from the proof techniques used for our main result. Moreover, due to the extensive work in parameterized complexity on connections between vertex-deletion problems to bounded treewidth graphs and vertex-deletion problems to minor-free graphs, there is a rich collection of powerful algorithmic techniques that we are able to draw from (see, for example, [14, 19, 13]).

We now state our central result formally and discuss our corollaries and further context.

▶ **Theorem 1.** *For every $d \in \mathbb{N}$, there is a computable function f_d and an algorithm $\mathcal{A}_d$ that satisfies the following:*

1. *$\mathcal{A}_d$ takes as input a $K_{d,d}$-free formula ϕ, a family $\mathcal{F}$ of graphs containing at least one planar graph, and an integer k.*
2. *$\mathcal{A}_d$ runs in time $f_d(\mathcal{F}, k) \cdot |\phi|^{\mathcal{O}(d)}$.*
3. *$\mathcal{A}_d$ either outputs a weak $\mathcal{F}$-minor-free-backdoor set of ϕ of size at most k, or concludes correctly that ϕ has no weak $\mathcal{F}$-minor-free-backdoor set of size at most k.*

Let us first make a few remarks regarding the time-complexity of the algorithm described in the above statement. First of all, in the running time of the above algorithm, the exponent in the term $|\phi|^{\mathcal{O}(d)}$ is independent of k and $\mathcal{F}$. Secondly, as our main goal is to obtain a fixed-parameter tractability classification result, and as we have proved our result in such general terms, we have not attempted to optimize the function f_d. Indeed, due to the standard step of invoking Courcelle's theorem as a subroutine when the input treewidth is already bounded (see, for example, [23, 26]) one should not expect this algorithm to be practical *as presented*. However, our result is an important proof of concept demonstrating

fixed-parameter tractability for such a general class of inputs. We believe that optimizing the function f_d is a more meaningful research question for specific subclasses of $K_{d,d}$-free formulas, e.g., formulas with incidence graphs that exclude a fixed graph as a minor.

We now describe the main consequences of our result. As discussed, it is well-known that satisfiability of a t-tw formula (i.e., a CNF formula whose incidence graph has treewidth at most t) can be tested in linear time for every fixed t and moreover, a satisfying assignment can be computed in the same time. Therefore, we can combine Theorem 1 with the fact that graphs of treewidth at most t have a finite and computable set of forbidden minors that include the $(t+1) \times (t+1)$-grid, which is a planar graph. This leads us to the first FPT algorithm for SAT on $K_{d,d}$-free formulas (for fixed d) parameterized by $t + \mathsf{wb}_t(\phi)$. Here, $\mathsf{wb}_t(\phi)$ denotes the size of a smallest weak t-tw-backdoor set of ϕ. The formal statement follows.

▶ **Theorem 2.** *For every $d \in \mathbb{N}$, there is a computable function f_d and an algorithm that, given $t \in \mathbb{N}$, a $K_{d,d}$-free formula ϕ and an integer k, runs in time $f_d(t,k)|\phi|^{\mathcal{O}(d)}$ and either finds a satisfying assignment of ϕ, or concludes correctly that ϕ has no weak t-tw-backdoor set of size at most k.*

An immediate corollary of the above theorem is that d-SAT is FPT parameterized by $t + \mathsf{wb}_t(\phi)$ for every fixed d. To obtain this, notice that a d-CNF formula is $K_{d+1,d+1}$-free.

A further consequence of the techniques used to prove Theorem 1 is in the case of weak backdoor sets to *nested formulas*. These form a subclass of $K_{3,3}$-free formulas and are known to have treewidth at most 3 (see preliminaries for the formal definition of nested formulas). However, this is not a precise characterization of these formulas, so we need a minor modification in the proof of Theorem 1 to obtain the following result.

▶ **Theorem 3.** *For every $d \in \mathbb{N}$, there is a computable function f_d and an algorithm that, given $t \in \mathbb{N}$, a $K_{d,d}$-free formula ϕ and an integer k, runs in time $f_d(t,k)|\phi|^{\mathcal{O}(d)}$ and either finds a satisfying assignment of ϕ, or concludes correctly that ϕ has no weak backdoor set of size at most k to the class of nested formulas.*

Hardness results for strong and weak backdoor detection

To complement our algorithmic results, we also show new hardness results that strengthen existing results. Towards this, we present a polynomial-time algorithm that, given a Set Cover instance $I = (U, \mathcal{S})$, numbers $d, d' \geq 2$ and a non-negative number k, outputs a CNF formula ϕ such that if I has a set cover of size k, then ϕ has a strong/weak backdoor set of size k to the class of empty formulas, which are formulas with no clauses. Conversely, if ϕ has a strong/weak backdoor set of size k to any subclass of $\mathcal{K}_{d,d'}$-free formulas, then I has a set cover of size k. This algorithm is effectively a reduction that implies that for any class $\mathcal{C}$ that is a superclass of empty formulas and a subclass of $\mathcal{K}_{d,d}$-free formulas for some $d \in \mathbb{N}$, the problem of detecting a strong/weak backdoor to $\mathcal{C}$ of size at most k is at least as hard as solving Set Cover parameterized by the solution size (i.e., it is W[2]-hard).

In particular, notice that for every $t \geq 0$, there exists a $d \geq 2$ such that t-tw-CNF is a subclass of $K_{d,d}$-free formulas. Moreover, nested formulas are subclasses of $K_{3,3}$-free formulas. Hence, we obtain the following consequences.

▶ **Theorem 4.** *For every $t \geq 0$, family $\mathcal{F}$ that contains a planar graph but does not contain an independent set, and for each class $\mathcal{C} \in \{t$-tw formulas, nested formulas, $\mathcal{F}$-minor-free formulas$\}$, the problem of detecting a strong/weak backdoor set to $\mathcal{C}$ of size at most k is $W[2]$-hard parameterized by k.*

Table 1 Our results for weak backdoor detection in relation to the state of the art. Each row corresponds to a base class into which we are finding a backdoor set. Each column corresponds to a possible class of inputs.

	Input: d-CNF	Input: $K_{d,d}$-free CNF	Input: CNF
Acyclic	FPT [23]	**FPT (Theorem 2)**	W[2]-hard [23]
Nested	**FPT**	**FPT (Theorem 3)**	W[2]-hard (**Theorem 4**)
tw-t (for every t)	FPT [13]	**FPT (Theorem 2)**	W[2]-hard (**Theorem 4**)
Planar-$\mathcal{F}$ minor-free	**FPT**	**FPT** (Theorem 1)	W[2]-hard (**Theorem 4**)

This theorem strengthens the hardness result of Gaspsers and Szeider [23] and shows that not only is it W[2]-hard to detect small weak backdoor sets (with no restriction on the input) to Acyclic formulas, but also to treewidth-t formulas for every fixed t.

2 Preliminaries

CNF Formulas

We consider propositional formulas in conjunctive normal form (CNF). In our algorithms to compute weak backdoor sets to bounded-treewidth formulas, we do not assume that the clauses exclude a pair of complementary literals. However, when the goal is to solve SAT, it can be assumed without loss of generality that the input formula does not contain a clause with a pair of complementary literals. For a CNF formula ϕ, we use $\mathsf{var}(\phi)$ and $\mathsf{cla}(\phi)$ to refer to the sets of variables and clauses in ϕ, respectively. We say that a variable x is positive (negative) in a clause C if $x \in C$ ($\overline{x} \in C$), and we write $\mathsf{var}(C)$ for the set of variables that are positive or negative in C, while we use $\mathsf{lit}(C)$ to denote the set of literals in C. For a set $\mathcal{C}$ of clauses, we use $\mathsf{var}(\mathcal{C})$ to denote $\bigcup_{C\in\mathcal{C}} \mathsf{var}(C)$ and call this set *the set of variables of* $\mathcal{C}$. Given a CNF formula ϕ and a truth assignment τ, $\phi[\tau]$ denotes the *truth assignment reduct* of ϕ under τ, which is the CNF formula obtained from ϕ by first removing all clauses that are satisfied by τ and second removing from the remaining clauses all literals $x, \overline{x}$ with $x \in \mathsf{var}(\tau)$. The assigned variables are also removed.

Incidence graphs

The *incidence graph* of a CNF formula ϕ, $\mathsf{inc}(\phi)$, is the bipartite graph whose vertices are the variables and clauses of ϕ, and where vertices corresponding to a variable x and a clause C are adjacent if and only if $x \in \mathsf{var}(C)$. Further, an edge between a vertex corresponding to $x \in \mathsf{var}(\phi)$ and $C \in \mathsf{cla}(\phi)$ has the *polarity* (or *label*) $+$ if $x \in \mathsf{lit}(C)$ and $\overline{x} \notin \mathsf{lit}(C)$ and is labeled $-$ if $\overline{x} \in \mathsf{lit}(C)$ and $x \notin \mathsf{lit}(C)$. If $x, \overline{x} \in \mathsf{lit}(C)$, then this edge is labeled $\mathfrak{b}$. For an incidence graph G, we abuse notation and use $\mathsf{var}(G)$ to refer to the vertices of G that correspond to variables in $\psi(G)$, and $\mathsf{cla}(G)$ to refer to the vertices of G that correspond to clauses in $\psi(G)$. Also, for a vertex subset $Y \subseteq V(G)$, we continue to use the notations $\mathsf{var}(Y)$ and $\mathsf{cla}(Y)$ to refer to the sets $\mathsf{var}(G) \cap Y$ and $\mathsf{cla}(G) \cap Y$, respectively. We say that a graph is $K_{d,d}$*-free* if it does not contain $K_{d,d}$ as a subgraph. Notice that the $K_{d,d}$-free incidence graphs correspond to CNF formulas where no set of d clauses contain literals from d distinct variables in their common intersection.

Nested formulas

A CNF formula ϕ is *nested* [28] precisely if the graph $\mathsf{inc}(\mathsf{univ}(\phi))$ is planar, where $\mathsf{univ}(\phi)$ is obtained from ϕ by adding any universal clause $c^\star$ containing a literal from each variable of ϕ. Moreover, $\mathsf{inc}(\phi)$ has treewidth at most 3 if ϕ is nested. Hence, one can determine satisfiability of nested formulas in polynomial time. Nested formulas are a subclass of planar CNF formulas and so, are also a subclass of $K_{3,3}$-free formulas.

Treewidth

Let G be a graph. A *tree decomposition* of G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ where T is a tree and $\mathcal{X}$ is a collection of subsets of $V(G)$ such that: (i) $\forall e = (u, v) \in E(G)$, there is $t \in V(T) : \{u, v\} \subseteq X_t$, and (ii) $\forall v \in V(G)$, $T[\{t \mid v \in X_t\}]$ is a non-empty connected subtree of T. We call the vertices of T *nodes* and the sets in $\mathcal{X}$ *bags* of the tree decomposition $(T, \mathcal{X})$. The *width* of $(T, \mathcal{X})$ is equal to $\max\{|X_t| - 1 \mid t \in V(T)\}$ and the *treewidth* of G is the minimum width over all tree decompositions of G. A *nice tree decomposition* is a pair $(T, \mathcal{X})$ where $(T, \mathcal{X})$ is a tree decomposition such that T is a rooted tree and the following conditions are satisfied: (i) Every node of the tree T has at most two children; (ii) if a node t has two children t_1 and t_2, then $X_t = X_{t_1} = X_{t_2}$; and (iii) if a node t has one child t_1, then either $|X_t| = |X_{t_1}| + 1$ and $X_{t_1} \subset X_t$ (in this case we call t_1 an *introduce node*) or $|X_t| = |X_{t_1}| - 1$ and $X_t \subset X_{t_1}$ (in this case we call t_1 a *forget node*). It is possible to transform a given tree decomposition $(T, \mathcal{X})$ into a nice tree decomposition $(T', \mathcal{X}')$ in time $O(|V| + |E|)$ [3]. A set $X \subseteq V(G)$ is a treewidth-t modulator for a graph G if the treewidth of $G - X$ is at most t.

Boundaried graphs

A t-boundaried graph is a triple (G, B, ℓ_G) where G is a graph, $B \subset V(G)$ of size t with each vertex $v \in B$ having a unique label $\ell_G(v) \in \{1, \ldots, t\}$. We refer to B as the *boundary* of G. We often refer to a t-boundaried graph simply as G with the function $\partial(G)$ returning the boundary of G and the function ℓ_G denoting the labeling function on the boundary. In other cases, it will be useful to mention the boundary explicitly, in which case we will use (G, B). When the size of the boundary is irrelevant to the discussion, we simply use "boundaried graph" instead of "t-boundaried graph".

Let $G = (X, C, E)$ be a boundaried incidence graph with boundary $\partial(G) = \{b_1, \ldots, b_{|\partial(G)|}\}$, where $\partial(G) \subseteq (X \cup C)$ and $b_i = \ell_G^{-1}(i)$. We use $\vartheta(G)$ to denote the $|\partial(G)|$-length binary word given as: $\vartheta(G)[i] = 0$ if $b_i \in X$ and $\vartheta(G)[i] = 1$ otherwise.

▶ **Definition 5** (Boundary types and mergeability). *We say that two boundaried incidence graphs $G_1 = (X_1, C_1, E_1)$ and $G_2 = (X_2, C_2, E_2)$ have the same* boundary type *if $\vartheta(G_1) = \vartheta(G_2)$. If G_1 and G_2 have the same boundary type, then we say that G_1 and G_2 are* mergeable *if for every u_1, $v_1 \in \partial(G_1)$ and u_2, $v_2 \in \partial(G_2)$ such that $\ell_{G_1}(u_1) = \ell_{G_2}(u_2)$ and $\ell_{G_1}(v_1) = \ell_{G_2}(v_2)$, at most one out of (u_1, v_1) and (u_2, v_2) is an edge in G_1 and G_2 respectively.*

Gluing operation and compatibility of boundaried graphs

A pair of boundaried incidence graphs G_1 and G_2 that are mergeable can be "glued" together along their boundaries to obtain a new incidence graph, which we denote by $G_1 \oplus G_2$. The gluing operation takes the disjoint union of G_1 and G_2 and identifies the vertices of $\partial(G_1)$ and $\partial(G_2)$ with the same label. If there are vertices u_1, $v_1 \in \partial(G_1)$ and u_2, $v_2 \in \partial(G_2)$ such that $\ell_{G_1}(u_1) = \ell_{G_2}(u_2)$ and $\ell_{G_1}(v_1) = \ell_{G_2}(v_2)$ then G has vertices u formed by unifying u_1 and u_2 and v formed by unifying v_1 and v_2. That is, u_1 and u_2 are merged to form the

supernode u and v_1 and v_2 are merged to form the supernode v. The new vertices u and v are adjacent if $(u_1, v_1) \in E(G_1)$ or $(u_2, v_2) \in E(G_2)$. The polarities of the edges in G_1 and G_2 are inherited in $G_1 \oplus G_2$ in the natural way.

▶ **Definition 6** (d-compatibility of boundaried graphs)**.** *Let G_1 and G_2 be two t-boundaried $K_{d,d}$-free incidence graphs. We say that G_1 and G_2 are* d-compatible *if they are mergeable and further, $G_1 \oplus G_2$ is also a $K_{d,d}$-free graph.*

Minors and nowhere-dense graph classes

Given an edge $e = (x, y)$ of a graph G, the graph G/e is obtained from G by contracting the edge e, that is, the endpoints x and y are replaced by a new vertex v_{xy} which is adjacent to the original neighbors of x and y (except x and y). A graph H obtained by a sequence of edge-contractions is said to be a *contraction* of G. We denote it by $H \leq_c G$. A graph H is a *minor* of a graph G if H is the contraction of some subgraph of G and we denote it by $H \leq_m G$. We say that a graph G is *H-minor-free* when it does not contain H as a minor. We also say that a graph class $\mathcal{G}$ is *H-minor-free* (or, excludes H as a minor) when all its members are H-minor-free. It is well-known [36] that if $H \leq_m G$ then $tw(H) \leq tw(G)$.

The notion of *shallow minors* is required to define nowhere-dense graph classes. The radius of a connected graph G is the minimum over all vertices v of G of the maximum distance between v and another vertex. For non-negative integer r, a graph H is a shallow minor at depth r of a graph G if there exists a subgraph X of G whose connected components have radius at most r, such that H is a simple graph obtained from G by contracting each component of X into a single vertex and then taking a subgraph.

We now recall the notion of *nowhere-dense* graph classes. Although the only property of a nowhere-dense graph class we require is that they are biclique-free, we give the formal definition here for the sake of completeness.

▶ **Definition 7.** *A class $\mathcal{C}$ of graphs is nowhere dense if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $r \geq 0$, $K_{f(r)}$ is* not *a shallow minor at depth r of the graph G, for every $G \in \mathcal{C}$.*

▶ **Proposition 8.** (see Fact 1, [40]) *Let $\mathcal{C}$ be a nowhere-dense class of graphs and let f be the corresponding function as described in the above definition. Then, there is a number t depending only on f such that for any $G \in \mathcal{C}$, G does not contain $K_{t,t}$ as a subgraph.*

Unbreakability

Roughly speaking, a graph is unbreakable if it is not possible to "break" it into two large parts by removing only a small number of vertices. We now give the formal definition. A pair (X, Y) where $X \cup Y = V(G)$ is a *separation* if there is no edge with one endpoint in $X \setminus Y$ and the other in $Y \setminus X$. The order of (X, Y) is $|X \cap Y|$. If there exists a separation (X, Y) of order at most c such that $|X \setminus Y| \geq s$ and $|Y \setminus X| \geq s$, called an *(s, c)-witnessing separation*, then G is *(s, c)-breakable*. Otherwise, G is *(s, c)-unbreakable*. The following lemma states that it is possible to determine (approximately) whether a graph is unbreakable or not using a fixed-parameter algorithm, and lemmas similar to it can be found in [5, 30].

▶ **Proposition 9** ([30])**.** *There exists an algorithm that, given $s, c \in \mathbb{N}$ and a graph G, runs in time $2^{O(c(s+c))} \cdot n^3 \log n$ and either returns an $(\frac{s}{2^c}, c)$-witnessing separation or correctly concludes that G is (s, c)-unbreakable.*

▶ **Observation 10.** *Let $t, \delta, \tau^\star \in \mathbb{N}$, where $t \leq 2\tau^\star$. Let (G, Z) be a t-boundaried graph and let (X, Y) be a $(\delta, \tau^\star)$-witnessing separation. Then, one of the pairs $(G[X], (X \cap Y) \cup (Z \cap X))$ or $(G[Y], (X \cap Y) \cup (Z \cap Y))$ is a t'-boundaried graph, where $t' \leq 2\tau^\star$.*

By repeatedly invoking the above observation, one can "zoom in" to a separation of small order in the graph such that one of the sides is large and induces an unbreakable subgraph.

▶ **Lemma 11.** *Let $\delta, \delta^\star, \tau^\star \in \mathbb{N}$ be such that $\delta^\star = 2^{\tau^\star} \cdot \delta$. Let (X, Y) be a separation of order at most $2\tau^\star$ in a graph G such that $|X| \geq \delta$. There is an algorithm that, given G, X, Y, runs in time $2^{\mathcal{O}(\tau^\star \cdot \delta^\star)} \cdot n^{\mathcal{O}(1)}$ and returns a separation $(\widetilde{X}, \widetilde{Y})$ of order at most $2\tau^\star$ in G such that $|\widetilde{X}| \geq \delta$ and $G[\widetilde{X}]$ is $(\delta^\star, \tau^\star)$-unbreakable.*

▶ Remark 12. Roughly speaking, the notions of boundaried incidence graphs, mergeability, and d-compatibility allows us to split our original input (formula's) incidence graph along a small balanced separator (of size at most some t) and treat the two sides of the separator as t-boundaried d-compatible graphs and focus our attention on one of these two graphs in order to make further progress. The notion of unbreakability plus Lemma 11 gives us a kind of "base case" where we are able to stop splitting our instance along such small separators and use the additional structure imposed by unbreakability to make progress.

3 Algorithm for Weak Planar-$\mathcal{F}$-minor-free-Backdoor Detection

In this section, we discuss the proof of Theorem 1. We begin with the following definitions and assumptions.

- Recall that d is fixed, i.e., it is a constant for us.
- We assume that $k > d$. Otherwise, we could simply guess (and verify) a weak-backdoor set of size at most k in time $|\phi|^{\mathcal{O}(d)}$. Here, k is the integer taken as input by the algorithm $\mathcal{A}$ described in Theorem 1.
- We assume that $\mathcal{F}$ is a family of graphs containing at least one planar graph and use η to denote a computable upper bound on the treewidth of $\mathcal{F}$-minor-free graphs.
- Define $\tau(k, d) = k^{d+1}$, $\tau'(\eta, k, d) = \tau(k, d) + 2\eta + 2$.
- We pick $\delta(\eta, k, d)$ to be a large enough function of $\tau'(\eta, k, d)$ to be decided later.
- Set $\delta'(\eta, k, d) = 2^{\tau'(\eta,k,d)} \cdot \delta(\eta, k, d)$.
- Let $\Delta(\eta, k, d) = \delta'(\eta, k, d) + 2\tau'(\eta, k, d)$.
- For ease of readability, we omit the arguments of these functions when they are clear from the context, e.g., we replace $\Delta(\eta, k, d)$ with Δ.

3.1 Overview of our algorithm

We first briefly describe the ideas used by Fomin et al. [13] in their randomized FPT algorithm to detect weak η-tw-backdoor sets when the input formula is d-CNF. We then outline the similarities and main differences with our algorithm.

Algorithm of Fomin et al [13]

Let G be the input incidence graph. Say that a set $X \subseteq V(G)$ is an η-modulator if $\mathsf{tw}(G - X) \leq \eta$. Fomin et al. observed that if X is a set of variables that form a weak η-tw-backdoor, then the set $N(X)$ of their neighbors in the incidence graph is an η-modulator. Note that $|N(X)|$ could be arbitrarily large compared to $|X|$. They then give a preprocessing procedure that ensures that for every η-tw-backdoor set X, the set $N(X)$ is incident to a large fraction of the edges in G. Therefore, if one picks an edge uniformly at random, then the clause endpoint of the edge belongs to $N(X)$ with constant probability. Further, since the clauses are of constant size d, we have that a randomly chosen variable from the clause belongs to X with a constant probability, some $f(d, \eta)$. The algorithm then simply branches

on the chosen variable x in the usual way – in one branch, the formula is simplified by setting x to 1 and recurse, and in the other branch, x is set to 0 and the residual instance is recursed upon.

The main obstacle in our setting is that simply obtaining a clause in $N(X)$ is not good enough since clauses can be arbitrarily large. This motivates us to look for a "domination core" within the incidence graph. That is, if we find sufficiently many clause vertices in $N(X)$, then it is possible to identify a small subset of variables that intersect X. However, being able to locate the required number of clause vertices in $N(X)$ is far from obvious even though we have access to the preprocessing rule of Fomin et al. For instance, each time we locate a vertex in $N(X)$ and remove it, the preprocessing rule could kick in again and even remove vertices of X. A similar issue arises in the recent work of Choudhary et al. [6] for Feedback Vertex Set on linear hypergraphs (whose incidence graphs are $K_{2,2}$-free). As a result, they needed to design problem specific preprocessing rules that heavily utilize the fact that they are dealing with deletion to graphs of treewidth 1. Moreover, their approach only works when the input incidence graph is $K_{2,2}$-free and extending it to our far more general setting appears to be a challenging task.

Our approach

To overcome the obstacle described above, we take a different route and use the notion of unbreakability [5] to argue that if we "zoom into" a subgraph that is large enough, has high treewidth, is separated from the rest of the graph by a small boundary and is unbreakable, then it is possible to identify sufficiently many vertices from $N(X)$ and a *domination core* can be extracted from this [34, 11, 12]. This is the main technical contribution of the paper and we believe that this approach can find further applications.

Our algorithm relies on Lemma 11 and Lemmas 13-16. Out of these, the next three lemmas correspond to the three main cases that we will encounter in our algorithm. In the following three lemmas, let $G^\star$ denote the input incidence graph, let G and H be $K_{d,d}$-free t-boundaried d-compatible incidence graphs ($t \leq 2\tau'$) with the same boundary type such that $G^\star = G \oplus H$. Moreover, suppose that $|\partial(G)| \leq 2\tau'$ and G is (δ', τ')-unbreakable. Recall that $\partial(G)$ and $\partial(H)$ denote the boundaries of G and H respectively. Moreover, we use n to denote the number of vertices in the incidence graph under consideration.

The first lemma states that if G has sufficiently high tree-width, then it is possible to compute a small family of clause-sets, each containing few clauses, such that if $G^\star$ contains a small variable set $S^\star$ such that deleting its neighborhood reduces the treewidth of G significantly, then in at least one of these clause-sets, every clause contains a variable of $S^\star$. In graph-theoretic terms, it shows that one can compute a small family of small clause-vertex sets, such that in at least one of these small clause-vertex sets, every vertex is adjacent to $S^\star$.

▶ **Lemma 13** (Case 1: High treewidth piece)**.** *Suppose* $\mathsf{tw}(G) > \Delta$*. Let* $S^\star \subseteq \mathsf{var}(G^\star)$ *be such that* $|S^\star| \leq k$ *and* $\mathsf{tw}(G^\star - N_{G^\star}[S^\star]) \leq \eta$*. There is an algorithm that, given* $G^\star$*,* G *and* H *as input, runs in time* $(\delta')^{\mathcal{O}(d\tau')}n^{\mathcal{O}(1)}$*, and outputs a family* $\mathscr{Q}$ *of subsets of* $\mathsf{cla}(G^\star)$ *of size* τ *each, with the following properties.*

1. $|\mathscr{Q}| = (\delta')^{\mathcal{O}(d\tau')}$*, and*
2. *there exists* $\mathcal{Q} \in \mathscr{Q}$ *such that each clause in* $\mathcal{Q}$ *contains a variable from* $S^\star$*.*

The following lemma is used to handle a situation similar to that above, with the only difference being that the treewidth of G is not too high. On the other hand, it is also high enough to ensure that any weak $\mathcal{F}$-minor-free-backdoor set of $G^\star$ intersects G in at least one vertex. The proof of this lemma uses Arnborg et al.'s extension [1] of Courcelle's Theorem and the fact that the existence of a small weak $\mathcal{F}$-minor-free-backdoor set is CMSO-expressible (see, for example, [23]). We refer the reader to [7, 1, 8] for a detailed introduction to CMSO.

▶ **Lemma 14** (Case 2: Moderately-high treewidth piece). *Suppose that* $\eta < \mathsf{tw}(G - \partial(G)) \leq \Delta$. *We can compute a set* $S \subseteq \mathsf{var}(G)$ *in time* $g(\eta, k, d) \cdot n^{\mathcal{O}(1)}$ *for some computable function* g, *such that the following hold.*

1. S *has size at most* $g(\eta, k, d)$, *and*
2. S *has a non-empty intersection with a weak* $\mathcal{F}$*-minor-free-backdoor set of size at most* k *for* $G \oplus H$ *(if one exists).*

The next lemma shows that if G has low-treewidth but has a large number of vertices, then one can obtain a strictly smaller, equivalent instance by closely following the graph-replacement arguments used by Fomin et al. [13], which in turn was inspired by ideas used in [19] for kernelization.

▶ **Lemma 15** (Case 3: Low treewidth piece). *Suppose that* $|V(G)| \geq \delta$ *and* $\mathsf{tw}(G - \partial(G)) \leq \eta$. *Then, there is a* $K_{d,d}$*-free incidence graph* G' *such that the following hold.*

1. $|V(G')| < |V(G^\star)|$.
2. *For every* $\gamma \leq k$, $\psi(G^\star)$ *has a weak* $\mathcal{F}$*-minor-free-backdoor set* S *of size at most* γ *if and only if* $\psi(G')$ *has a weak* $\mathcal{F}$*-minor-free-backdoor set* S' *of size at most* γ.
3. *Given* $G^\star, G$ *and* H, *one can obtain* G' *in time* $\widehat{g}(\mathcal{F}, k, d) \cdot n^{\mathcal{O}(1)}$ *for some computable function* $\widehat{g}$.

Having stated the three central lemmas that we will be using, we next demonstrate an immediate application of Lemma 13, following which we give the proof of Theorem 1 assuming the correctness of the three central lemmas.

Recall that Lemma 13 provides us a family of subsets of $\mathsf{cla}(G^\star)$ such that there is one subset where each clause contains a variable from S. In the following lemma we explain how to get a variable in S from such a family. The proof the lemma uses the concept of *domination core*, which is used in the study of parameterized complexity of dominating set on $K_{d,d}$-free graphs [34, 11, 12]. Recall that $k > d$.

▶ **Lemma 16** (Domination core). *There is an algorithm that, given a set* $Z_C \subseteq \mathsf{cla}(\phi)$ *of* τ *clauses of a* $K_{d,d}$*-free formula* ϕ, *each of whose variable set intersects some fixed variable set* $S \subseteq \mathsf{var}(\phi)$ *of size at most* k, *runs in polynomial time and outputs a set* Z_V *of at most* d *variables that intersects* S.

Proof. Let $Z_C^0 = Z_C$. We define $v_1, \ldots, v_r \in \mathsf{var}(\phi)$ and $\mathcal{C}_1, \ldots, \mathcal{C}_r \subseteq Z_C$ as follows. For every $i \in [r]$, we select v_i to be the lexicographically first variable disjoint from the previously selected variables, that occurs most frequently among the clauses in the set Z_C^{i-1}. We set $Z_C^i := N_{\mathsf{inc}(\phi)}(v_i) \cap Z_C^{i-1}$. We now observe that for every $i \in [d]$, either S intersects $\{v_1, \ldots, v_i\}$ or $|Z_C^i| \geq |Z_C^{i-1}|/k$. This is because S has size at most k and every clause in Z_C contains a variable of S. In particular, we have that either S intersects $Z_V = \{v_1, \ldots, v_d\}$ or it must be the case that $|Z_C^d| \geq k$ (since $|Z_C^0| \geq k^{d+1}$), implying the existence of a $K_{d,d}$ in $\mathsf{inc}(\phi)$, which is a contradiction. This completes the proof of the lemma. ◀

We are now ready to complete the proof of Theorem 1 assuming Lemmas 11-16. We restate it here for the reader's convenience.

▶ **Theorem 1.** *For every* $d \in \mathbb{N}$, *there is a computable function* f_d *and an algorithm* $\mathcal{A}_d$ *that satisfies the following:*

1. $\mathcal{A}_d$ *takes as input a* $K_{d,d}$*-free formula* ϕ, *a family* $\mathcal{F}$ *of graphs containing at least one planar graph, and an integer* k.
2. $\mathcal{A}_d$ *runs in time* $f_d(\mathcal{F}, k) \cdot |\phi|^{\mathcal{O}(d)}$.
3. $\mathcal{A}_d$ *either outputs a weak* $\mathcal{F}$*-minor-free-backdoor set of* ϕ *of size at most* k, *or concludes correctly that* ϕ *has no weak* $\mathcal{F}$*-minor-free-backdoor set of size at most* k.

Proof. We focus on solving the decision version of the problem. This is because, if the input is a yes-instance, then a weak $\mathcal{F}$-minor-free backdoor set of size at most k can be computed using self-reducibility. Let $I = (\phi, \mathcal{F}, k)$ denote the input instance and let G be the incidence graph of ϕ. Suppose that G has treewidth at most Δ. Then, one can use Courcelle's theorem to solve the problem (see, for example, [23, 26]). Hence, we may assume that G has treewidth $> \Delta$. In particular, $|V(G)| > \Delta$. Now, using Proposition 9, we either conclude that G is (δ', τ')-unbreakable or we compute a (δ, τ')-witnessing separation.

In the first case, we define $\partial(G) = \emptyset$ and H to be an edgeless graph with $\partial(H) = \emptyset$. We then invoke Lemma 13 to compute a set $\mathscr{Q}$ of size $\delta'^{\mathcal{O}(d\tau')}$ comprising sets of τ clauses each such that for some $\mathcal{Q} \in \mathscr{Q}$, each clause in $\mathcal{Q}$ contains a variable in some fixed weak backdoor set $S^\star$ (if one exists). Recall that $\mathsf{inc}(\phi) - N_{\mathsf{inc}(\phi)}[S^\star]$ has treewidth at most η, satisfying the premise of Lemma 13. We then invoke Lemma 16 on every $\mathcal{Q} \in \mathscr{Q}$ to compute a set Z_V of at most $d \cdot \delta'^{\mathcal{O}(d\tau')}$ variables that intersects $S^\star$. For every $x \in Z_V$ and $\alpha \in \{0,1\}$, we recursively solve the problem on the instance $I_{x,\alpha} = (\phi[x = \alpha], k-1)$. We conclude that I is a yes-instance if and only if there is an $x \in Z_V$ and $\alpha \in \{0,1\}$ such that $I_{x,\alpha}$ is a yes-instance.

Now, consider the second case, i.e., the case where we compute a (δ, τ')-witnessing separation (X, Y) for G. Notice that this separation satisfies the premises of Lemma 11 (by taking the same δ, taking $\delta^\star$ be δ' and $\tau^\star$ to be τ'). Hence, we execute the algorithm of this lemma to compute a separation $(\widetilde{X}, \widetilde{Y})$ of order at most $2\tau'$ in G such that $|\widetilde{X}| \geq \delta$ and $G[\widetilde{X}]$ is (δ', τ')-unbreakable. Let $\widetilde{G} = G[\widetilde{X}]$ and let $\widetilde{H} = G[\widetilde{Y}]$. At this point, we have that $|V(\widetilde{G})| \geq \delta$, $|\partial(\widetilde{G})| \leq 2\tau'$ and $\widetilde{G}$ is (δ', τ')-unbreakable.

Consider the following three cases depending on whether the "piece" $\widetilde{G} - \partial(\widetilde{G})$ has high treewidth or moderately-high treewidth or low treewidth. In each case, we will either strictly reduce the size of the instance or compute a bounded set of variables upon which to branch.

Case 1: $\mathsf{tw}(\widetilde{G} - \partial(\widetilde{G})) > \Delta$. We invoke Lemma 13 with $G := \widetilde{G}$ and $H := \widetilde{H}$ to compute $\mathscr{Q}$ and then Lemma 16 on the sets in $\mathscr{Q}$ as described above. This gives us a set Z_V of at most $d \cdot \delta'^{\mathcal{O}(d\tau')}$ variables that intersects a weak $\mathcal{F}$-minor-free-backdoor set $S^\star$ of ϕ of size at most k. Now, for every $x \in Z_V$ and $\alpha \in \{0,1\}$, we recursively solve the problem on the instance $I_{x,\alpha} = (\phi[x = \alpha], \mathcal{F}, k-1)$. We conclude that I is a yes-instance if and only if there is an $x \in Z_V$ and $\alpha \in \{0,1\}$ such that $I_{x,\alpha}$ is a yes-instance.

Case 2: $\eta < \mathsf{tw}(\widetilde{G} - \partial(\widetilde{G})) \leq \Delta$. We invoke Lemma 14 with $G := \widetilde{G}$ and $H := \widetilde{H}$ to compute a set $S \subseteq \mathsf{var}(G)$ of size at most $g(\eta, k, d)$ that has a non-empty intersection with a weak $\mathcal{F}$-minor-free-backdoor set of G of size at most k (if one exists). For every $x \in S$ and $\alpha \in \{0,1\}$, we recursively solve the problem on the instance $I_{x,\alpha} = (\phi[x = \alpha], \mathcal{F}, k-1)$. We conclude that I is a yes-instance if and only if there is an $x \in S$ and $\alpha \in \{0,1\}$ such that $I_{x,\alpha}$ is a yes-instance.

Case 3: $\mathsf{tw}(\widetilde{G} - \partial(\widetilde{G})) \leq \eta$. We invoke Lemma 15 with $G := \widetilde{G}$ and $H := \widetilde{H}$ to compute an equivalent instance $(\phi', \mathcal{F}, k)$ such that $|\phi'| < |\phi|$ and recurse on $(\phi', \mathcal{F}, k)$.

This completes the description of our algorithm. Each step of the algorithm has running time $\chi(\eta, d, k)$ for some computable function χ and at the end of each, we either make $2d \cdot \delta'^{\mathcal{O}(d\tau')}$ recursive calls with a strictly smaller budget k or strictly reduce the size of the instance. Hence, the running time follows. The correctness is derived from that of the three main lemmas (Lemmas 13, 14, 15) and the domination core lemma (Lemma 16). ◀

It remains to prove the three main lemmas. Lemma 13 captures the new technical insight at the heart of this work, and so we focus on that lemma in this extended abstract. We also give a proof sketch of Lemma 14 and as the proof of Lemma 15 closely mirrors the algorithm of Fomin et al. [13], we omit the details of this lemma in this extended abstract.

3.2 Handling unbreakable boundaried instances

Protrusions

For a graph G and $S \subseteq V(G)$, we define $\partial_G(S)$ as the set of vertices in S that have a neighbor in $V(G) \setminus S$. For a set $S \subseteq V(G)$ the *neighborhood* of S is $N_G(S) = \partial_G(V(G) \setminus S)$. When it is clear from the context, we omit the subscripts. An *r-protrusion* in a graph G is a set $X \subseteq V$ such that $|\partial(X)| \leq r$ and $\mathsf{tw}(G[X]) \leq r$. The *size* of a protrusion is the number of vertices in it. An α-cover in G is a set S such that $\sum_{v \in S} d(v)) \geq \alpha \cdot \sum_{v \in V(G)} d(v) = 2\alpha|E(G)|$. Here, $d(v)$ denotes the degree of the vertex v.

We begin with the following lemma that states that if a graph does not have large r-protrusions and there is a set Z of vertices whose deletion results in a graph of constant treewidth, then a constant fraction of the edge-set is incident on Z.

▶ **Lemma 17.** *Let $r = 2\eta + 2$, and $\alpha = \frac{1}{18(\eta+1)^2}$. Let J be a graph and $Z \subseteq V(J)$ such that $\mathsf{tw}(J - Z) \leq \eta$. If J has no r-protrusion of size at least r', then Z is an $\frac{\alpha}{r'}$-cover of J.*

Lemma 17 can be inferred directly from the proof of Lemma 3 in [14], although it does not state concrete values for the various parameters involved in the statement. The following lemma provides a procedure to extract, from a given $K_{d,d}$-free incidence graph with no *large* protrusions, a small set of clauses such that at least one of the clauses in this set contains a backdoor variable.

▶ **Lemma 18** (Protrusion-free instances)**.** *Let J be a $K_{d,d}$-free incidence graph and let $k, r' \in \mathbb{N}$. Let $S \subseteq \mathsf{var}(J)$ be a non-empty inclusion-wise minimal set such that $|S| \leq k$ and $\mathsf{tw}(J - N[S]) \leq \eta$. Moreover, suppose that J has no $(2\eta + 2)$-protrusion of size at least r'. There is an algorithm that, given J, runs in polynomial time, and outputs a subset $\mathcal{D}$ of $\mathsf{cla}(J)$ such that $|\mathcal{D}| = (k \cdot \eta \cdot r')^{\mathcal{O}(d)}$, and there exists a clause $C \in \mathcal{D}$ that contains a variable from S.*

Proof. Since J is an incidence graph, S is a variable set, and $\mathsf{tw}(J - N[S]) \leq \eta$, it follows that $\mathsf{tw}(J - N(S)) \leq \eta$. Invoking Lemma 17 with $Z = N(S)$, we infer that $N(S)$ is an $\frac{\alpha'}{r'}$-cover of J, where $\alpha' = \frac{1}{18(\eta+1)^2}$. Let $\alpha = \frac{\alpha'}{k \cdot r'}$, $g = \frac{\alpha}{2d}$, and $h = \prod_{i=0}^{d-1}(\alpha - ig)$. Let $X = \mathsf{var}(J)$ and $\mathcal{C} = \mathsf{cla}(J)$. Now we define a weight function w on $\mathcal{C}$ as follows. For every $C \in \mathcal{C}$, $w(C) = \frac{d(C)}{m}$, where m is the number of edges in J and $d(C)$ denotes the degree of the vertex C in J. Extending the weight function to sets in the natural way, we conclude that $w(\mathcal{C}) = \sum_{C \in \mathcal{C}} w(C) = 1$ by definition. It will be useful to keep in mind that picking an edge of J uniformly at random and picking the clause-vertex that appears as one of the endpoints of this edge corresponds to picking a vertex $C \in \mathcal{C}$ with probability $w(C)$.

Next, we identify sufficiently small sets $Y \subseteq X$ and $\mathcal{D}' \subseteq \mathcal{C}$ and prove that either $S \cap Y \neq \emptyset$ or $N_J(S) \cap \mathcal{D}' \neq \emptyset$. The sets $\mathcal{D}'$ and Y are defined as follows: $\mathcal{D}'$ is the set of vertices in $\mathcal{C}$ whose weight is greater than g and Y is the set of vertices in X such that it has no neighbor in $\mathcal{D}'$, and the sum of weights on its neighbors is at least α. Formally,

$$\mathcal{D}' = \{y \in \mathcal{C} \mid w(y) > g\}$$

$$Y = \{x \in X \setminus N(\mathcal{D}') \mid w(N(x)) \geq \alpha\}$$

Notice that for every $x \in Y$, $|N(x) \setminus \mathcal{D}'| \geq 2d$. This is because $w(N(x)) \geq \alpha = 2gd$ and every clause in $\mathcal{C} \setminus \mathcal{D}'$ has weight at most g.

▷ Claim 19. Either $N(S) \cap \mathcal{D}' \neq \emptyset$ or $S \cap Y \neq \emptyset$.

We next claim that the sets $\mathcal{D}'$ and Y are sufficiently small for our purposes.

▷ Claim 20. $|\mathcal{D}'| \leq 36(\eta+1)^2 d \cdot k \cdot r'$ and if $N(S) \cap \mathcal{D}' = \emptyset$, then $|Y| \leq d \cdot (36k \cdot r'(\eta+1)^2)^d$.

Given the above two claims, the algorithm is straightforward. We first construct $\mathcal{D}'$ and Y. If $|Y| \leq d \cdot (36k \cdot r'(\eta+1)^2)^d$, then we output a set $\mathcal{D}$ which is the union of $\mathcal{D}'$ and a set of clauses comprising one arbitrary neighbor of each vertex in Y. Otherwise, we simply output $\mathcal{D} = \mathcal{D}'$. From Claim 20, it follows that the size of $\mathcal{D}$ is bounded by $(k \cdot \eta \cdot r')^{\mathcal{O}(d)}$. Moreover, from both claims above, we conclude that some clause in $\mathcal{D}$ contains a variable in S. Since computing $\mathcal{D}'$ and Y can be done in polynomial time, this algorithm runs in polynomial time as required. This completes the proof of the lemma. ◀

In the following, let G and H be $K_{d,d}$-free t-boundaried d-compatible incidence graphs ($t \leq 2\tau'$) with the same boundary type, $|\partial(G)| \leq 2\tau'$ and G is (δ', τ')-unbreakable. Recall that $\partial(G)$ and $\partial(H)$ denote the boundaries of G and H respectively. Throughout this section, we denote by $G^\star$, the graph $G \oplus H$.

▶ **Lemma 13** (Case 1: High treewidth piece)**.** *Suppose* $\mathsf{tw}(G) > \Delta$. *Let* $S^\star \subseteq \mathsf{var}(G^\star)$ *be such that* $|S^\star| \leq k$ *and* $\mathsf{tw}(G^\star - N_{G^\star}[S^\star]) \leq \eta$. *There is an algorithm that, given* $G^\star$, G *and* H *as input, runs in time* $(\delta')^{\mathcal{O}(d\tau')} n^{\mathcal{O}(1)}$, *and outputs a family* $\mathscr{Q}$ *of subsets of* $\mathsf{cla}(G^\star)$ *of size* τ *each, with the following properties.*

1. $|\mathscr{Q}| = (\delta')^{\mathcal{O}(d\tau')}$, *and*
2. *there exists* $\mathcal{Q} \in \mathscr{Q}$ *such that each clause in* $\mathcal{Q}$ *contains a variable from* $S^\star$.

Proof. First we prove the following claim that will allow us to invoke Lemma 18 on certain subraphs of G.

▷ Claim 21. For every set $R \subseteq V(G)$ of size at most τ, $G - R$ does not contain a $(2\eta+2)$-protrusion of size at least $\delta' + 2\eta + 2$

Proof. For the sake of contradiction, suppose there exists $R \subseteq V(G)$ of size at most τ and $A \subseteq V(G-R)$ such that $G[A]$ is a $(2\eta+2)$-protrusion of size at least $\delta' + 2\eta + 2$. That is, $|\partial_{G-R}(A)| \leq 2\eta+2$ and $|A| \geq \delta'+2\eta+2$. Set $X = A \cup R$ and $Y = R \cup (V(G) \setminus (A \setminus \partial_{G-R}(A)))$ and observe that (X, Y) is a separation in G of order at most $|R|+|\partial_{G-R}(A)| \leq \tau+2\eta+2 = \tau'$, and $|X \setminus Y| = |A \setminus \partial_{G-R}(A)| \geq \delta'$. We now claim that $|Y \setminus X| = |V(G) \setminus (A \cup R)| \geq \delta'$. Suppose to the contrary that $|V(G) \setminus (A \cup R)| < \delta'$. Then, $\mathsf{tw}(G) \leq \mathsf{tw}(G[A]) + |V(G) \setminus (A \cup R)| + |R| \leq 2\eta + 2 + \delta' + \tau \leq \Delta$, which is a contradiction to the premise of the lemma. ◁

We next define a recursive procedure FindClauses. The input to FindClauses is G, a set $R \subseteq \mathsf{cla}(G)$ and an integer $q \in \{1, \ldots, \tau\}$ such that $|R| + q = \tau$. The output is a family $\mathscr{Q}'$ of subsets of $\mathsf{cla}(G) \setminus R$ of size q each, such that $|\mathscr{Q}'| \leq (k \cdot \eta \cdot \delta')^{c \cdot d \cdot q}$, for some constant $c > 0$. The following are the steps of the procedure FindClauses(G, R, q).

1: First we apply Lemma 18 where $J = G - R$ and $r' = \delta' + 2\eta + 2$ (the above claim guarantees that the premise of this lemma is satisfied by this choice of J and r'). Let $\mathcal{D}$ be the output of the algorithm of Lemma 18.
2: If $q = 1$, then we output $\{\{C\} \mid C \in \mathcal{D}\}$ and stop.
3: If $q > 1$, then for each $C \in \mathcal{D}$, we recursively call the procedure FindClauses with input G, $R \cup \{C\}$ and $q - 1$ to obtain a set $\mathscr{Q}_C$. Let $\mathscr{Q}'_C = \{Q \cup \{C\} \mid Q \in \mathscr{Q}_C\}$. We return $\mathscr{Q}' = \bigcup_{C \in \mathcal{D}} \mathscr{Q}'_C$ and stop.

Let c be a constant such that the output of Lemma 18 (i.e., $\mathcal{D}$) has cardinality bounded by $(k \cdot \eta \cdot r')^{c \cdot d}$. It can be proved that $|\mathscr{Q}'| \leq (k \cdot \eta \cdot \delta')^{c \cdot d \cdot q}$ by induction on q.

As the number of nodes in the recurrence tree is bounded by $|\mathscr{Q}'|$ and the algorithm in Lemma 18 runs in polynomial time, the running time of the procedure FindClauses is bounded by $|\mathscr{Q}'| \cdot n^{\mathcal{O}(1)}$.

Before moving to the description of the main algorithm of this lemma, we develop some additional notation. For every $Z \subseteq \partial(G)$, let $Z' = \mathsf{cla}(Z) \cup \{c \in \mathsf{cla}(G) \mid \exists v \in c : v \in \mathsf{var}(Z)\}$ and let Z'' be the subset of Z' comprising the (lexicographically) first $\min\{\tau, |Z'|\}$ elements. In other words, if Z' has at least τ elements, then we truncate it to size τ and obtain Z'' and otherwise, $Z'' = Z'$.

We are now ready to describe our main algorithm. We begin by setting $\mathscr{Q} := \emptyset$. We then compute Z'' for every $Z \subseteq \partial(G)$ and for every Z'' such that $|Z''| = \tau$, we set $\mathscr{Q} = \mathscr{Q} \cup \{Z''\}$. For every Z'' such that $|Z''| < \tau$, we set $\mathscr{Q}'_{Z''} \leftarrow \mathsf{FindClauses}(G, Z'', \tau - |Z''|)$ and we set $\mathscr{Q} = \mathscr{Q} \cup \{\{Q\} \cup Z'' \mid Q \in \mathscr{Q}'_{Z''}\}$. When we have completed iterating over all $Z \subseteq \partial(G)$ and updating $\mathscr{Q}$, we return it.

The correctness of the algorithm results from the following claim.

▷ Claim 22. There exists $\mathcal{Q} \in \mathscr{Q}$ such that each clause in $\mathcal{Q}$ contains a variable from $S^\star$.

It remains to argue the size bound on the output family and the running time of the algorithm. Since $q \leq \tau$ in every call to FindClauses and we have at most $2^{|\partial(G)|} \leq 2^{\tau'}$ such calls it follows that the cardinality of the output family is bounded by $(k \cdot \eta \cdot \delta')^{c \cdot d \cdot \tau} \cdot 2^{2\tau'}$ which is upper bounded by $(\delta')^{\mathcal{O}(d\tau')}$. By the same reasoning and from the upper bound on the running time of FindClauses, the running time of this algorithm is $(\delta')^{\mathcal{O}(d\tau')} n^{\mathcal{O}(1)}$. ◀

3.3 Handling boundaried instances with moderately high treewidth

▶ **Definition 23** (Label-wise isomorphism). *Let G_1 and G_2 be two graphs, and let t be a fixed positive integer. For $i \in \{1, 2\}$, let f_{G_i} be a function that associates with every vertex of $V(G_i)$ some subset of $[t]$. The image of a vertex $v \in G_i$ under f_{G_i} is called the label of that vertex. We say that that G_1 is* label-wise isomorphic *to G_2, and denote it by $G_1 \cong_t G_2$, if there is a map $h : V(G_1) \to V(G_2)$ such that (a) h is one to one and onto; (b) $(u, v) \in E(G_1)$ if and only if $(h(u), h(v)) \in E(G_2)$ and (c) $f_{G_1}(v) = f_{G_2}(h(v))$. We call h a* label-preserving isomorphism.

Notice that the first two conditions of Definition 23 simply indicate that G_1 and G_2 are isomorphic. Now, let G be a t-boundaried graph with t distinguished vertices, uniquely labeled from 1 to t. Given such a t-boundaried graph G, we define a canonical labeling function $\mu_G : V(G) \to 2^{[t]}$. The function μ_G maps every distinguished vertex v with label $\ell \in [t]$ to the set $\{\ell\}$, that is $\mu_G(v) = \{\ell\}$, and for all vertices $v \in V(G) \setminus \partial(G)$ we have that $\mu_G(v) = \emptyset$.

Next we define a notion of labeled edge contraction. Let H be a graph together with a function $f_H : V(H) \to 2^{[t]}$ and $(u, v) \in E(H)$. Furthermore, let H' be the graph obtained from H by identifying the vertices u and v into w_{uv}, removing all the parallel edges and removing all the loops. Then by *labeled edge contraction* of an edge (u, v) of a graph H, we mean obtaining a graph H' with the label function $f_{H'} : V(H') \to 2^{[t]}$. For $x \in V(H') \cap V(H)$ we have that $f_{H'}(x) = f_H(x)$ and for w_{uv} we define $f_{H'}(w_{uv}) = f_H(u) \cup f_H(v)$. Now we introduce a notion of labeled minors of a t-boundaried graph. Let H be a graph together with a function $f : V(H) \to 2^{[t]}$ and G be a t-boundaried graph with canonical labeling function μ_G. A graph H is called a labeled minor of G, if we can obtain a labeled isomorphic copy of H from G by performing edge deletion and labeled edge contraction. The *h-folio* of a t-boundaried graph G is the set $\mathcal{M}_h(G)$ of all t-labeled minors of G on at most h vertices.

▶ **Lemma 14** (Case 2: Moderately-high treewidth piece)**.** *Suppose that* $\eta < \mathsf{tw}(G - \partial(G)) \leq \Delta$. *We can compute a set* $S \subseteq \mathsf{var}(G)$ *in time* $g(\eta, k, d) \cdot n^{\mathcal{O}(1)}$ *for some computable function* g, *such that the following hold.*

1. S *has size at most* $g(\eta, k, d)$, *and*
2. S *has a non-empty intersection with a weak* $\mathcal{F}$*-minor-free-backdoor set of size at most* k *for* $G \oplus H$ *(if one exists).*

Proof. Fix a weak $\mathcal{F}$-minor-free-backdoor set $S^\star$ in $G^\star$. Since $G - \partial(G)$ has treewidth at least $\eta + 1$, it must be the case that $S^\star$ intersects $V(G)$. We ensure that the variables in $\partial(G)$ are added to the returned set S and so, we may assume henceforth that $S^\star$ is disjoint from $\partial(G)$.

Let $S_1^\star = S^\star \cap V(G)$. Let $\tau^\star : S^\star \to \{0,1\}$ be a partial assignment such that the incidence graph of $\phi[\tau^\star]$ is $\mathcal{F}$-minor free and let $\tau : \mathsf{var}(\phi) \to \{0,1\}$ be a satisfying assignment for ϕ that extends $\tau^\star$. Let $\tau_1^\star$ denote the restriction of $\tau^\star$ to $S_1^\star$. Let $Z_{kill} \subseteq \partial(G)$ denote those clauses in $\partial(G)$ that are satisfied by $\tau^\star$ and hence do not appear in $\phi[\tau^\star]$. Let $Z_{satext} \subseteq \partial(G)$ denote those clauses on the boundary that survive in $\phi[\tau^\star]$ and have at least one neighbor disjoint from $V(G)$ that satisfies it according to τ. Consider the t'-boundaried graph G_1' obtained from G by restricting the vertex set to those vertices that survive in $\mathsf{inc}(\phi[\tau^\star])$. Notice that the boundary of G' is precisely $\partial(G) \setminus Z_{kill}$. Finally, consider the h-folio $\mathcal{M}_h(G_1')$ where h is an upper bound on the largest graph in $\mathcal{F}$. The labels of the graphs in $\mathcal{M}_h(G_1')$ correspond to the vertices in $\partial(G) \setminus Z_{kill}$.

Now, let $S_2^\star \subseteq \mathsf{var}(G) \setminus \partial(G)$, $\tau_2^\star : S_2^\star \to \{0,1\}$ and $\widetilde{\tau} : \mathsf{var}(G) \to \{0,1\}$ such that the following hold:

- $|S_2^\star| \leq |S_1^\star|$,
- $\widetilde{\tau}$ extends $\tau_2^\star$, $\widetilde{\tau}$ coincides with τ on $\mathsf{var}(\partial(G))$ and $\widetilde{\tau}$ satisfies all clauses in $\mathsf{cla}(G) \setminus (Z_{satext} \cup Z_{kill})$,
- $\mathcal{M}_h(G_1') = \mathcal{M}_h(G_2')$ where G_2' is the graph obtained from G by deleting Z_{kill} and then restricting the vertex set to those vertices that survive after instantiating the vertices in $S_2^\star$ with $\tau_2^\star$.

▷ Claim 24. $\widehat{S} = (S^\star \setminus S_1^\star) \cup S_2^\star$ is also a weak backdoor set of ϕ of size at most k to $\mathcal{F}$-minor-free formulas.

In order to compute the required set S, it is sufficient to iterate over all $r \in [k]$ (guessing $|S_1^\star|$), assignments to the variables in $\partial(G)$ (guessing τ on the variables in the boundary), all possible (pairs) of disjoint subsets of $\mathsf{cla}(\partial(G))$ (guessing Z_{satext} and Z_{kill}) and all possible t'-labeled minors on at most h vertices (guessing $\mathcal{M}_h(G_1')$) and decide whether there exists a set $S_2^\star$ that satisfies the properties listed above and if yes, compute one. Finally, we return the union of all such sets. Notice that the number of such iterations is bounded by a function of η, k, d and hence the returned set has size at most $g(\eta, k, d)$ for some computable function g. Moreover, a set of the required kind in each iteration can be computed using Courcelle's theorem (see [23, 26] for MSO sentences capturing weak backdoors and minor exclusion). ◀

4 Hardness results for Backdoor Detection

Recall that an empty formula is a CNF formula with no clauses. Such formulas are trivially satisfiable.

▶ **Lemma 25.** *Let* $d, d' \geq 2$ *be two positive integers. There is a polynomial-time algorithm that takes as input* d, d', *an instance* $(U, \mathcal{S}, k)$ *of* Set Cover *and outputs a CNF formula* ϕ *with the following properties:*

1. *If* $(U, \mathcal{S}, k)$ *is a yes-instance of* Set Cover*, then* ϕ *has a strong/weak backdoor set of size at most* k *to the class of empty formulas.*
2. *If* ϕ *has strong/weak backdoor set of size at most* k *to any subclass of* $\mathcal{K}_{d,d'}$*-free formulas, then* $(U, \mathcal{S}, k)$ *is a yes-instance of* Set Cover*.*

Proof. Let $(U, \mathcal{S}, k)$ be the given instance of Set Cover. Now, we construct a CNF formula ϕ as follows. For each element $x \in U$, we have a set of $d'(k+1)$ variables $\{x_{i,j} \mid i \in [d'], j \in [k+1]\}$. For each $S \in \mathcal{S}$, we have a set of d variables $\{y_{S,i} \mid i \in [d]\}$. Thus,

$$\mathsf{var}(\phi) = \{x_{i,j} \mid x \in U, i \in [d'], j \in [k+1]\} \cup \{y_{S,i} \mid S \in \mathcal{S}, i \in [d]\}.$$

Next, we explain the construction of the set of clauses of ϕ. For each element $x \in U$, let $\mathcal{S}_x$ be the family of sets in $\mathcal{S}$ that contain x. For each $x \in U$, we make $d'(k+1)+1$ clauses as follows.

$$C_x = \{y_{S,i}, \overline{y}_{S,i} \mid S \in \mathcal{S}_x, i \in [d]\}, \text{ and}$$

$$\text{for all } i \in [d'] \text{ and } j \in [k+1], \quad T_x(i,j) = C_x \cup \{x_{i,j}, \overline{x}_{i,j}\}.$$

This completes the construction of ϕ. It is easy to verify that the construction can be done in polynomial time.

Next we prove the correctness of the algorithm. Consider the first statement. Let $\mathcal{P}$ be a solution for $(U, \mathcal{S}, k)$ of Set Cover. We claim that $Z = \{y_{S,1} \mid S \in \mathcal{P}\}$ is a strong backdoor set as well as a weak backdoor set of ϕ of size at most k to the class of empty formulas. Since $\mathcal{P}$ is a set cover, we have that for any $x \in U$, C_x contains a variable from Z, which appears both positively and negatively in C_x. Thus, for any assignment for the variables in Z, C_x is satisfied for all $x \in U$. This also implies that for any $x \in U, i \in [d']$, and $j \in [k+1]$, $T_x(i,j)$ is satisfied because $C_x \subseteq T_x(i,j)$. In other words, every clause is satisfied by every assignment to Z. This shows that Z is a strong backdoor set of ϕ of size at most k to the class of empty formulas. Since each such sub-formula is trivially satisfiable, we also conclude that every assignment to the variables in Z can be extended to a satisfying assignment of ϕ. This implies that Z is also a weak backdoor set of ϕ of size at most k to the class of empty formulas.

We now prove the second statement. Let Z be a strong or weak $\mathcal{C}$-backdoor set of size at most k for the CNF formula ϕ, where $\mathcal{C}$ is a subclass of $\mathcal{K}_{d,d'}$-free formulas. Let,

$$\mathcal{P} = \{S \in \mathcal{S} \mid \text{ there exists } j \in [d] \text{ such that } y_{S,j} \in Z\}.$$

We prove that $\mathcal{P}$ is indeed a solution for the Set Cover instance $(U, \mathcal{S}, k)$. Since $|Z| \leq k$, we have that $|\mathcal{P}| \leq k$. For the sake of contradiction assume that $\mathcal{P}$ is not a set cover of $(U, \mathcal{S})$. Then, there exists an element $x \in U$ that does not belong to any set in $\mathcal{P}$. This implies that none of the variables in C_x is present in Z. Consider the set of clauses $\mathcal{T} = \{T_x(i,j) \mid i \in [d'], j \in [k+1]\}$. C_x is a subset of every clause in $\mathcal{T}$. Moreover, the clauses in $\{T_x(i,j) \setminus C_x \mid i \in [d'], j \in [k+1]\}$ have pairwise disjoint variable sets. Since $\mathsf{var}(C_x) \cap Z = \emptyset$ and $|Z| \leq k$, there exists $j \in [k+1]$ such that the clauses $T_x(1,j), \ldots T_x(d',j)$ do not contain a variable from Z. Let Q be an arbitrary set in $\mathcal{S}$ containing x. Notice that $y_{Q,1}, \ldots, y_{Q,d} \notin Z$ because of our assumption that x is not covered by $\mathcal{P}$. As $y_{Q,1}, \ldots, y_{Q_d} \in \mathsf{var}(C_x)$, we have that $y_{Q,1}, \ldots, y_{Q,d} \in \mathsf{var}(T_x(i,j))$ for all $i \in [d']$. That is, the subgraph of $\mathsf{inc}(\phi)$ induced on $y_{Q,1}, \ldots, y_{Q,d}$ and $T_x(1,j), \ldots, T_x(d',j)$ forms a $K_{d,d'}$. Moreover this subgraph remains in the incidence graph of the formula obtained by simplifying ϕ based on any assignment to Z. This is a contradiction to the assumption that Z is a strong/weak $\mathcal{C}$-free backdoor set for ϕ. This completes the proof of the second statement. ◀

Lemma 25 implies Theorem 4 as argued in Section 1.

5 Conclusion

We have shown the fixed-parameter tractability of SAT parameterized by the size of a smallest weak backdoor to bounded-treewidth formulas, when the input belongs to the class of formulas whose incidence graphs are $K_{d,d}$-free. This implies the same result when the incidence graph of the input formula excludes a fixed (topological) minor or is from a class of bounded expansion or even from a nowhere-dense graph class. Thus, our main result advances our understanding of the parameterized complexity of SAT well beyond previously studied sparse inputs. In fact, we have shown that all nowhere-dense input formulas are tractable in this setting and moreover, by extending our result to biclique-free formulas, we have also identified natural classes of tractable formulas incomparable with nowhere-dense classes, i.e., formulas of bounded degeneracy.

We also reiterate that our main objective was to obtain a classification result and so, we have not attempted to optimize the running time and relied on invocations to Courcelle's theorem where it is standard in the literature on backdoor set detection. However, given the single-exponential (in k) running time of the algorithms in [13] that significantly improved upon similarly large exponential factors in [23, 26], it is not unreasonable to expect targeted research in this direction to yield similar speed-ups. A promising starting point towards this could be to consider subclasses of biclique-free graphs (e.g., graphs of bounded expansion) and identify those input classes where such a single-exponential dependence on k can be achieved.

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

2 Christian Bessiere, Clément Carbonnel, Emmanuel Hebrard, George Katsirelos, and Toby Walsh. Detecting and exploiting subproblem tractability. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI, 2013.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

4 Clément Carbonnel, Martin C. Cooper, and Emmanuel Hebrard. On backdoors to tractable constraint languages. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 224–239. Springer Verlag, 2014.

5 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

6 Pratibha Choudhary, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Parameterized complexity of feedback vertex sets on hypergraphs. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.18.

7 Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

8 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. *Handbook of Graph Grammars*, pages 313–400, 1997.

9 Y. Crama, O. Ekin, and P. L. Hammer. Variable and term removal from Boolean formulae. *Discr. Appl. Math.*, 75(3):217–230, 1997.

10 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artif. Intell.*, pages 353–366, 1989.

11 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.31.

12 Eduard Eiben, Mithilesh Kumar, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy kernels for connected dominating set on sparse graphs. *SIAM J. Discret. Math.*, 33(3):1743–1771, 2019. doi:10.1137/18M1172508.

13 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, M. S. Ramanujan, and Saket Saurabh. Solving d-sat via backdoors to small treewidth. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 630–641. SIAM, 2015. doi:10.1137/1.9781611973730.43.

14 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 470–479. IEEE, 2012.

15 Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, 1985. doi:10.1145/4221.4225.

16 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 815–821. AAAI Press, 2017. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14272.

17 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1670–1681. SIAM, 2016. doi:10.1137/1.9781611974331.ch114.

18 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining treewidth and backdoors for CSP. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.36.

19 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M Thilikos. Explicit Linear Kernels via Dynamic Programming. In *STACS*, pages 312–324, 2014.

20 Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Zivny. Backdoors into heterogeneous classes of SAT and CSP. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2652–2658. AAAI Press, 2014.

21 Serge Gaspers, Sebastian Ordyniak, M. S. Ramanujan, Saket Saurabh, and Stefan Szeider. Backdoors to q-horn. *Algorithmica*, 74(1):540–557, 2016. doi:10.1007/s00453-014-9958-5.

22 Serge Gaspers, Sebastian Ordyniak, and Stefan Szeider. Backdoor sets for CSP. In Andrei A. Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 137–157. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/DFU.Vol7.15301.5.

23 Serge Gaspers and Stefan Szeider. Backdoors to acyclic SAT. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7391 of *Lecture Notes in Computer Science*, pages 363–374. Springer, 2012.

24 Serge Gaspers and Stefan Szeider. Strong backdoors to nested satisfiability. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 72–85. Springer, 2012. doi:10.1007/978-3-642-31612-8_7.

25 Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In *The Multivariate Algorithmic Revolution and Beyond*, pages 287–317, 2012. doi:10.1007/978-3-642-30891-8_15.

26 Serge Gaspers and Stefan Szeider. Strong backdoors to bounded treewidth sat. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 489–498. IEEE Computer Society, 2013.

27 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

28 Donald E. Knuth. Nested satisfiability. *Acta Informatica*, 28(1):1–6, 1990. doi:10.1007/BF02983372.

29 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Trans. Algorithms*, 15(2):24:1–24:19, 2019. doi:10.1145/3274652.

30 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 135:1–135:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.135.

31 Jaroslav Nesetril and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *J. Symb. Log.*, 75(3):868–887, 2010. doi:10.2178/jsl/1278682204.

32 Jaroslav Nesetril and Patrice Ossona de Mendez. On nowhere dense graphs. *Eur. J. Comb.*, 32(4):600–617, 2011. doi:10.1016/j.ejc.2011.01.006.

33 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proceedings of SAT 2004 (Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May, 2004, Vancouver, BC, Canada)*, pages 96–103, 2004.

34 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):11:1–11:23, 2012. doi:10.1145/2390176.2390187.

35 Igor Razgon and Barry O'Sullivan. Almost 2-SAT is fixed parameter tractable. *J. of Computer and System Sciences*, 75(8):435–450, 2009.

36 Neil Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Combin. Theory Ser. B*, 41(1):92–114, 1986.

37 Marko Samer and Stefan Szeider. Backdoor sets of quantified boolean formulas. In J. Marques-Silva and K. A. Sakallah, editors, *Proceedings of SAT 2007, Tenth International Conference on Theory and Applications of Satisfiability Testing, May 28-31, 2007, Lisbon, Portugal,*, volume 4501 of *Lecture Notes in Computer Science*, pages 230Â–243, 2007.

38 Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010. doi:10.1016/j.jcss.2009.04.003.

39 Stefan Szeider. Matched formulas and backdoor sets. In J. Marques-Silva and K. A. Sakallah, editors, *Proceedings of SAT 2007, Tenth International Conference on Theory and Applications of Satisfiability Testing, May 28-31, 2007, Lisbon, Portugal,*, volume 4501 of *Lecture Notes in Computer Science*, pages 94–99, 2007.

40 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in sparse graphs and beyond. *Theor. Comput. Sci.*, 770:62–68, 2019. doi:10.1016/j.tcs.2018.10.030.

41 Ryan Williams, Carla Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003*, pages 1173–1178. Morgan Kaufmann, 2003.

Optimal Coding Theorems in Time-Bounded Kolmogorov Complexity

Zhenjian Lu ✉
University of Warwick, Coventry, UK

Igor C. Oliveira ✉
University of Warwick, Coventry, UK

Marius Zimand ✉
Towson University, MD, USA

Abstract

The classical coding theorem in Kolmogorov complexity states that if an n-bit string x is sampled with probability δ by an algorithm with prefix-free domain then $\mathsf{K}(x) \leq \log(1/\delta) + O(1)$. In a recent work, Lu and Oliveira [31] established an unconditional time-bounded version of this result, by showing that if x can be efficiently sampled with probability δ then $\mathsf{rKt}(x) = O(\log(1/\delta)) + O(\log n)$, where rKt denotes the randomized analogue of Levin's Kt complexity. Unfortunately, this result is often insufficient when transferring applications of the classical coding theorem to the time-bounded setting, as it achieves a $O(\log(1/\delta))$ bound instead of the information-theoretic optimal $\log(1/\delta)$.

Motivated by this discrepancy, we investigate optimal coding theorems in the time-bounded setting. Our main contributions can be summarised as follows.

• **Efficient coding theorem for rKt with a factor of 2.** Addressing a question from [31], we show that if x can be efficiently sampled with probability at least δ then $\mathsf{rKt}(x) \leq (2 + o(1)) \cdot \log(1/\delta) + O(\log n)$. As in previous work, our coding theorem is *efficient* in the sense that it provides a polynomial-time probabilistic algorithm that, when given x, the code of the sampler, and δ, it outputs, with probability ≥ 0.99, a probabilistic representation of x that certifies this rKt complexity bound.

• **Optimality under a cryptographic assumption.** Under a hypothesis about the security of cryptographic pseudorandom generators, we show that no efficient coding theorem can achieve a bound of the form $\mathsf{rKt}(x) \leq (2 - o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$. Under a weaker assumption, we exhibit a gap between *efficient* coding theorems and *existential* coding theorems with near-optimal parameters.

• **Optimal coding theorem for pK^t and unconditional Antunes-Fortnow.** We consider pK^t complexity [17], a variant of rKt where the randomness is public and the time bound is fixed. We observe the existence of an optimal coding theorem for pK^t, and employ this result to establish an *unconditional* version of a theorem of Antunes and Fortnow [5] which characterizes the worst-case running times of languages that are in average polynomial-time over all P-samplable distributions.

2012 ACM Subject Classification Theory of computation

Keywords and phrases computational complexity, randomized algorithms, Kolmogorov complexity

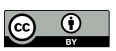
Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.92

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2204.08312` [33]

Funding M. Zimand was supported in part by the National Science Foundation through grant CCF 1811729. Z. Lu and I.C. Oliveira received support from the Royal Society University Research Fellowship URF\R1\191059 and from the EPSRC New Horizons Grant EP/V048201/1.

Acknowledgements We thank Bruno Bauwens for discussions and useful insights.



49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 92; pp. 92:1–92:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Context and Background

A sampler is a probabilistic function that outputs Boolean strings. For any string $x \in \{0,1\}^*$ in its range, let $\mu(x)$ denote the probability with which x is generated. The Coding Theorem in Kolmogorov complexity states that if the sampler is computable and its domain is a prefix-free set, then for every x in its range

$$\mathsf{K}(x) \leq \log(1/\mu(x)) + O(1),$$

where $\mathsf{K}(\cdot)$ is the prefix-free Kolmogorov complexity. In other words, strings that are sampled with non-trivial probability have short representations. Note that the coding theorem achieves *optimal* expected length, since no uniquely decodable code can have expected length smaller than $\sum \mu(x) \log_2(1/\mu(x))$, the entropy of the sampler (the sum is over all x in the range of the sampler, assumed here to be finite).

The coding theorem is a central result in Kolmogorov complexity.[1] While it has found a number of applications in theoretical computer science (see, e.g., [28, 25, 1]), it comes with an important caveat: many aspects of the theory of Kolmogorov complexity are *non-constructive*. For instance, there is provably no algorithm that estimates $\mathsf{K}(x)$. Similarly, for arbitrary samplers, there is no effective compressor achieving the short representation provided by the coding theorem[2] and also no upper bound on the running time required to decompress x from it.

In order to translate results and techniques from Kolmogorov complexity to the setting of *efficient* algorithms and computations, several *time-bounded* variants of Kolmogorov complexity have been proposed. We refer to the book [29], thesis [25], and surveys [2, 3, 16, 4] for a comprehensive treatment of this area and its numerous applications to algorithms, complexity, cryptography, learning, and pseudorandomness, among other fields. We highlight that many exciting new results, which include worst-case to average-case reductions for NP problems [20, 21] and complexity-theoretic characterizations of one-way functions [30, 36], rely in a crucial way on time-bounded Kolmogorov complexity. These recent developments further motivate the investigation of key results from Kolmogorov complexity in the time-bounded setting.

In time-bounded Kolmogorov complexity we consider the minimum description length of a string x *with respect to machines that operate under a time constraint*. We informally review next two central notions in this area (see Section A for precise definitions). For a Turing machine $\mathcal{M}$, we let $|\mathcal{M}|$ denote its description length according to a fixed universal machine U. $\mathcal{M}(\varepsilon)$ denotes the computation of $\mathcal{M}$ over the empty string.

Kt Complexity. [26] This notion simultaneously considers description length and running time when measuring the complexity of a string x.

$$\mathsf{Kt}(x) \;=\; \min_{\text{TM } \mathcal{M},\, t \geq 1} \{|\mathcal{M}| + \log t \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps}\}\,.$$

K^t Complexity. [37] In contrast with Kt, here we fix the time bound $t\colon \mathbb{N} \to \mathbb{N}$, and consider the minimum description with respect to machines that run in time at most $t(|x|)$.

$$\mathsf{K}^t(x) \;=\; \min_{\text{TM } \mathcal{M}} \{|\mathcal{M}| \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t(|x|) \text{ steps}\}\,.$$

[1] For instance, [25] describes it as one of the four pillars of Kolmogorov complexity.

[2] However, there exists a probabilistic polynomial-time compressor that given x and an integer $m \geq \log(1/\mu(x))$ outputs a description of x of length m + small polylogarithmic overhead [6].

While Kt complexity is tightly related to optimal search algorithms (see [24] for a recent application), K^t is particularly useful in settings where maintaining a polynomial bound on the running time t is desired (see, e.g., [20]).

Antunes and Fortnow [5] introduced techniques that can be used to establish (conditional) coding theorems for K^t and Kt. In particular, if a sampler runs in *polynomial time* and outputs a string x with probability at least δ, then $\mathsf{Kt}(x) \leq \log(1/\delta) + O(\log n)$. Note that this coding theorem also achieves an optimal dependence on the probability parameter δ. However, the results of [5] rely on a strong derandomization assumption. For this reason, their application often lead to *conditional* results.

More recently, [31] established an *unconditional* coding theorem for a *randomized* analogue of Kt complexity. Before explaining their result, we review the definitions of rKt and rK^t.[3]

rKt Complexity. [35] In this definition, we consider randomized machines that output x with high probability.

$$\mathsf{rKt}(x) = \min_{\text{RTM } \mathcal{M},\, t \geq 1} \{|\mathcal{M}| + \log t \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps with probability} \geq 2/3\}.$$

rK^t Complexity. [10, 32][4] This is the randomized analogue of K^t, where the time bound t is fixed in advance.

$$\mathsf{rK}^t(x) = \min_{\text{RTM } \mathcal{M}} \{|\mathcal{M}| \mid \mathcal{M}(\varepsilon) \text{ outputs } x \text{ in } t(|x|) \text{ steps with probability} \geq 2/3\}.$$

In both cases, we can think of the randomized Turing machine $\mathcal{M}$ as a *probabilistic representation* of the input string x, in the sense that x can be recovered with high probability from its description. These measures allow us to employ methods from time-bounded Kolmogorov complexity in the setting of randomized computation, which is ubiquitous in modern computer science. For instance, [35, 32] employed rKt and rK^t to obtain bounds on the compressibility of prime numbers and other objects and to show that certain problems about time-bounded Kolmogorov complexity can be intractable. We note that, under derandomization assumptions (see [35]), for every string x, $\mathsf{rKt}(x) = \Theta(\mathsf{Kt}(x))$. Similarly, one can conditionally show that $\mathsf{K}^t(x)$ is essentially $\mathsf{rK}^t(x)$, up to a $O(\log |x|)$ additive term (see [17]). Consequently, insights obtained in the context of probabilistic notions of Kolmogorov complexity can often inform the study of more classical notions such as Kt and K^t.

Among other results, [31] established the following unconditional coding theorem in time-bounded Kolmogorov complexity: if a sampler runs in polynomial time and outputs a string x with probability at least δ, then $\mathsf{rKt}(x) = O(\log(1/\delta) + O(\log n)$. While this result can be used to port some applications of the coding theorem from Kolmogorov complexity to the time-bounded setting, in many cases it is still insufficient. This is because its dependence on the probability parameter δ is not optimal, which is often crucial in applications (see, e.g., [5, 1]).

2 Results

In this work, we investigate optimal coding theorems in time-bounded Kolmogorov complexity. We describe our results next.

[3] See Appendix A for a formal treatment.
[4] [10] refers to this notion as CBP^t complexity.

2.1 A Tighter Efficient Coding Theorem

Our first result addresses the question posed in [31, Problem 37].

▶ **Theorem 1.** *Suppose there is an efficient algorithm A for sampling strings such that $A(1^n)$ outputs a string $x \in \{0,1\}^n$ with probability at least δ. Then*

$$\mathsf{rKt}(x) \;\leq\; 2\log(1/\delta) + O\big(\log n + \log^2 \log(1/\delta)\big)\,,$$

where the constant behind the $O(\cdot)$ depends on A and is independent of the remaining parameters. Moreover, given x, the code of A, and δ, it is possible to compute in time $\mathsf{poly}(n, |A|)$, with probability ≥ 0.99, a probabilistic representation of x certifying this rKt-complexity bound.

In [9, Lemma 4], it was observed that by hashing modulo prime numbers one can obtain short descriptions of strings. As discussed in [31, Section A.2.1], for each efficient sampling algorithm, this technique implies that if some string x is produced with probability $\geq \delta$, then $\mathsf{rKt}(x) \leq 3\log(1/\delta) + O(\log n)$.[5] In contrast, Theorem 1 achieves a bound of the form $(2 + o(1)) \cdot \log(1/\delta) + O(\log n)$.

Theorem 1 readily improves some parameters in the applications of the coding theorem for rKt discussed in [31], such as the efficient instance-based search-to-decision reduction for rKt. We omit the details.

In [33, Section 3.1], we discuss extensions of this result. In particular, we describe precise bounds on the running time used in producing the corresponding probabilistic representation, and discuss computational aspects of the compression and decompression of x in detail. In [33, Appendix A], we discuss the computation of a probabilistic representation of the string x when one does not know a probability bound δ.

2.2 Matching Lower Bound Under a Cryptographic Assumption

It is possible to extend techniques from [5] to show the following conditional result (see [33, Section 3.2]).

▶ **Proposition 2.** *Assume there is a language $L \in \mathsf{BPTIME}\left[2^{O(n)}\right]$ that requires nondeterministic circuits of size $2^{\Omega(n)}$ for all but finitely many n. Suppose there is an efficient algorithm A for sampling strings such that $A(1^n)$ outputs a string $x \in \{0,1\}^n$ with probability at least $\delta > 0$. Then*

$$\mathsf{rKt}(x) \;\leq\; \log(1/\delta) + O(\log n).$$

While Proposition 2 provides a better bound than Theorem 1, the result is only *existential*, i.e., it does not provide an efficient algorithm that produces a probabilistic representation of x. In other words, Proposition 2 does not establish an *efficient* coding theorem. Our next result shows that the bound achieved by Theorem 1 is optimal for efficient coding theorems, under a cryptographic assumption.

[5] The bound from [31, Section A.2.1] is different because it does not take into account the running time, which incurs an additional overhead of $\log(1/\delta)$.

The Cryptographic Assumption. For a constant $\gamma \in (0,1)$, we introduce the γ-Crypto-ETH assumption, which can be seen as a cryptographic analogue of the well-known exponential time hypothesis about the complexity of k-CNF SAT [23]. Informally, we say that γ-Crypto-ETH holds if there is a pseudorandom generator $G\colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$ computable in time $\mathrm{poly}(n)$ that fools *uniform algorithms* running in time $2^{\gamma \cdot \ell(n)}$. Any seed length $(\log n)^{\omega(1)} \leq \ell(n) \leq n/2$ is sufficient in our negative results.

In analogy with the well-known ETH and SETH hypotheses about the complexity of k-CNF SAT, we say that Crypto-ETH holds if γ-Crypto-ETH is true for some $\gamma > 0$, and that Crypto-SETH holds if γ-Crypto-ETH is true for every $\gamma \in (0,1)$. Since a candidate PRG of seed length $\ell(n)$ can be broken in time $2^{\ell(n)}\mathsf{poly}(n)$ by trying all possible seeds, these hypotheses postulate that for some PRGs one cannot have an attack that does sufficiently better than this naive brute-force approach.

We stress that these assumptions refer to uniform algorithms. In the case of non-uniform distinguishers, it is known that Crypto-SETH does not hold (see [15, 14, 13] and references therein). We provide a formal treatment of the cryptographic assumption in [33, Section 4].

▶ **Theorem 3** (Informal). *Let $\gamma \in (0,1)$ be any constant. If γ-Crypto-ETH holds, there is no efficient coding theorem for* rKt *that achieves bounds of the form* $(1+\gamma-o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$.

Theorem 3 shows that if Crypto-ETH holds then the best parameter achieved by an *efficient* coding theorem for rKt is $(1+\Omega(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$. This exhibits an inherent gap in parameters between the efficient coding theorem (Theorem 1) and its existential analogue (Proposition 2). On the other hand, if the stronger Crypto-SETH hypothesis holds, then no efficient coding theorem for rKt achieves parameter $(2-o(1)) \cdot \log(1/\delta) + \mathsf{poly}(\log n)$. In this case, Theorem 1 is essentially optimal with respect to its dependence on δ.

Fine-grained complexity of coding algorithms for polynomial-time samplers. An rKt bound refers to the time necessary to *decompress* a string x from its probabilistic representation. On the other hand, an *efficient* coding theorem provides a routine that can *compress* x in polynomial time. More generally, a coding procedure for a sampler A consists of a pair of probabilistic algorithms (Compress, Decompress) that aim to produce a "good" codeword p for every string y sampled by A. The quality of p depends on three values: the length of p, the number of steps t_C used to produce p from y (the compression time), and the number of steps t_D used to produce y from p (the decompression time). It is interesting to understand the trade-off between these three values. Toward this goal, we aggregate them in a manner similar to rKt, by defining the 2-sided-rKt complexity of y to be, roughly, $|p| + \log(t_C + t_D)$ (the formal definition, see [33, Definition 25]), is more complicated because it takes into account that Compress and Decompress are probabilistic). Thus according to 2-sided-rKt, each bit gained by a shorter codeword is worth doubling the compression/decompression time. For instance, for simple samplers (say, having a finite range, or generating strings with the uniform distribution), there exist trivial polynomial time Compress and Decompress, which in case y is sampled with probability at least δ, produce a codeword p with $|p| = \log(1/\delta)$ (provided Compress and Decompress know δ). Such a coding procedure certifies for each sampled string a 2-sided-rKt complexity of $\log(1/\delta) + O(\log n)$. We say that the sampler admits coding with 2-sided-rKt complexity bounded by $\log(1/\delta) + O(\log n)$. In general, we have to include also the error probability of Compress and Decompress, which we omit in this informal discussion.

Similarly to Theorem 1 and Theorem 3 (and also with similar proofs), we establish the following theorem.

▶ **Theorem 4** (Informal). *The following results hold.*

(a) (Upper Bound) *Every polynomial-time sampler admits coding with* 2*-sided-*rKt *complexity* $2\log(1/\delta) + O(\log^2\log(1/\delta)) + O(\log n)$.

(b) (Conditional Lower Bound) *Let* $\gamma \in (0,1)$ *be any constant. If* γ*-*Crypto-ETH *holds, there exists a polynomial-time sampler that does not admit coding with* 2*-sided-*rKt *complexity bounded by* $(1+\gamma-o(1))\cdot\log(1/\delta) + \mathsf{poly}(\log n)$*, unless the error probability is greater than* $1/7$.

2.3 An Optimal Coding Theorem and Unconditional Antunes-Fortnow

While Theorem 1 improves the result from [31] to achieve a bound that is tight up to a factor of 2 and that is possibly optimal among efficient coding theorems, it is still insufficient in many applications. We consider next a variant of rKt that allows us to establish an *optimal* and *unconditional* coding theorem in time-bounded Kolmogorov complexity.

Fix a function $t\colon \mathbb{N}\to\mathbb{N}$. For a string $x\in\{0,1\}^*$, the probabilistic t-bounded Kolmogorov complexity of x (see [17]) is defined as

$$\mathsf{pK}^t(x) = \min\left\{ k \;\middle|\; \Pr_{w\sim\{0,1\}^{t(|x|)}}\left[\exists\,\mathcal{M}\in\{0,1\}^k,\ \mathcal{M}(w) \text{ outputs } x \text{ within } t(|x|) \text{ steps}\right] \geq \frac{2}{3}\right\}.$$

In other words, if $k = \mathsf{pK}^t(x)$, then with probability at least $2/3$ over the choice of the random string w, x admits a time-bounded encoding of length k. In particular, if two parties share a typical random string w, then x can be transmitted with k bits and decompressed in time $t = t(|x|)$. (Recall that here the time bound t is fixed, as opposed to rKt, where a $\log t$ term is added to the description length.)

It is possible to show that $\mathsf{K}^t(x)$, $\mathsf{rK}^t(x)$, and $\mathsf{pK}^t(x)$ correspond essentially to the same time-bounded measure, under standard derandomization assumptions [17].[6] One of the main benefits of pK^t is that it allows us to establish unconditional results that are currently unknown in the case of the other measures.[7]

▶ **Theorem 5.** *Suppose there is a randomized algorithm* A *for sampling strings such that* $A(1^n)$ *runs in time* $T(n)\geq n$ *and outputs a string* $x\in\{0,1\}^n$ *with probability at least* $\delta>0$. *Then*

$$\mathsf{pK}^t(x) \;=\; \log(1/\delta) + O(\log T(n))\,,$$

where $t(n) = \mathrm{poly}(T(n))$ *and the constant behind the* $O(\cdot)$ *depends on* $|A|$ *and is independent of the remaining parameters.*

Theorem 5 provides a time-bounded coding theorem that can be used in settings where the optimal dependence on δ is crucial. As an immediate application, it is possible to show an equivalence between efficiently sampling a fixed sequence $w_n\in\{0,1\}^n$ of objects (e.g., n-bit prime numbers) with probability at least $\delta_n/\mathsf{poly}(n)$ and the existence of bounds for the

---

[6] More precisely, under standard derandomization assumptions, $\mathsf{pK}^t(x)$ and $\mathsf{rK}^{t'}(x)$ coincide up to an additive term of $O(\log|x|)$, provided that $t' = \mathrm{poly}(t)$. A similar relation holds between K^t and rK^t.

[7] While in this work we focus on coding theorems, we stress that pK^t is a key notion introduced in [17] that enables the investigation of meta-complexity in the setting of probabilistic computations. It has applications in worst-case to average-case reductions and in learning theory.

corresponding objects of the form $\mathsf{pK}^{\mathrm{poly}}(w_n) = \log(1/\delta_n) + O(\log n)$.[8] This is the first tight equivalence of this form in time-bounded Kolmogorov complexity that does not rely on an unproven assumption.

As a more sophisticated application of Theorem 5, we establish an unconditional form of the main theorem from Antunes and Fortnow [5], which provides a characterization of the worst-case running times of languages that are in average polynomial-time over all P-samplable distributions.

We recall the following standard notion from average-case complexity (see, e.g., [8]). For an algorithm A that runs in time $T_A\colon \{0,1\}^* \to \mathbb{N}$ and for a distribution $\mathcal{D}$ supported over $\{0,1\}^*$, we say that A runs in polynomial-time on average with respect to $\mathcal{D}$ if there is some constant $\varepsilon > 0$ such that

$$\mathop{\mathbf{E}}_{x\sim\mathcal{D}}\left[\frac{T_A(x)^\varepsilon}{|x|}\right] < 1.$$

As usual, we say that a distribution $\mathcal{D}$ is P-samplable if it can be sampled in polynomial time.

▶ **Theorem 6.** *The following conditions are equivalent for any language $L \subseteq \{0,1\}^*$.*

(i) *For every P-samplable distribution $\mathcal{D}$, L can be solved in polynomial-time on average with respect to $\mathcal{D}$.*

(ii) *For every polynomial p, there exists a constant $b > 0$ such that the running time of some algorithm that computes L is bounded by $2^{O(\mathsf{pK}^p(x) - \mathsf{K}(x) + b\log(|x|))}$ for every input x.*

In contrast, [5] shows a *conditional* characterisation result that employs K^t complexity in the expression that appears in Item (*ii*).

3 Techniques

In this section, we provide an informal overview of our proofs and techniques.

Efficient Coding Theorem for rKt (Theorem 1). Breaking down the result into its components, Theorem 1 shows that for any polynomial-time sampler A, there exist a probabilistic polynomial-time algorithm Compress and an algorithm Decompress with the following properties: Compress on input an n-bit string x and δ (which estimates from below the probability with which A samples x), returns a codeword c_x of length $\log(1/\delta) + \mathrm{poly}(\log n)$ such that Decompress with probability ≥ 0.99 reconstructs x in time $1/\delta \cdot \exp(\mathrm{poly}(\log n))$. Note that the probabilistic representation of x certifying the rKt bound in Theorem 1 can be obtained from the codeword c_x and Decompress, and that obtaining a running time with a factor of $(1/\delta)^{1+o(1)}$ is crucial in order to get a final rKt bound of the form $(2 + o(1)) \cdot \log(1/\delta)$. (Actually, Compress does not have to depend on A, the 0.99 can be $1 - \varepsilon$ for arbitrary $\varepsilon > 0$, and the $\mathrm{poly}(\log n)$ term is $O(\log n + \log^2 \log(1/\delta))$, but we omit these details in our discussion). We explain what are the challenges in obtaining Compress and Decompress and how they are overcome. We remark that the construction is different from the approaches described in [31].

[8] An efficient sampler immediately implies the corresponding pK^t bounds via Theorem 5. On the other hand, objects of bounded pK^t complexity can be sampled by considering a random sequence of bits and a random program of appropriate length. We refer to [31, Theorem 6] for a weaker relation and its proof. Since the argument is essentially the same, we omit the precise details.

Decompress can run the sampler $K := O(1/\delta)$ times and obtain a list of elements S^* (the list of suspects) that with high probability contains x. Compress has to provide information that allows Decompress to prune S^* and find x. Since the algorithms do not share randomness, Compress does not know S^*, and so compression has to work for any $S \subseteq \{0,1\}^n$ of size K, only assuming that $x \in S$. Compress can use a bipartite lossless expander graph G, which is a graph with the property that any set S of left nodes with size $|S| \leq K$ has at least $(1-\varepsilon)D|S|$ neighbors, where D is the left degree. Such graphs are called $((1-\varepsilon)D, K)$ lossless expanders and they have numerous applications (see e.g., [11, 22]). An extension of Hall's matching theorem shows that for any set S of K left nodes, there is a matching that assigns to each $x \in S$, $(1-\varepsilon)D$ of its neighbors, so that no right node is assigned twice (i.e., the matching defines a subgraph with no collisions). Compress can just pick the codeword c_x to be one random neighbor of x. Then, Decompress can do the pruning of S^* as follows. Having S^* and c_x, it does the matching, and, since with probability $1-\varepsilon$, c_x is only assigned to x, Decompress can find x. There is one problem though. The algorithms for maximum matching in general bipartite graphs do not run in linear time (see [12, 34], and the references therein). Therefore, the decompression time would have a dependency on δ too large for us. Fortunately, lossless expanders can be used to do "almost" matching faster. [6] introduces *invertible functions* (see [33, Definition 12]) for the more demanding task in which the elements of S appear one-by-one and the matching has to be done in the online manner. We do not need online matching, but we take advantage of the construction in [6] to obtain a fast matching algorithm. It follows from [6], that in a lossless expander it is possible to do a greedy-type of "almost" matching, which means that every left node in S is matched to $(1-\varepsilon)D$ of its neighbors (exactly what we need), but with $\mathrm{poly}(\log n)$ collisions. The collisions can be eliminated with some additional standard hashing (see the discussion on [33, Page 14] for details). As we explain on [33, Page 14], this leads to decompression time $K \cdot D \cdot \mathrm{poly}(n)$ and the length of the codeword c_x is $\log |R| + |\text{hash-code}|$, where R is the right set of the lossless expander. To obtain our result, the degree D has to be $2^{\mathrm{poly}(\log n)}$ and $|R|$ has to be $K \cdot 2^{\mathrm{poly}(\log n)}$.

Building on results and techniques from [18], [6] constructs a $((1-\varepsilon)D, K)$ explicit lossless expander with left side $\{0,1\}^n$, degree $D = 2^d$ for $d = O(\log(n/\varepsilon) \cdot \log k)$, and right side R, with size verifying $\log |R| = k + \log(n/\varepsilon) \cdot \log k$ (where $k := \log K$). To obtain in Theorem 1 the dependency on n to be $O(\log n)$ (which is optimal up to the constant in $O(\cdot)$), we show the existence of a $((1-\varepsilon)D, K)$ explicit expander with $d = O(\log n + \log(k/\varepsilon) \cdot \log k)$ and $\log |R| = k + O(\log n + \log(k/\varepsilon) \cdot \log k)$. This lossless expander is constructed by a simple composition of the above lossless expander from [6] with a lossless expander from [18], with an appropriate choice of parameters (see [33, Section 3.1.1]).

Conditional Lower Bound for Efficient Coding Theorems (Theorem 3). Our goal is to show that there is no *efficient* coding theorem for rKt that achieves bounds of the form $(1+\gamma-o(1)) \cdot \log(1/\delta) + \mathrm{poly}(\log n)$, under the assumption that γ-Crypto-ETH holds for $\gamma \in (0,1)$. We build on an idea attributed to L. Levin (see e.g. [25, Section 5.3]). To provide an overview of the argument, let $G_n \colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$ be a cryptographic generator of seed length $\ell(n) = n/2$ witnessing that γ-Crypto-ETH holds. In other words, G_n has security $2^{\gamma \cdot \ell(n)}$ against uniform adversaries. We define a sampler S_n as follows. On input $x \in \{0,1\}^n$, which we interpret as a random string, it outputs $G_n(x')$, where x' is the prefix of x of length $\ell(n)$. We argue that if an *efficient* algorithm F is able to compress every string y in the support of $\mathsf{Dist}(S_n)$, the distribution induced by the sampler S_n, to an rKt encoding of complexity $(1+\gamma-\varepsilon) \cdot \log(1/\delta'(y)) + C \cdot (\log n)^C$, where $\delta'(y)$ is a lower bound on $\delta(y)$ (the probability of y under $\mathsf{Dist}(S_n)$), we can use F to break G_n. (Note that F expects as input n, y, δ', and $\mathsf{code}(S)$.)

The (uniform) distinguisher D computes roughly as follows. Given a string $z \in \{0,1\}^n$, which might come from the uniform distribution U_n or from $G_n(U_{\ell(n)}) \equiv \mathsf{Dist}(S_n)$, D attempts to use F to *compress* z to a "succinct" representation, then checks if the computed representation *decompresses* to the original string z. If this is the case, it outputs 1, otherwise it outputs 0. (Note that we haven't specified what "succinct" means, and it is also not immediately clear how to run F, since it assumes knowledge of a probability bound δ'. For simplicity of the exposition, we omit this point here.) We need to argue that a test of this form can be implemented in time $2^{\gamma \cdot \ell(n)}$, and that it distinguishes the output of G from a random string.

To achieve these goals, first note that a typical random string cannot be compressed to representations of length, say, $n - \mathsf{poly}(\log n)$, even in the much stronger sense of (time-unbounded) Kolmogorov complexity. Therefore, with some flexibility with respect to our threshold for succinctness, the proposed distinguisher is likely to output 0 on a random string. On the other hand, if F implements an efficient coding theorem that achieves rKt encodings of complexity $(1+\gamma-\varepsilon)\cdot\log(1/\delta'(y))+\mathsf{poly}(\log n)$, the following must be true. Using that the expected *encoding length* of *any* (prefix-free) encoding scheme is at least $H(\mathsf{Dist}(S_n))$, where $\mathsf{Dist}(S_n)$ is the distribution of strings sampled by S_n and H is the entropy function, we get (via a slightly stronger version of this result) that a non-trivial measure of strings y in the support of $\mathsf{Dist}(S_n)$ have rKt *encoding length* at least $(1-\varepsilon/4)\cdot\log(1/\delta(y))$. Consequently, for such strings, an upper bound on rKt complexity of $(1+\gamma-\varepsilon)\cdot\log(1/\delta'(y))+\mathsf{poly}(\log n)$ when $\delta'(y)$ is sufficiently close to $\delta(y)$ implies that the running time t of the underlying machine satisfies $\log t \leq (\gamma-\varepsilon/2)\log(1/\delta(y))+\mathsf{poly}(\log n)$. Using that $\ell(n)=n/2$ and $\delta(y)\geq 2^{-\ell(n)}$ for any string y in the support of $\mathsf{Dist}(S_n)$, it is easy to check that (asymptotically) $t \leq 2^{(\gamma-\varepsilon/4)\cdot\ell(n)}$. For this reason, we can implement a (slightly modified) distinguisher D in time less than $2^{\gamma\cdot\ell(n)}$, by trying different approximations $\delta'(z)$ for an input string z and by running the decompressor on the produced representation for at most t steps on each guess for $\delta(z)$. By our previous discussion, a non-trivial measure of strings from $\mathsf{Dist}(S_n)$ will be accepted by D, while only a negligible fraction of the set of all strings (corresponding to the random case) will be accepted by D.

Implementing this strategy turns out to be more subtle than this. This happens because F is a *probabilistic* algorithm which does not need to commit to a *fixed* succinct encoding. We refer to the formal presentation in [33, Section 4] for details, where we also discuss the bound on the seed length $\ell(n)$.

Coding Theorem for pK^t (Theorem 5) and Unconditional [5] (Theorem 6). The proof of our optimal coding theorem for pK^t builds on that of the *conditional* coding theorem for K^t from [5], which can be viewed as a two-step argument. Roughly speaking, the first step is to show that if there is a polynomial-time sampler that outputs a string $x \in \{0,1\}^n$ with probability δ, then the polynomial-time-bounded Kolmogorov complexity of x is about $\log(1/\delta)+O(\log n)$ *if we are given a random string*. After this, they "derandomize" the use of random strings using a certain pseudorandom generator, which exists under a strong derandomization assumption. Our key observation is that the use of random strings arises naturally in probabilistic Kolmogorov complexity, and particularly in this case the random strings can be "embedded" into the definition of pK^t. As a result, we don't need to perform the afterward derandomization as in original proof of [5], and hence get rid of the derandomization assumption.

Next, we describe how to use Theorem 5, together with other useful properties of pK^t, to obtain an unconditional version of Antunes and Fortnow's main result. Let μ be a Kolmogorov complexity measure, such as $\mathsf{K}^{\mathsf{poly}}$, $\mathsf{rK}^{\mathsf{poly}}$ or $\mathsf{pK}^{\mathsf{poly}}$. The key notion in the proof

is the distribution (in fact, a class of semi-distributions) called m_μ, which is defined as $m_\mu(x) := 1/2^{\mu(x)}$. More specifically, following [5], it is not hard to show that, for every language L, L can be decided in polynomial-time on average with respect to m_μ if and only if its worst-case running time is $2^{O(\mu(x)-\mathsf{K}(x))}$ on input x (see [33, Lemma 25]). Then, essentially, to show our result we argue that L can be decided in polynomial time on average with respect to m_μ if and only if the same holds with respect to all P-samplable distributions.

Recall that if a distribution $\mathcal{D}$ *dominates* another distribution $\mathcal{D}'$ (i.e., $\mathcal{D}(x) \gtrsim \mathcal{D}'(x)$ for all x) and L is polynomial-time on average with respect to $\mathcal{D}$, then the same holds with respect to $\mathcal{D}'$ (see Definition 9 and Fact 10). Therefore, to replace m_μ above with P-samplable distributions, it suffices to show that m_μ is "universal" with respect to the class of P-samplable distributions, in the following sense.

1. m_μ dominates *every* P-samplable distribution. (This is essentially an optimal source coding theorem for the Kolmogorov measure μ.)
2. m_μ is dominated by *some* P-samplable distribution.

The above two conditions require two properties of the Kolmogorov measure μ that are somewhat conflicting: the first condition requires the notion of μ to be *general* enough so that m_μ can "simulate" *every* P-samplable distribution, while the second condition needs μ to be *restricted* enough so that m_μ can be "simulated" by *some* P-samplable (i.e., simple) distribution. For example, if μ is simply the time-unbounded Kolmogorov complexity K (or even the polynomial-space-bounded variant), then it is easy to establish an optimal source coding theorem for such a general Kolmogorov measure; however it is unclear how to sample in polynomial-time a string x with probability about $1/2^{\mathsf{K}(x)}$, so in this case μ does not satisfy the second condition. On the other hand, if μ is some restricted notion of time-bounded Kolmogrov complexity measure such as $\mathsf{K}^{\mathsf{poly}}$ or $\mathsf{rK}^{\mathsf{poly}}$, then one can obtain polynomial-time samplers that sample x with probability about $1/2^{\mathsf{K}^{\mathrm{poly}}(x)}$ or $1/2^{\mathsf{K}^{\mathrm{poly}}(x)}$ (up to a polynomial factor); however, as in [5], we only know how to show an *optimal* source coding theorem for $\mathsf{K}^{\mathsf{poly}}$ (or $\mathsf{rK}^{\mathsf{poly}}$) under a derandomization assumption. Therefore, in this case μ does not satisfies the first condition. Our key observation is that the notion $\mathsf{pK}^{\mathsf{poly}}$, which sits in between K and $\mathsf{K}^{\mathrm{poly}}$ (or $\mathsf{rK}^{\mathrm{poly}}$),[9] satisfies both conditions described above (see [33, Lemmas 36 and 37]).

4 Concluding Remarks and Open Problems

Our results indicate that Theorem 1 might be optimal among *efficient* coding theorems for rKt, i.e., those that efficiently produce representations matching the existential bounds. In the case of pK^t, the corresponding coding theorem (Theorem 5) is optimal. We have described a concrete application of Theorem 5 (Theorem 6). A second application appears in [17]. In both cases, achieving an optimal dependence on the probability parameter δ is critical, and for this reason, the result from [31] is not sufficient.

Naturally, we would like to understand the possibility of establishing an *unconditional* coding theorem for rKt with an optimal dependence on the probability parameter δ. While the validity of $\mathsf{Crypto\text{-}ETH}$ implies that no *efficient* coding theorem with this property exist, we have an *existential* coding theorem of this form under a derandomization assumption (Proposition 2). In the case of K^t complexity, it is known that an unconditional coding theorem with optimal dependence on δ implies that $\mathsf{EXP} \neq \mathsf{BPP}$ (see [25, Theorem 5.3.4]).

[9] We can show that for every $x \in \{0,1\}^*$ and every computable time bound $t\colon \mathbb{N} \to \mathbb{N}$, $\mathsf{K}(x) \lesssim \mathsf{pK}^t(x) \leq \mathsf{rK}^t(x) \leq \mathsf{K}^t(x)$.

However, the techniques behind this connection do not seem to lead to an interesting consequence in the case of rKt and rK^t. Consequently, an optimal coding theorem for rKt might be within the reach of existing techniques.

It would also be interesting to establish Theorem 3 under a weaker assumption, or to refute Crypto-SETH. A related question is the possibility of basing Crypto-ETH on the existence of one-way functions of exponential hardness. Existing reductions are not strong enough to provide an equivalence between one-way functions and cryptographic pseudorandomness in the exponential regime (see [38, 19]).

Finally, are there more applications of pK^t complexity and of Theorem 5? Since this coding theorem is both optimal and unconditional, we expect more applications to follow.

References

1 Scott Aaronson. The equivalence of sampling and searching. *Theory Comput. Syst.*, 55(2):281–298, 2014.

2 Eric Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In *Kolmogorov complexity and computational complexity*, pages 4–22. Springer, 1992.

3 Eric Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *International Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), pages 1–15. Springer, 2001.

4 Eric Allender. The complexity of complexity. In *Computability and Complexity*, pages 79–94. Springer, 2017.

5 Luis Filipe Coelho Antunes and Lance Fortnow. Worst-case running times for average-case algorithms. In *Conference on Computational Complexity* (CCC), pages 298–303, 2009.

6 Bruno Bauwens and Marius Zimand. Universal almost optimal compression and Slepian-Wolf coding in probabilistic polynomial time. *CoRR*, abs/1911.04268, 2019. arXiv:1911.04268.

7 Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992. doi:10.1016/0022-0000(92)90019-F.

8 Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Found. Trends Theor. Comput. Sci.*, 2(1), 2006.

9 Harry Buhrman, Lance Fortnow, and Sophie Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM J. Comput.*, 31(3):887–905, 2001.

10 Harry Buhrman, Troy Lee, and Dieter van Melkebeek. Language compression and pseudorandom generators. *Comput. Complex.*, 14(3):228–255, 2005.

11 M. R. Capalbo, O. Reingold, S. P. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *STOC*, pages 659–668, 2002. doi:10.1145/509907.510003.

12 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *CoRR*, abs/2203.00671, 2022. doi:10.48550/arXiv.2203.00671.

13 Kai-Min Chung, Siyao Guo, Qipeng Liu, and Luowen Qian. Tight quantum time-space tradeoffs for function inversion. In *Symposium on Foundations of Computer Science* (FOCS), pages 673–684, 2020.

14 Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *Annual International Cryptology Conference* (CRYPTO), pages 649–665, 2010.

15 Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. doi:10.1137/S0097539795280512.

16 Lance Fortnow. Kolmogorov complexity and computational complexity. *Complexity of Computations and Proofs. Quaderni di Matematica*, 13, 2004.

17 Halley Goldberg, Valentine Kabanets, Zhenjian Lu, and Igor C. Oliveira. Probabilistic Kolmogorov complexity with applications to average-case complexity. Preprint, 2022.

18 Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009.

19 Iftach Haitner, Omer Reingold, and Salil P. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. *SIAM J. Comput.*, 42(3):1405–1430, 2013.

20 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *Symposium on Foundations of Computer Science* (FOCS), pages 247–258, 2018.

21 Shuichi Hirahara. Average-case hardness of NP from exponential worst-case hardness assumptions. In *Symposium on Theory of Computing* (STOC), pages 292–302, 2021.

22 S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.

23 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

24 Jan Krajíček. Information in propositional proofs and algorithmic proof search. *The Journal of Symbolic Logic*, pages 1–22, 2021.

25 Troy Lee. *Kolmogorov complexity and formula lower bounds.* PhD thesis, University of Amsterdam, 2006.

26 Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984.

27 Leonid A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, 1986. doi:10.1137/0215020.

28 Ming Li and Paul M. B. Vitányi. Average case complexity under the universal distribution equals worst-case complexity. *Inf. Process. Lett.*, 42(3):145–149, 1992.

29 Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Springer-Verlag, 2019. 4th edition (1st edition in 1993).

30 Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. In *Symposium on Foundations of Computer Science* (FOCS), pages 1243–1254, 2020.

31 Zhenjian Lu and Igor C. Oliveira. An efficient coding theorem via probabilistic representations and its applications. In *International Colloquium on Automata, Languages, and Programming* (ICALP), pages 94:1–94:20, 2021.

32 Zhenjian Lu, Igor C. Oliveira, and Rahul Santhanam. Pseudodeterministic algorithms and the structure of probabilistic time. In *Symposium on Theory of Computing* (STOC), pages 303–316, 2021.

33 Zhenjian Lu, Igor C. Oliveira, and Marius Zimand. Optimal coding theorems in time-bounded Kolmogorov complexity, 2022. doi:10.48550/ARXIV.2204.08312.

34 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Symposium on Foundations of Computer Science* (FOCS), pages 253–262, 2013. doi:10.1109/FOCS.2013.35.

35 Igor C. Oliveira. Randomness and intractability in Kolmogorov complexity. In *International Colloquium on Automata, Languages, and Programming* (ICALP), pages 32:1–32:14, 2019.

36 Hanlin Ren and Rahul Santhanam. Hardness of KT characterizes parallel cryptography. In *Computational Complexity Conference* (CCC), pages 35:1–35:58, 2021.

37 Michael Sipser. A complexity theoretic approach to randomness. In *Symposium on Theory of Computing* (STOC), pages 330–335, 1983.

38 Salil P. Vadhan and Colin Jia Zheng. A uniform min-max theorem with applications in cryptography. In *Annual Cryptology Conference* (CRYPTO), pages 93–110, 2013.

A Definitions and Basic Results

Time-bounded Kolmogorov complexity. For a function $t\colon \mathbb{N} \to \mathbb{N}$, a string x, and a universal Turing machine U, let the time-bounded Kolmogorov complexity be defined as

$$\mathsf{K}^t_U(x) = \min_{p\in\{0,1\}^*} \{|p| \mid U(p) \text{ outputs } x \text{ in at most } t(|x|) \text{ steps}\}.$$

A machine U is said to be *time-optimal* if for every machine M there exists a constant c such that for all $x \in \{0,1\}^n$ and $t\colon \mathbb{N} \to \mathbb{N}$ satisfying $t(n) \geq n$,

$$\mathsf{K}_U^{ct \log t}(x) \leq \mathsf{K}_M^t(x) + c,$$

where for simplicity we write $t = t(n)$. It is well known that there exist time-optimal machines [29, Th. 7.1.1]. In this paper, we fix such a machine U, and drop the index U when referring to time-bounded Kolmogorov complexity measures. It is also possible to consider prefix-free notions of Kolmogorov complexity. However, since all our results hold up to additive $O(\log |x|)$ terms, we will not make an explicit distinction.

Henceforth we will not distinguish between a Turing machine $\mathcal{M}$ and its encoding p according to U. If p is a probabilistic Turing machine, we define $t_p \in \mathbb{N} \cup \{\infty\}$ to be the maximum number steps it takes p to halt on input λ (the empty string), where the maximum is over all branches of the probabilistic computation.

rKt complexity and probabilistic representations. A probabilistic representation of a string x is a probabilistic Turing machine p that on input λ halts with x on the output tape with probability at least $2/3$. The rKt-complexity of a string x is the minimum, over all probabilistic representations p of x, of $p + \log t_p$. A *probabilistic representation* p of x *certifies* rKt*-complexity bounded by* Γ if $|p| + \log t_p \leq \Gamma$.

Distributions and semi-distributions. We consider distributions over the set $\{0,1\}^*$. We will identify a distribution with its underlying probability density function of the form $\mathcal{D}\colon \{0,1\}^* \to [0,1]$. A distribution $\mathcal{D}$ is a *semi-distribution* if $\sum_{x \in \{0,1\}^*} \mathcal{D}(x) \leq 1$, and is simply called a *distribution* if the sum is exactly 1. In this subsection, we will use the word "distribution" to refer to both distribution and semi-distribution.

Samplers. A *sampler* is a probabilistic algorithm A with inputs in $\{1\}^n$ such that $A(1^n)$ outputs a string $x \in \{0,1\}^n$.[10] It defines a family of distributions $\{\mu_{A,n}\}_{n \in \mathbb{N}}$, where $\mu_{A,n}$ is the distribution on $\{0,1\}^n$ defined by $\mu_{A,n}(x) = \mathbf{Pr}_A[A(1^n) = x]$.

Average-case complexity. We now review some standard definitions and facts from average-case complexity. We refer to the survey [8] for more details.

▶ **Definition 7** (Polynomial-time Samplable [7])**.** *A distribution $\mathcal{D}$ is called* P*-samplable if there exists a polynomial p and a probabilistic algorithm M such that for every $x \in \{0,1\}^*$, M outputs x with probability $\mathcal{D}(x)$ within $p(|x|)$ steps.*

▶ **Definition 8** (Polynomial Time on Average [27])**.** *Let A be an algorithm and $\mathcal{D}$ be a distribution. We say that A runs in polynomial-time on average with respect to $\mathcal{D}$ if there exist constants ε and c such that,*

$$\sum_{x \in \{0,1\}^*} \frac{t_A(x)^\varepsilon}{|x|} \mathcal{D}(x) \leq c,$$

[10] For simplicity, we assume that $A(1^n)$ samples a string of length n. Our coding theorems also hold for algorithms used to define P-samplable distributions, see Definition 7, with obvious changes in the proofs. Also, as in [31], our results can be easily generalised to samplers that on 1^n output strings of arbitrary length. In this case, while the length of x might be significantly smaller than n, an additive overhead of $\log n + O(1)$ is necessary in our coding theorems, as we need to encode 1^n.

where $t_A(x)$ denotes the running time of A on input x. For a language L we say that L can be solved in polynomial time on average with respect to $\mathcal{D}$ if there is an algorithm that computes L and runs in polynomial-time on average with respect to $\mathcal{D}$.

▶ **Definition 9** (Domination). *Let $\mathcal{D}$ and $\mathcal{D}'$ be two distributions. We say that $\mathcal{D}$* dominates *$\mathcal{D}'$ if there is a constant $c > 0$ such that for every $x \in \{0,1\}^*$,*

$$\mathcal{D}(x) \geq \frac{\mathcal{D}'(x)}{|x|^c}.$$

▶ **Fact 10** (See e.g., [5, Lemma 3.3]). *Let $\mathcal{D}, \mathcal{D}'$ be two distributions, and let A be an algorithm. If*

- *A runs in polynomial time on average with respect to $\mathcal{D}$, and*
- *$\mathcal{D}$ dominates $\mathcal{D}'$*

Then A also runs in polynomial time on average with respect to $\mathcal{D}'$.

Max Weight Independent Set in Graphs with No Long Claws: An Analog of the Gyárfás' Path Argument

Konrad Majewski
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Tomáš Masařík
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Jana Novotná
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Karolina Okrasa
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Marcin Pilipczuk
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Paweł Rzążewski
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Marek Sokołowski
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Abstract

We revisit recent developments for the Maximum Weight Independent Set problem in graphs excluding a subdivided claw $S_{t,t,t}$ as an induced subgraph [Chudnovsky, Pilipczuk, Pilipczuk, Thomassé, SODA 2020] and provide a subexponential-time algorithm with improved running time $2^{\mathcal{O}(\sqrt{n}\log n)}$ and a quasipolynomial-time approximation scheme with improved running time $2^{\mathcal{O}(\varepsilon^{-1}\log^5 n)}$.

The Gyárfás' path argument, a powerful tool that is the main building block for many algorithms in P_t-free graphs, ensures that given an n-vertex P_t-free graph, in polynomial time we can find a set P of at most $t-1$ vertices, such that every connected component of $G-N[P]$ has at most $n/2$ vertices. Our main technical contribution is an analog of this result for $S_{t,t,t}$-free graphs: given an n-vertex $S_{t,t,t}$-free graph, in polynomial time we can find a set P of $\mathcal{O}(t\log n)$ vertices and an extended strip decomposition (an appropriate analog of the decomposition into connected components) of $G-N[P]$ such that every particle (an appropriate analog of a connected component to recurse on) of the said extended strip decomposition has at most $n/2$ vertices.

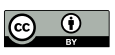
2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases Max Independent Set, subdivided claw, QPTAS, subexponential-time algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.93

Category Track A: Algorithms, Complexity and Games

Related Version *Preprint Version*: https://arxiv.org/abs/2203.04836 [22]



49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 93; pp. 93:1–93:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This research is part of projects that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement 714704 (JN, KO, MP, PRz) and 948057 (KM, TM, MS).
Paweł Rzążewski: Partially supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.

1 Introduction

The complexity of the Maximum Weight Independent Set problem (MWIS for short), one of the classic combinatorial optimization problems, varies depending on the restrictions imposed on the input graph from polynomial-time solvable (e.g., in bipartite or chordal graphs) through known to admit a quasipolynomial-time algorithm (graphs with bounded longest induced path [15]), a polynomial-time approximation scheme and a fixed-parameter algorithm (planar graphs [8]), a quasipolynomial-time approximation scheme (graphs excluding a fixed subdivided claw as an induced subgraph [10, 11]), to being NP-hard and hard to approximate within $n^{1-\varepsilon}$ factor in general graphs [20, 26]. A methodological study of this behavior leads to the following question:

> For which structures in the input graph, the assumption of their absence from the input graph makes MWIS easier and by how much?

The "absence of structures" notion can be made precise by specifying the forbidden structure and the containment relation, for example as a minor, topological minor, induced minor, subgraph, or induced subgraph. The last one – induced subgraph relation – is the weakest one, and thus the most expressible. This leads to the study of the complexity of MWIS in various hereditary graph classes, that is, graph classes closed under vertex deletion and thus definable by a (possibly infinite) list of forbidden induced subgraphs.

While a general classification of *all* hereditary graph classes with regards to the complexity of MWIS (or other classic graph problems) may be too complex, classifying graph classes with one forbidden induced subgraph looks more feasible. That is, we focus on H-free graphs, graphs excluding a fixed graph H as an induced subgraph. Furthermore, the complexity of a given problem (here, MWIS) in H-free graphs may indicate the impact of forbidding H as an induced subgraph on the complexity of MWIS in more general settings.

As observed by Alekseev [5, 6], the fact that MWIS remains NP-hard and APX-hard in subcubic graphs, together with the observation that subdividing every edge twice in a graph increases the size of the maximum independent set by exactly the number of edges of the original graph, leads to the conclusion that MWIS remains NP-hard and APX-hard in H-free graphs unless every connected component of H is a path or a tree with three leaves.

In what follows, for integers $t, a, b, c > 0$, by P_t we denote the path on t vertices, and by $S_{a,b,c}$ we denote the tree with three leaves within distance a, b, and c from the unique vertex of degree 3 of the tree. Since 1980s, it has been known that MWIS is polynomial-time solvable in P_4-free graphs (because of their strong structural properties) and in $S_{1,1,1}$-free graphs [23, 25] (because the notion of an augmenting path from the matching problem generalizes to MWIS in $S_{1,1,1}$-free, i.e., claw-free graphs). For many years, only partial results in subclasses were obtained until the area started to develop rapidly around 2014.

Lokshtanov, Vatshelle, and Villanger [21] adapted the framework of potential maximal cliques [9] to show a polynomial-time algorithm for MWIS in P_5-free graphs; this was later generalized to P_6-free graphs [17] and other related graph classes [3, 4]. More importantly for this work, Bacsó et al. [7] observed that the classic Gyárfás' path argument, developed to

show that for every fixed t the class of P_t-free graphs is χ-bounded [18, 19], also easily gives a subexponential-time algorithm for MWIS in P_t-free graphs. The crucial corollary of the Gyárfás' path argument lies in the following.

▶ **Theorem 1.** *Given an n-vertex graph G, one can in polynomial time find an induced path Q in G such that every connected component of $G - N[V(Q)]$ has at most $n/2$ vertices.*

For P_t-free graphs the said path Q has at most $t-1$ vertices. Bacsó et al. [7] observed that branching either on the highest degree vertex (if this degree is larger than $\sqrt{n}$) or on the whole set $N[V(Q)]$ for the path Q coming from Theorem 1 (otherwise) gives an algorithm with running time bound exponential in $\sqrt{n} \cdot \mathrm{poly}(t, \log n)$.

Chudnovsky, Pilipczuk, Pilipczuk, and Thomassé [10, 11] added to the mix an observation that a simple branching algorithm is able to get rid of *heavy* vertices: vertices of the input graph whose neighborhood contains a large fraction of the sought independent set. Once this branching is executed and the graph does not have heavy vertices, the set $N[Q]$ from Theorem 1 contains only a small fraction of the sought solution and, if one aims for an approximation algorithm, can be just sacrificed, yielding a quasipolynomial-time approximation scheme (QPTAS) for MWIS in P_t-free graphs. Using this as a starting point and leveraging on the celebrated *three-in-a-tree* theorem of Chudnovsky and Seymour [13], they developed a much more involved QPTAS and a subexponential algorithm (with running time bound $2^{n^{8/9}\mathrm{poly}(\log n, t)}$) for MWIS in $S_{t,t,t}$-free graphs.

Consider the following simple template for a branching algorithm for MWIS: if the current graph is disconnected, solve independently every connected component; otherwise, pick a vertex (*pivot*) v and branch whether v is in the sought independent set (recursing on $G - N[v]$) or not (recursing on $G - v$). The performance of such an algorithm highly depends on how we choose the pivot v. Theorem 1 suggests that in P_t-free graphs the vertices of Q may be good choices: there is only a bounded number of them, and the deletion of *the whole* neighborhood $N[V(Q)]$ splits G into multiplicatively smaller pieces. In a breakthrough result, Gartland and Lokshtanov [15] showed how to choose the pivot and measure the progress of the algorithm, obtaining a quasipolynomial-time algorithm for MWIS in P_t-free graphs. Later, Pilipczuk, Pilipczuk, and Rzążewski [24] provided an arguably simpler measure, leading to an improved (but still quasipolynomial) running time bound. These developments have been subsequently generalized to a larger class of problems beyond MWIS and to $C_{>t}$-free graphs (graphs without induced cycle of length more than t) [16].

This progress suggests that MWIS may be actually solvable in polynomial time in H-free graphs for all open cases, that is, whenever H is a forest whose every connected component has at most three leaves. However, we seem still far from proving it: not only we do not know how to improve the quasipolynomial bounds of [15, 24] to polynomial ones, but also it remains unclear how to merge the approach of [15, 24] with the way how [10, 11] used the three-in-a-tree theorem [13].

In this work, we make a step in this direction, providing an analog of Theorem 1 for $S_{t,t,t}$-free graphs. Before we state it, let us briefly discuss what we can hope for in the class of $S_{t,t,t}$-free graphs.

Consider an example of a graph G being the line graph of a clique K. The graph G is $S_{1,1,1}$-free, but does not admit any (balanced in any useful sense) separator of the form $N[P]$ for a small set $P \subseteq V(G)$. The MWIS problem on G translates back to the maximum weight matching problem in the clique K; this problem is polynomial-time solvable, but with very different methods than branching. In particular, we are not aware of any way of solving maximum weight matching in a clique in quasipolynomial time by simple branching.

Thus, we expect that an algorithm for MWIS in $S_{t,t,t}$-free graphs, given such a graph G, will discover that it is actually working with the line graph of a clique and apply maximum weight matching techniques to the preimage graph K.

Chudnovsky and Seymour, in their project to understand claw-free graphs [12], developed a good way of describing that a graph "looks like a line graph" by the notion of an *extended strip decomposition*. The formal definition can be found in Section 2. Here, we remark that in an extended strip decomposition of a graph, one can distinguish *particles* being induced subgraphs of the graph; an algorithm for MWIS can recurse on individual particles, compute the maximum weight independent sets there, and combine the results into a maximum weight independent set in the whole graph using a maximum weight matching algorithm on an auxiliary graph (cf. [10, 11]). Thus, an extended strip decomposition of a graph with particles of multiplicatively smaller size is very useful for recursion; it can be seen as an analog of splitting into connected components of multiplicatively smaller size, as it is in the case of the components of $G - N[V(Q)]$ in Theorem 1.

With the above discussion in mind, we can now state our main technical result.

▶ **Theorem 2.** *Given an n-vertex graph G and $t \geqslant 1$, one can in polynomial time either:*

- *output an induced copy of $S_{t,t,t}$ in G, or*
- *output a set $\mathcal{P}$ consisting of at most $11 \log n + 6$ induced paths in G, each of length at most $t+1$, and a rigid extended strip decomposition of $G - N[\bigcup_{P \in \mathcal{P}} V(P)]$ whose every particle has at most $n/2$ vertices.*

Combining Theorem 2 with previously known algorithmic techniques, we derive two algorithms for MWIS in $S_{t,t,t}$-free graphs. Actually, our algorithms work in a slightly more general setting. For integers $s, t \geqslant 1$, by $sS_{t,t,t}$ we denote the graph with s connected components, each isomorphic to $S_{t,t,t}$. Recall that by the observation of Alekseev [5, 6] the only graphs H, for which we can hope for tractability results for MWIS in H-free graphs, are forests whose every component has at most three leaves. We observe that each such H is contained in $sS_{t,t,t}$, for some s and t depending on H. Thus algorithms for $sS_{t,t,t}$-free graphs, for every s and t, cover *all* potential positive cases.

First, we observe that the statement of Theorem 2 seamlessly combines with the method how [7] obtained a subexponential-time algorithm for MWIS in P_t-free graphs. As a result, we obtain a subexponential-time algorithm for MWIS in $sS_{t,t,t}$-free graphs with improved running time as compared to [10, 11].

▶ **Theorem 3.** *Let $s, t \geqslant 1$ be constants. Given an n-vertex $sS_{t,t,t}$-free graph G with weights on vertices, one can in time exponential in $\mathcal{O}(\sqrt{n} \log n)$ compute an independent set in G of maximum possible weight.*

Second, we observe that the statement of Theorem 2 again seamlessly combines with the method how [10, 11] obtained a QPTAS for MWIS in P_t-free graphs, obtaining an arguably simpler QPTAS for MWIS in $sS_{t,t,t}$-free graphs with improved running time (compared to [10, 11]).

▶ **Theorem 4.** *Let $s, t \geqslant 1$ be constants. Given an n-vertex $sS_{t,t,t}$-free graph G with weights on vertices, and a real $\varepsilon > 0$, one can in time exponential in $\mathcal{O}(\varepsilon^{-1} \log^5 n)$ compute an independent set in G that is within a factor of $(1-\varepsilon)$ of the maximum possible weight.*

After preliminaries in Section 2, we prove Theorem 2 in Section 3. Proofs of Theorems 3 and 4 are provided in Section 4. Finally, we discuss future steps in Section 5.

2 Preliminaries

Notation. For a family $\mathcal{Q}$ of sets, by $\bigcup \mathcal{Q}$ we denote $\bigcup_{Q \in \mathcal{Q}} Q$. If the base of a logarithmic function is not specified, we mean the logarithm of base 2, i.e., $\log n := \log_2 n$. For a function $\mathfrak{w} : V \to \mathbb{Z}_{\geqslant 0}$ and subset $V' \subseteq V$, we denote $\mathfrak{w}(V') := \sum_{v \in V'} \mathfrak{w}(v)$.

Let G be a graph. For $X \subseteq V(G)$, by $G[X]$ we denote the subgraph of G induced by X, i.e., $(X, \{uv \in E(G) : u, v \in X\})$. If the graph G is clear from the context, we will often identify induced subgraphs with their vertex sets. The sets $X, Y \subseteq V(G)$ are *complete* to each other if for every $x \in X$ and $y \in Y$ the edge xy is present in G. Note that this, in particular, implies that X and Y are disjoint. We say that two sets X, Y *touch* if $X \cap Y \neq \emptyset$ or there is an edge with one end in X and another in Y.

For a vertex v, by $N_G(v)$ we denote the set of neighbors of v, and by $N_G[v]$ we denote the set $N_G(v) \cup \{v\}$. For a set $X \subseteq V(G)$, we also define $N_G(X) := \bigcup_{v \in X} N_G(v) - X$, and $N_G[X] = N_G(X) \cup X$. If it does not lead to confusion, we omit the subscript and write simply $N(\cdot)$ and $N[\cdot]$.

By $T(G)$, we denote the set of all triangles in G. Similarly to writing $xy \in E(G)$, we will write $xyz \in T(G)$ to indicate that $G[\{x, y, z\}] \simeq K_3$.

Extended strip decompositions. Now let us define a certain graph decomposition which will play an important role in the paper. An *extended strip decomposition* of a graph G is a pair (H, η) that consists of:

- a simple graph H,
- a set $\eta(x) \subseteq V(G)$ for every $x \in V(H)$,
- a set $\eta(xy) \subseteq V(G)$ for every $xy \in E(H)$, and its subsets $\eta(xy, x), \eta(xy, y) \subseteq \eta(xy)$,
- a set $\eta(xyz) \subseteq V(G)$ for every $xyz \in T(H)$,

which satisfy the following properties (also see Figure 1):

1. $\{\eta(o) \mid o \in V(H) \cup E(H) \cup T(H)\}$ is a partition of $V(G)$,
2. for every $x \in V(H)$ and every distinct $y, z \in N_H(x)$, the set $\eta(xy, x)$ is complete to $\eta(xz, x)$,
3. every $uv \in E(G)$ is contained in one of the sets $\eta(o)$ for $o \in V(H) \cup E(H) \cup T(H)$, or is as follows:
   - $u \in \eta(xy, x), v \in \eta(xz, x)$ for some $x \in V(H)$ and $y, z \in N_H(x)$, or
   - $u \in \eta(xy, x), v \in \eta(x)$ for some $xy \in E(H)$, or
   - $u \in \eta(xyz)$ and $v \in \eta(xy, x) \cap \eta(xy, y)$ for some $xyz \in T(H)$.

Note that for an extended strip decomposition (H, η) of a graph G, the number of vertices of H can be much larger than the number of vertices of G. However, in such case many sets $\eta(\cdot)$ are empty and thus H is "unnecessarily complicated." An extended strip decomposition (H, η) is *rigid* if (i) for every $xy \in E(H)$ it holds that $\eta(xy, x) \neq \emptyset$, and (ii) for every $x \in V(H)$ such that x is an isolated vertex it holds that $\eta(x) \neq \emptyset$. Observe that if we *restrict* η to $V' \subset V(G)$, i.e. we keep in η only vertices of V', (H, η) after the restriction remains an extended strip decomposition, but it might not be rigid any more.

▶ **Observation 5.** *Let (H, η) be a rigid extended strip decomposition of an n-vertex graph G. Then $|E(H)| \leqslant n$ and $|V(H)| \leqslant 2n$.*

Proof. Recall that since (H, η) is rigid, for every $xy \in E(H)$ we have that $\emptyset \neq \eta(xy, x) \subseteq \eta(xy)$, and for every isolated vertex x of H we have $\eta(x) \neq \emptyset$.

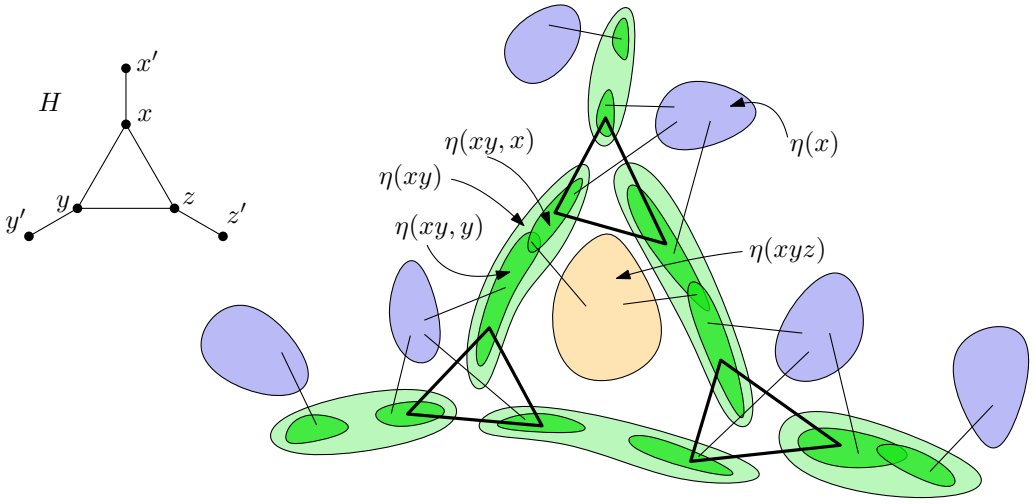


Figure 1 A graph H and an extended strip decomposition (H, η) of a graph G. Sets $\eta(\cdot)$ corresponding to vertices, edges, and the triangle of H are marked green, blue, and orange, respectively. The edges between distinct sets are drawn thick if they must exist, and thin if they may exist.

Let V_0 and V_+ denote, respectively, the sets of vertices of H with degree 0 and more than 0. As the family $\{\eta(xy) \mid xy \in E(H)\} \cup \{\eta(x) \mid x \in V_0\}$ consists of pairwise disjoint nonempty subsets of $V(G)$, we conclude that $|E(H)| + |V_0| \leqslant n$ and therefore $|E(H)| \leqslant n$.

Note that by the handshaking lemma we have $|E(H)| \geqslant |V_+|/2$, and so $|V(H)| = |V_0| + |V_+| \leqslant |V_0| + 2|E(H)| \leqslant 2n$ by the previous argument. ◀

We say that a vertex $v \in V(G)$ is *peripheral* in (H, η) if there is a degree-one vertex x of H, such that $\eta(xy, x) = \{v\}$, where y is the (unique) neighbor of x in H. For a set $Z \subseteq V(G)$, we say that (H, η) is an *extended strip decomposition of* (G, Z) if H has $|Z|$ degree-one vertices and each vertex of Z is peripheral in (H, η).

The following theorem by Chudnovsky and Seymour [13] is a slight strengthening of their celebrated solution of the famous *three-in-a-tree* problem. We will use it as a black-box to build extended strip decompositions.

▶ **Theorem 6** (Chudnovsky, Seymour [13, Section 6]). *Let G be an n-vertex graph and consider $Z \subseteq V(G)$ with $|Z| \geqslant 2$. There is an algorithm that runs in time $\mathcal{O}(n^5)$ and returns one of the following:*

- *an induced subtree of G containing at least three elements of Z,*
- *a rigid extended strip decomposition (H, η) of (G, Z).*

Let us point out that actually, an extended strip decomposition produced by Theorem 6 satisfies more structural properties, but of our purpose, we will only use the fact that it is rigid.

Particles of extended strip decompositions. Let (H, η) be an extended strip decomposition of a graph G. We introduce some special subsets of $V(G)$ called *particles*, divided into five *types*.

$$\begin{aligned}
\text{vertex particle:} &\quad A_x := \eta(x) \text{ for each } x \in V(H),\\
\text{edge interior particle:} &\quad A^{\perp}_{xy} := \eta(xy) - (\eta(xy,x) \cup \eta(xy,y)) \text{ for each } xy \in E(H),\\
\text{half-edge particle:} &\quad A^{x}_{xy} := \eta(x) \cup \eta(xy) - \eta(xy,y) \text{ for each } xy \in E(H),\\
\text{full edge particle:} &\quad A^{xy}_{xy} := \eta(x) \cup \eta(y) \cup \eta(xy) \cup \bigcup_{z\,:\,xyz \in T(H)} \eta(xyz) \text{ for each } xy \in E(H),\\
\text{triangle particle:} &\quad A_{xyz} := \eta(xyz) \text{ for each } xyz \in T(H).
\end{aligned}$$

Observe that the number of all particles of (H,η) is at most $\mathcal{O}(|V(H)|^3)$. However, the number of nonempty particles is linear in the number of vertices of G.

▶ **Observation 7.** *Let (H,η) be an extended strip decomposition of an n-vertex graph. Then the number of nonempty particles of (H,η) is bounded by $4n$.*

Proof. Let V', E', T', respectively, be the subsets consisting of those elements o of $V(H)$, $E(H)$, or $T(H)$, for which $\eta(o) \neq \emptyset$. Observe that each $o \in V' \cup T'$ gives rise to one nonempty particle A_o, and each $xy \in E'$ gives rise to at most four nonempty particles: $A^{\perp}_{xy}, A^{x}_{xy}, A^{y}_{xy}, A^{xy}_{xy}$. Moreover, since $\{\eta(o) \mid o \in V' \cup E' \cup T'\}$ are pairwise disjoint subsets of $V(G)$, we have that $|V'| + |E'| + |T'| \leqslant n$. Hence, the number of nonempty particles is bounded by $|V'| + |T'| + 4|E'| = (|V'| + |T'| + |E'|) + 3|E'| \leqslant 4n$. ◀

A vertex particle A_x is *trivial* if x is an isolated vertex in H. Similarly, an extended strip decomposition (H,η) is *trivial* if H is an edgeless graph. The following observation follows immediately from the definitions of an extended strip decomposition and particles.

▶ **Observation 8.** *Let (H,η) be an extended strip decomposition of a graph G. For each $xy \in E(H)$ the following hold:*

1. $A^{\perp}_{xy} \subseteq A^{x}_{xy} \subseteq A^{xy}_{xy}$,
2. *for any $v_x \in \eta(xy,x)$ and $v_y \in \eta(xy,y)$ we have $N(A^{xy}_{xy}) = N(v_x) \cup N(v_y) - A^{xy}_{xy}$.*

We conclude this section by recalling an important property of particles of extended strip decompositions, observed by Chudnovsky et al. [10].

▶ **Theorem 9** (Chudnovsky et al. [10, Lemma 6.8])**.** *Let (H,η) be an extended strip decomposition of G. Suppose P_1, P_2, P_3 are three induced paths in G that do not touch each other, and moreover each of P_1, P_2, P_3 has an endvertex that is peripheral in (H,η). Then in (H,η) there is no particle that touches each of P_1, P_2, P_3.*

3 Main result

In this section, we prove our main result, i.e., Theorem 2. Let us first give an overview of our approach. We present a recursive algorithm that, for a given graph G, will return one of the outcomes of Theorem 2. Let $n := |V(G)|$ be the number of vertices in the input graph; the value of n will not change throughout the recursive steps of the algorithm. We start with finding a Gyárfás path Q navigating towards the largest component in G. That is, by Theorem 1, we find Q such that each connected component of $G - N[Q]$ is of size at most $\frac{n}{2}$. Finding such a small connected component is a great outcome as we can readily include it as a small trivial vertex particle of an extended strip decomposition we are constructing. We say that a particle is *small* if its size is at most $\frac{n}{2}$, and an extended strip decomposition is *refined* if all its particles are small. Observe that if $|Q| \leq 3t+1$, we immediately get the

desired refined extended strip decomposition of G. Otherwise, we proceed to the main part of the algorithm. At each step, we will remove some vertices from Q, and will measure the progress of our algorithm in the number of the remaining vertices of Q.

Formally, we create a set $\mathcal{Q}$ of at most two non-touching induced paths such that $\bigcup \mathcal{Q} \subseteq Q$. At each step of recursion we obtain a set $\hat{\mathcal{Q}}$ with $|\bigcup \hat{\mathcal{Q}}| \leq \frac{2}{3}|\bigcup \mathcal{Q}|$ that represents $\mathcal{Q}$ for the next step. Hence, in $11 \log n$ recursive steps, $|\bigcup \mathcal{Q}|$ drops below $3t+1$. In the base case of the recursion, when $|\bigcup \mathcal{Q}| \leq 3t+1$, we return the refined trivial extended strip decomposition ensured by maintaining the property that $G - N[\bigcup \mathcal{Q}]$ has connected components of size at most $\frac{n}{2}$ throughout the recursive steps. In each step of recursion, we further split the induced path(s) in $\mathcal{Q}$ so we are able to use Theorem 6 to obtain an extended strip decomposition (H, η). If (H, η) is already refined, then we are done. Otherwise, it contains a particle A that is not small. We use Theorem 9 to select at most two paths touching A. Then it is easy to separate A with the respective touching paths from the rest of the graph. The graph induced by A and the touching paths form a smaller instance, i.e., an instance where $|\bigcup \mathcal{Q}|$ drops by a factor of $\frac{2}{3}$. We need to ensure that at every recursive step, we include only a constant number of paths of length $t+1$ into $\mathcal{P}$ (i.e., the set of paths in the second outcome of Theorem 2). We now prove the core recursive formulation of the algorithm formally.

▶ **Lemma 10** (Recursion). *Given a graph G and a set $\mathcal{Q}$ of at most two induced paths (vertex disjoint non-adjacent), and a refined extended strip decomposition of $G - N[\bigcup \mathcal{Q}]$. In polynomial time, we can output one of the following:*

- *an induced copy of $S_{t,t,t}$ in G, or*
- *$\mathcal{P}$, $X \subseteq N[\bigcup \mathcal{P}]$, and a refined extended strip decomposition (H, η) of $G - X$, so that $|\mathcal{P}| \leq 6 \log_{3/2}(|\bigcup \mathcal{Q}|) + 6$ and the longest path in $\mathcal{P}$ has at most $t+1$ vertices.*

Proof. If the longest path of $\mathcal{Q}$ has at most $3t+1$ vertices, return $\mathcal{P} \coloneqq \mathcal{Q}$ where each path in $\mathcal{P}$ may be further split in at most three paths on at most $t+1$ vertices, and $X \coloneqq N[\bigcup \mathcal{P}]$. Hence, we output the extended strip decomposition we were given by the assumptions of the lemma.

Otherwise, let Q_1 be the longest path in $\mathcal{Q}$. Let u_1 and u_2 be the $\left(\left\lfloor \frac{|Q_1|}{3} \right\rfloor + 1\right)$-th and the $\left(2\left\lfloor \frac{|Q_1|}{3} \right\rfloor + 2\right)$-th vertex of Q_1, respectively. The removal of u_1 and u_2 from Q_1 divides the path into three induced non-touching subpaths Q_1^1, Q_1^2, and Q_1^3, each of length at least t. Let Q_2 be the remaining path of $\mathcal{Q}$, should it exist. We define $\mathcal{S} \coloneqq \{Q_1^1, Q_1^2, Q_1^3, Q_2\}$ if Q_2 exists, or $\mathcal{S} \coloneqq \{Q_1^1, Q_1^2, Q_1^3\}$, otherwise. Consult Figure 2 to see an overview of the definitions described in this paragraph. For each path $P \in \mathcal{S}$ we define $\mathrm{pref}(P)$ as the set comprising:

- first $t-1$ vertices of P (or all vertices of P if $|P| < t-1$), and
- the separating vertex of Q_1 directly preceding P if $P \in \{Q_1^2, Q_1^3\}$.

It can be easily seen that the set of vertices $\mathrm{pref}(P)$ forms an induced path of length at most t. We finally define *shells* of paths in $\mathcal{S}$. Given a path $P \in \mathcal{S}$, we set $\mathrm{shell}(P) \coloneqq N[\mathrm{pref}(P)] - \bigcup \mathcal{S}$ if $|P| \geq t$ and $\mathrm{shell}(P) \coloneqq N[\mathrm{pref}(P)]$ otherwise. Intuitively, if $|P| < t$, the shell of P takes the whole neighborhood as we do not have a use for a short path in the next stage of our algorithm. For a long enough path P ($|P| > t$), the shell of P intersects all short paths (shorter than t) connecting the first vertex of P with the rest of the graph. In other words, each path from the first vertex of P to any vertex of $G - \mathrm{shell}(P)$ outside of P will have length at least t. To ease the notation, we define $\mathcal{S}_{\geq t} \coloneqq \{P \in \mathcal{S} \mid |P| \geq t\}$, $\mathrm{shell}(\mathcal{S}) \coloneqq \bigcup_{P \in \mathcal{S}} \mathrm{shell}(P)$, and $\mathrm{pref}(\mathcal{S}) \coloneqq \bigcup_{P \in \mathcal{S}} \mathrm{pref}(P)$.

Now, we use the algorithm from Theorem 6 on Z being the set of the first vertices of paths in $\mathcal{S}_{\geq t}$ and the graph defined as $G - \mathrm{shell}(\mathcal{S})$. If Theorem 6 produced an induced tree with three leaves among Z, it contains an induced $S_{t,t,t}$, since those must have been

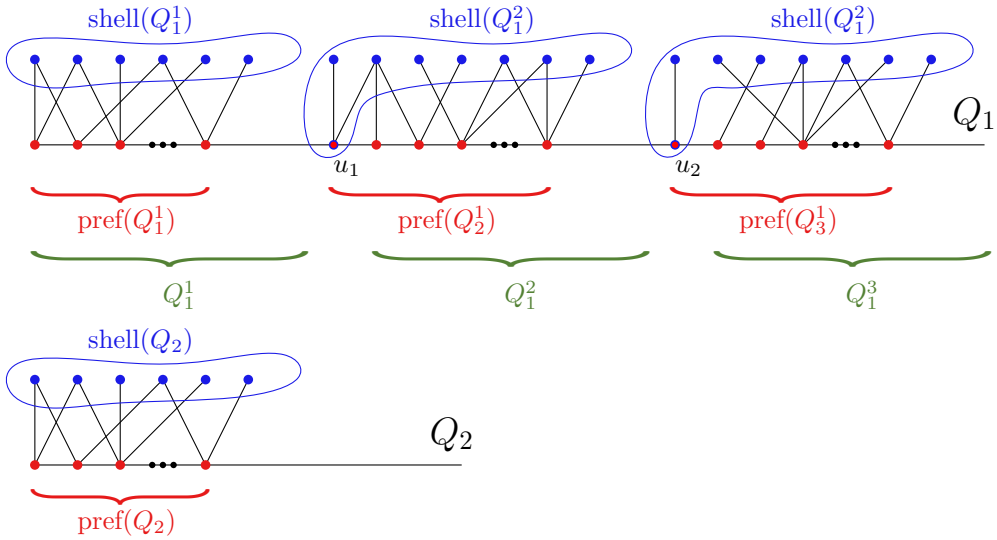


Figure 2 Definitions of pref($\mathcal{S}$) and shell($\mathcal{S}$) in case of $|Q_2| \geq t$.

induced branches at least t vertices long in $G - \text{shell}(\mathcal{S})$. Hence, we obtained an extended strip decomposition (H', η') of $G - \text{shell}(\mathcal{S})$. If the obtained decomposition is refined, we return $\mathcal{P} \coloneqq \text{pref}(\mathcal{S})$, $X \coloneqq \text{shell}(\mathcal{S})$, and the extended strip decomposition $(H \coloneqq H', \eta \coloneqq \eta')$.

Therefore, the obtained extended strip decomposition (H', η') of $G - \text{shell}(\mathcal{S})$ contains a particle A which is not small, i.e., A is composed of at least $\frac{n}{2}$ vertices. As every vertex in Z is peripheral in (H', η'), we know that no three paths in $\mathcal{S}_{\geq t}$ touch one particle by Theorem 9. Therefore, we take the set $\hat{\mathcal{Q}}$ of at most two paths, say P_1 and P_2, touching A (for convenience, let P_1 or P_2 be an empty set if it does not exist). We now compute the maximum proportion of $\bigcup \mathcal{Q}$ put to $\hat{\mathcal{Q}}$. If both $P_1, P_2 \subseteq Q_1$, then this fraction is at most $\frac{2}{3}$ as by the definition $|Q_1^i| \leq \frac{|Q_1|}{3}$, for $i \in \{1, 2, 3\}$. If one is Q_2 and the other comes from Q_1, then we estimate $a + \frac{1-a}{3} = \frac{2a+1}{3} \leq \frac{2}{3}$ for $a = |Q_2|/|\bigcup \mathcal{Q}| \leq \frac{1}{2}$. Hence, we know that $|\bigcup \hat{\mathcal{Q}}| \leq \frac{2}{3}|\bigcup \mathcal{Q}|$. We define $\hat{G} \coloneqq A \cup P_1 \cup P_2$ to use Lemma 10 on a smaller instance. Now, we need to verify that the assumption of the lemma holds. We claim the following:

▷ Claim 11. $\hat{G} - N[\bigcup \hat{\mathcal{Q}}]$ has a refined extended strip decomposition.

Proof. As $\hat{G}$ is an induced subgraph of G and $G - N[\bigcup \mathcal{Q}]$ has a refined extended strip decomposition, we know that $\hat{G} - N[\bigcup \mathcal{Q}]$ has a refined extended strip decomposition. First, recall that $N[u_1] - (Q_1^1 \cup Q_1^2) \subseteq \text{shell}(Q_1^2)$, which is disjoint with $V(\hat{G})$. Analogously $N[u_2] - (Q_1^2 \cup Q_1^3)$ is disjoint with $V(\hat{G})$. Also, if $|Q_2| < t$ then Q_2 is disjoint with $V(\hat{G})$ as well. Hence, $\hat{G} - N[\bigcup \mathcal{Q}] \simeq \hat{G} - N[\bigcup \mathcal{S}_{\geq t}]$. Also, recall that the only paths among $\mathcal{S}_{\geq t}$ that touch A are in $\hat{\mathcal{Q}}$. Hence, observe that $\hat{G} - N[\bigcup \mathcal{S}_{\geq t}] \simeq \hat{G} - N[\bigcup \hat{\mathcal{Q}}]$. ◁

Therefore, we can apply Lemma 10 inductively on $\hat{G}$ and $\hat{\mathcal{Q}}$, obtaining $\hat{\mathcal{P}}$ and $\hat{X}$, and a refined extended strip decomposition $(\hat{H}, \hat{\eta})$ of $\hat{G} - \hat{X}$. We need to combine the extended strip decomposition obtained from the recursion with the extended strip decomposition (H', η') we obtained earlier.

We can always suppose that particle A is of type A_{xy}^{xy} for some edge $xy \in E(H')$, unless A is of type A_x for an isolated vertex $x \in V(H')$. That is because A_{xy}^{xy} is the superset of all possible particle types. As Theorem 6 gives us that both $\eta'(xy, x)$ and $\eta'(xy, y)$ are nonempty, we can select $v_x \in \eta'(xy, x)$ and $v_y \in \eta'(xy, y)$ (possibly $v_x = v_y$). By Observation 8, the set

$$X' \coloneqq (N(v_y) \cup N(v_x)) - V(A)$$

separates A from the rest of G. Set $\mathcal{P}' \coloneqq \{\{v_x\}, \{v_y\}\}$. In the case of A_x such that $x \in V(H)$ is an isolated vertex, we set $\mathcal{P}' \coloneqq \emptyset$ and $X' \coloneqq \emptyset$ and still such A is separated from the rest of G by X'. We return:

- $\mathcal{P} \coloneqq \hat{\mathcal{P}} \cup \mathcal{P}' \cup \operatorname{pref}(\mathcal{S})$,
- $X \coloneqq \hat{X} \cup X' \cup \operatorname{shell}(\mathcal{S})$,
- an extended strip decomposition (H, η) of $G - X$, where H is $\hat{H}$ with an additional isolated vertex w, and η is $\hat{\eta}$ restricted only to vertices in $A - X$ with an additional trivial vertex particle $\eta(w)$ containing all vertices of $G - X - A$.

Recall that during the recursion, we do not require rigidity, therefore, we do not mind restricting η only to a subset of vertices. Note that indeed, $G - X - A$ may contain parts of P_1 or P_2, however, $\eta(w)$ does not touch any vertices contained in $\hat{\eta}$ restricted to $A - X$ as $X' \subseteq X$ completely separated $A - X$ from $G - X - A$.

We compute that $|\mathcal{P}| \leq 6 + 6\log_{3/2}\left(\left|\bigcup \hat{\mathcal{Q}}\right|\right) + 6 \leq 6\log_{3/2}(|\bigcup \mathcal{Q}|) + 6$ as we added at most six new paths into $\mathcal{P}$. Observe that the described algorithm runs in polynomial time as we just computed that the depth of recurrence is logarithmic in $|\bigcup(\mathcal{Q})| \leq |V(G)|$ and each recursive call takes polynomial time in the size of G. ◀

Proof of Theorem 2. Using Theorem 1 we find a Gyárfás path Q. We get the desired outcome by Lemma 10 on G with $\mathcal{Q} \coloneqq \{Q\}$. The extended strip decomposition needed by the lemma's assumption is trivial. That is, each connected component of $G - Q$ is represented by a vertex particle of small size. Note that Lemma 10 provides an extended strip decomposition of $G - X$, where $X \subseteq \bigcup \mathcal{P}$ and Theorem 2 only requires an extended strip decomposition of $G - \bigcup \mathcal{P}$, so we can restrict the obtained extended strip decomposition to $V(G) - \bigcup \mathcal{P}$. We conclude the proof of Theorem 2 by the following calculation:

$$6\log_{3/2} n + 6 \leq 11 \log n + 6.$$

Note that for any extended strip decomposition (H, η) we can easily add the assumption that sets $\eta(xy, x) \neq \emptyset$ for any edge $xy \in E(H)$. As suppose $\eta(xy, x) = \emptyset$; then we can update (H, η) by adding $\eta(xy)$ to $\eta(y)$ and removing xy from H. Moreover, we can simply remove any empty trivial vertex particle form η and the corresponding isolated vertex from H. Therefore, we may suppose that the obtained extended strip decomposition is rigid. ◀

In the following simple corollary we apply Theorem 2 to $sS_{t,t,t}$-free graphs, for some $s, t \geq 1$.

▶ **Corollary 12.** *Let $s \geqslant 1, t \geqslant 1$ be constants. Let G be an $sS_{t,t,t}$-free graph on n vertices. Then in polynomial time we can find a set X consisting of at most*

$$(s-1)(3t+1) + (11\log n + 6)(t+1)$$

vertices and a rigid extended strip decomposition of $G - N[X]$ whose every particle has at most $n/2$ vertices.

Proof. Induction on s. If $s = 1$, then we obtain the result immediately by Theorem 2. Thus let us assume that $s \geqslant 2$ and the theorem holds for $(s-1)S_{t,t,t}$-free graphs.

We exhaustively check if there is some $Y \subseteq V(G)$ with $|Y| = 3t+1$, such that $G[Y] \simeq S_{t,t,t}$; we can do it in time $n^{3(t+1)+\mathcal{O}(1)} = n^{\mathcal{O}(1)}$. If such Y does not exist, then we can immediately apply Theorem 2, and the proof is complete. Thus suppose that Y exists.

We observe that the graph $G' \coloneqq G - N[Y]$ is $(s-1)S_{t,t,t}$-free. Denote $n' \coloneqq |V(G')|$. By the inductive assumption, in time $(n')^{\mathcal{O}(1)} = n^{\mathcal{O}(1)}$ we can obtain a set $X' \subseteq V(G')$ of size at most $(s-2)(3t+1) + (11\log n' + 6)(t+1)$ and a rigid extended strip decomposition (H, η) of $G' - N[X']$ whose every particle is of size at most $n'/2$.

We set $X = Y \cup X'$. Now X and (H, η) satisfy the statement of the theorem, as $G' - N[X'] = G - N[X]$ and $n' \leqslant n$. The total running time is polynomial in n as the depth of the recursion is $s - 1$. ◀

4 Algorithmic applications

In this section we will show how to combine Theorem 2 with the approach of Chudnovsky et al. [10, 11] in order to obtain a QPTAS and a subexponential-time algorithm for MWIS in $S_{t,t,t}$-free graphs, i.e., we prove Theorems 3 and 4.

Both algorithms follow the same general outline; let us sketch it before we get into the details of each particular case. Each algorithm is a recursive procedure, which consists of two phases. In the first one, we deal with the vertices of G that are *heavy*, which means that their neighborhood is "large", where the exact meaning of "large" depends on the particular algorithm.

Once there are no heavy vertices, i.e., the neighborhood of each vertex is "small", we proceed to the second phase. We call Corollary 12 for the current instance G, obtaining a small-sized set X and a rigid extended strip decomposition (H, η) of $G - N[X]$, whose every particle is of small size. The crux is that since we are in the second phase, all vertices in X are not heavy, and since X is of small size, the whole set $N[X]$ is "small". We treat $N[X]$ separately in a way that depends on the particular algorithm.

Next, for each particle A of (H, η), we call the algorithm recursively for $G[A]$, obtaining (a good approximation of) a maximum-weight independent set in $G[A]$. Finally, we combine the obtained results to derive (a good approximation of) a maximum-weight independent set in G. This last step is based on the idea of Chudnovsky et al. [10, 11] to reduce the problem to finding a maximum-weight matching in a graph obtained by a simple modification of H. Since the size of H is linear in $|V(G)|$ (by Observation 5), this problem can be solved in time polynomial in $|V(G)|$ using, e.g., the classic algorithm of Edmonds [14]. The last step is encapsulated in the following lemma, whose exact statement comes from Abrishami et al. [1].

▶ **Lemma 13** (Chudnovsky et al. [10, 11]). *Let $\varsigma \in [0, 1]$ be a real number. Let G be an n-vertex graph equipped with a weight function $\mathfrak{w} : V(G) \to \mathbb{Z}_{\geqslant 0}$. Suppose that G is given along with an extended strip decomposition (H, η), where H has N vertices.*
Let $I_0 \subseteq V(G)$ be a fixed independent set in G. Furthermore, assume that for each particle A of (H, η) we are given an independent set $I(A)$ in $G[A]$ such that $\mathfrak{w}(I(A)) \geqslant \varsigma \cdot \mathfrak{w}(I_0 \cap A)$. Then in time polynomial in $n + N$ we can compute an independent set I in G such that $\mathfrak{w}(I) \geqslant \varsigma \cdot \mathfrak{w}(I_0)$.

Let us stress out that the algorithm from Lemma 13 does not need to know the value of ς or the independent set I_0.

The main difference between our approach and the one of Chudnovsky et al. [11] is that we use Theorem 2 and its consequence, i.e., Corollary 12. The previous algorithms used a similar statement but with a worse (and much more involved) guarantee on the size of X and each particle. Furthermore, the way we obtain our set X is significantly simpler.

4.1 Proof of Theorem 3

Before we proceed to the proof, let us first explain the meaning of "small", and how to deal with $N[X]$ in this particular case. Here the neighborhood of a vertex is "small" if it has few vertices (more specifically, at most $\sqrt{n/t}$). In the first phase, we deal with heavy vertices v (i.e., of large degree) with simple branching: we guess whether v is included in our optimum solution or not. Since the degree of v is large, in the first branch, we obtain significant progress, which is enough to obtain a subexponential running time.

In the second phase, since $N[X]$ is the neighborhood of $\mathcal{O}(\log n)$ vertices, each of degree $\mathcal{O}(\sqrt{n})$, the total size of $N[X]$ is $\mathcal{O}(\sqrt{n}\log n)$. Thus we can afford to exhaustively guess the intersection of our optimum solution with $N[X]$.

Proof of Theorem 3. Let $s,t \geqslant 1$ be constants and let $(G,\mathfrak{w})$ be an instance of MWIS, where G is $sS_{t,t,t}$-free and has n vertices. We observe that if n is small, i.e., bounded by a constant, then we can solve the problem by brute force. Thus we assume that $n \geqslant n_0$, where n_0 is a constant (depending on s and t) whose exact value follows from the reasoning below.

First, consider the case that there exists $v \in V(G)$ such that $\deg v \geqslant \sqrt{n/t}$. We branch on including v in the final solution: we either delete v from G, or we delete $N[v]$ and add v to the solution returned by the recursive call. Then we output the one of these two solutions that has a larger weight. The correctness of this step of the algorithm is straightforward.

Hence, we can assume that for every $v \in V(G)$ it holds that $\deg v \leqslant \sqrt{n/t}$. By Corollary 12, since G is $sS_{t,t,t}$-free, we obtain a set X of size $(s-1)(3t+1)+(11\log n+6)(t+1) \leqslant 12(t+1)\log n$ (here we use that n is large), and a rigid extended strip decomposition (H,η) of $G' = G - N[X]$ whose every particle has at most $n/2$ vertices.

We exhaustively guess an independent set $J \subseteq N[X]$; think of it as an intersection of the intended optimum solution with $N[X]$. Consider the graph $G'' := G' - N[J]$. We modify (H,η) by removing the vertices from $N[J]$ from the sets $\eta(\cdot)$. Let us call the obtained strip decomposition (H,η'); note that it might not be rigid. We call the algorithm recursively for the subgraph $G''[A]$ for every nonempty particle A of (H,η'). Let $I(A)$ be the solution. If $A = \emptyset$, then $I(A) = \emptyset$. By the inductive assumption $I(A)$ is a maximum-weight independent set in $G''[A]$. Then we use Lemma 13 for $\varsigma = 1$ to combine the solutions into a maximum-weight independent set I_J of G''. Finally, we return the independent set $J \cup I_J$ whose weight is maximum over all choices of J. Note that the correctness of this step is guaranteed by the exhaustive guessing of J and Lemma 13.

Running time. Let $F(n)$ denote the running time of our algorithm for n-vertex instances. We prove that $F(n) = 2^{\mathcal{O}(\sqrt{tn}\log n)}$. If $n < n_0$, then the claim clearly holds. So let us assume that $n \geqslant n_0$.

In the first case we call the algorithm for two instances, one of size $n-1$ and one of size at most $n - \sqrt{n/t}$. Hence,

$$F(n) \leqslant F(n-1) + F(n-\sqrt{n/t}) = 2^{\mathcal{O}\left(\frac{n\log n}{\sqrt{n/t}}\right)} \leqslant 2^{\mathcal{O}(\sqrt{tn}\log n)}.$$

Here we skip the description how this recursion is solved, as it is pretty standard. For a formal proof we refer the reader to Bacsó et al. [7, Lemma 1].

It remains to analyze the running time of the step in which the maximum degree of vertices in G is bounded by $\sqrt{n/t}$. Corollary 12 asserts that we obtain X and the rigid extended strip decomposition (H,η) of $G' = G - N[X]$ in time polynomial in n. There are $2^{\mathcal{O}(\sqrt{n/t}\cdot t\log n)} = 2^{\mathcal{O}(\sqrt{nt}\log n)}$ ways of choosing the set J. In polynomial time we modify (H,η) into (H,η').

Observe that while (H,η') might not be rigid, it was obtained from a rigid extended strip decomposition (H,η) by deleting some vertices from the sets $\eta(\cdot)$. In particular, both decompositions have the same sets of particles, and every nonempty particle of (H,η') is also a nonempty particle of (H,η). Thus by Observation 7 we call the algorithm recursively for at most $4n$ nonempty particles, each of size at most $n/2$. By Observation 5, the total number of particles of (H,η') is polynomial in n. Finally, having computed a maximum-weight

independent set contained in each particle, by Lemma 13, we can compute the final solution in time polynomial in n. Hence, there are constants c, c_1, c_2, where $c \gg c_1, c_2$, such that total running time of this step is bounded by:

$$F(n) \leqslant 2^{c_1 \cdot \sqrt{nt} \log n} \left(n^{c_2} + 4n \cdot 2^{c \cdot \sqrt{tn/2} \log (n/2)} \right) \overset{c \gg c_1, c_2}{\leqslant} 2^{c \cdot \sqrt{tn} \log n}, \tag{1}$$

and so is the total complexity of the algorithm. ◀

4.2 Proof of Theorem 4

Again let us start with explaining the algorithm-specific details of the outline presented at the start of Section 4.

We will use the notion of β-heavy vertices from [10, 11]. Consider a graph G, a weight function $\mathfrak{w} : V(G) \to \mathbb{Z}_{\geqslant 0}$, and an independent set $I \subseteq V(G)$. Let $\beta \in (0, 1/2]$ be a real. We say that a vertex $v \in V(G)$ is *β-heavy* (with respect to I) if $\mathfrak{w}(N[v] \cap I) > \beta \cdot \mathfrak{w}(I)$. A set J is *good for I* if $J \subseteq I$ and $N[J]$ contains all vertices that are β-heavy with respect to I.

▶ **Lemma 14** (Chudnovsky et al. [10, 11]). *Let G be an n-vertex graph for $n > 2$, $\mathfrak{w} : V(G) \to \mathbb{Z}_{\geqslant 0}$ be a weight function, $I \subseteq V(G)$ be an independent set, and $\beta \in (0, 1/2]$ be a real. Then there exists a set J of size at most $\lceil \beta^{-1} \log n \rceil$ which is good for I.*

Now the vertex is heavy if it is β-heavy for some carefully chosen parameter β. This means that a neighborhood of a vertex is "large" if it contains a significant ($\geqslant \beta$) fraction of the weight of I_{OPT}. In the first phase, we exhaustively guess the set J that is good for a fixed optimum solution I_{OPT}. Note that J is of small size and since $J \subseteq I_{\mathsf{OPT}}$, we know that $N(J)$ contains no vertices from I_{OPT} and thus can be safely removed from the graph.

Since J is good for I_{OPT}, we know that $G - N[J]$ contains no heavy vertices, and for this graph we call Corollary 12. Now, as $N[X]$ is a neighborhood of few non-heavy vertices, we know that the total weight of $I_{\mathsf{OPT}} \cap N[X]$ is small and thus can be sacrificed, as we aim for an approximation.

Proof of Theorem 4. Let $s, t \geqslant 1$ be constants and let $(G, \mathfrak{w})$ be an instance of MWIS, where G is $sS_{t,t,t}$-free and has n vertices. Let $\varepsilon \in (0, 1)$ be fixed. Fix a maximum-weight independent set I_{OPT} in G with respect to $\mathfrak{w}$. We describe a procedure that finds in G an independent set I of weight at least $(1 - \varepsilon) \cdot \mathfrak{w}(I_{\mathsf{OPT}})$.

Let N be the minimum power of two greater than or equal to the size of our initial instance. Note that $n \leqslant N < 2n$. The value of N will not change throughout the execution of the algorithm.

The algorithm itself is a recursive procedure. The arguments of each call are a graph G', which is an induced subgraph of G, the weight function on $V(G')$ obtained by restricting the domain of $\mathfrak{w}$, and an integer h, which can be intuitively understood as the depth of the current call in the recursion tree. Since it does not lead to confusion, we will always denote the weight function by $\mathfrak{w}$. We will keep the invariant that for each call $(G', \mathfrak{w}, h)$ it holds that $|V(G')| \leqslant N/2^h$. The initial call, corresponding to the root of the recursion tree, is for $(G, \mathfrak{w}, 0)$.

Consider a call for the instance $(G', \mathfrak{w}, h)$. If $|V(G')| < n_0$, where n_0 is a constant (depending on s and t) that follows from the reasoning below, then we can solve the problem by brute force. Thus let us assume that $n \geqslant n_0$. In particular, $N > 1$.

We set

$$\beta(h,\varepsilon) := \frac{\varepsilon}{12(t+1)\log(N/2^h)\cdot((1-\varepsilon)\log N + \varepsilon(h+1))}. \tag{2}$$

It is straightforward to verify that for $h < \log N$ we have $\beta(h,\varepsilon) \in (0, 1/2]$. On the other hand, if $h \geqslant \log N$, then G' is of constant size and thus $\beta(h,\varepsilon)$ is not computed for such h.

Let $\mathcal{J}$ be the family of all independent sets in G' of size at most $\lceil \beta(h,\varepsilon)^{-1} \log(N/2^h)\rceil$. For each $J \in \mathcal{J}$ we proceed as follows. If $|V(G'-N[J])| < n_0$, then we compute a maximum-weight independent set I_J in $G'-N[J]$ by brute force. Otherwise, we use Corollary 12, to obtain a set $X_J \subseteq V(G'-N[J])$ and a rigid extended strip decomposition (H,η) of $G'-N[J]-N[X_J]$ such that each particle of (H,η) is of size at most $|V(G'-N[J])|/2$. By Corollary 12, we obtain

$$\begin{aligned} |X_J| &\leqslant (s-1)(3t+1) + (11\log|V(G'-N[J])| + 6)(t+1) \\ &\leqslant 12(t+1)\log|V(G')| \leqslant 12(t+1)\log(N/2^h). \end{aligned} \tag{3}$$

Let $Y_J := N(J) \cup N[X_J]$. We modify (H,η) into an extended strip decomposition of $G'-Y_J$ as follows. For each $v \in J$, we add to H an isolated vertex x_v, and set $\eta(x_v) = \{v\}$.[1] Let us call this extended strip decomposition (H',η'). Observe that each particle of (H',η') is of size at most $|V(G'-N[J])|/2 \leqslant |V(G')|/2$. Furthermore, since (H,η) is rigid, so is (H',η').

For each nonempty particle A of (H',η') we call the algorithm recursively on an instance $(G'[A], \mathfrak{w}, h+1)$. Let $I(A)$ be the value returned by the algorithm. For each empty particle A we set $I(A) := \emptyset$. Finally, we apply the algorithm from Lemma 13, in order to obtain an independent set I_J of $G'-Y_J$ and thus of G'. Recall that the value of ς is not needed to apply Lemma 13; we will define it in the next paragraph when we discuss the approximation guarantee. As the solution, we return the set I_J of maximum weight, over all choices of $J \in \mathcal{J}$.

Approximation guarantee. Consider the recursion tree of our algorithm. We mark some nodes of the recursion tree. First, we mark the root. Now consider some marked node z corresponding to a call $(G',\mathfrak{w},h)$, such that z is not a leaf node. Observe that by Lemma 14, there is some $J \in \mathcal{J}$ (for this particular instance) which is good for $I_{\mathsf{OPT}} \cap V(G')$. Fix such J. If there is more than one, we choose one arbitrarily. We mark the children of z that correspond to the calls on the particles of the extended strip decomposition of $G'-Y_J$.

Let $\mathcal{T}$ be the subtree of the recursion tree induced by the marked nodes. Note that each leaf of $\mathcal{T}$ is a leaf of the whole recursion tree, i.e., it corresponds to an instance of constant size. Since at each level of the recursion, the size of the instance drops by at least half, we observe that each instance at level h (where the root is at level 0) is of size at most $N/2^h$. Consequently, the depth of $\mathcal{T}$ is at most $\log N$.

Consider a call for an instance $(G',\mathfrak{w},h)$ and let J be good for I_{OPT}. Let us estimate $\mathfrak{w}(I_{\mathsf{OPT}} \cap Y_J)$. First, observe that since $J \subseteq I_{\mathsf{OPT}}$, we have that $\mathfrak{w}(I_{\mathsf{OPT}} \cap N(J)) = 0$. Moreover, since J was chosen to be good, there are no $\beta(h,\varepsilon)$-heavy vertices in $V(G'-N[J])$, and in particular, in $N[X_J]$. Hence,

$$\begin{aligned} \mathfrak{w}(I_{\mathsf{OPT}} \cap Y_J) = \mathfrak{w}(I_{\mathsf{OPT}} \cap N[X_J]) &\leqslant |X_J| \cdot \beta(h,\varepsilon) \cdot \mathfrak{w}(I_{\mathsf{OPT}} \cap V(G')) \\ &\overset{(2) \text{ and } (3)}{\leqslant} \frac{\varepsilon}{(1-\varepsilon)\log N + \varepsilon(h+1)} \cdot \mathfrak{w}(I_{\mathsf{OPT}} \cap V(G')). \end{aligned} \tag{4}$$

[1] Another possible way of dealing with the set J would be to add it directly in the computed solution. However, we decided to restore J to the graph, so that these vertices are handled by Lemma 13 and do not require any special treatment.

The following claim shows that the solution computed for the instance $(G', \mathfrak{w}, h)$ at each node of $\mathcal{T}$ is a reasonable approximation of $I_{\mathsf{OPT}} \cap V(G')$.

▷ Claim 15. Let z be a node of $\mathcal{T}$, and let $(G', \mathfrak{w}, h)$ be the instance corresponding to z. Let I be the independent set returned by the algorithm for the call at z. Then $\mathfrak{w}(I) \geqslant \left(1 - \varepsilon + \frac{\varepsilon h}{\log N}\right) \cdot \mathfrak{w}\left(I_{\mathsf{OPT}} \cap V(G')\right)$.

Proof. First, observe that if z is a leaf of $\mathcal{T}$, then the statement of the claim is satisfied. Indeed, in this case I is computed by brute force, and hence $\mathfrak{w}(I) = \mathfrak{w}(I_{\mathsf{OPT}} \cap V(G'))$.

Recall that the algorithm returns the solution of maximum weight among all choices of $J \in \mathcal{J}$, so clearly we have $\mathfrak{w}(I) \geqslant \mathfrak{w}(I_J)$, where J is good for $I_{\mathsf{OPT}} \cap V(G')$.

We proceed by induction on h. First, consider a node z at the level $h = \log N$. As the depth of $\mathcal{T}$ is at most $\log N$, we observe that z must be a leaf, so the claim follows by the observation above.

Assume that the claim holds for $h+1 \in [\log N]$ and consider a node z at level h. If z is a leaf, then again, we are done. Otherwise, let $\mathcal{A}$ be the set of nonempty particles of the extended strip decomposition of $G' - Y_J$. For every such particle A, we recursively computed an independent set $I(A)$. By the inductive assumption, we have that $\mathfrak{w}(I(A)) \geqslant \left(1 - \varepsilon + \frac{\varepsilon(h+1)}{\log N}\right) \mathfrak{w}(I_{\mathsf{OPT}} \cap V(G'[A]))$; note that these recursive calls are at level $h+1$. Clearly, the same holds for empty particles because $\emptyset$ is there an optimum solution.

Thus, by Lemma 13 applied to I_{OPT} and $\varsigma = 1 - \varepsilon + \frac{\varepsilon(h+1)}{\log N}$, we obtain an independent set I_J in $G' - Y_J$, such that

$$\begin{aligned} \mathfrak{w}(I_J) &\geqslant \left(1 - \varepsilon + \frac{\varepsilon(h+1)}{\log N}\right) \mathfrak{w}(I_{\mathsf{OPT}} \cap V(G' - Y_J)) \\ &= \left(1 - \varepsilon + \frac{\varepsilon(h+1)}{\log N}\right) \left(\mathfrak{w}(I_{\mathsf{OPT}} \cap V(G')) - \mathfrak{w}(I_{\mathsf{OPT}} \cap Y_J)\right). \end{aligned} \tag{5}$$

Combining (5) with (4) and simplifying the formula, we obtain

$$\mathfrak{w}(I_J) \geqslant \left(1 - \varepsilon + \frac{\varepsilon h}{\log N}\right) \mathfrak{w}(I_{\mathsf{OPT}} \cap V(G')),$$

which concludes the proof of the claim. ◁

Since the root of the recursion tree belongs to $\mathcal{T}$, the final result I returned for the call at the root (i.e., for $(G, \mathfrak{w}, 0)$) satisfies

$$\mathfrak{w}(I) \geqslant (1 - \varepsilon) \cdot \mathfrak{w}\left(I_{\mathsf{OPT}} \cap V(G)\right) = (1 - \varepsilon) \cdot \mathfrak{w}(I_{\mathsf{OPT}}).$$

This concludes the discussion of the approximation guarantee.

Running time. Recall that the recursion tree has depth at most $\log N$. Let us show the following claim concerning the running time.

▷ Claim 16. Let z be a node of the recursion tree, and let $(G', \mathfrak{w}, h)$ be the instance corresponding to z. Then the algorithm solves this instance in time $2^{\mathcal{O}\left(\varepsilon^{-1} \log^4 N \log\left(N/2^{h-1}\right)\right)}$.

Proof. Let $F(h)$ denote the upper bound for the running time of our algorithm, depending on the level of the call in the recursion tree. We aim to show that there is an absolute constant c, such that for N sufficiently large we have

$$F(h) \leqslant 2^{c \cdot \varepsilon^{-1} \log^4 N \log\left(N/2^{h-1}\right)}.$$

Recall that $|V(G')| \leqslant N/2^h$. If z is a leaf, then the instance is of constant size, and thus the claim holds (assuming that c is sufficiently large). In particular this happens if $h = \log N$. So let us assume that the claim holds for the calls at level $h+1$ and that $h < \log N$.

Recall that we first enumerate the family $\mathcal{J}$ of all independent sets of size at most $\lceil \beta(h,\varepsilon)^{-1} \log(N/2^h) \rceil$. Observe that

$$|\mathcal{J}| \leqslant |V(G')|^{\lceil \beta(h,\varepsilon)^{-1} \log(N/2^h) \rceil} \leqslant 2^{\log(N/2^h) \lceil \beta(h,\varepsilon)^{-1} \log(N/2^h) \rceil},$$

and the family $\mathcal{J}$ can be enumerated in time polynomial in its size.

For each $J \in \mathcal{J}$, using Corollary 12 and modifying its outcome, in polynomial time we obtain a set X_J and a rigid extended strip decomposition (H', η') of $G - Y_J$, where $Y_J = N[X_J] \cup N(J)$.

Next, we call the algorithm recursively for at most $4 \cdot |V(G')| \leqslant 4 \cdot N/2^h$ instances, each at depth $h+1$. Finally, use use Lemma 13 to obtain our solution in time polynomial in $|V(G')|$ and thus in $N/2^h$.

Thus the running time is bounded by the following expression (here c_1, c_2, c_3 are absolute constants, such that c_1 and c_2 are much smaller than c_3, and $c_3 = c/12(t+1)$):

$$\begin{aligned}
F(h) &\leqslant 2^{c_1 \cdot \beta(h,\varepsilon)^{-1} \log^2(N/2^h)} \cdot \left((N/2^h)^{c_2} + 4 \cdot (N/2^h) \cdot F(h+1) \right) \\
&\overset{c_3 \gg c_1, c_2}{\leqslant} 2^{c_3 \cdot \beta(h,\varepsilon)^{-1} \log^2(N/2^h)} \cdot 2^{c \cdot \varepsilon^{-1} \log^4 N \log(N/2^h)} \\
&= \exp\left\{ c_3 \cdot \beta(h,\varepsilon)^{-1} \log^2(N/2^h) + c \cdot \varepsilon^{-1} \log^4 N \log(N/2^h) \right\} \\
&\leqslant \exp\left\{ c_3 \cdot 12(t+1) \cdot \left(\frac{1-\varepsilon}{\varepsilon} \log N + (h+1) \right) \log^3(N/2^h) + c \cdot \varepsilon^{-1} \log^4 N \log(N/2^h) \right\} \\
&\overset{h < \log N}{\leqslant} \exp\left\{ c \cdot \varepsilon^{-1} \log^4 N + c \cdot \varepsilon^{-1} \log^4 N \log(N/2^h) \right\} \\
&= \exp\left\{ c \cdot \varepsilon^{-1} \log^4 N (\log(N/2^h) + 1) \right\} = \exp\left\{ c \cdot \varepsilon^{-1} \log^4 N \log(N/2^{h-1}) \right\}.
\end{aligned}$$

This completes the proof of the claim. ◁

Now we apply Claim 16 to the initial call $(G, \mathfrak{w}, 0)$ and obtain that the overall running time is

$$2^{\mathcal{O}(\varepsilon^{-1} \log^5 N)} = 2^{\mathcal{O}(\varepsilon^{-1} \log^5 n)},$$

as $N < 2n$. This completes the proof. ◀

5 Conclusion

In the QPTAS of Chudnovsky, Pilipczuk, Pilipczuk, and Thomassé [10, 11] it was more convenient to measure the weight of parts of the graph not by the number of vertices, but by the weight of the intersection of the sought solution with the part in question. We observe that we can adapt Theorem 2 to this setting of unknown weight function.

▶ **Theorem 17.** *Given an n-vertex graph G and an integer t, one can in time $n^{\mathcal{O}(t \log n)}$ either:*

- *output an induced copy of $S_{t,t,t}$ in G, or*
- *output a family $\mathcal{F}$ satisfying the following:*
  1. *every element of $\mathcal{F}$ is a pair of a set $\mathcal{P}$ consisting of at most $11 \log n + 6$ induced paths in G, each of length at most $t+1$, and an extended strip decomposition of $G - N[\bigcup \mathcal{P}]$;*

2. *for every weight function* $\mathfrak{w} : V(G) \to \mathbb{Z}_{\geqslant 0}$ *there exists a pair in* $\mathcal{F}$ *such that every particle in the extended strip decomposition of the pair has weight at most half of the total weight of* G*;*
3. *the size of* $\mathcal{F}$ *is bounded by* $n^{\mathcal{O}(\log n)}$*.*

Proof sketch. As observed in [10, 11], in G one can identify at most n^2 induced paths such that for every weight function $\mathfrak{w} : V(G) \to \mathbb{Z}_{\geqslant 0}$, at least one of the identified path is a Gyárfás' path for $\mathfrak{w}$, that is, a path Q such that every connected component of $G - N[Q]$ is of weight at most half of the weight of G. Thus, we can guess the path Q as in the proof in Theorem 2 out of at most n^2 candidates.

Then, in the recursive step in the proof of Theorem 2, instead of choosing the heavy particle to recurse on, we guess which particle is heavy (or that none exists). It is easy to see that any extended strip decomposition in the process will have fewer than n inclusion-wise maximal particles; thus, this gives $n^{\mathcal{O}(\log n)}$ possible outputs to enumerate. ◀

We think the $\log n$ factor in Theorem 2 is an artifact of our technique, and is not necessary. Therefore, we pose the following conjecture.

▶ **Conjecture 18.** *For every integer* $t \geqslant 1$ *there exists a constant* $\varepsilon > 0$ *and an integer* s *such that every* $S_{t,t,t}$*-free graph* G *admits a set* $P \subseteq V(G)$ *of size at most* s *such that* $G - N[P]$ *admits a rigid extended strip decomposition whose every particle has at most* $(1-\varepsilon)|V(G)|$ *vertices.*

Abrishami, Chudnovsky, Dibek, and Rzążewski [2] very recently announced a polynomial-time algorithm for MWIS in $S_{t,t,t}$-free graphs of bounded degree. Their argument is quite involved and revisits the proof of the three-in-a-tree theorem [13].

Confirming Conjecture 18 would imply the same result almost immediately, possibly with a better running time. Indeed, one needs to branch on $N[P]$ and recurse on the remainder of every particle of (H, η). The maximum degree of H is bounded by a function of the maximum degree of G (i.e., is a constant), which ensures that the sum of sizes of all particles is linear in $|V(G)|$. This in turns implies that the total complexity of the algorithm can be bounded by a polynomial function. Note that the same approach using Theorem 2 yields quasipolynomial running time bound.

We see Theorem 2 as the analog of Theorem 1 in the classes of $S_{t,t,t}$-free graphs: with its help, obtaining a QPTAS or a subexponential algorithm was relatively simple, following the ideas of [7, 10, 11]. We expect it is a first step to get a quasipolynomial-time algorithm for MWIS in $S_{t,t,t}$-free graphs, similarly as Theorem 1 is an essential ingredient of the algorithms for P_t-free graphs [15, 24]. However, there is a lot of work to be done: the way how [15, 24] measure the progress of the branching algorithm is quite intricate; furthermore, for the class of $C_{>t}$-free graphs (graphs excluding all cycles of length more than t as induced subgraphs, a proper superclass of P_t-free graphs) while an analog of Theorem 1 is known, the corresponding measure of the progress of the branching algorithm is much more involved [16].

References

1 Tara Abrishami, Maria Chudnovsky, Cemil Dibek, and Paweł Rzążewski. Polynomial-time algorithm for maximum independent set in bounded-degree graphs with no long induced claws. *CoRR*, abs/2107.05434, 2021. arXiv:2107.05434.

2 Tara Abrishami, Maria Chudnovsky, Cemil Dibek, and Paweł Rzążewski. Polynomial-time algorithm for maximum independent set in bounded-degree graphs with no long induced claws. In Niv Buchbinder Joseph (Seffi) Naor, editor, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference, January 9-12, 2022*, pages 1448–1470. SIAM, 2022. doi:10.1137/1.9781611977073.61.

3 Tara Abrishami, Maria Chudnovsky, Cemil Dibek, Stéphan Thomassé, Nicolas Trotignon, and Kristina Vušković. Graphs with polynomially many minimal separators. *J. Comb. Theory, Ser. B*, 152:248–280, 2022. doi:10.1016/j.jctb.2021.10.003.

4 Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, Paweł Rzążewski, and Paul D. Seymour. Induced subgraphs of bounded treewidth and the container method. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1948–1964. SIAM, 2021. doi:10.1137/1.9781611976465.116.

5 Vladimir E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982.

6 Vladimir E. Alekseev. On easy and hard hereditary classes of graphs with respect to the independent set problem. *Discret. Appl. Math.*, 132(1-3):17–26, 2003. doi:10.1016/S0166-218X(03)00387-1.

7 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zsolt Tuza, and Erik Jan van Leeuwen. Subexponential-time algorithms for Maximum Independent Set in P_t-free and broom-free graphs. *Algorithmica*, 81(2):421–438, 2019. doi:10.1007/s00453-018-0479-5.

8 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.

9 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.

10 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the Maximum Weight Independent Set Problem in H-free graphs. *CoRR*, abs/1907.04585, 2019. arXiv:1907.04585.

11 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the Maximum Weight Independent Set Problem in H-free graphs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2260–2278. SIAM, 2020. doi:10.1137/1.9781611975994.139.

12 Maria Chudnovsky and Paul D. Seymour. The structure of claw-free graphs. In Bridget S. Webb, editor, *Surveys in Combinatorics, 2005 [invited lectures from the Twentieth British Combinatorial Conference, Durham, UK, July 2005]*, volume 327 of *London Mathematical Society Lecture Note Series*, pages 153–171. Cambridge University Press, 2005. doi:10.1017/cbo9780511734885.008.

13 Maria Chudnovsky and Paul D. Seymour. The three-in-a-tree problem. *Comb.*, 30(4):387–417, 2010. doi:10.1007/s00493-010-2334-4.

14 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

15 Peter Gartland and Daniel Lokshtanov. Independent set on P_k-free graphs in quasi-polynomial time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

16 Peter Gartland, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Finding large induced sparse subgraphs in $C_{>t}$-free graphs in quasipolynomial time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 330–341. ACM, 2021. doi:10.1145/3406325.3451034.

17 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6-free graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1257–1271. SIAM, 2019. doi:10.1137/1.9781611975482.77.

18 András Gyárfás. On Ramsey covering-numbers. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. II*, number 10 in Colloq. Math. Soc. Janos Bolyai, pages 801–816. North-Holland, Amsterdam, 1975.

19 András Gyárfás. Problems from the world surrounding perfect graphs. In *Proceedings of the International Conference on Combinatorial Analysis and its Applications, (Pokrzywna, 1985)*, number 19 in Zastos. Mat., pages 413–441, 1987. doi:10.4064/am-19-3-4-413-441.

20 Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Math.*, 182(1):105–142, 1999. doi:10.1007/BF02392825.

21 Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.

22 Konrad Majewski, Tomáš Masařík, Jana Novotná, Karolina Okrasa, Marcin Pilipczuk, Paweł Rzążewski, and Marek Sokołowski. Max weight independent set in graphs with no long claws: An analog of the gyárfás' path argument, 2022. doi:10.48550/ARXIV.2203.04836.

23 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

24 Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for independent set in P_t-free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

25 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalite maximum dans un graphe sans etoile. *Discrete Mathematics*, 29(1):53–76, 1980. doi:10.1016/0012-365X(90)90287-R.

26 David Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.

Listing, Verifying and Counting Lowest Common Ancestors in DAGs: Algorithms and Fine-Grained Lower Bounds

Surya Mathialagan ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Virginia Vassilevska Williams ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Yinzhan Xu ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

The AP-LCA problem asks, given an n-node directed acyclic graph (DAG), to compute for every pair of vertices u and v in the DAG a lowest common ancestor (LCA) of u and v if one exists, i.e. a node that is an ancestor of both u and v but no proper descendent of it is their common ancestor. Recently [Grandoni et al. SODA'21] obtained the first sub-$n^{2.5}$ time algorithm for AP-LCA running in $O(n^{2.447})$ time. Meanwhile, the only known conditional lower bound for AP-LCA is that the problem requires $n^{\omega-o(1)}$ time where ω is the matrix multiplication exponent.

In this paper we study several interesting variants of AP-LCA, providing both algorithms and fine-grained lower bounds for them. The lower bounds we obtain are the first conditional lower bounds for LCA problems higher than $n^{\omega-o(1)}$. Some of our results include:

- In any DAG, we can detect all vertex pairs that have at most two LCAs and list all of their LCAs in $O(n^{\omega})$ time. This algorithm extends a result of [Kowaluk and Lingas ESA'07] which showed an $\tilde{O}(n^{\omega})$ time algorithm that detects all pairs with a unique LCA in a DAG and outputs their corresponding LCAs.
- Listing 7 LCAs per vertex pair in DAGs requires $n^{3-o(1)}$ time under the popular assumption that 3-uniform 5-hyperclique detection requires $n^{5-o(1)}$ time. This is surprising since essentially cubic time is sufficient to list all LCAs (if $\omega = 2$).
- Counting the number of LCAs for every vertex pair in a DAG requires $n^{3-o(1)}$ time under the Strong Exponential Time Hypothesis, and $n^{\omega(1,2,1)-o(1)}$ time under the 4-Clique hypothesis. This shows that the algorithm of [Echkardt, Mühling and Nowak ESA'07] for listing all LCAs for every pair of vertices is likely optimal.
- Given a DAG and a vertex $w_{u,v}$ for every vertex pair u, v, verifying whether all $w_{u,v}$ are valid LCAs requires $n^{2.5-o(1)}$ time assuming 3-uniform 4-hyperclique requires $n^{4-o(1)}$ time. This defies the common intuition that verification is easier than computation since returning some LCA per vertex pair can be solved in $O(n^{2.447})$ time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases All-Pairs Lowest Common Ancestor, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.94

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2204.10932

Funding *Surya Mathialagan*: Supported by the MIT Akamai Presidential Fellowship.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-2129139 and CCF-1909429, a BSF Grant BSF:2020356, a Google Research Fellowship and a Sloan Research Fellowship.
Yinzhan Xu: Partially supported by NSF Grants CCF-1528078 and CCF-2129139.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 94; pp. 94:1–94:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A lowest common ancestor (LCA) of two nodes u and v in a directed acyclic graph (DAG) is a common ancestor c of u and v such that no proper descendent of c is a common ancestor of u and v. The AP-LCA problem asks to compute for every pair of nodes in a given DAG, some LCA, provided a common ancestor exists.

Computing LCAs is an important problem with a wide range of applications. For instance, LCA computation is a key ingredient in verification of the correctness of distributed computation (e.g. [10]), object inheritance in object oriented programming languages such as C++ and Java (e.g. [2, 19, 26]), and computational biology for finding the closest ancestor of species in rooted phylogenetic networks (e.g. [22]).

Computing LCAs is very well-understood in trees [7, 8, 17, 24, 27, 37, 38, 40]. Aït-Kaci, Boyer, Lincoln and Nasr [2] were one of the first to consider LCAs in DAGs, focusing on lattices and lower semilattices with object inheritance in mind. Nykänen and Ukkonen [36] obtained efficient algorithms for directed trees and asked if there is a subcubic time algorithm for AP-LCA in DAGs.

Bender, Martin Farach-Colton, Pemmasani, Skiena and Sumazin [6] gave the first subcubic, $O(n^{(3+\omega)/2}) \leq O(n^{2.687})$, time algorithm for AP-LCA in DAGs, where $\omega < 2.37286$ is the matrix multiplication exponent [4]. They also showed that AP-LCA is equivalent to the so-called All-Pairs Shortest LCA Distance problem. Czumaj, Kowaluk and Lingas [30, 18] improved the AP-LCA running time to $O(n^{2.575})$ using a reduction to the Max-Witness Product problem. With the current best bounds for rectangular matrix multiplication [33], their algorithm runs in $O(n^{2.529})$ time.

Notice that all subcubic algorithms above would run in $\tilde{O}(n^{2.5})$ time[1] if $\omega = 2$. For more than a decade, this running time remained unchallenged. It seemed that AP-LCA might actually require $n^{2.5-o(1)}$ time, similar to several other $n^{2.5}$ time problems such as computing the Max-Witness product (see e.g. [34]).

Recently, Grandoni, Italiano, Lukasiewicz, Parotsidis and Uznanski [25] showed that this is not the case, giving an algorithm that runs in $O(n^{2.447})$ time, or in $\tilde{O}(n^{7/3})$ time if $\omega = 2$.

It is not hard to show (see [6, 18]) that any algorithm for AP-LCA can be used to solve Boolean Matrix Multiplication (BMM), and hence beating $O(n^\omega)$ time for AP-LCA would likely be difficult. No higher conditional lower bounds are known for the problem. It is still open whether $O(n^\omega)$ time can actually be achieved for AP-LCA.

Partial progresses have been made for DAGs with special structures or for variants of AP-LCA. Czumaj, Kowaluk and Lingas [18] showed that AP-LCA is in $O(n^\omega)$ time for low-depth DAGs. Kowaluk and Lingas [31] showed that in $O(n^\omega \log n)$ time one can return an LCA for every vertex pair that has a unique LCA. Eckhardt, Mühling and Nowak [20] showed that one can solve the AP-All-LCA problem, which asks to output all LCAs for every pair of vertices, in $O(n^{\omega(1,2,1)})$ time. Here $\omega(1,2,1) \leq 3.252$ is the exponent of multiplying an $n \times n^2$ by an $n^2 \times n$ matrix. AP-LCA was also studied in the weighted setting [5], the dynamic setting [20] and the space-efficient setting [32].

This paper considers the following questions:

1. Can we return all LCAs for every pair of nodes that has at most 2 LCAs, in $\tilde{O}(n^\omega)$ time, extending Kowaluk and Lingas's algorithm [31]?
2. The AP-LCA problem asks us to exhibit a single LCA for each vertex pair. What if we want to list 2, 3, ..., k LCAs? How fast can we do it?

[1] $\tilde{O}$ hides poly-logarithmic factors.

So far two variants of LCA are studied: list a single LCA per pair and list all LCAs per pair. What about listing numbers in between? This is just as natural. In phylogenetic networks for instance, there can be multiple LCAs per species pair, but typically not too many. Then listing a constant number of LCAs fast can give a better picture than listing a single representative. Other applications of AP-LCA would similarly make more sense for listing multiple LCAs.

3. How fast can we count the number of LCAs each vertex pair has?
4. Suppose that for every pair of nodes u, v in a DAG we are given a node $w_{u,v}$. Can we efficiently determine whether $w_{u,v}$ is an LCA of u and v, for each u, v? One would think that if AP-LCA can be solved faster than $O(n^{2.5})$ time, then this verification version of the problem should also be solvable faster.

We provide algorithms and fine-grained conditional lower bounds to address the above questions. Our lower bounds are the first conditional lower bounds higher than $n^{\omega-o(1)}$ for LCA problems.

1.1 Our results

Detecting and listing $O(1)$ LCAs. Our results for this part are summarized in Table 1.

Table 1 A summary of our results for detecting and listing LCAs. In the second and third columns, we give the best known runtime exponents for AP-AtLeast$k$-LCA and AP-List-k-LCA respectively. An exponent of 3 above corresponds to the trivial brute-force algorithm. In the fourth and fifth columns, we give the best conditional lower bounds for the exponent of AP-AtLeast$k$-LCA, and the corresponding hardness sources for the lower bounds. The exponents and lower bounds in the last row are for AP-All-LCA problem. All values in parentheses are the corresponding values when $\omega = 2$.

k	AP-AtLeast$k$-LCA Exponent		AP-List-k-LCA Exponent		Best Lower Bound		Source of LB
1	ω (2)	Folklore	2.447 (7/3)	[25]	ω (2)	[6]	BMM
2	ω (2)	[31], Thm 26	2.529 (2.5)	Thm 3	ω (2)	[6]	BMM
3	ω (2)	Thm 27	2.529 (2.5)	Thm 3	ω (2)	[6]	BMM
4	3		3		2.5	Thm 30	$(4,3)$-Hyperclique
5	3		3		2.666	Thm 30	$(5,3)$-Hyperclique
6	3		3		2.8	Thm 30	$(6,3)$-Hyperclique
7	3		3		3	Thm 30	$(5,3)$-Hyperclique
All	N/A		$\omega(1,2,1)$ (3)	[20]	$\omega(1,2,1)$ (3)	Thm 4, 5	SETH, 4-Clique

Let us define AP-Exact$k$-LCA, AP-AtLeast$k$-LCA and AP-AtMost$k$-LCA as the problems of deciding for every pair of vertices in a given DAG, whether they have exactly, greater than or equal to, and less than or equal to k LCAs, respectively.

We study how fast AP-Exact$k$-LCA, AP-AtLeast$k$-LCA and AP-AtMost$k$-LCA can be solved for constant k. More generally, we study the problem of returning k LCAs per vertex pair if it has at least k LCAs, or all LCAs if it has fewer. We call the latter problem AP-List-k-LCA.

For any constant k, one can return up to k LCAs for every vertex pair in a DAG in cubic time using a trivial brute-force algorithm[2]. More generally, if $\omega = 2$, the $O(n^{\omega(1,2,1)})$ time AP-All-LCA algorithm in [20] would also run in essentially cubic time.

[2] We first compute a topological ordering of the graph in $O(n^2)$ time and the transitive closure in $O(n^\omega)$ time using [23]. For each vertex pair (u, v), we scan the vertices in the reverse order of the topological ordering, and declare the current vertex w a new LCA if w can reach both u and v and w cannot reach any LCAs found so far. We stop the scan as soon as we find k LCAs or reach the end of the topological ordering. Given the transitive closure, each reachability check can be finished in $O(1)$ time, so the overall running time of the algorithm is $O(kn^3)$.

It is thus interesting to study *for what values of k,* AP-Exact$k$-LCA*,* AP-AtLeast$k$-LCA*,* AP-AtMost$k$-LCA *and* AP-List-k-LCA *are solvable in truly subcubic, $O(n^{3-\varepsilon})$ for $\varepsilon > 0$, time.*

We show that for every constant k, the listing problem AP-List-k-LCA and the decision problem AP-AtLeast$k$-LCA are subcubically equivalent. This statement appears as Theorem 28 in the main text. Thus, the values k for which one problem is in subcubic time are exactly the same for the other problem.

We also prove a convenient equivalence between AP-Exact$k$-LCA, AP-AtLeast$k$-LCA and AP-AtMost$k$-LCA:

▶ **Theorem 1.** *For any constant $k \geq 0$, the running times of* AP-Exact$k$-LCA*,* AP-AtMost$k$-LCA *and* AP-AtLeast$(k+1)$-LCA *are the same up to constant factors.*

Now we can focus on AP-Exact$k$-LCA, and due to the above equivalence, we also obtain results for the other variants.

Next, we extend the result of Kowaluk and Lingas [31] for pairs with unique LCAs to pairs with two LCAs by showing that AP-Exact$k$-LCA can be solved in $O(n^\omega)$ time for both $k = 1, 2$. Moreover, the corresponding witness LCAs can be listed in the same time.

▶ **Theorem 2.** AP-Exact1-LCA *and* AP-Exact2-LCA *can be solved in $O(n^\omega)$ time with high probability by Las Vegas algorithms. Moreover, finding the LCAs for vertex pairs (u, v) that have exactly 1 or 2 LCAs can also be solved in $O(n^\omega)$ time with high probability.*

This theorem appears as Theorems 26 and 27 in the main text. By our equivalence theorem, the same result applies to AP-AtLeast$(k+1)$-LCA and AP-AtMost$k$-LCA for $k = 1, 2$.

Our algorithm for AP-Exact1-LCA is different from that of [31]. The algorithm of [31] is deterministic while ours is randomized, so it is seemingly weaker. We nevertheless include our approach to AP-Exact1-LCA as it is simple and saves a factor of $\log n$. Additionally, our approach generalizes to AP-Exact2-LCA.

As our techniques no longer seem to work for the case of deciding if there are exactly 3 LCAs, we turn to conditional lower bounds. We prove that under popular fine-grained hypotheses, the following hold in the word-RAM model with $O(\log n)$ bit words (Theorem 30): AP-Exact$k$-LCA requires time $n^{2.5-o(1)}$ for $k = 3$, $n^{8/3-o(1)}$ for $k = 4$, $n^{2.8-o(1)}$ for $k = 5$ and $n^{3-o(1)}$ for $k = 6$.

With our earlier equivalence theorem in mind, our conditional lower bound for AP-Exact3-LCA means that detecting for each pair whether it has at least 4 LCAs, or listing 4 LCAs per vertex pair also requires $n^{2.5-o(1)}$ time. In particular, this shows that listing 4 LCAs is more difficult than listing just one LCA per vertex pair, as the latter has an $O(n^{2.447})$ time algorithm [25].

Furthermore, our conditional lower bound for AP-Exact6-LCA also implies that AP-AtLeast7-LCA requires $n^{3-o(1)}$ time, and hence the clearly even harder problem of listing 7 LCAs per vertex pair requires $n^{3-o(1)}$ time. This is intriguing since as we mentioned earlier, we can list all LCAs per pair in essentially cubic time if $\omega = 2$.

We also show the following algorithmic results for AP-List-2-LCA and AP-List-3-LCA.

▶ **Theorem 3.** *For $k = 2$ and $k = 3$, the* AP-List-k-LCA *problem can be deterministically solved in $\tilde{O}(n^{2+\lambda})$ time, where λ satisfies the equation $\omega(1, \lambda, 1) = 1 + 2\lambda$. Here, $\omega(1, \lambda, 1)$ is the exponent of multiplying an $n \times n^\lambda$ by an $n^\lambda \times n$ matrix.*

The running time for AP-List-k-LCA above matches the best known running time for Max-Witness product [18]. Using the current best bounds for rectangular matrix multiplication [33], the runtime we get for AP-List-k-LCA is $O(n^{2.529})$ for $k = 2$ and 3.

Counting LCAs. We now turn our attention to computing the number of LCAs for every pair of vertices in a DAG. We call this problem AP-#LCA. As shown in [20], we can list all LCAs for every pair of vertices in $O(n^{\omega(1,2,1)})$ time, which is essentially cubic time if $\omega = 2$. Thus in particular, we can also count all the LCAs in the same amount of time.

One might wonder, can the counts be computed faster, in truly subcubic time? We show that under the Strong Exponential Time (SETH) Hypothesis [29, 13, 14], this is impossible, even if we are only required to return the count for a vertex pair if it is smaller than some superconstant function $g(n)$. Notice that we can solve this restrained case in $O(n^3 g(n))$ time using the brute-force algorithm, so the following theorem is tight up to $n^{o(1)}$ factors when $g(n)$ is $\tilde{O}(1)$.

▶ **Theorem 4.** *Assuming SETH,* AP-#LCA *requires* $n^{3-o(1)}$ *time, even if we only need to return the minimum between the count and* $g(n)$ *for any* $g(n) = \omega(1)$.

The current best running time $O(n^{\omega(1,2,1)})$ for listing LCAs and also for AP-#LCA is actually supercubic, however. For the current best bounds on $\omega(1,2,1)$, it is $O(n^{3.252})$ [33]. In fact, there are serious limitations of the known matrix multiplication techniques [3, 15, 16] that show that current techniques cannot be used to prove that $\omega(1,2,1) < 3.05$.

In this case, the cubic lower bound for AP-#LCA under SETH would not be entirely satisfactory. We thus present a tight conditional lower bound from the 4-Clique problem.

The 4-Clique problem asks, does a given n-node graph contain a clique on 4 nodes? The fastest known algorithm for 4-Clique runs in $\tilde{O}(n^{\omega(1,2,1)})$ time [21], which has remained unchallenged for almost two decades. We show that an improvement over the $O(n^{\omega(1,2,1)})$ time for AP-#LCA would also solve 4-Clique faster.

▶ **Theorem 5.** *If the* AP-#LCA *problem can be solved in* $T(n)$ *time, then* 4-Clique *can be computed in* $O(T(n) + n^{\omega})$ *time.*

Verifying LCAs. Oftentimes in algorithms, one is also concerned with the problem of verifying an answer besides computing an answer. In many cases, verification is an easier problem than computation. For instance, even though computing the product of two $n \times n$ matrices A and B currently is only known to be possible in $O(n^{2.373})$ time, verifying whether the product of A and B is a matrix C can be done in randomized $\tilde{O}(n^2)$ time. This was the basis of the Blum-Luby-Rubinfeld linearity test [9].

We consider the following two verification variants of AP-LCA which we call Ver-LCA and AP-Ver-LCA. In both variants, we are given an n-node DAG, and for every pair of nodes u, v in the DAG, we are also given a node $w_{u,v}$. In Ver-LCA, we want to determine whether all $w_{u,v}$ are LCAs for their respective pair u, v, i.e. that the matrix w of candidate LCAs is all correct (or conversely, that there is *some* pair that has an incorrect entry). In the AP-Ver-LCA variant we want to know for every u, v whether $w_{u,v}$ is an LCA of u and v, so this variant is potentially more difficult. After we compute the transitive closure of the graph, it takes $O(n)$ time to verify whether a vertex $w_{u,v}$ is indeed an LCA of u and v. Thus, both Ver-LCA and AP-Ver-LCA can be solved in $O(n^3)$ time. No faster algorithm is known to the best of our knowledge.

Kowaluk and Lingas [31] solved a variant of AP-Ver-LCA concerning vertex pairs that have at most 2 LCAs. Specifically, given one or two nodes per pair they showed how to verify that those nodes are all the LCAs for the pair, in $O(n^{\omega})$ time. However, their algorithm is not able to compute 2 LCAs for vertex pairs that have exactly 2 LCAs in $O(n^{\omega})$ time.

Surprisingly, we provide strong evidence that Ver-LCA and AP-Ver-LCA are actually *harder* than AP-LCA, as AP-LCA can be solved in $O(n^{2.5-\varepsilon})$ time for $\varepsilon > 0$, while under popular fine-grained hypotheses, Ver-LCA and AP-Ver-LCA require $n^{2.5-o(1)}$ time.

Our first hardness result is that the running time of AP-Ver-LCA is at least as high as that of the Max-Witness problem, whose current best running time is $O(n^{2.529})$ [30, 33]. If $\omega = 2$, then Max-Witness would be solvable in $\tilde{O}(n^{2.5})$ time, and it is hypothesized [34] that no $n^{2.5-o(1)}$ time algorithms exist for it.

▶ **Theorem 6.** *If the* AP-Ver-LCA *problem can be solved in* $T(n)$ *time, then the* Max-Witness *problem can be solved in* $\tilde{O}(T(n))$ *time.*

Note that Czumaj, Kowaluk and Lingas's algorithm [18] for AP-LCA is essentially a reduction from AP-LCA to Max-Witness. Combined with their algorithm, the above theorem says that we can solve AP-Ver-LCA in $T(n)$ time, then we can solve AP-LCA in $\tilde{O}(T(n))$ time.

Our second result is the hardness of Ver-LCA based on the hardness of the $(4,3)$-Hyperclique problem: given a 3-uniform hypergraph on n nodes, return whether it contains a 4-hyperclique. This problem is hypothesized to require $n^{4-o(1)}$ time [35], and solving it in $O(n^{4-\varepsilon})$ time for $\varepsilon > 0$ would imply improved algorithms for Max-3-SAT and other problems (see [35] and the discussion therein).

▶ **Theorem 7.** *Assuming the* $(4,3)$-Hyperclique *hypothesis,* Ver-LCA *requires* $n^{2.5-o(1)}$ *time.*

Thus, verifying candidate LCAs is most likely harder than finding LCAs, defying the common intuition that verification should be easier than computation.

1.2 Paper Organization

In Section 2, we give necessary definitions. In Section 3, we list basic relationships among AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA, including Theorem 1. In Section 4, we show $O(n^{\omega})$ time algorithms for AP-Exact1-LCA and AP-Exact2-LCA, proving Theorem 2. In Section 5, we consider the AP-List-k-LCA problem. In Section 6, we prove several conditional lower bounds for AP-Exact$k$-LCA and AP-#LCA, including Theorem 4 and Theorem 5. In Section 7, we show conditional lower bounds for AP-Ver-LCA, proving Theorem 6 and Theorem 7. Finally, in Section 8, we conclude with several open problems.

2 Preliminaries

2.1 Notation

Let $G = (V, E)$ be a DAG. For every $u, v \in V$, we use $\mathsf{LCA}(u, v)$ to denote the set of vertices that are LCAs for vertex pair u and v. We use $u \rightsquigarrow v$ to denote that u can reach v via zero or more edges and use $u \not\rightsquigarrow v$ to denote that u cannot reach v. In particular, $u \rightsquigarrow u$ for every $u \in V$. We also use $\mathsf{Anc}(u)$ to denote the set of vertices that can reach u. For any $V' \subseteq V$, we use $G[V']$ to denote the subgraph in G induced by the vertex set V'.

We use $\omega < 2.37286$ to denote the matrix multiplication exponent [4]. For any constants $a, b, c \geq 0$, we use $\omega(a, b, c)$ to denote the exponent of multiplying an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix, in the arithmetic circuit model. Note that the fastest known algorithms for square [4] and rectangular [33] matrix multiplication all work in the arithmetic circuit model.

It is well-known that $\omega(a, b, c) = \omega(b, c, a)$ (see e.g. [12]).

2.2 Variants of AP-LCA

Given a DAG $G = (V, E)$, we study the following variants of AP-LCA.

▶ **Definition 8** (AP-Exact$k$-LCA)**.** *Decide if* $|\mathsf{LCA}(u, v)| = k$ *for every pair* $u, v \in V$*.*

▶ **Definition 9** (AP-AtMost$k$-LCA). *Decide if* $|\mathsf{LCA}(u,v)| \leq k$ *for every pair* $u, v \in V$.

▶ **Definition 10** (AP-AtLeast$k$-LCA). *Decide if* $|\mathsf{LCA}(u,v)| \geq k$ *for every pair* $u, v \in V$.

▶ **Definition 11** (AP-#LCA). *Compute* $|\mathsf{LCA}(u,v)|$ *for every pair* $u, v \in V$.

▶ **Definition 12** (AP-List-k-LCA). *Compute for every pair* $u, v \in V$ *a list of* k *distinct LCAs. If any pair* $u, v \in V$ *has fewer than* k *LCAs, output all of their LCAs.*

▶ **Definition 13** (AP-All-LCA). *For every pair* $u, v \in V$, *output* $\mathsf{LCA}(u,v)$.

▶ **Definition 14** (AP-Ver-LCA). *Given a candidate vertex* $w_{u,v}$ *for each pair* $u, v \in V$, *decide if* $w_{u,v} \in \mathsf{LCA}(u,v)$ *for every pair* $u, v \in V$.

▶ **Definition 15** (Ver-LCA). *Given a candidate vertex* $w_{u,v}$ *for each pair* $u, v \in V$, *decide if there exists* $u, v \in V$ *such that* $w_{u,v}$ *is not an LCA for* u *and* v.

2.3 Fine-Grained Hypotheses

In this section, we list the hypotheses we use in this paper.

Eisenbrand et al. [21] gave the current best algorithm for 4-Clique that runs in $O(n^{\omega(1,2,1)})$ time. The 4-Clique hypothesis states that we cannot improve this algorithm much.

▶ **Hypothesis 16** (4-Clique Hypothesis [11, 1]). *On a Word-RAM with* $O(\log n)$ *bit words, detecting a 4-clique in an* n*-node graph requires* $n^{\omega(1,2,1)-o(1)}$ *time.*

▶ **Hypothesis 17** ((ℓ,k)-Hyperclique Hypothesis, [35]). *Let* $\ell > k > 2$ *be constant integers. On a Word-RAM with* $O(\log n)$ *bit words, detecting whether an* n*-node* k*-uniform hypergraph contains an* ℓ*-hyperclique requires* $n^{\ell-o(1)}$ *time.*

Using common techniques (see e.g. [39]), the (ℓ,k)-Hyperclique hypothesis actually implies the hardness of the following unbalanced version of (ℓ,k)-Hyperclique.

▶ **Fact 18.** *Assuming the* (ℓ,k)-Hyperclique *hypothesis, on a Word-RAM with* $O(\log n)$ *bit words, detecting whether a* k*-uniform* ℓ*-partite hypergraph with* $n^{a_1}, \ldots, n^{a_\ell}$ *vertices on each part for* $a_1, \ldots, a_\ell > 0$ *requires* $n^{a_1+\cdots+a_\ell-o(1)}$ *time.*

▶ **Hypothesis 19** (Max-k-SAT Hypothesis, [35]). *On a Word-RAM with* $O(\log n)$ *bit words, for any* $k \geq 3$, *given a* k*-CNF formula on* n *variables and* $\mathrm{poly}(n)$ *clauses, determining the maximum number of clauses that can be satisfied by a Boolean assignment of the variables requires* $2^{n-o(n)}$ *time.*

▶ **Hypothesis 20** (Strong Exponential Time Hypothesis (SETH), [28, 13, 14]). *On a Word-RAM with* $O(\log n)$ *bit words, for every* $\epsilon > 0$, *there exists* k *such that* k-SAT *on* n *variables cannot be solved in* $O(2^{(1-\epsilon)n})$ *time.*

▶ **Definition 21.** *The* Max-Witness *product* C *of two* $n \times n$ *Boolean matrices* A *and* B *is defined as*

$$C[i,j] = \max\{k \mid A[i,k] = B[k,j] = 1\}$$

where the maximum is defined to be $-\infty$ *if no such witness exists.*

The best running time to compute the Max-Witness product is $O(n^{2+\lambda})$ where λ satisfies the equation $\omega(1,\lambda,1) = 1 + 2\lambda$ [18]. This running time is $\tilde{O}(n^{2.5})$ if $\omega = 2$. It is used as a hypothesis that this running time cannot be improved much.

▶ **Hypothesis 22** (Max-Witness Hypothesis, [34]). *On a Word-RAM with* $O(\log n)$ *bit words, computing the* Max-Witness *product of two* $n \times n$ *matrices requires* $n^{2.5-o(1)}$ *time.*

3 Relationships among AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA

In this section, we consider the relationships between AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA. Our results are depicted in Figure 1 and are proven in the full version of the paper.

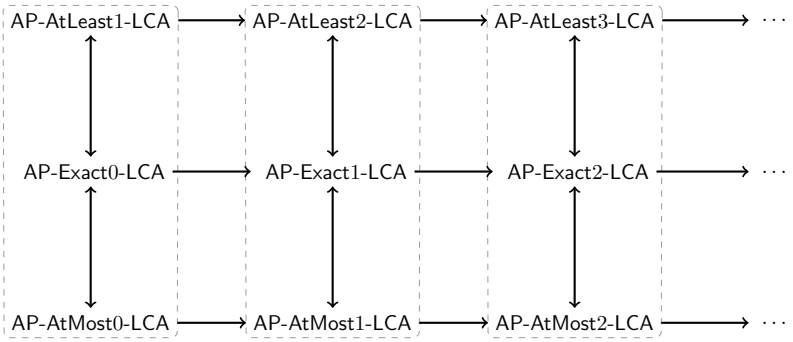


Figure 1 Reductions between AP-AtMost$k$-LCA, AP-Exact$k$-LCA and AP-AtLeast$k$-LCA. All arrows in this figure represent $O(n^2)$ time reductions from an instance to another instance with the same input sizes up to constant factors.

We first show the following lemma (whose proof is deferred to the full version) which then allows us to show that AP-Exact$(k+1)$-LCA (resp. AP-AtMost$(k+1)$-LCA, AP-AtLeast$(k+1)$-LCA) is harder than AP-Exact$k$-LCA (resp. AP-AtMost$k$-LCA, AP-AtLeast$k$-LCA) for $k \geq 0$.

▶ **Lemma 23.** *Given a DAG G with n vertices, we can create another DAG G' with $2n+1$ vertices and a map $\rho : V(G) \to V(G')$ in $O(n^2)$ time such that for every $u, v \in V(G)$, the number of LCAs of u and v in G is exactly one fewer than the number of LCAs of $\rho(u)$ and $\rho(v)$ in G'.*

▶ **Corollary 24.** *For any $k \geq 0$, an instance of* AP-Exact$k$-LCA *(resp.* AP-AtMost$k$-LCA, AP-AtLeast$k$-LCA*) with n vertices reduces to an instance of* AP-Exact$(k+1)$-LCA *(resp.* AP-AtMost$(k+1)$-LCA, AP-AtLeast$(k+1)$-LCA*) with $O(n)$ vertices in $O(n^2)$ time.*

Finally, we recall the relationship among AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA.

▶ **Theorem 1.** *For any constant $k \geq 0$, the running times of* AP-Exact$k$-LCA, AP-AtMost$k$-LCA *and* AP-AtLeast$(k+1)$-LCA *are the same up to constant factors.*

4 Algorithms for AP-Exact$k$-LCA

As noted in the introduction, AP-Exact$k$-LCA can be solved in $O(n^3)$ time for any constant k. Interestingly, an algorithm by Kowaluk and Lingas [31] that finds and verifies the LCAs for vertex pairs with a unique LCA implies that AP-Exact1-LCA can be solved in $\tilde{O}(n^\omega)$ time deterministically. In this section, we present an alternative randomized algorithm for AP-Exact1-LCA, and also extend the algorithm for AP-Exact2-LCA.

The following claim is essential to our AP-Exact1-LCA algorithm. We defer its proof to the full version fo the paper.

▷ Claim 25. Given a DAG $G = (V, E)$, for every pair of vertices $u, v \in V$, we have that

$$\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \bigcup_{w \in \mathsf{LCA}(u,v)} \mathsf{Anc}(w). \tag{1}$$

Moreover, if $\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \bigcup_{w \in S} \mathsf{Anc}(w)$ for some $S \subseteq V$, it must be the case that $\mathsf{LCA}(u, v) \subseteq S$.

▶ **Theorem 26.** *There exists an $O(n^\omega)$ time Las Vegas algorithm for* AP-Exact1-LCA *that succeeds with high probability. Additionally, this algorithm can find the unique LCA for all pairs of vertices that have exactly* 1 *LCA.*

Proof. For every pair of vertices u and v with a unique LCA w, we rewrite Equation (1) as $\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(w)$. In fact, Claim 25 gives us that this holds if and only if w is a unique LCA of the pair u and v.

Let $f : V \to \mathbb{Z}_p$ be a random function for some $p = \Theta(n^{10})$. For every $S \subseteq V$, we will use $f(S)$ to denote $\sum_{x \in S} f(x)$. Then with high probability, for any $u, v, x \in V$,

$$\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(x) \quad \text{if and only if} \quad f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v)) = f(\mathsf{Anc}(x)).$$

To see this, note that for $S, S' \subseteq V$, if $S \neq S'$, then $f(S) - f(S')$ is a sum of a nonzero number of independent uniform random variables from $\mathbb{Z}_p$. Thus if $S \neq S'$, then $\Pr[f(S) = f(S')] = \frac{1}{p}$. Since we are comparing $O(n^2)$ such sets of the form $f(\mathsf{Anc}(x))$ and $f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$, by a union bound, the probability that two distinct sets collide is $O(n^4/p)$.

Therefore, it suffices to compute $f(\mathsf{Anc}(x))$ and $f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$ for all $u, v, x \in V$. For each $x \in V(G)$, it is easy to compute $f(\mathsf{Anc}(x)) = \sum_{v \in \mathsf{Anc}(x)} f(v)$ in $O(n)$ time. To compute $F(u, v) = f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$ for all $u, v \in V$, we construct the following matrices. Let A be the transitive closure of G and let $B[x, v] = f(x) \cdot A[x, v]$. Now, note that the (u, v)-th entry of $C = A^T B$ gives us $C[u, v] = \sum_{x \in \mathsf{Anc}(u) \cap \mathsf{Anc}(v)} f(x) = F(u, v)$, as desired. Therefore, we can compute all $F(u, v)$ in $O(n^\omega)$ time.

Now, we sort the list $L = \{f(v) \mid v \in V(G)\}$ in $\tilde{O}(n)$ time. For each $u, v \in V$, we can find an arbitrary $w_{u,v}$ such that $F(u, v) = f(w_{u,v})$ in $\tilde{O}(1)$ time. Assuming none of the $O(n^2)$ sets we are interested in collide, which happens with probability at least $1 - O(n^4/p) = 1 - O(1/n^6)$, we find such a $w_{u,v}$ if and only if it is the unique LCA of $u, v \in V$.

To make this algorithm Las Vegas, we first notice that if our algorithm does not report a $w_{u,v}$, then u and v does not have a unique LCA. For the vertex pairs that our algorithm does find a $w_{u,v}$, we run [31]'s verification algorithm (Theorem 2 in [31]) to verify if each $w_{u,v}$ is in fact the unique LCA of u, v in $O(n^\omega)$ time. If we find any errors, we can simply repeat the algorithm. ◀

Now we show how to extend our AP-Exact1-LCA algorithm to AP-Exact2-LCA.

▶ **Theorem 27.** *There exists an $O(n^\omega)$ time Las Vegas algorithm for* AP-Exact2-LCA *that succeeds with high probability. Additionally, this algorithm can find the two LCAs for all pairs of vertices with exactly 2 LCAs.*

Proof. For all pair of vertices u and v with exactly two LCAs, say a and b, we rewrite (1) as $\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(a) \cup \mathsf{Anc}(b)$. Moreover, for any u, v, a, b such that the above equation holds, it must be the case that either both a and b are the only LCAs of u and v, or exactly one of them is the unique LCA (and the other is a common ancestor). We can detect the latter case with high probability by performing the algorithm as described in Theorem 26.

Let $f : V(G) \to \mathbb{Z}_p$ be a random function for some $p = \Theta(n^{10})$. By the same argument as Theorem 26, with high probability, for any $u, v, a, b \in V$,

$$\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(a) \cup \mathsf{Anc}(b) \quad \text{if and only if} \quad f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v)) = f(\mathsf{Anc}(a) \cup \mathsf{Anc}(b)).$$

Let $F(u, v) = f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$ and $H(a, b) = f(\mathsf{Anc}(a) \cup \mathsf{Anc}(b))$. As we saw in Theorem 26, we can compute $F(u, v)$ in $O(n^\omega)$ time.

To compute $H(a, b)$, note that $\mathsf{Anc}(a) \cup \mathsf{Anc}(b) = \overline{\overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}}$. First, we compute the transitive closure A of G in $O(n^\omega)$ time. Then, we construct an $n \times n$ matrix M by setting $M[x, a] = 1 - A[x, a]$. Now, construct another matrix N by setting $N[x, b] = f(x) \cdot M[x, b]$. Then, it is easy to see that

$$(M^T N)[a, b] = \sum_{x \in \overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}} f(x) = f(\overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}).$$

Therefore, one can compute

$$H(a, b) = f(\mathsf{Anc}(a) \cup \mathsf{Anc}(b)) = f(V) - f(\overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}) = f(V) - (M^T N)[a, b]$$

for all $a, b \in V$ in $O(n^\omega)$ time.

Now, sort $L = \{H(a, b) \mid a, b \in V\}$. For each u, v which does not have a unique LCA, search for an arbitrary pair $a_{u,v}, b_{u,v}$ (if one exists) such that $F(u, v) = H(a_{u,v}, b_{u,v})$ in $\tilde{O}(1)$ time. With probability $1 - O(1/n^6)$, we find such a pair for each u, v if and only if $a_{u,v}$ and $b_{u,v}$ are the only two LCAs of u and v.

To make this algorithm Las Vegas, we first notice that if our algorithm does not report a pair $a_{u,v}, b_{u,v}$, then u and v does not have exactly two LCAs. For vertex pairs for which our algorithm does find two LCA candidates, we run [31]'s verification algorithm (it is described in a remark in [31]) to verify that $a_{u,v}$ and $b_{u,v}$ are the only two LCAs of u and v in $O(n^\omega)$ time. If we find any errors, we can simply repeat the algorithm from the beginning. ◀

Note that our technique for AP-Exact1-LCA and AP-Exact2-LCA does not extend to AP-Exact3-LCA because it would require us to list $f(\mathsf{Anc}(x) \cup \mathsf{Anc}(y) \cup \mathsf{Anc}(z))$ for all $x, y, z \in V$, which easily exceeds n^ω time. In fact, in Section 6, we show it is unlikely to obtain an $\tilde{O}(n^\omega)$ time algorithm for AP-Exact3-LCA by proving that any $O(n^{2.5-\epsilon})$ time algorithm for $\epsilon > 0$ for AP-Exact3-LCA would refute the $(4, 3)$-Hyperclique hypothesis. Thus, AP-Exact3-LCA is indeed (conditionally) harder than AP-Exact1-LCA and AP-Exact2-LCA.

5 AP-LCA Listing Algorithms

In this section, we consider the AP-List-k-LCA problem. First, we show that AP-AtLeast$k$-LCA and AP-List-k-LCA are *subcubically equivalent*, i.e. either both or neither have a truly subcubic time algorithm.

▶ **Theorem 28.** *Suppose* AP-AtLeast$k$-LCA *can be computed in $T(n)$ time for a constant k. Then,* AP-List-k-LCA *can be computed in $O(\sqrt{n^3 \cdot T(n)})$ time. In particular,* AP-AtLeast$k$-LCA *and* AP-List-k-LCA *are subcubically equivalent.*

Proof. Suppose we are given a DAG $G = (V, E)$. First compute a topological ordering π of the vertices in $O(n^2)$ time, and the transitive closure D in $O(n^\omega)$ time. Now, for every pair of vertices u and v, we inductively find their k topologically latest (with respect to π) LCAs.

Suppose we have found the set $S(u, v)$ of the topologically latest $\ell - 1$ LCAs for every pair of vertices u, v, for some $1 \leq \ell \leq k$ with respect to π. Now, partition the vertices into sets $V = V_1 \sqcup V_2 \sqcup \cdots \sqcup V_{n/L}$, where V_1 contains the first L vertices in the topological ordering, V_2 contains the next L and so on for a parameter L that we will set later.

Let $\mathsf{LCA}_{G[W]}(u,v)$ denote the set of LCAs of u and v in the subgraph induced by W (note the distinction between $\mathsf{LCA}_{G[W]}(u,v)$ and $\mathsf{LCA}(u,v) \cap W$). Consider the vertex set $U_i = V_i \sqcup V_{i+1} \sqcup \cdots \sqcup V_{n/L}$ and the induced subgraph $G_i = G[U_i]$. We will prove the following claim in the full version of the paper.

▷ Claim 29. For $u, v \in U_i$, it must be the case that $\mathsf{LCA}_{G_i}(u,v) = \mathsf{LCA}(u,v) \cap U_i$.

Now we describe our algorithm. For $i = n/L, n/L-1, \ldots, 1$, run AP-AtLeast$\ell$-LCA on each G_i. For each $(u,v) \in V \times V$, keep track of the largest index $i_{u,v}$ where AP-AtLeastℓ-LCA outputs 1, i.e. largest index such that $|\mathsf{LCA}_{G_i}(u,v)| \geq \ell$. By Claim 29, this must mean that $|\mathsf{LCA}(u,v) \cap U_{i_{u,v}}| \geq \ell$ whereas $|\mathsf{LCA}(u,v) \cap U_{i_{u,v}-1}| < \ell$. In other words, the ℓth LCA lies in $V_{i_{u,v}}$. By Corollary 24, we can compute AP-AtLeastℓ-LCA in time $O(T(n))$. Therefore, this step takes $O\left(\frac{n}{L} \cdot T(n)\right)$ time in total.

Next, for each $u, v \in V$, note that the topologically ℓth LCA must lie in the vertex partition $V_{i_{u,v}}$, if $i_{u,v}$ exists. Therefore, it suffices to find the latest vertex $x \in V_{i_{u,v}}$ such that $x \in \mathsf{Anc}(u) \cap \mathsf{Anc}(v)$ and no $y \in S(u,v)$ is a descendent of x. Such an x must in fact be the ℓth LCA. Note that these checks can be done in $O(1)$ time for each $x \in V_{i_{u,v}}$ using the transitive closure D. If there is no $i_{u,v}$ such that AP-AtLeastℓ-LCA outputs 1, then u and v have fewer than ℓ LCAs. This step takes $O(\ell \cdot L) = O(L)$ time for each pair $u, v \in V$.

Since we have to iteratively find up to k LCAs per vertex pair, the overall runtime of the algorithm is $O(n^\omega + k(\frac{n}{L} \cdot T(n) + n^2 \cdot L))$. Choosing $L = \sqrt{T(n)/n}$, we have a runtime of $O(\sqrt{n^3 \cdot T(n)})$.

Moreover, it is clear that if there is a subcubic algorithm for AP-List-k-LCA, we can use the same algorithm to solve AP-AtLeast$k$-LCA with an $\tilde{O}(n^2)$ additional cost. Therefore the two problems are in fact subcubically equivalent. ◀

In Theorem 26 and Theorem 27, we showed $O(n^\omega)$ time algorithms for AP-Exact1-LCA and AP-Exact2-LCA. By their equivalences with AP-AtLeast2-LCA and AP-AtLeast3-LCA respectively, we can also solve AP-AtLeast2-LCA and AP-AtLeast3-LCA in $O(n^\omega)$ time. By Theorem 28, these imply $O(n^{(\omega+3)/2})$ time algorithms for AP-List-2-LCA and AP-List-3-LCA.

In the following theorem, we show that we can further improve the $O(n^{(3+\omega)/2})$ running time for AP-List-2-LCA and AP-List-3-LCA to $\tilde{O}(n^{2+\lambda})$ time where $\omega(1,\lambda,1) = 1 + 2\lambda$. Interestingly this running time matches the current best running time of the Max-Witness problem [18]. For these algorithms, we use an idea from [31] about comparing the sizes of two sets for verifying whether a set of one or two vertices are all the LCAs.

▶ **Theorem 3.** *For $k = 2$ and $k = 3$, the* AP-List-k-LCA *problem can be deterministically solved in $\tilde{O}(n^{2+\lambda})$ time, where λ satisfies the equation $\omega(1,\lambda,1) = 1 + 2\lambda$. Here, $\omega(1,\lambda,1)$ is the exponent of multiplying an $n \times n^\lambda$ by an $n^\lambda \times n$ matrix.*

We defer the proof of Theorem 3 to the full version of the paper.

6 Lower Bounds

In this section, we show our conditional lower bounds for AP-Exact$k$-LCA and AP-#LCA. These lower bounds are the first conditional lower bounds for LCA problems that are higher than $n^{\omega - o(1)}$.

6.1 Lower Bounds for AP-Exact$k$-LCA

First, we show lower bounds for the AP-Exact$k$-LCA problem by reducing from 3-uniform hypercliques. Combined with Corollary 24, the following theorem also shows that, for all constant $k \geq 6$, AP-Exact$k$-LCA requires $n^{3-o(1)}$ time.

▶ **Theorem 30.** *Assuming the* $(4,3)$-Hyperclique *hypothesis,* AP-Exact3-LCA *requires* $n^{2.5-o(1)}$ *time. Assuming the* $(5,3)$-Hyperclique *hypothesis,* AP-Exact4-LCA *and* AP-Exact6-LCA *require* $n^{8/3-o(1)}$ *and* $n^{3-o(1)}$ *time respectively. Also, assuming the* $(6,3)$-Hyperclique *hypothesis,* AP-Exact5-LCA *requires* $n^{14/5-o(1)}$ *time.*

Proof. All four reductions share the same underlying ideas. Thus, we only give full details for the first reduction. The remaining reductions are deferred to the full version of the paper.

$(4,3)$-Hyperclique $\rightarrow$ AP-Exact**3**-LCA. Suppose we are given a 3-uniform 4-partite hypergraph G on vertex sets A, B, C, U, where $|A| = |B| = |C| = \sqrt{n}$, and $|U| = n$. By Fact 18, the $(4,3)$-Hyperclique hypothesis implies that it requires $(|A||B||C||U|)^{1-o(1)} = n^{2.5-o(1)}$ time to determine whether G contains a 4-hyperclique.

We construct the following instance of AP-Exact3-LCA as depicted in Figure 2. The graph G' contains 3 layers of vertices V_1, V_2, V_3. Vertex set V_1 is a copy of U, vertex set V_2 equals $(A \times B) \sqcup (B \times C) \sqcup (C \times A)$ and vertex set V_3 equals $(A \times B) \sqcup C$. To distinguish vertices from V_2 and V_3, we use subscript 2 and 3, e.g. $(a,b)_2$ and $(a,b)_3$, to denote vertices from V_2 and V_3 respectively.

We also add the following edges to the graph G':

- Add a directed edge from every vertex in V_1 to every vertex in V_3.
- Add a directed edge from any vertex in V_2 to any vertex in V_3 as long as they do not have *inconsistent* labels. For instance, for every $a \in A, b \in B, c \in C$, we add an edge from $(a,b)_2$ to $(a,b)_3$ and to c_3, but we do not add an edge from $(a,b)_2$ to $(a,b')_3$ if $b \neq b'$.
- For every $u \in V_1$ and every $(x,y)_2 \in V_2$, add a directed edge from u to (x,y) if and only if there is *not* a 3-hyperedge among u, x and y.

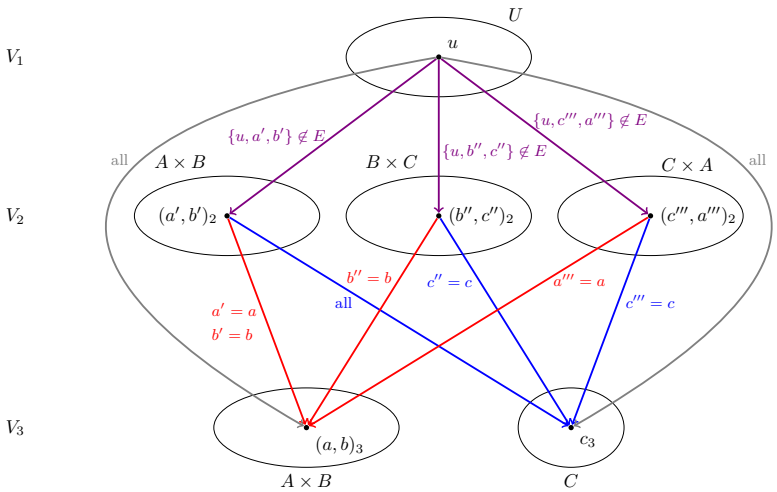


Figure 2 Construction of G' in Theorem 30 from the 3-uniform 4-hyperclique instance. Between the parts where we mark "all", we add all possible edges. Between the parts where we mark a condition, we only add an edge when the corresponding condition holds.

We consider the set of LCAs for every pair of $(a,b)_3, c_3 \in V_3$.

First, since G' is a three layered graph, all common ancestors of $(a,b)_3$ and c_3 in V_2 are their LCAs. Since we only add edges from V_2 to V_3 when the labels are consistent, it is easy to verify that the set of LCAs in V_2 is $\{(a,b)_2, (b,c)_2, (c,a)_2\}$.

Now we claim that a,b,c are in a 4-hyperclique in G if and only if $(a,b)_3$ and c_3 have an LCA in V_1 and there is a 3-hyperedge among a,b,c in G.

Suppose a,b,c are in a 4-hyperclique with a vertex $u \in V(G)$. Since G is 4-partite, we must have $u \in U$. The copy of u in G' is clearly a common ancestor of $(a,b)_3$ and c_3, since we add all possible edges from V_1 to V_3. Because a,b,c,u is in a 4-hyperclique, $\{a,b,u\}, \{b,c,u\}, \{c,a,u\} \in E(G)$. Therefore, in G' we do not add edges from u to any of $(a,b)_2, (b,c)_2$ and $(c,a)_2$. Since these are the only common ancestors of $(a,b)_3$ and c_3 in V_2, u in fact cannot reach any other vertex that can reach both $(a,b)_3$ and c_3, which makes u an LCA. Clearly, there is a 3-hyperedge among a,b,c in G.

To prove the converse, suppose $u \in V_1$ is an LCA of $(a,b)_3$ and c_3 and there is a 3-hyperedge among a,b,c in G. In that case, u cannot reach any vertex that can reach both $(a,b)_3$ and c_3. In particular, u cannot reach any of $(a,b)_2, (b,c)_2, (c,a)_2$. When we add edges from V_1 to V_2, we have that $\{a,b,u\}, \{b,c,u\}, \{c,a,u\}$ are all 3-hyperedges in G. Also, since $\{a,b,c\}$ is a 3-hyperedge, there is indeed a 4-hyperclique with vertices a,b,c,u.

Thus, a,b,c are in a 4-hyperclique in G if and only if the number of LCAs of $(a,b)_3$ and c_3 is not 3 and there is a 3-hyperedge among a,b,c in G. Thus, given the result of an AP-Exact3-LCA computation of G', we can easily determine if G has a 4-hyperclique. Therefore, assuming the $(4,3)$-Hyperclique hypothesis, AP-Exact3-LCA requires $n^{2.5-o(1)}$ time. ◀

▶ Remark 31. Note that in all our reductions to AP-Exact$k$-LCA for $3 \leq k \leq 5$, we only need to output the results for $o(n^2)$ pairs of u and v. For instance, in the reduction from $(4,3)$-Hyperclique to AP-Exact3-LCA, we only need to output whether (u,v) has exactly 3 LCAs for $u \in A \times B$ and $v \in C$. The total number of such pairs is only $O(n^{1.5})$. This is the main reason why we do not get $n^{3-o(1)}$ conditional lower bounds for AP-Exact$k$-LCA for $3 \leq k \leq 5$. On the other hand, in the reduction to AP-Exact6-LCA, we do have $\Theta(n^2)$ queries.

Williams [41] showed that Max-3-SAT reduces to 3-uniform hypercliques. Lincoln, Vassilevska Williams and Williams [35] further generalized this reduction to a reduction from Constraint Satisfaction Problem (CSP) on degree-3 formulas to 3-uniform hypercliques. Therefore, Theorem 30 also works assuming the Max-3-SAT hypothesis or the hardness of maximizing the number of satisfying clauses in degree-3 CSP formulas.

▶ **Corollary 32.** *Assuming* Max-3-SAT *(or even max degree* 3 *CSP formulas) on* N *variables and* $\text{poly}(n)$ *clauses requires* $2^{N-o(N)}$ *time,* AP-Exact3-LCA*,* AP-Exact4-LCA*,* AP-Exact5-LCA *and* AP-Exact6-LCA *requires* $n^{2.5-o(1)}$*,* $n^{8/3-o(1)}$*,* $n^{14/5-o(1)}$ *and* $n^{3-o(1)}$ *time respectively.*

6.2 Lower Bounds for Counting LCAs

In this section, we show two conditional lower bounds for AP-#LCA, one based on SETH and one based on the 4-Clique hypothesis.

The next lemma is a crucial tool for the SETH lower bound. It is a generalization of our previous reduction from $(5,3)$-Hyperclique to AP-Exact6-LCA.

▶ **Lemma 33.** *If there exists a* $T(N)$ *time algorithm for* AP-Exact$\binom{2(k-1)}{k-1}$-LCA *for graphs with* N *vertices, then there exists an* $O(f(k)\,\text{poly}(n)^{f(k)}T(2^{n/3}))$ *time algorithm for* Max-k-SAT *with* n *variables and* $\text{poly}(n)$ *clauses for some function* f*.*

To prove the lemma, we first reduce Max-k-SAT to k-uniform $(2k-1)$-hyperclique, which is a straightforward generalization of Williams' Max-2-SAT algorithm [41]. Then we reduce k-uniform $(2k-1)$-hyperclique to AP-Exact$\binom{2(k-1)}{k-1}$-LCA, building on ideas similar to the proof of Theorem 30. The full proof can be found in the full version of the paper.

▶ Remark 34. Lemma 33 implies that if we assume the Max-k-SAT hypothesis, then AP-Exact$\binom{2(k-1)}{k-1}$-LCA requires $n^{3-o(1)}$ time. Since our reduction uses $(2k-1,k)$-Hyperclique as an intermediate problem, the same lower bound also holds assuming the $(2k-1,k)$-Hyperclique hypothesis.

Now we show our SETH lower bound using Lemma 33.

▶ **Theorem 4.** *Assuming SETH,* AP-#LCA *requires $n^{3-o(1)}$ time, even if we only need to return the minimum between the count and $g(n)$ for any $g(n)=\omega(1)$.*

Proof. For the sake of contradiction, assume AP-#LCA has an $O(n^{3-\epsilon})$ time algorithm for $\epsilon>0$ when the algorithm only needs to return the minimum between the count and $g(n)$. For any fixed k, when n is large enough, we have $\binom{2(k-1)}{k-1}<g(n)$, so we can solve AP-Exact$\binom{2(k-1)}{k-1}$-LCA in $O(n^{3-\epsilon})$ time. Thus, by Lemma 33, we can solve Max-k-SAT (and thus k-SAT) with n variables and $\mathrm{poly}(n)$ clauses in time

$$O(f(k)\,\mathrm{poly}(n)^{f(k)}(2^{n/3})^{3-\epsilon})=O(f(k)\,\mathrm{poly}(n)^{f(k)}2^{(1-\epsilon/3)n})=O(\mathrm{poly}(n)\cdot 2^{(1-\epsilon/3)n}),$$

which would refute SETH. ◀

Finally, we present our reduction from 4-Clique to AP-#LCA, showing an $n^{\omega(1,2,1)-o(1)}$ lower bound for AP-#LCA assuming the current algorithm for 4-Clique is optimal.

▶ **Theorem 5.** *If the* AP-#LCA *problem can be solved in $T(n)$ time, then* 4-Clique *can be computed in $O(T(n)+n^{\omega})$ time.*

Proof. Suppose we are given a 4-Clique instance $G=(V,E)$. Without loss of generality, we assume G is a 4-partite graph with four vertex parts $V=A\sqcup B\sqcup C\sqcup D$ of size n each.

First, make a copy $G'=(V',E')$ of G, and modify the edge set of G' as follows:

- Remove all edges between A and B.
- Direct all edges from D to A and B.
- Direct all edges from C to A,B and D.

Then we add two additional vertex sets A' and B' to G', where A' is a copy of A and B' is a copy of B. We use a' to denote the copy of $a\in A$ in A' and use b' to denote the copy of $b\in B$ in B'. We also add the following edges:

- For every $a\in A$, add an edge (a',a).
- For every $b\in B$, add an edge (b',b).
- For every $a\in A,b\in B$, add two edges (a',b) and (b',a).
- For every $a\in A,c\in C$, add an edge (c,a') if $\{c,a\}\notin E$.
- For every $b\in B,c\in C$, add an edge (c,b') if $\{c,b\}\notin E$.

This construction of the graph is also depicted in Figure 3. From there, it is clear that G' is a 3-layered graph.

▷ Claim 35. For every $a\in A,b\in B,c\in C$, c is an LCA of a and b in G' if and only if $\{c,a\},\{c,b\}\in E$ and there doesn't exist any $d\in D$ such that $\{c,d\},\{d,a\},\{d,b\}\in E$.

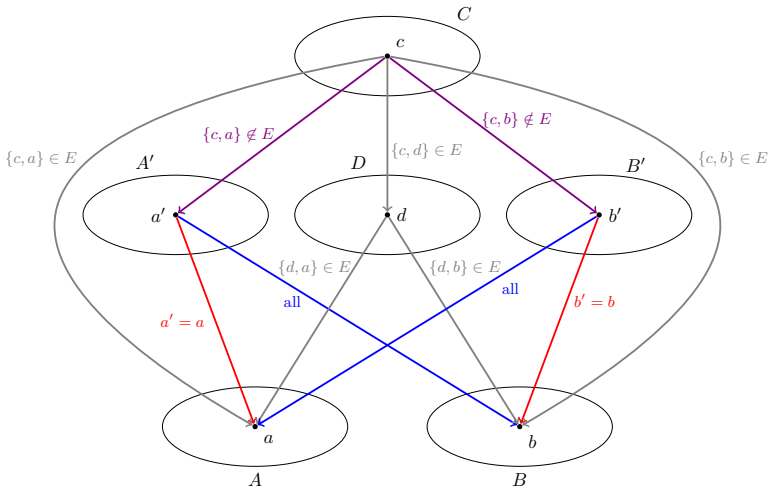


Figure 3 Construction of G' in Theorem 5 given a 4-partite 4-Clique instance. Between parts where we mark "all", we add all possible edges. Between parts where we mark a condition, we only add an edge when the corresponding condition holds.

Proof. First, suppose c is an LCA of a and b. For the sake of contradiction, suppose $\{c, a\} \notin E$. Then by the construction of G', $(c, a') \in E'$. Also, $(a', a), (a', b) \in E'$, so c cannot be an LCA. This leads to a contradiction, so we must have $\{c, a\} \in E$. Similarly, we must have $\{c, b\} \in E$. Finally, suppose for the sake of contradiction that there exists a $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$, then by construction, $(c, d), (d, a), (d, b) \in E'$, so c cannot be an LCA. Thus, there doesn't exist any $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$.

Now we prove the converse direction. Suppose $\{c, a\}, \{c, b\} \in E$ and there doesn't exist any $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$. By our construction, $(c, a), (c, b) \in E'$, so c is at least a common ancestor of a and b. Since G' is a 3-layered graph, it suffices to show that there isn't any vertex u in the middle layer such that $(c, u), (u, a), (u, b) \in E'$. First, for any $u \in A'$, if $u \neq a'$, then $(u, a) \notin E'$; if $u = a'$, then $(c, u) \notin E'$ because $\{c, a\} \in E$. Therefore, there isn't any $u \in A'$ such that $(c, u), (u, a), (u, b) \in E'$. Similarly, there isn't any $u \in B'$ such that $(c, u), (u, a), (u, b) \in E'$. For any $d \in D$, we already have the condition that at least one of $\{c, d\}, \{d, a\}, \{d, b\}$ is not in E, so at least one of $(c, d), (d, a), (d, b)$ is not in E'. Therefore, c is an LCA. ◁

Using this claim, we describe our algorithm below.

First, run AP-#LCA to compute $|\mathsf{LCA}(a, b)|$ for all $(a, b) \in A \times B$. Since G' is a three-layered graph, the set of LCAs of a and b in the middle layer is exactly the set of their common neighbors in the middle layer. Therefore, we can easily compute $|\mathsf{LCA}(a, b) \cap (A' \cup B' \cup D)|$ in $O(n^\omega)$ time by using matrix multiplication to count the number of their common neighbors in the middle layer. Also, clearly, there isn't any LCA of a and b in A or B. Thus, we can compute the number of $c \in C$ that is an LCA of a and b by

$$|\mathsf{LCA}(a, b) \cap C| = |\mathsf{LCA}(a, b)| - |\mathsf{LCA}(a, b) \cap (A' \cup B' \cup D)|.$$

By Claim 35, $|\mathsf{LCA}(a, b) \cap C|$ is exactly the number of $c \in C$ such that $\{c, a\}, \{c, b\} \in E$ and there doesn't exist any $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$.

Next, in $O(n^\omega)$ we can use matrix multiplication again to compute $Q(a, b)$ for every (a, b) where $Q(a, b)$ is defined as the number of $c \in C$ such that $\{c, a\}, \{c, b\} \in E$.

Note that $Q(a,b) - |\mathsf{LCA}(a,b) \cap C|$ is exactly the number of $c \in C$ such that $\{c,a\}, \{c,b\} \in E$ and there *exists* $d \in D$ such that $\{c,d\}, \{d,a\}, \{d,b\} \in E$. Thus, a and b are in a 4-clique if and only if $\{a,b\} \in E$ and $Q(a,b) - |\mathsf{LCA}(a,b) \cap C| > 0$.

Overall, if we can compute AP-#LCA in $T(n)$ time, then we can solve the 4-partite 4-Clique instance in $O(T(n) + n^\omega)$ time. ◀

Since AP-#LCA is easier than AP-All-LCA, this lower bound shows that the AP-All-LCA algorithm in [20] is in fact conditionally optimal.

7 AP-Ver-LCA

In this section, we show two conditional lower bounds for AP-Ver-LCA, based on the Max-Witness hypothesis and the $(4,3)$-Hyperclique hypothesis. First, recall the following theorem:

▶ **Theorem 6.** *If the* AP-Ver-LCA *problem can be solved in $T(n)$ time, then the* Max-Witness *problem can be solved in $\tilde{O}(T(n))$ time.*

At the high level, we reduce the MaxWitness problem to $O(\log n)$ calls of the AP-Ver-LCA problem using a parallel binary search technique.

Proof. Without loss of generality, suppose $n = 2^\ell$ for some integer ℓ. Suppose we have two $n \times n$ Boolean matrices A and B, each already padded with a column and row of ones to ensure that there always exists a Boolean witness. Now, we will describe an algorithm to compute $C = \mathsf{Max\text{-}Witness}(A,B)$ using an AP-Ver-LCA algorithm $\ell = \log n$ times. At the high level, we will be using a parallel binary search to find the maximum witness corresponding to each entry of C.

Construct a tripartite graph G on vertices $V = I \sqcup J \sqcup K$, where $|I| = |J| = |K|$ and identify each of the sets with $[n]$. Add a directed edge from $k \in K$ to $i \in I$ if $A[i,k] = 1$ and an edge from $k \in K$ to $j \in J$ if $B[k,j] = 1$. Then, computing $C[i,j]$ is the same as determining the largest $k \in K$ that is a common ancestor of both $i \in I$ and $j \in J$. Now, we will iteratively find the tth bit in the binary representation of each $C[i,j]$ for $t = 1, \ldots, \ell$ (the first bit is the highest order bit, and the last bit is the lowest order bit). In the first iteration, we do the following.

Phase 1: Construct a graph G_1 by first making a copy of G and adding a vertex w. Then, we add a directed edge from w to every vertex in $I \cup J$. Now, add a directed edge from w to all vertices $k \in K$ whose binary representation starts with 1. Finally, run AP-Ver-LCA where we guess w is an LCA for all pairs $(i,j) \in I \times J$. If w is in fact an LCA, set $c_{i,j}^{(1)} = 0$, and otherwise, set $c_{i,j}^{(1)} = 1$.

More generally, at the tth iteration of the algorithm, we do the following.

Phase t: At the tth iteration of the algorithm for $1 \le t \le \ell$, construct the graph G_t as follows. First, make a copy of G. Then, for each string $b = b_1 b_2 \ldots b_{t-1} \in \{0,1\}^{t-1}$, create a vertex w_b. Now, add an edge from w_b to all vertices in K whose binary representation starts with $b_1 b_2 \ldots b_{t-1} || 1$. Then, add an edge from every w_b to every vertex in $I \cup J$. If $c_{i,j}^{(t-1)} = b$, guess that w_b is an LCA for $(i,j) \in I \times J$. Run AP-Ver-LCA with all of these guesses. If the algorithm outputs YES for (i,j), set $c_{i,j}^{(t)} = b||0$. Otherwise, set $c_{i,j}^{(t)} = b||1$.

We show by induction that at Phase t, $c_{i,j}^{(t)}$ is the first t bits of $C[i,j]$. In Phase 1, note that w is an LCA for $(i,j) \in I \times J$ exactly when none of its children are common ancestors of (i,j). In other words, (i,j) has no common ancestor (and hence no witness) $k \in K$ whose first bit is 1.

Suppose at iteration $t-1$, this claim is true. In other words, for each i, j, $d_{i,j} = c_{i,j}^{(t-1)}$ corresponds to the first $t-1$ bits of $C[i,j]$. Then, at iteration t, we guessed that $w_{d_{i,j}}$ is an ancestor. Since $w_{d_{i,j}}$ only has children whose first t bits are $d_{i,j}||1$, it is an LCA of (i,j) exactly when none of these children are common ancestors, i.e. the largest common ancestor of (i,j) has binary representation starting with $d_{i,j}||0$. Otherwise, it starts with $d_{i,j}||1$, as desired.

Therefore, after ℓ iterations, we have that $C[i,j] = c_{i,j}^{(\ell)}$ (where we interpret $c_{i,j}$ as an ℓ-bit binary integer). The algorithm does $O(n^2)$ work at each phase to construct G_t, and then invokes an AP-Ver-LCA algorithm. Hence the overall runtime is $\tilde{O}(n^2 + T(n)) = \tilde{O}(T(n))$, as desired. ◀

▶ **Theorem 7.** *Assuming the* $(4,3)$-Hyperclique *hypothesis,* Ver-LCA *requires* $n^{2.5-o(1)}$ *time.*

Proof. Suppose we are given a 3-uniform 4-partite hypergraph G on vertex sets A, B, C, U, where $|A| = |B| = |C| = \sqrt{n}$, and $|U| = n$. The $(4,3)$-Hyperclique hypothesis implies that it requires $n^{2.5-o(1)}$ time to determine whether G contains a 4-hyperclique by Fact 18.

We construct the following Ver-LCA instance G' on $O(n)$ vertices.

The graph G' contains 3 layers of vertices V_1, V_2, V_3 with an additional vertex s. We set V_1 to be $A \times B$, set V_2 to be a copy of U and set V_3 to be $(B \times C) \sqcup (C \times A)$.

We also add the following edges to the graph G'.

- Add a directed edge from every $v_1 \in V_1$ to every $v_3 \in V_3$.
- Add a directed edge from $(a,b) \in V_1$ to $u \in V_2$ if and only if $\{u,a,b\} \in E(G)$.
- Add a directed edge from $u \in V_2$ to $(b,c) \in V_3$ if and only if $\{u,b,c\} \in E(G)$. Similarly, add a directed edge from $u \in V_2$ to $(c,a) \in V_3$ if and only if $\{u,c,a\} \in E(G)$.
- Add a directed edge from s to every other vertex in G'. This ensures that every pair of vertices has some common ancestors, and thus has at least one LCA.

We claim that for every $a \in A, b \in B, c \in C$, a, b, c are in a 4-hyperclique in G if and only if $\{a,b,c\} \in E(G)$ and (a,b) is *not* an LCA of (b,c) and (c,a) in G.

First, if a, b, c are in a 4-hyperclique with u, then clearly $\{a,b,c\} \in E(G)$. Also, by the construction of G', $((a,b),u), (u,(b,c)), (u,(c,a))$ are all edges in G'. Thus, (a,b) can reach a vertex u which can reach both (b,c) and (c,a), so (a,b) is not an LCA of (b,c) and (c,a).

Conversely, if $\{a,b,c\} \in E(G)$ and (a,b) is *not* an LCA of (b,c) and (c,a), then since (a,b) can reach both (b,c) and (c,a) via edges added from V_1 to V_3, (a,b) must be able to reach some vertex that can reach both (b,c) and (c,a). Such a vertex must belong to V_2. Say the vertex is u, then by the construction of G', we must have $\{a,b,u\}, \{b,c,u\}, \{c,a,u\} \in E(G)$. Together with the hyperedge $\{a,b,c\}$, a, b, c is in a 4-hyperclique.

Therefore, we can run Ver-LCA on G' with the following set of LCA candidates:

- For every $a \in A, b \in B, c \in C$ such that $\{a,b,c\} \in E(G)$, let $w_{(b,c),(c,a)} = (a,b)$.
- For every other pair of vertices $u, v \in V(G')$, we use Grandoni et al.'s algorithm [25] to find an actual LCA $\ell_{u,v}$ for them in $O(n^{2.447})$ time and set $w_{u,v} = \ell_{u,v}$.

If some LCA candidate is incorrect, it must be that (a,b) is not an LCA for (b,c) and (c,a) for some $a \in A, b \in B, c \in C$ such that $\{a,b,c\} \in E(G)$ and thus by previous discussion, the hypergraph G has a 4-hyperclique. On the other hand, if all LCA candidates are correct, then the hypergraph G does not have a 4-hyperclique.

Therefore, assuming the $(4,3)$-Hyperclique hypothesis, Ver-LCA requires $n^{2.5-o(1)}$ time. ◀

Our conditional lower bounds for AP-Ver-LCA and Ver-LCA are surprising because they suggest that AP-Ver-LCA and Ver-LCA require $n^{2.5-o(1)}$ time, while AP-LCA can be computed in $O(n^{2.447})$ time [25]. This defies the common intuition that verification should be easier than computation.

8 Open problems

We conclude this work by pointing out some potential future directions.

1. Does there exist a subcubic time algorithm for AP-Exact$k$-LCA for any $3 \leq k \leq 5$? Or, can we show an $n^{3-o(1)}$ conditional lower bound for AP-Exact$k$-LCA for any such k? How about AP-Ver-LCA?
2. Is it possible to show conditional lower bounds for AP-List-k-LCA without using AP-AtLeast$k$-LCA as an intermediate problem? For instance, since AP-AtLeast$k$-LCA has $O(n^\omega)$ time algorithms for $k \leq 3$, we cannot hope to get a higher than n^ω lower bound for AP-List-k-LCA for $k \leq 3$ using AP-AtLeast$k$-LCA as an intermediate problem. However, the current best algorithm for AP-List-1-LCA runs in $O(n^{2.447})$ and the best algorithm for AP-List-2-LCA and AP-List-3-LCA runs in $O(n^{2.529})$ time.
3. All our reductions reduce to instances of LCA variants in graphs with $O(1)$ layers. In such graphs, some variants could have faster algorithms. In particular, AP-LCA has an $\tilde{O}(n^\omega)$ time algorithm [18] for graphs with $O(1)$ layers, and thus we cannot hope to show a higher conditional lower bound using our techniques. In order to overcome this, we need to find reductions that show hardness for LCA variants in graphs with many layers.
4. Are there any other related problems whose verification version is easier than the computation version? Can we reduce these problems to or from AP-LCA?

References

1 Amir Abboud, Loukas Georgiadis, Giuseppe F Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemysław Uznański, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 7:1–7:15. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

2 Hassan Aït-Kaci, Robert S. Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Trans. Program. Lang. Syst.*, 11(1):115–146, 1989.

3 Josh Alman. Limits on the universal method for matrix multiplication. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, volume 137, pages 12:1–12:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

4 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

5 Matthias Baumgart, Stefan Eckhardt, Jan Griebsch, Sven Kosub, and Johannes Nowak. All-pairs ancestor problems in weighted dags. In *Proceedings of the First International Conference on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, ESCAPE'07, pages 282–293. Springer-Verlag, 2007.

6 Michael A. Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *J. Algorithms*, 57(2):75–94, 2005.

7 Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993.

8 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994.

9 Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

10 Vincent Bouchitté and Jean-Xavier Rampon. On-line algorithms for orders. *Theor. Comput. Sci.*, 175(2):225–238, 1997.

11 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE, 2017.

12 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic Complexity Theory*. Springer Verlag, 1997.

13 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the Exact Complexity of Evaluating Quantified k-CNF. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 6478, pages 50–59. Springer, 2010.

14 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the Exact Complexity of Evaluating Quantified k-CNF. *Algorithmica*, 65(4):817–827, 2013.

15 Matthias Christandl, François Le Gall, Vladimir Lysikov, and Jeroen Zuiddam. Barriers for rectangular matrix multiplication. *CoRR*, abs/2003.03019, 2020. `arXiv:2003.03019`.

16 Matthias Christandl, Péter Vrana, and Jeroen Zuiddam. Barriers for fast matrix multiplication from irreversibility. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, volume 137, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

17 Richard Cole and Ramesh Hariharan. Dynamic LCA queries on trees. *SIAM J. Comput.*, 34(4):894–923, 2005.

18 Artur Czumaj, Miroslaw Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theor. Comput. Sci.*, 380(1-2):37–46, 2007.

19 Roland Ducournau and Michel Habib. On some algorithms for multiple inheritance in object-oriented programming. In *Proceedings of ECOOP' 87 European Conference on Object-Oriented Programming*, volume 276, pages 243–252. Springer, 1987.

20 Stefan Eckhardt, Andreas Michael Mühling, and Johannes Nowak. Fast lowest common ancestor computations in dags. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA)*, volume 4698, pages 705–716, 2007.

21 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

22 Johannes Fischer and Daniel H. Huson. New common ancestor problems in trees and directed acyclic graphs. *Inf. Process. Lett.*, 110(8-9):331–335, 2010.

23 Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 129–131. IEEE, 1971.

24 Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 135–143. ACM, 1984.

25 Fabrizio Grandoni, Giuseppe F. Italiano, Aleksander Lukasiewicz, Nikos Parotsidis, and Przemyslaw Uznanski. All-pairs LCA in dags: Breaking through the $O(n^{2.5})$ barrier. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 273–289. SIAM, 2021.

26 Michel Habib, Marianne Huchard, and Jeremy P. Spinrad. A linear algorithm to decompose inheritance graphs into modules. *Algorithmica*, 13(6):573–591, 1995.

27 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

28 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

29 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

30 Miroslaw Kowaluk and Andrzej Lingas. LCA queries in directed acyclic graphs. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580, pages 241–248, 2005.

31 Miroslaw Kowaluk and Andrzej Lingas. Unique lowest common ancestors in dags are almost as easy as matrix multiplication. In *Proceedings of the 15th Annual European Symposium (ESA)*, volume 4698, pages 265–274. Springer, 2007.

32 Mirosław Kowaluk, Andrzej Lingas, and Johannes Nowak. A path cover technique for lcas in dags. In *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 222–233. Springer, 2008.

33 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046. SIAM, 2018.

34 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic Triangles, Intermediate Matrix Products, and Convolutions. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151, pages 53:1–53:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

35 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1236–1252. SIAM, 2018.

36 Matti Nykänen and Esko Ukkonen. Finding lowest common ancestors in arbitrarily directed trees. *Inf. Process. Lett.*, 50(6):307–310, 1994.

37 Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.

38 Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, 1979.

39 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

40 Zhaofang Wen. New algorithms for the LCA problem and the binary tree reconstruction problem. *Inf. Process. Lett.*, 51(1):11–16, 1994.

41 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

A PTAS for Capacitated Vehicle Routing on Trees

Claire Mathieu
CNRS Paris, France

Hang Zhou
École Polytechnique, Institut Polytechnique de Paris, France

Abstract

We give a polynomial time approximation scheme (PTAS) for the unit demand capacitated vehicle routing problem (CVRP) on trees, for the entire range of the tour capacity. The result extends to the splittable CVRP.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases approximation algorithms, capacitated vehicle routing, graph algorithms, combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.95

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2111.03735

Funding This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

1 Introduction

Given an edge-weighted graph with a vertex called *depot*, a subset of vertices with demands, called *terminals*, and an integer *tour capacity* k, the *capacitated vehicle routing problem (CVRP)* asks for a minimum length collection of tours starting and ending at the depot such that those tours together cover all the demand and the total demand covered by each tour is at most k. In the *unit demand* version, each terminal has unit demand, which is covered by a single tour;[1] in the *splittable* version, each terminal has a positive integer demand and the demand at a terminal may be covered by multiple tours.

The CVRP was introduced by Dantzig and Ramser in 1959 [15] and is arguably one of the most important problems in Operations Research. Books have been dedicated to vehicle routing problems, e.g., [26, 18, 14, 3]. Yet, these problems remain challenging, both from a practical and a theoretical perspective.

Here we focus on the special case when the underlying metric is a tree. That case has been the object of much research. The splittable tree CVRP was proved NP-hard in 1991 [24], so researchers turned to approximation algorithms. Hamaguchi and Katoh [21] gave a simple lower bound: every edge must be traversed by enough tours to cover all terminals whose shortest paths to the depot contain that edge. Based on this lower bound, they designed a 1.5-approximation in polynomial time [21]. The approximation ratio was improved to $(\sqrt{41}-1)/4$ by Asano, Katoh, and Kawashima [4] and further to 4/3 by Becker [6], both results again based on the lower bound from [21]. On the other hand, it was shown in [4] that using this lower bound one cannot achieve an approximation ratio better than 4/3. More recently, researchers tried to go beyond a constant factor so as to get a $(1+\epsilon)$-approximation, at the cost of relaxing some of the constraints. When the tour capacity is allowed to be

[1] Thus we may identify the demand coverd with the number of terminals covered.

violated by an ϵ fraction, there is a bicriteria PTAS for the unit demand tree CVRP due to Becker and Paul [10]. When the running time is allowed to be quasi-polynomial, Jayaprakash and Salavatipour [22] very recently gave a *quasi-polynomial time approximation scheme (QPTAS)* for the unit demand and the splittable versions of the tree CVRP. In this paper, we close this line of research by obtaining a $(1+\epsilon)$-approximation without relaxing any of the constraints – in other words, a *polynomial-time approximation scheme (PTAS)*.

▶ **Theorem 1.** *There is an approximation scheme for the unit demand capacitated vehicle routing problem (CVRP) on trees with polynomial running time.*

▶ **Corollary 2.** *There is an approximation scheme for the splittable capacitated vehicle routing problem (CVRP) on trees with running time polynomial in the number of vertices n and the tour capacity k.*

To the best of our knowledge, this is the first PTAS for the CVRP in a non-trivial metric and for the entire range of the tour capacity. Previously, PTASs for small capacity as well as QPTASs were given for the CVRP in several metrics, see Section 1.1.3.

1.1 Related Work

Our algorithms build on [22] and [10] but with the addition of significant new ideas, as we now explain.

1.1.1 Comparison with the QPTAS in [22]

Jayaprakash and Salavatipour noted in [22] that

"*it is not clear if it (the QPTAS) can be turned into a PTAS without significant new ideas.*"

The running time in [22] is $n^{O_\epsilon(\log^4 n)}$. Where do those four $\log n$ factors in the exponent come from? At a high level, the QPTAS in [22] consists of three parts: (1) reducing the height of the tree; (2) designing a bicriteria QPTAS; (3) going from the bicriteria QPTAS to a true QPTAS. Our approach builds on [22] but differs from it in each of the three parts, so that in the end we get rid of all of four $\log n$ factors, hence a PTAS.

(1) Jayaprakash and Salavatipour [22] reduce the input tree height from $O(n)$ to $O_\epsilon(\log^2 n)$; whereas instead of the input tree, we consider a *tree of components* (Lemma 9) and reduce its height to $O_\epsilon(1)$, see Figure 1. Pleasingly, the height reduction (Section 4) is much simpler than in [22]. The analysis differs from [22] and uses the structure of a near-optimal solution established in Section 3 and the *bounded distance* property (Definition 3 and Theorem 5).

(2) In the *adaptive rounding* used in [22], they consider the entire range $[1, k]$ of the demands of subtours and partition the subtour demands into *buckets*, resulting in $\Omega_\epsilon(\log k)$ different subtour demands after rounding. In our approach, we define *large* and *small* subtours inside components, depending on whether their demands are $\Omega_\epsilon(k)$ (Definition 14). Then we transform the solution structure to eliminate small subtours (Section 3), hence only $O_\epsilon(1)$ different subtour demands after rounding. This elimination requires a delicate handling of small subtours. Thanks to the additional structure, our analysis of the adaptive rounding is simpler than in [22], and in particular, we do not need the concept of buckets.

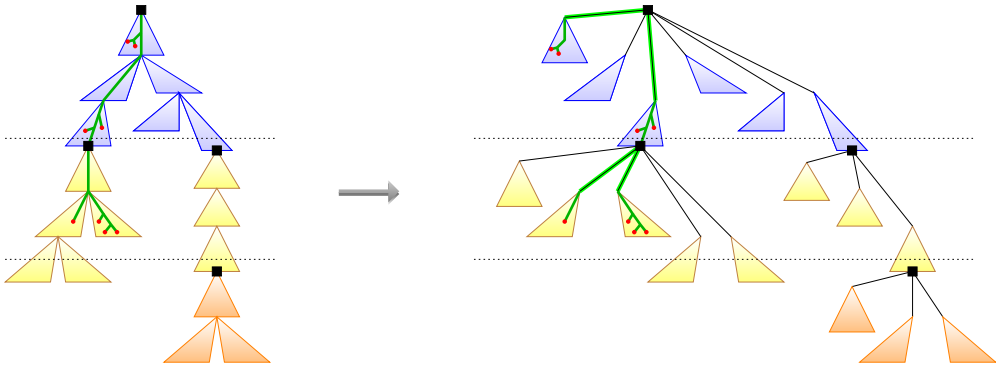


Figure 1 Height reduction for a tree of components. The left figure represents the initial tree of components, where each triangle represents a component. We partition the components into classes (indicated by blue, yellow, and orange), according to the distances from the roots of the components to the root of the tree, and we reduce the height within each class to 1 (right figure), see Section 4. The thick green path in the left figure represents a tour in an optimal solution. The red circular nodes are the terminals visited by that tour. The corresponding tour in the new tree (right figure) spans the same set of terminals.

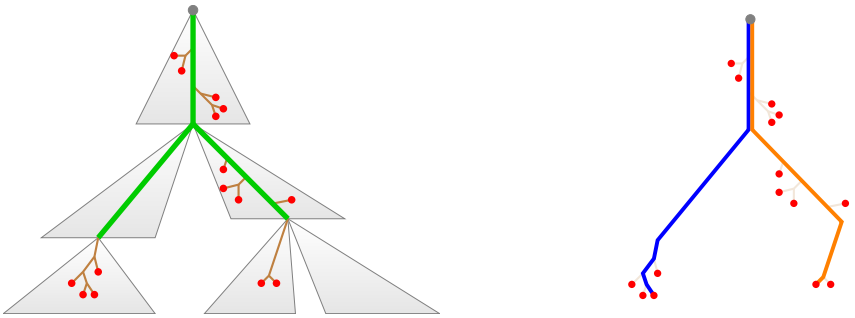
(3) Jayaprakash and Salavatipour show that the *orphan tokens*, which are removed from the tours exceeding capacity, can probably be covered by duplicating a small random set of tours in the optimal solution. Their approach requires remembering the demands of $\Omega(\log n)$ subtours passing through each edge. To avoid this $\Omega(\log n)$ factor, our approach to cover the orphan tokens (Section 3.1) is different, see Figure 2. The analysis of our approach (Sections 3.2 and 3.3) contains several novelties of this paper.

1.1.2 Comparison with the Bicriteria PTAS in [10]

Why is the algorithm in [10] a bicriteria PTAS, but not a PTAS?

Becker and Paul [10] start by decomposing the tree into *clusters*. (1) They require that each *leaf* cluster is visited by a single tour. When the violation of the tour capacity is not allowed, this requirement does not preserve a $(1+\epsilon)$-approximate solution. (2) They also require that each *small* internal cluster is visited by a single tour. To that end, they modify the optimal solution by reassigning all terminals of a small cluster to some existing tour at the cost of possibly violating the tour capacity. Such modifications do not seem achievable in the design of a PTAS.

In this paper, we start by defining *components* (Lemma 9), inspired by *clusters* in [10]. Unlike [10], we allow terminals in any component to be visited by multiple tours. However, allowing many subtours inside a component could result in an exponential running time for a dynamic program. To prevent that, we modify the solution structure inside a component so that the number of subtours becomes bounded (Theorem 13). Instead of considering all subtours simultaneously as in [10], we distinguish *small* subtours from *large* subtours (Definition 14). Inspired by [10], we combine small subtours and reallocate them to existing tours such that the violation of the tour capacity is an $O(\epsilon)$ fraction, see Steps 1 to 3 of the construction in Section 3.1. Next, we use the *iterated tour partitioning (ITP)* and its postprocessing to reduce the demand of the tours exceeding capacity (Figure 2), which is a novelty in this paper, see Steps 4 to 6 of the construction in Section 3.1. The ITP algorithm and its postprocessing are analyzed in Sections 3.2 and 3.3. In particular, we bound the cost due to the ITP algorithm thanks to the *bounded distance* property (Theorem 5) and to



(a) Constructing a traveling salesman tour. **(b)** Adding connections to the depot.

Figure 2 Covering the *orphan tokens* (in red). In Figure 2a, the orphan tokens are contained in the small pieces (in brown) that are removed from the tours exceeding capacity. We add the thick paths (in green) to connect all of the small pieces to the root of the tree. The cost of the thick paths is an $O(\epsilon)$ fraction of the optimal cost (Lemma 17 and Corollary 18), thanks to the bounded distance property. The induced traveling salesman tour is a double cover of the tree spanning the orphan tokens. Next, we apply the *iterated tour partitioning (ITP)* algorithm on that tour. In Figure 2b, the two paths (in blue and in orange) represent the connections to the depot added by the ITP algorithm. Their cost is again an $O(\epsilon)$ fraction of the optimal cost (Lemma 19), thanks to the bound on the number of orphan tokens and the bounded distance property.

the parameters in our component decomposition that are different from those in [10], see Remark 10. Besides the above novelties in our approach, the height reduction (Figure 1, see also Section 4), the adaptive rounding (Section 5), the reduction to bounded distances, as well as part of the dynamic program are new compared with [10]. These additional techniques are essential in the design of our PTAS, because of the more complicated solution structure inside components in our approach compared with the solution structure inside clusters in [10].

1.1.3 Other Related Work

Constant-factor approximations in general metric spaces

The CVRP is a generalization of the *traveling salesman problem (TSP)*. In general metric spaces, Haimovich and Rinnooy Kan [20] introduced a simple heuristics, called *iterated tour partitioning (ITP)*. Altinkemer and Gavish [2] showed that the approximation ratio of the ITP algorithm for the unit demand and the splittable CVRP is at most $1 + \left(1 - \frac{1}{k}\right) C_{\text{TSP}}$, where $C_{\text{TSP}} \geq 1$ is the approximation ratio of a TSP algorithm. Bompadre, Dror, and Orlin [12] improved this bound to $1 + \left(1 - \frac{1}{k}\right) C_{\text{TSP}} - \Omega\left(\frac{1}{k^3}\right)$. The ratio for the unit demand and the splittable CVRP on general metric spaces was recently improved by Blauth, Traub, and Vygen [11] to $1 + C_{\text{TSP}} - \epsilon$, for some small constant $\epsilon > 0$.

QPTASs

Das and Mathieu [16] designed a QPTAS for the CVRP in the Euclidean space; Jayaprakash and Salavatipour [22] designed a QPTAS for the CVRP in trees and extended that algorithm to QPTASs in graphs of bounded treewidth, bounded doubling or highway dimension. When the tour capacity is fixed, Becker, Klein, and Saulpic [7] gave a QPTAS for planar graphs and bounded-genus graphs.

PTASs for small capacity

In the Euclidean space, there have been PTAS algorithms for the CVRP with small capacity k: work by Haimovich and Rinnooy Kan [20], when k is constant; by Asano et al. [5] extending techniques in [20], for $k = O(\log n / \log\log n)$; and by Adamaszek, Czumaj, and Lingas [1], when $k \leq 2^{\log^{f(\epsilon)}(n)}$. For higher dimensional Euclidean metrics, Khachay and Dubinin [23] gave a PTAS for fixed dimension ℓ and $k = O(\log^{\frac{1}{\ell}}(n))$. Again when the capacity is bounded, Becker, Klein and Schild [9] gave a PTAS for planar graphs; Becker, Klein, and Saulpic [8] gave a PTAS for graphs of bounded highway dimension; and Cohen-Addad et al. [13] gave PTASs for bounded genus graphs and bounded treewidth graphs.

Unsplittable CVRP

In the *unsplittable* version of the CVRP, every terminal has a positive integer demand, and the entire demand at a terminal should be served by a single tour. On general metric spaces, the best-to-date approximation ratio for the unsplittable CVRP is roughly 3.194 due to the recent work of Friggstad et al. [17]. For tree metrics, the unsplittable CVRP is APX-hard: indeed, it is NP-hard to approximate the unsplittable tree CVRP to better than a 1.5 factor [19] using a reduction from the bin packing problem. Labbé, Laporte and Mercure [24] gave a 2-approximation for the unsplittable tree CVRP. The approximation ratio for the unsplittable tree CVRP was improved to $(1.5 + \epsilon)$ very recently by Mathieu and Zhou [25], building upon several techniques in the current paper.

1.2 Overview of Our Techniques

The main part of our work focuses on the unit demand tree CVRP, and we extend our results to the splittable tree CVRP in the end of this work.

▶ **Definition 3** (bounded distances). *Let $D_{\min}$ (resp. $D_{\max}$) denote the minimum (resp. maximum) distance between the depot and any terminal in the tree. We say that an instance has* bounded distances *if $D_{\max} < \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min}$.*

Theorem 1 follows directly from Theorems 4 and 5.

▶ **Theorem 4.** *There is a polynomial time $(1 + 4\epsilon)$-approximation algorithm for the unit demand CVRP on the tree T with bounded distances.*

▶ **Theorem 5.** *For any $\rho \geq 1$, if there is a polynomial time ρ-approximation algorithm for the unit demand (resp. splittable, or unsplittable) CVRP on trees with* bounded distances*, then there is a polynomial time $(1 + 5\epsilon)\rho$-approximation algorithm for the unit demand (resp. splittable, or unsplittable) CVRP on trees with* general distances*.*

Theorem 5 may be of independent interest for the splittable and the unsplittable versions of the tree CVRP.

Outline for unit demand instances with bounded distances (Theorem 4)

First, we show that there exists a near-optimal solution with a simple structure, and afterwards, we use a dynamic program to compute the best solution with that structure.

In Section 3, we consider the *components* of the tree T (Lemma 9) and we show that there exists a near-optimal solution such that terminals within each component are visited by a constant $O_\epsilon(1)$ number of tours and that each of those tours visits $\Omega_\epsilon(k)$ terminals in that component (Theorem 13). The proof of Theorem 13 contains several novelties in our

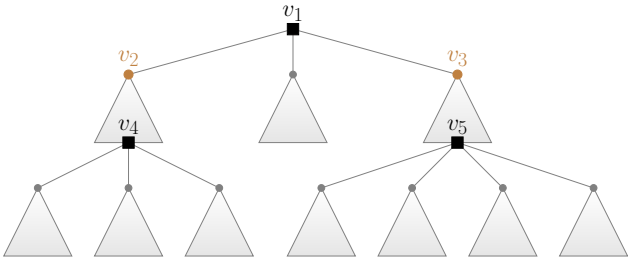


Figure 3 At each vertex of the tree, the dynamic program memorizes the capacities used by the subtours in the subtree and their total cost. Terminals within each component are visited by $O_\epsilon(1)$ tours and each of those tours visits $\Omega_\epsilon(k)$ terminals in that component. Here is an example of the flow of execution in the dynamic program. First, independent computation in each component. Next, computation in the subtrees rooted at vertices v_4 and v_5. Then computation for the subtrees rooted at vertices v_2 and v_3. Finally, computation for the subtree rooted at vertex v_1. The output is the best solution in the subtree rooted at v_1. Vertices v_1, v_4, and v_5, whose degrees may be arbitrarily large, are where adaptive rounding of the subtour demands is needed to maintain polynomial time. The cost due to the rounding is small thanks to the way components are defined and the bounded distance property, and does not accumulate excessively because the height is bounded (Section 4).

work. We start by defining *large* and *small* subtours inside a component, depending on the number of terminals visited by the subtours. To construct a near-optimal solution with that structure, first, we detach small subtours from their initial tours, combine small subtours in the same component, and reallocate the combined subtours to existing tours. Then we remove subtours from tours exceeding capacity. To connect the removed subtours to the root of the tree, we include the *spines subtours* (Definition 12) of all *internal components*, and we obtain a traveling salesman tour. Next, we apply the *iterated tour partitioning (ITP)* algorithm on that tour, see Figure 2. Finally, in a postprocessing step, we eliminate the small subtours created due to the ITP algorithm. The complete construction is in Section 3.1; the feasibility of the construction is in Section 3.2; and the analysis on the constructed solution is in Section 3.3, which in particular uses the bounded distance property.

In Section 4, we transform the tree T into a tree $\hat{T}$ that has $O_\epsilon(1)$ levels of components (Figure 1) and satisfies the following property.

▶ **Fact 6.** *The tree $\hat{T}$ defined in Section 4 can be computed in polynomial time. The components in the tree $\hat{T}$ are the same as those in the tree T. Any solution for the unit demand CVRP on the tree $\hat{T}$ can be transformed in polynomial time into a solution for the unit demand CVRP on the tree T without increasing the cost.*

Thanks to the structure of the near-optimal solution on T (Section 3) and to the bounded distance property, the optimal cost for $\hat{T}$ is increased by an $O(\epsilon)$ fraction compared with the optimal cost for T (Theorem 23).

In Section 5, we apply the *adaptive rounding* on the demands of the subtours in a near-optimal solution on $\hat{T}$. Recall that in the design of the QPTAS by Jayaprakash and Salavatipour [22], the main technique is to show the existence of a near-optimal solution in which the demand of a subtour can be rounded to the nearest value from a set of only *poly-logarithmic* threshold values. In our work, we reduce the number of threshold values to *a constant* $O_\epsilon(1)$ (Theorem 26). To analyze the adaptive rounding, observe that an extra cost occurs whenever we detach a subtour and complete it into a separate tour by connecting it to the depot. We bound the extra cost thanks to the structure of a near-optimal solution inside components (Section 3), the reduced height of the components (Section 4), as well as the bounded distance property.

In the full version of the paper, we design a *polynomial-time dynamic program* that computes the best solution on the tree $\hat{T}$ that satisfies the constraints on the solution structure imposed by previous sections. The algorithm combines a dynamic program inside components and two dynamic programs in subtrees, see Figure 3. Thus we obtain the following Theorem 7.

▶ **Theorem 7.** *Consider the unit demand CVRP on the tree $\hat{T}$. There is a dynamic program that computes in polynomial time a solution with cost at most $(1+4\epsilon)\cdot \mathrm{opt}$, where* opt *denotes the optimal cost for the unit demand CVRP on the tree T.*

Theorem 4 follows directly from Theorem 7 and Fact 6.

Reduction from general distances to bounded distances (Theorem 5)

In the full version of the paper, we prove Theorem 5. We use Baker's technique to split tours into pieces such that each piece covers terminals that are within a certain range of distances from the depot. This requires duplicating some parts of the tours so that each piece of the tour is connected to the depot.

Extension to the splittable setting (Corollary 2)

In the full version of the paper, we extend the result in Theorem 1 to the splittable setting, thus obtaining Corollary 2.

Open questions

Previously, Jayaprakash and Salavatipour [22] extended their QPTAS on trees to QPTASs on graphs of bounded treewidth and beyond, including Euclidean spaces. While some of our techniques extend to those settings, others do not seem to carry over without significant additional ideas, so it is an interesting open question whether the techniques in our paper could be used in the design of PTAS algorithms for other metrics, such as graphs of bounded treewidth, planar graphs, and Euclidean spaces.

2 Preliminaries

Let T be a rooted tree (V, E) with root $r \in V$ and edge weights $w(u, v) \geq 0$ for all $(u, v) \in E$. The root r represents the *depot* of the tours. Let n denote the number of vertices in V. Let $V' \subseteq V$ denote the set of *terminals*, such that a *token* is placed on each terminal $v \in V'$. Let $k \in [1, n]$ be an integer *capacity* of the tours. The *cost* of a tour t, denoted by $\mathrm{cost}(t)$, is the overall weight of the edges on that tour. We say that a tour *visits* a terminal $v \in V'$ if the tour picks up the token at v.[2]

▶ **Definition 8** (unit demand tree CVRP)**.** *An instance of the* unit demand *version of the* capacitated vehicle routing problem (CVRP) *on* trees *consists of*

- *an edge weighted* tree $T = (V, E)$ *with* $n = |V|$ *and with* root $r \in V$ *representing the* depot*,*
- *a set* $V' \subseteq V$ *of* terminals*,*
- *a positive integer* tour capacity k *such that* $k \leq n$*.*

[2] Note that a tour might go through a terminal v without picking up the token at v.

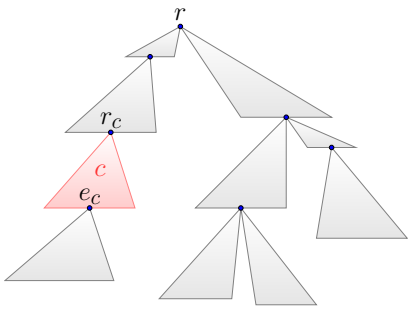


Figure 4 Decomposition into components. In this example, the tree is decomposed into four *leaf* components and six *internal* components. An internal component c has a *root* vertex r_c and an *exit* vertex e_c.

A feasible solution is a set of tours such that

- *each tour starts and ends at r,*
- *each tour visits at most k terminals,*
- *each terminal is visited by one tour.*

The goal is to find a feasible solution such that the total cost of the tours is minimum.

Let OPT (resp. OPT_1, OPT_2, or OPT_3) denote an optimal (resp. near-optimal) solution to the unit demand CVRP, and let opt (resp. opt_1, opt_2, or opt_3) denote the value of that solution.

Without loss of generality, we assume that every vertex in the input tree T has exactly two children, and that the terminals are the same as the leaf vertices of the tree. Indeed, general instances can be reduced to instances with these properties by inserting edges of weight 0, removing leaf vertices that are not terminals, and slicing out internal vertices of degree two, see, e.g. [10] for details.

For any vertex $v \in V$, a *subtour at the vertex v* is a path that starts and ends at v and only visits vertices in the subtree rooted at v. The *demand* of a subtour is the number of terminals visited by that subtour. For each vertex $v \in V$, let $\mathrm{dist}(v)$ denote the distance between v and the depot in the tree T. For technical reasons, we allow *dummy* terminals to be included in the solution at internal vertices of the tree.

Throughout the paper, we define several constants depending on ϵ: Γ (Lemma 9), α (Theorem 13), H_ϵ (Lemma 21), and β (Theorem 26). They satisfy the relation that $H_\epsilon \gg \Gamma \gg 1 \gg \epsilon \gg \alpha \gg \beta$.

Decomposition of the Tree into Components

The component decomposition (Lemma 9) is inspired by the cluster decomposition by Becker and Paul [10]. The proof of Lemma 9 is similar to arguments in [10]; we give its proof in the full version of the paper for completeness.

▶ **Lemma 9.** *Let $\Gamma = \frac{12}{\epsilon}$. There is a polynomial time algorithm to compute a partition of the edges of the tree T into a set $\mathcal{C}$ of* components *(see Figure 4), such that all of the following properties are satisfied:*

- *Every component $c \in \mathcal{C}$ is a connected subgraph of T; the* root *vertex of the component c, denoted by r_c, is the vertex in c that is closest to the depot.*
- *We say that a component $c \in \mathcal{C}$ is a* leaf *component if all descendants of r_c in tree T are in c, and is an* internal *component otherwise. A leaf component c interacts with other components at vertex r_c only. An internal component c interacts with other components at two vertices only: at vertex r_c, and at another vertex, called the* exit *vertex of the component c, and denoted by e_c.*

- *Every component $c \in \mathcal{C}$ contains at most $2\Gamma \cdot k$ terminals. We say that a component is* big *if it contains at least $\Gamma \cdot k$ terminals. Each leaf component is big.*
- *If the number of components in $\mathcal{C}$ is strictly greater than one, then we have: (1) there exists a map from all components to big components, such that the image of a component is among its descendants (including itself), and each big component has at most three pre-images; and (2) the number of components in $\mathcal{C}$ is at most $3/\Gamma$ times the total demand in the tree T.*

▶ Remark 10. The root and the exit vertices of components are a rough analog of *portals* used in approximation schemes for other problems: they are places where the dynamic program will gather and synthesize information about partial solutions before passing it on.

Compared with [10], the decomposition in Lemma 9 uses different parameters: the number of terminals inside a *leaf component* is $\Theta(k/\epsilon)$, whereas in [10] the number of terminals inside a *leaf cluster* is $\Theta(\epsilon \cdot k)$; the threshold demand to define *big components* is $\Theta(k/\epsilon)$, whereas in [10] the threshold demand to define *small clusters* is $\Theta(\epsilon^2 \cdot k)$.

▶ **Definition 11** (subtours in components and subtour types)**.** *Let c be any component. A* subtour in the component c *is a path that starts and ends at the root r_c of the component, and such that every vertex on the path is in c. The* type *of a subtour is "*passing*" if c is an internal component and the exit vertex e_c belongs to that subtour; and is "*ending*" otherwise.*

A passing subtour in c is to be combined with a subtour at e_c. In a leaf component, there is no passing subtour.

▶ **Definition 12** (spine subtour)**.** *For an internal component c, we define the* spine subtour *in the component c, denoted by spine_c, to be the connection (in both directions) between the root vertex r_c and the exit vertex e_c of the component c, without visiting any terminal.*

From the definition, a spine subtour in a component is also a passing subtour in that component.

Without loss of generality, we assume that any subtour in a component c either visits at least one terminal in c or is a spine subtour; that any tour traverses each edge of the tree at most twice (once in each direction); and that any tour contains at most one subtour in any component.[3]

3 Solutions Inside Components

In this section, we prove Theorem 13, which is a main novelty in this paper.

▶ **Theorem 13.** *Let $\alpha = \epsilon^{(\frac{1}{\epsilon}+1)}$. Consider the unit demand CVRP on the tree T with bounded distances. There exist dummy terminals and a solution OPT_1 visiting all of the real and the dummy terminals, such that all of the following holds:*

1. *For each component c, there are at most $\frac{2\Gamma}{\alpha} + 1$ tours visiting terminals in c;*
2. *For each component c and each tour t visiting terminals in c, the number of the terminals in c visited by t is at least $\alpha \cdot k$;*
3. *We have $\text{opt}_1 \leq (1+\epsilon) \cdot \text{opt}$.*

[3] If a tour contains several subtours in a component c, we may combine those subtours into a single subtour in c without increasing the cost of the tour.

3.1 Construction of OPT_1

▶ **Definition 14** (large and small subtours). *We say that a subtour is* large *if its demand is at least $\alpha \cdot k$, and* small *otherwise.*

The construction of OPT_1, starting from an optimal solution OPT, is in several steps.

Step 1: Detaching small subtours

Prune each tour of OPT so that it only visits the terminals that do *not* belong to a small subtour in any component, and is minimal. In other words, if a tour t in OPT contains a small ending subtour t_e in a component c, then we remove t_e from t; and if a tour t in OPT contains a non-spine small passing subtour t_p in a component c, then we remove t_p from t, except for the spine subtour of c.

Let A denote the set of the resulting tours. Note that each tour in A is connected. The removed pieces of a non-spine small passing subtour t_p may be disconnected from one another.

The parts of OPT that have been pruned consist of the set $\mathcal{E}$, each element being a small ending subtours in a component, and of the set $\mathcal{P}$, each element being a group of pieces in a component obtained from a non-spine small passing subtour by removing the corresponding spine subtour. The *demand* of a group of pieces in $\mathcal{P}$ is the number of terminals in all of the pieces in that group.

Step 2: Combining small subtours within components

- For each component c, repeatedly concatenate subtours in c from the set $\mathcal{E}$ so that in the end, all resulting subtours in c from the set $\mathcal{E}$ have demands between $\alpha \cdot k$ and $2\alpha \cdot k$ except for at most one subtour.
- For each component c, repeatedly merge groups in c from the set $\mathcal{P}$ so that in the end, all resulting groups in c from the set $\mathcal{P}$ have demands between $\alpha \cdot k$ and $2\alpha \cdot k$ except for at most one group.

Let $\mathcal{E}'$ and $\mathcal{P}'$ denote the corresponding sets after the modifications for all components c. Let $B = \mathcal{E}' \cup \mathcal{P}'$. An *element* of B is either a *subtour* or a *group of pieces* in a component c. For each component c, all elements of B in the component c have demands between $\alpha \cdot k$ and $2\alpha \cdot k$ except for at most two elements with smaller demands.

Step 3: Reassigning small subtours

Construct a bipartite graph with vertex sets A and B and with edge set E. Let a be any tour in A, and let a_0 denote the corresponding tour in OPT. Let b be any element in B. The set E contains an edge between a and b if and only if the element b contains terminals from a_0; the weight of the edge (a, b) is the number of terminals in both b and a_0. By Lemma 1 from [10], there exists an assignment $f : B \to A$ such that each element $b \in B$ is assigned to a tour $a \in A$ with $(a, b) \in E$ and that, for each tour $a \in A$, the demand of a plus the overall demand of the elements $b \in B$ that are assigned to a is at most the demand of the corresponding tour a_0 plus the maximum demand of any element in B.

Let A_1 denote the set of *pseudo-tours* induced by the assignment f. Each pseudo-tour in A_1 is the union of a tour $a \in A$ and the elements $b \in B$ that are assigned to a.

Step 4: Correcting tour capacities

For each pseudo-tour a_1 in A_1, if the demand of a_1 exceeds k, we repeatedly remove an element $b \in B$ from a_1, until the demand of a_1 is at most k.

Let A_2 denote the resulting set of pseudo-tours. Every pseudo-tour in A_2 is a connected tour of demand at most k (Lemma 15). Let $B^* \subseteq B$ denote the set of the removed elements $b \in B$. The elements in B^* are represented by the small pieces in Figure 2a (Page 4).

Step 5: Creating additional tours

We connect the elements of B^* to the depot by creating additional tours as follows.

- Let Q denote the union of the spine subtours for all internal components. Q is represented by the green thick paths in Figure 2a. Let X denote a multi-subgraph of T that is the union of the elements in B^* and the edges in Q. Observe that each element in B^* is connected to the depot through edges in Q. Thus X is connected, and induces a traveling salesman tour t_{TSP} visiting all terminals in B^*. Without loss of generality, we assume that, for any component c, if t_{TSP} visits terminals in c, then those terminals belong to a single subpath p_c of t_{TSP}, such that p_c does not visit any terminal from other components.
- If the traveling salesman tour t_{TSP} is within the tour capacity, then let A_3 denote the set consisting of a single tour t_{TSP}. Otherwise, we apply the *iterated tour partitioning (ITP)* algorithm [20] on t_{TSP}: we partition t_{TSP} into segments with exactly k terminals each, except possibly the last segments containing less than k terminals. For each segment, we connect its endpoints to the depot so as to make a tour, see Figure 2b. Let A_3 denote the resulting set of tours.

Let $A_4 = A_2 \cup A_3$.

Step 6: Postprocessing

For each component c, we rearrange the small subtours in A_4 as follows.

- For each tour t in A_4 that contains a small subtour in c, letting t_c denote this small subtour, if t_c is an ending subtour, we remove t_c from t; if t_c is a passing subtour, we remove t_c from t, except for the spine subtour in c. The total demand of all of the removed small subtours in c is at most k (Lemma 16).
- We create an additional tour t_c^* from the depot that connects all of the removed small subtours in c. If the demand of t_c^* is less than $\alpha \cdot k$, then we include dummy terminals at r_c into the tour t_c^* so that its demand is exactly k.

Let A_5 denote the resulting set of tours after removing small subtours from A_4. Let A_6 denote the set of newly created tours $\{t_c^*\}_{c \in \mathcal{C}}$.

Finally, let $\text{OPT}_1 = A_5 \cup A_6$.

In Section 3.2, we show that OPT_1 is a feasible solution to the unit demand tree CVRP; in Section 3.3, we prove the three properties of OPT_1 in the claim of Theorem 13.

3.2 Feasibility of the Construction

▶ **Lemma 15.** *Every pseudo-tour in A_2 is a connected tour of demand at most k.*

Proof. Let a_2 be any pseudo-tour in A_2. By the construction in Step 4, the demand of a_2 is at most k. It suffices to show that a_2 is a connected tour.

Observe that a_2 is the union of a tour $a \in A$ and some elements $b_1, \ldots, b_q$ from B, for $q \geq 0$. From the construction, any tour $a \in A$ is connected. Consider any b_i for $i \in [1, q]$. Observe that $f(b_i) = a$, so the edge (a, b_i) belongs to the bipartite graph (A, B). If b_i is a

subtour at r_c for some component c, then r_c must belong to the tour a; and if b_i is a group of pieces in some component c, then the spine subtour of c must belong to the tour a. So the union of a and b_i is connected. Thus $a_2 = a \cup b_1 \cup \cdots \cup b_q$ is a connected tour. ◀

▶ **Lemma 16.** *In any component c, the total demand of all of the removed small subtours in c at the beginning of Step 6 is at most k.*

Proof. Let c be any component. The key is to show that the number of non-spine small subtours in c that are contained in tours in A_4 is at most 4. Since $A_4 = A_2 \cup A_3$, we analyze the number of non-spine small subtours in c that are contained in tours in A_2 and in A_3, respectively.

The tours in A_2 contain at most two non-spine small subtours in c, since at most two elements of B in component c have demands less than $\alpha \cdot k$.

We claim that the tours in A_3 contain at most two non-spine small subtours in c. If A_3 consists of a single tour t_{TSP}, the claim follows trivially since any tour contains at most one subtour in c from our assumption. Next, we consider the case when A_3 is generated by the ITP algorithm. From our assumption, if t_{TSP} visits terminals in c, then those terminals belong to a single subpath p_c of t_{TSP}, such that p_c does not visit any terminal from other components. By applying the ITP algorithm on t_{TSP}, we obtain a collection of segments. All segments that intersect p_c visit exactly k terminals in c, except for possibly the first and the last of those segments visiting less than k terminals in c. Hence at most two non-spine small subtours in c among the tours in A_3.

Therefore, the number of non-spine small subtours in c in the solution A_4 is at most 4. Since each small subtour has demand at most $\alpha \cdot k$, the total demand of the removed small subtours is at most $4 \cdot \alpha \cdot k < k$. ◀

3.3 Analysis of $\mathbf{OPT_1}$

Let $c \in \mathcal{C}$ be any component. From Lemma 9, c contains at most $2\Gamma \cdot k$ real terminals. Each tour in A_5 visiting terminals in c visits at least $\alpha \cdot k$ real terminals in c, so there are at most $\frac{2\Gamma}{\alpha}$ tours in A_5 visiting terminals in c. There is a single tour in A_6, the tour t_c^*, that visits terminals in c. Hence the first property of the claim. From the construction of t_c^*, the second property of the claim follows.

It remains to analyze the cost of OPT_1. Compared with OPT, the extra cost in OPT_1 is due to Step 5 and Step 6 of the construction. This extra cost consists of the cost of the edges in Q (Step 5), the cost in the ITP algorithm to connect the endpoints of all segments to the depot (Step 5), and the cost of the postprocessing (Step 6), which we bound in Corollary 18 and Lemmas 19 and 20, respectively. Both Corollary 18 and Lemma 20 are based on Lemma 17.

▶ **Lemma 17.** *We have* $\displaystyle\sum_{\text{component } c} \text{dist}(r_c) \le \frac{\epsilon}{8} \cdot \text{opt}.$

Proof. For any edge e in T, let u and v denote the two endpoints of e such that u is the parent of v. Let T_e denote the subtree of T rooted at v. Let n_e denote the number of terminals in T_e. From the lower bound in [21], we have

$$\text{opt} \ge \sum_{e \in T} 2 \cdot w(e) \cdot \frac{n_e}{k}.$$

Since each big component contains at least $\Gamma \cdot k$ terminals, we have

$$n_e \ge \sum_{\text{big component } c \subseteq T_e} \Gamma \cdot k.$$

Thus

$$\begin{aligned}\text{opt} &\geq \sum_{e\in T} 2\cdot w(e)\cdot \sum_{\text{big component } c\subseteq T_e} \Gamma \\ &= \sum_{\text{big component } c} \Gamma\cdot \sum_{e\in T \text{ such that } c\subseteq T_e} 2\cdot w(e) \\ &= \sum_{\text{big component } c} \Gamma\cdot 2\cdot \text{dist}(r_c).\end{aligned}$$

From Lemma 9, there exists a map from all components to big components such that the image of a component is among its descendants (including itself) and each big component has at most three pre-images. Thus

$$3\cdot \sum_{\text{big component } c} \text{dist}(r_c) \geq \sum_{\text{component } c} \text{dist}(r_c).$$

Therefore,

$$\text{opt} \geq \frac{2\cdot\Gamma}{3}\cdot \sum_{\text{component } c} \text{dist}(r_c).$$

The claim follows since $\Gamma = \frac{12}{\epsilon}$. ◀

▶ **Corollary 18.** *We have* $\text{cost}(Q) \leq \frac{\epsilon}{4}\cdot \text{opt}$.

Proof. Observe that every edge in Q belongs to the connection (in both directions) between the depot and the root of some component c. By Lemma 17, we have

$$\text{cost}(Q) \leq 2\cdot \sum_{\text{component } c} \text{dist}(r_c) \leq \frac{\epsilon}{4}\cdot \text{opt}.$$ ◀

▶ **Lemma 19.** *Let* Δ_1 *denote the cost in the ITP algorithm to connect the endpoints of all segments to the depot in Step 5. We have* $\Delta_1 \leq \frac{\epsilon}{4}\cdot \text{opt}$.

Proof. Let n' denote the number of terminals in the tree T. First, we show that the number of terminals on t_{TSP} is at most $4\alpha\cdot n'$. Observe that the number of terminals on t_{TSP} is the overall removed demand in Step 4. Consider any pseudo-tour $a_1 \in A_1$ whose demand exceeds k. Let a_0 denote the corresponding tour in OPT. By the construction, the demand of a_1 is at most the demand of a_0 plus the maximum demand of any element in B. Since the demand of a_0 is at most k and the demand of any element in B is at most $2\alpha\cdot k$, the demand of a_1 is at most $k + 2\alpha\cdot k$. Let a_2 denote the corresponding tour after the correction of capacity in Step 4. Since any element in B has demand at most $2\alpha\cdot k$, the demand of a_2 is at least $k - 2\alpha\cdot k$. So the total removed demand from a_1 in Step 4 is at most $4\alpha\cdot k$. There are at most $\frac{n'}{k}$ pseudo-tours $a_1 \in A_1$ whose demands exceed k. Summing over all those pseudo-tours, the overall removed demand in Step 4 is at most $\frac{n'}{k}\cdot 4\alpha\cdot k = 4\alpha\cdot n'$. Hence the number of terminals on t_{TSP} is at most $4\alpha\cdot n'$.

If $4\alpha\cdot n' \leq k$, then t_{TSP} is within the tour capacity, so the ITP algorithm is not applied and $\Delta_1 = 0$. It remains to consider the case in which $4\alpha\cdot n' > k$. By the construction in the ITP algorithm, every segment visits exactly k terminals except possibly the last segment. Thus the number ℓ_{ITP} of resulting tours in the ITP algorithm is

$$\ell_{\text{IPT}} \leq \frac{4\alpha\cdot n'}{k} + 1. \tag{1}$$

Since $4\alpha \cdot n' > k$, we have $\ell_{\mathrm{IPT}} < \frac{8\alpha \cdot n'}{k}$. Since $\Delta_1 \le \ell_{\mathrm{ITP}} \cdot 2 \cdot D_{\max}$ and using Definition 3 and the definition of α in the claim of Theorem 13, we have

$$\Delta_1 < \frac{8\alpha \cdot n'}{k} \cdot 2 \cdot \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min} < \frac{\epsilon}{2} \cdot \frac{n'}{k} \cdot D_{\min}.$$

On the other hand, the solution OPT consists of at least $\frac{n'}{k}$ tours, so $\mathrm{opt} \ge \frac{2n'}{k} \cdot D_{\min}$. Therefore, $\Delta_1 \le \frac{\epsilon}{4} \cdot \mathrm{opt}$. ◀

▶ **Lemma 20.** *Let Δ_2 denote the cost of the postprocessing (Step 6). Then $\Delta_2 \le \frac{\epsilon}{2} \cdot \mathrm{opt}$.*

Proof. For each leaf component c, the cost to connect the small subtours in c to the depot in Step 6 is at most $2 \cdot \mathrm{dist}(r_c)$; and for each internal component c, the cost to connect the small subtours in c to the depot is at most $2 \cdot \mathrm{dist}(r_c) + \mathrm{cost}(\mathrm{spine}_c)$. Summing over all components, we have

$$\Delta_2 \le \sum_{\text{component } c} 2 \cdot \mathrm{dist}(r_c) + \sum_{\text{internal component } c} \mathrm{cost}(\mathrm{spine}_c).$$

By Lemma 17, we have

$$\sum_{\text{component } c} 2 \cdot \mathrm{dist}(r_c) \le \frac{\epsilon}{4} \cdot \mathrm{opt}$$

and

$$\sum_{\text{internal component } c} \mathrm{cost}(\mathrm{spine}_c) = \mathrm{cost}(Q) \le \frac{\epsilon}{4} \cdot \mathrm{opt}.$$

Thus $\Delta_2 \le \frac{\epsilon}{2} \cdot \mathrm{opt}$. ◀

From Corollary 18 and Lemmas 19 and 20, we have $\mathrm{opt}_1 \le \mathrm{opt} + \mathrm{cost}(Q) + \Delta_1 + \Delta_2 \le (1+\epsilon) \cdot \mathrm{opt}$. This completes the proof for Theorem 13.

4 Height Reduction

In this section, we transform the tree T into a tree $\hat{T}$ so that $\hat{T}$ has $O_\epsilon(1)$ levels of components, see Figure 1. We assume that the tree T has bounded distances. To begin with, we partition the components according to the distances from their roots to the depot.

▶ **Lemma 21.** *Let $\tilde{D} = \alpha \cdot \epsilon \cdot D_{\min}$. Let $H_\epsilon = (\frac{1}{\epsilon})^{\frac{2}{\epsilon}+1}$. For each $i \in [1, H_\epsilon]$, let $\mathcal{C}_i \subseteq \mathcal{C}$ denote the set of components $c \in \mathcal{C}$ such that $\mathrm{dist}(r_c) \in \big[(i-1) \cdot \tilde{D}, i \cdot \tilde{D}\big)$. Then any component $c \in \mathcal{C}$ belongs to a set $\mathcal{C}_i$ for some $i \in [1, H_\epsilon]$.*

Proof. Let $c \in \mathcal{C}$ be any component. We have

$$\mathrm{dist}(r_c) \le D_{\max} < \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min} = H_\epsilon \cdot \tilde{D},$$

where the second inequality follows from Definition 3, and the equality follows from the definition of α in Theorem 13 and the definitions of $\tilde{D}$ and H_ϵ. Thus there exists $i \in [1, H_\epsilon]$ such that $c \in \mathcal{C}_i$. ◀

▶ **Definition 22** (maximally connected sets and critical vertices). *We say that a set of components $\tilde{\mathcal{C}} \subseteq \mathcal{C}_i$ is* maximally connected *if the components in $\tilde{\mathcal{C}}$ are connected to each other and $\tilde{\mathcal{C}}$ is maximal within $\mathcal{C}_i$. For a maximally connected set of components $\tilde{\mathcal{C}} \subseteq \mathcal{C}_i$, we define the* critical vertex *of $\tilde{\mathcal{C}}$ to be the root vertex of the component $c \in \tilde{\mathcal{C}}$ that is closest to the depot.*

Figure 1 (Page 3) is an example with three levels of components: $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, indicated by different colors. There are four maximally connected sets of components. The critical vertices are represented by rectangular nodes.

Algorithm 1 Construction of the tree $\hat{T}$ (see Figure 1).

```
for each i ∈ [1, H_ε] do
    for each maximally connected set of components C̃ ⊆ C_i do
        z ← critical vertex of C̃
        for each component c ∈ C̃ do
            δ ← r_c-to-z distance in T
            Split the tree T at the root vertex r_c of the component c    ▷ r_c is duplicated
            Add an edge between the root of the component c and z with weight δ
T̂ ←the resulting tree
```

Let $\hat{T}$ be the tree constructed in Algorithm 1. We observe that Algorithm 1 is in polynomial time, and Fact 6 follows from the construction. We show in Theorem 23 that the optimal cost for $\hat{T}$ is increased by an $O(\epsilon)$ fraction compared with the optimal cost for T.

▶ **Theorem 23.** *Consider the unit demand CVRP on the tree $\hat{T}$. There exist dummy terminals and a solution* OPT_2 *visiting all of the real and the dummy terminals, such that all of the following holds:*

1. *For each component c, there are at most $\frac{2\Gamma}{\alpha} + 1$ tours visiting terminals in c;*
2. *For each component c and each tour t visiting terminals in c, the number of the terminals in c visited by t is at least $\alpha \cdot k$;*
3. *We have* $\mathrm{opt}_2 < (1 + 3\epsilon) \cdot \mathrm{opt}$, *where* opt *denotes the optimal cost for the unit demand CVRP on the tree T.*

In the rest of the section, we prove Theorem 23.

4.1 Construction of OPT_2

Consider any tour t in OPT_1. Let U denote the set of terminals visited by t.[4] We define the tour $\hat{t}$ as the minimal tour in the tree $\hat{T}$ that spans all terminals in U, see Figure 1. Let OPT_2 denote the set of the resulting tours on the tree $\hat{T}$ constructed from every tour t in OPT_1. Then OPT_2 is a feasible solution to the unit demand CVRP on $\hat{T}$.

4.2 Analysis of OPT_2

The first two properties in Theorem 23 follow from the construction and Theorem 13.

In the rest of the section, we analyze the cost of OPT_2.

▶ **Lemma 24.** *Let t denote any tour in* OPT_1. *Let $\hat{t}$ denote the corresponding tour in* OPT_2. *Then* $\mathrm{cost}(\hat{t}) \leq (1 + \epsilon) \cdot \mathrm{cost}(t)$.

[4] We assume that t is a minimal tour in T spanning all terminals in U.

Proof. We follow the notation on U from Section 4.1. Let $\mathcal{C}(U)$ denote the set of components $c \in \mathcal{C}$ that contains a (possibly spine) subtour of $\hat{t}$. Observe that the cost of $\hat{t}$ consists of the following two parts:

1. for each component $c \in \mathcal{C}(U)$, the cost of the subtour in c from the tour t; we *charge* that cost to the subtour in t;
2. for each component $c \in \mathcal{C}(U)$, the cost of the edge (r_c, z), where z denotes the father vertex of r_c in $\hat{T}$. Note that z is a critical vertex on $\hat{t}$. We analyze that cost over all components $c \in \mathcal{C}(U)$ as follows.

Let $Z \subseteq V$ denote the set of critical vertices $z \in V$ on $\hat{t}$. For any critical vertex $z \in Z$, let $Y(z)$ denote the set of edges (z, v) in the tree $\hat{T}$ such that v is a child of z and that the edge (z, v) belongs to the tour $\hat{t}$. The overall cost of the second part is the total cost of the edges in $Y(z)$ for all $z \in Z$.

Fix a critical vertex $z \in Z$. Let (z, v_1) denote the edge in $Y(z)$ such that $\mathrm{dist}(v_1)$ is minimized, breaking ties arbitrarily. From the minimality of $\mathrm{dist}(v_1)$, the z-to-v_1 path in T does not go through any component in $\mathcal{C}(U)$. From the construction, the cost of the edge (z, v_1) in $\hat{T}$ equals the cost of the z-to-v_1 path in T. It is easy to see that the z-to-v_1 path in T belongs to the tour t. Indeed, tour $\hat{t}$ traverses the edge (z, v_1) on its way to visit some terminals of U in the subtree rooted at v_1. In order to visit the corresponding terminals in T, tour t must traverse the z-to-v_1 path. We *charge* the cost of the edge (z, v_1) in $\hat{T}$ to the z-to-v_1 path in T. Next, we analyze the cost due to the other edges in $Y(z)$. Consider one such edge (z, v). From the construction, there exists $i \in [1, H_\epsilon]$, such that both $\mathrm{dist}(z)$ and $\mathrm{dist}(v)$ belong to $\big[(i-1) \cdot \tilde{D}, i \cdot \tilde{D}\big)$. Thus the cost of the z-to-v path in T equals $\mathrm{dist}(v) - \mathrm{dist}(z) < \tilde{D}$, so the extra cost in $\hat{t}$ due to the edge (z, v) is at most $2 \cdot \tilde{D}$ (for both directions). Therefore, the extra cost in $\hat{t}$ due to those $|Y(z)| - 1$ edges in $Y(z)$ is at most $2 \cdot \tilde{D} \cdot (|Y(z)| - 1)$.

Summing over all vertices $z \in Z$, and observing that all charges are to disjoint parts of t, we have

$$\mathrm{cost}(\hat{t}) \leq \mathrm{cost}(t) + 2 \cdot \tilde{D} \cdot \sum_{z \in Z} (|Y(z)| - 1). \tag{2}$$

It remains to bound $\sum_{z \in Z} (|Y(z)| - 1)$. The analysis uses the following basic fact in trees.

▶ **Fact 25.** *Let H be a tree with L leaves. For each vertex u in H, let $m(u)$ denote the number of children of u in H. Then*

$$\sum_{u \in H} (m(u) - 1) \leq L - 1.$$

We construct a tree H as follows. Starting from the tree spanning U in $\hat{T}$, we contract vertices in each component $c \in \mathcal{C}(U)$ into a single vertex; let H denote the resulting tree. It is easy to see that each leaf in H corresponds to a component $c \in \mathcal{C}(U)$ that contains at least one terminal in U (using the definition of $\mathcal{C}(U)$ and the fact that any descending component of c do not belong to $\mathcal{C}(U)$). From the second property of Theorem 23 (which follows from Theorem 13), terminals in U belong to at most $1/\alpha$ components. Thus, by Fact 25 we have

$$\sum_{z \in Z} (|Y(z)| - 1) \leq (1/\alpha) - 1.$$

Combined with Equation (2), we have

$$\mathrm{cost}(\hat{t}) - \mathrm{cost}(t) < 2 \cdot \tilde{D} \cdot (1/\alpha) = 2 \cdot \alpha \cdot \epsilon \cdot D_{\min} \cdot (1/\alpha) = 2\epsilon \cdot D_{\min},$$

using the definition of $\tilde{D}$ in Lemma 21. Since $\mathrm{cost}(t) \geq 2 \cdot D_{\min}$, the claim follows. ◀

Applying Lemma 24 on each tour t in OPT_1 and summing, we have $\mathrm{opt}_2 \leq (1+\epsilon)\cdot \mathrm{opt}_1$. By Theorem 13, $\mathrm{opt}_1 \leq (1+\epsilon)\cdot \mathrm{opt}$, thus $\mathrm{opt}_2 \leq (1+3\epsilon)\cdot \mathrm{opt}$. This completes the proof of Theorem 23.

5 Adaptive Rounding on the Subtour Demands

In this section, we prove Theorem 26. We use the adaptive rounding to show that, in a near-optimal solution, the demands of the subtours at any critical vertex are from a set of $O_\epsilon(1)$ values. This property enables us to later guess those values in polynomial time by a dynamic program.

▶ **Theorem 26.** *Let $\beta = \frac{1}{4}\cdot\epsilon^{(\frac{4}{\epsilon}+1)}$. Consider the unit demand CVRP on the tree $\hat{T}$. There exist dummy terminals and a solution OPT_3 visiting all of the real and the dummy terminals, such that all of the following holds:*

1. *For each component c, there are at most $\frac{2\Gamma}{\alpha}+1$ tours visiting terminals in c;*
2. *For each critical vertex z, there exist $\frac{1}{\beta}$ integer values in $[\alpha\cdot k, k]$ such that the demands of the subtours at the children of z are among these values;*
3. *We have $\mathrm{opt}_3 < (1+4\epsilon)\cdot \mathrm{opt}$, where opt denotes the optimal cost for the unit demand CVRP on the tree T.*

5.1 Construction of OPT_3

We construct the solution OPT_3 by modifying the solution OPT_2.

Let $I \subseteq V$ denote the set of vertices $v \in V$ that is either the root of a component or a critical vertex. Consider any vertex $v \in I$ in the bottom up order. Let $\mathrm{OPT}_2(v)$ denote the set of subtours at v in OPT_2. We construct a set $A(v)$ of subtours at v satisfying the following invariants:

- the subtours in $A(v)$ have a one-to-one correspondence with the subtours in $\mathrm{OPT}_2(v)$; and
- the demand of each subtour of $A(v)$ is at most that of the corresponding subtour in $\mathrm{OPT}_2(v)$.

The construction of $A(v)$ is according to one of the following three cases on v.

Case 1: v is the root vertex r_c of a leaf component c in $\hat{T}$

Let $A(v) = \mathrm{OPT}_2(v)$.

Case 2: v is the root vertex r_c of an internal component c in $\hat{T}$

For each subtour $a \in \mathrm{OPT}_2(v)$, if a contains a subtour at the exit vertex e_c of component c, letting t denote this subtour and t' denote the subtour in $A(e_c)$ corresponding to t, we replace the subtour t in a by the subtour t'. Let $A(v)$ be the resulting set of subtours at v.

Case 3: v is a critical vertex in $\hat{T}$

We apply the technique of the *adaptive rounding*, previously used by Jayaprakash and Salavatipour [22] in their design of a QPTAS the tree CVRP. The idea is to *round up* the demands of the subtours at the children of v so that the resulting demands are among $\frac{1}{\beta}$ values.

Let $r_1, \ldots, r_m$ be the children of v in $\hat{T}$. For each subtour $a \in \mathrm{OPT}_2(v)$ and for each $i \in [1, m]$, if a contains a subtour at r_i, letting t denote this subtour and t' denote the subtour

in $A(r_i)$ corresponding to t, we replace t in a by t'. Let $A_1(v)$ denote the resulting set of subtours at v.

Let W_v denote the set of the subtours at the children of v in $A_1(v)$, i.e., $W_v = A(r_1) \cup \cdots \cup A(r_m)$. If $|W_v| \leq \frac{1}{\beta}$, let $A(v) = A_1(v)$. In the following, we consider the non-trivial case when $|W_v| > \frac{1}{\beta}$. We sort the subtours in W_v in non-decreasing order of their demands, and partition these subtours into $\frac{1}{\beta}$ groups of equal cardinality.[5] We *round* the demands of the subtours in each group to the maximum demand in that group. The demand of a subtour is increased to the rounded value by adding *dummy* terminals at the children of v. We rearrange the subtours in W_v as follows.

- Each subtour $t \in W_v$ in the last group is discarded, i.e., detached from the subtour in $A_1(v)$ to which it belongs.
- Each subtour $t \in W_v$ in other groups is associated in a one-to-one manner to a subtour $t' \in W_v$ in the next group. Letting a (resp. a') denote the subtour in $A_1(v)$ to which t (resp. t') belongs, we detach t from a and reattach t to a'.

Let $A(v)$ be the set of the resulting subtours at v after the rearrangement for all $t \in W_v$.

For each subtour t that is discarded in the construction, we complete t into a separate tour by adding the connection (in both directions) to the depot. Let B denote the set of these newly created tours. Let $\mathrm{OPT}_3 = A(r) \cup B$.

It is easy to see that OPT_3 is a feasible solution to the unit demand CVRP, i.e., each tour in OPT_3 is connected and visits at most k terminals, and each terminal is covered by some tour in OPT_3.

5.2 Analysis of OPT_3

From the construction, in any component $c \in \mathcal{C}$, the non-spine subtours in OPT_3 are the same as those in OPT_2. From Theorem 23, we obtain the first property in Theorem 26, and in addition, each subtour at a child of a critical vertex in OPT_2 has demand at least $\alpha \cdot k$. The second property of the claim follows from the construction of OPT_3.

It remains to analyze the cost of OPT_3. Let $\Delta = \mathrm{opt}_3 - \mathrm{opt}_2$. Observe that Δ is due to adding connections to the depot to create the tours in the set B.

Fix any $i \in [1, H_\epsilon]$. Let $Z \subseteq V$ denote the set of vertices $v \in V$ such that v is the critical vertex of a maximally connected component $\tilde{\mathcal{C}} \subseteq \mathcal{C}_i$. For any $v \in Z$, we analyze the number of discarded subtours in the set W_v defined in Section 5.1. If $|W_v| \leq \frac{1}{\beta}$, there is no discarded subtour in W_v; if $|W_v| > \frac{1}{\beta}$, the number of discarded subtours in W_v is $\lceil \beta \cdot |W_v| \rceil < \beta \cdot |W_v| + 1 < 2\beta \cdot |W_v|$. Let W denote the disjoint union of W_v for all vertices $v \in Z$. Thus W contains at most $2\beta \cdot |W|$ discarded subtours. Let Δ_i denote the cost to connect the discarded subtours in W to the depot. We have

$$\Delta_i \leq 2\beta \cdot |W| \cdot 2 \cdot D_{\max} < \frac{1}{2} \cdot \epsilon^{(\frac{4}{\epsilon}+1)} \cdot |W| \cdot 2 \cdot \left(\frac{1}{\epsilon}\right)^{\frac{1}{\epsilon}-1} \cdot D_{\min} = \frac{\epsilon}{H_\epsilon} \cdot \alpha \cdot |W| \cdot D_{\min}, \quad (3)$$

where the second inequality follows from the definition of β (Theorem 26) and Definition 3, and the equality follows from the definitions of α (Theorem 13) and of H_ϵ (Lemma 21). From the second property of the claim, each subtour in W has demand at least $\alpha \cdot k$, so there are at least $\alpha \cdot |W|$ tours in OPT_3. Any tour in OPT_3 has cost at least $2 \cdot D_{\min}$, so we have

$$\mathrm{opt}_3 \geq 2 \cdot \alpha \cdot |W| \cdot D_{\min}. \quad (4)$$

[5] We add empty subtours to the first groups if needed in order to achieve equal cardinality among all groups.

From Equations (3) and (4), we have

$$\Delta_i < \frac{\epsilon}{2 \cdot H_\epsilon} \cdot \mathrm{opt}_3.$$

Summing over all integers $i \in [1, H_\epsilon]$, we have $\Delta = \sum_i \Delta_i \leq \frac{\epsilon}{2} \cdot \mathrm{opt}_3$. Thus

$$\mathrm{opt}_3 \leq \frac{2}{2-\epsilon} \cdot \mathrm{opt}_2.$$

By Theorem 23, $\mathrm{opt}_2 \leq (1+3\epsilon) \cdot \mathrm{opt}$. Therefore, $\mathrm{opt}_3 \leq (1+4\epsilon) \cdot \mathrm{opt}$. This completes the proof of Theorem 26.

References

1. Anna Adamaszek, Artur Czumaj, and Andrzej Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. *International Journal of Foundations of Computer Science*, 21(06):893–904, 2010.
2. Kemal Altinkemer and Bezalel Gavish. Heuristics for delivery problems with constant error guarantees. *Transportation Science*, 24(4):294–297, 1990.
3. S. P. Anbuudayasankar, K. Ganesh, and Sanjay Mohapatra. *Models for practical routing problems in logistics*. Springer, 2016.
4. Tetsuo Asano, Naoki Katoh, and Kazuhiro Kawashima. A new approximation algorithm for the capacitated vehicle routing problem on a tree. *Journal of Combinatorial Optimization*, 5(2):213–231, 2001.
5. Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: towards a polynomial time approximation scheme for general k. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 275–283, 1997.
6. Amariah Becker. A tight 4/3 approximation for capacitated vehicle routing in trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 116 of *LIPIcs*, pages 3:1–3:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
7. Amariah Becker, Philip N. Klein, and David Saulpic. A quasi-polynomial-time approximation scheme for vehicle routing on planar and bounded-genus graphs. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
8. Amariah Becker, Philip N. Klein, and David Saulpic. Polynomial-time approximation schemes for k-center, k-median, and capacitated vehicle routing in bounded highway dimension. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
9. Amariah Becker, Philip N. Klein, and Aaron Schild. A PTAS for bounded-capacity vehicle routing in planar graphs. In *Workshop on Algorithms and Data Structures*, pages 99–111. Springer, 2019.
10. Amariah Becker and Alice Paul. A framework for vehicle routing approximation schemes in trees. In *Workshop on Algorithms and Data Structures*, pages 112–125. Springer, 2019.
11. Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 1–14. Springer, 2021.
12. Agustín Bompadre, Moshe Dror, and James B. Orlin. Improved bounds for vehicle routing solutions. *Discrete Optimization*, 3(4):299–316, 2006.
13. Vincent Cohen-Addad, Arnold Filtser, Philip N. Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 589–600. IEEE, 2020.

14 Teodor G. Crainic and Gilbert Laporte. *Fleet management and logistics*. Springer Science & Business Media, 2012.

15 George B. Dantzig and John H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

16 Aparna Das and Claire Mathieu. A quasipolynomial time approximation scheme for Euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015.

17 Zachary Friggstad, Ramin Mousavi, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Improved approximations for CVRP with unsplittable demands. *arXiv preprint*, 2021. `arXiv:2111.08138`.

18 Bruce Golden, S. Raghavan, and Edward Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*. Springer, 2008.

19 Bruce L. Golden and Richard T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

20 Mordecai Haimovich and Alexander H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

21 Shin-ya Hamaguchi and Naoki Katoh. A capacitated vehicle routing problem on a tree. In *International Symposium on Algorithms and Computation*, pages 399–407. Springer, 1998.

22 Aditya Jayaprakash and Mohammad R. Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 877–893, 2022.

23 Michael Khachay and Roman Dubinin. PTAS for the Euclidean capacitated vehicle routing problem in $\mathbb{R}^d$. In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer, 2016.

24 Martine Labbé, Gilbert Laporte, and Hélene Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4):616–622, 1991.

25 Claire Mathieu and Hang Zhou. A tight $(1.5 + \epsilon)$-approximation for unsplittable capacitated vehicle routing on trees, 2022. `arXiv:2202.05691`.

26 Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

Graph Reconstruction from Random Subgraphs

Andrew McGregor
University of Massachusetts, Amherst, MA, USA

Rik Sengupta
University of Massachusetts, Amherst, MA, USA

Abstract

We consider the problem of reconstructing a graph G in two natural sampling models: 1) each sample corresponds to a random induced subgraph and 2) for a fixed adjacency matrix A_G for G, each sample corresponds to a random principal submatrix (i.e., a submatrix formed by deleting the same set of rows and columns) of A_G. We refer to these models as the "unordered" and "ordered" models respectively. The two models are motivated by work on the *reconstruction conjecture* in combinatorics and *trace reconstruction* in theoretical computer science. Despite the superficial similarities between the models, we show that the sample complexity of reconstruction can be exponentially different. Our main results are as follows:

- In the unordered model, we show that almost all graphs can be reconstructed with $\Theta(p^{-2}\log n)$ samples if each node is included in the random subgraph with *any* constant probability p; this is optimal. We show our upper bound extends to smaller values of p as well. In contrast, for arbitrary graphs, we show that $\exp(\Omega(n))$ samples are required for reconstruction even for 2-regular graphs. One of the key technical steps in the first result is showing that, with high probability, any subgraph isomorphism in a random graph has at most $O(\log n)$ non-fixed points.
- In the ordered model, we show that any graph with constant arboricity or degeneracy (i.e., every induced subgraph has constant average degree) can be reconstructed with $\exp(\widetilde{O}(n^{1/3}))$ samples and that arbitrary graphs can be reconstructed with $\exp(\widetilde{O}(n^{1/2}))$ samples. The results about almost all graphs in the first model carry over to the second. One of the key technical steps in the first result is showing that reconstruction of low degeneracy graphs can be reduced to learning a small number of moments of sets of the form $\{i-j : j < i, (i,j) \in E\}$ and $\{j-i : i < j, (i,j) \in E\}$ where $G = ([n], E)$ is the unknown graph.

2012 ACM Subject Classification Mathematics of computing → Probability and statistics

Keywords and phrases graph reconstruction, sample complexity, deletion channel

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.96

Category Track A: Algorithms, Complexity and Games

Funding *Andrew McGregor*: Work supported in part by NSF CCF-1934846, CCF-1908849, and CCF-1637536.

Acknowledgements We thank the reviewers for numerous helpful suggestions.

1 Introduction

We consider the problem of reconstructing an undirected graph G on n nodes in the following two natural sampling models:

- **Unordered Model**: Each node is sampled independently with probability p. The returned sample is the induced subgraph $G[A]$ where A is the set of sampled nodes. We wish to reconstruct G *up to isomorphism*.
- **Ordered Model**: Let A_G be a fixed adjacency matrix for G. Each node is sampled independently with probability p. For each node not sampled, the corresponding row and column of A_G are deleted and the returned sample corresponds to the resulting submatrix. We wish to reconstruct A_G.


We are interested in the sample complexity of reconstruction in each model, i.e., the number of samples required to reconstruct the original graph with high probability. When $p = 1/2$, the problem in the unordered model is how many uniformly random induced subgraphs are required to reconstruct a graph. The problem in the ordered model is how many random principal submatrices (i.e., submatrices formed by symmetric row/column deletions) are required to reconstruct a symmetric binary matrix with 0s on the diagonal. We will be interested in reconstructing both arbitrary graphs and random graphs (i.e., *almost all* graphs). In the ordered model, we are also interested in reconstructing low-degeneracy graphs. Degeneracy is perhaps the most natural notion of sparsity in graphs and sparsity has played an important role in other reconstruction problems such as compressed sensing [12] and trace reconstruction [20].

Unordered Model and Reconstruction from k-Decks. Reconstruction in the unordered model is closely connected to the problem of reconstruction from k-decks. Given an undirected graph G, its *k-deck* is the multiset of all $\binom{n}{k}$ induced subgraphs on k of the vertices of G. The $(n-1)$-deck is typically referred to as just the *deck*. The Reconstruction Conjecture, due to Kelly [17] and Ulam [31], asks whether there exist two different graphs with at least three nodes, that have the same deck. Bollobas [3] proved that almost all graphs can be reconstructed by taking three graphs from its deck. Furthermore, almost all sets of three graphs from the deck of G suffice to reconstruct it. Recently, Spinoza and West [30] generalized this result to the following. Let $\varepsilon > 0$ be an arbitrarily small constant and let $\ell \leq (1-\varepsilon)n/2$. Then, almost all graphs can be reconstructed from some subset of $\binom{\ell+2}{2}$ induced subgraphs from the $(n-\ell)$-deck. Note that if a k-deck is sufficient to reconstruct a graph, then one approach to bounding the sample complexity in our problem is to analyze the sample complexity of reconstructing the k-deck; this could be done by repeatedly sampling subgraphs of the appropriate size and estimating the number of copies of each such subgraph in the graph. However, the results above suggest it might be possible to reconstruct random graphs more efficiently. Many of the above results rely on showing that for almost all graphs G, any two "large" subgraphs of G are not isomorphic. In our problem we need to consider subgraphs of G that are significantly smaller and have to bound the number of isomorphisms rather than ensuring there are none.

Ordered Model and Trace Reconstruction. There is also a natural variant of the k-deck problem for matrices where now the k-deck corresponds to the multiset of all submatrices or principal submatrices. For example, reconstruction from such k-decks was studied by Kós et al. [18] and they showed that the $O(n^{2/3})$-deck was sufficient for reconstruction. Our problem in the ordered model can be thought of a stochastic variant of this problem.

Our problem is also closely related to the *trace reconstruction* problem. In this problem the goal is to reconstruct an unknown binary string $x \in \{0,1\}^n$ from independent random subsequences, or "traces", where each subsequence is formed by deleting each bit with probability $q = 1 - p$ and then concatenating the remaining bits. The trace reconstruction problem was first proposed by Batu et al. [2]. Since then, the problem attracted a lot of attention and ended up branching out into several directions and variants [4, 10, 10, 11, 13–16, 20–22, 26, 27, 29, 32]. The best upper and lower bounds known for this problem are $\exp(\widetilde{O}(n^{1/5}))$ and $\widetilde{\Omega}(n^{2/3})$ traces respectively, and were both proved recently by Chase [6, 7]. Our approach for reconstructing arbitrary graphs is very similar to the approach in [20, 27], but the result on reconstructing low-degenerary graphs requires combining those ideas with a "peeling" approach that iteratively reconstructs the neighborhood of low degree nodes, removes these nodes, and recurses on the remaining graph.

There is a natural variant of the classical string trace reconstruction problem where we have an unknown $n \times n$ binary matrix, and each trace is a sub-matrix obtained by deleting each row and column independently with probability q. The best known upper bound for this variant is $\exp(\widetilde{O}(n^{1/2}))$ traces [20]. The matrix variant is different from the string version because there are now dependencies between the bits that are deleted. The matrix variant is very closely related to the ordered model we consider; the only slight difference is that in the ordered model there is the symmetric constraint when deleting rows and columns of the adjacency matrix. There has also recently been work on tree reconstruction [4, 10, 21] that, although related somewhat to our work, primarily deals with different models of deletion channels that are only defined for rooted trees. There have been recent advances in a "smoothed" variant of the problem where each bit of the string is replaced by a uniform random bit with some probability [8]. Another variant, *coded* trace reconstruction, involving efficiently encodable codes that can be recovered despite some constant probability of edit errors, has also been studied extensively [5, 9, 10]. A generalized formulation of the problem where instead of a single unknown string, we draw a string at random from some distribution over strings and pass it through a deletion channel, has also garnered some recent interest [1, 25]. Note that in both the ordered and unordered model, the main challenge to reconstruction is that the nodes are not labelled. However, we note that are also other interesting reconstruction problems arising in the context of labelled graph, see e.g., Mossel and Ross [23], but these consider very different models from those considered here.

1.1 Our Results

1. **Unordered Model:** We show that for almost all graphs, $\Theta(p^{-2} \log n \log(1/\delta))$ traces (where in this model a trace is a randomly induced subgraph) suffice for reconstruction with probability at least $1 - \delta$, as long as the retention probability p is $\widetilde{\Omega}(1/n^{1/6})$. Note that this is optimal for the range of p considered since $\Theta(p^{-2} \log n)$ traces are required to ensure every edge appears in at least one trace.[1] In contrast, we show that reconstructing arbitrary graphs is hard: even distinguishing between a pair of 2-regular graphs may require $\exp(\Omega(n))$ traces. We show, however, that if the maximum degree of G is at most one, then it can be reconstructed with $\Theta(n)$ traces. One of the key technical steps in the first result is showing that, with high probability, any subgraph isomorphism in a random graph has at most $O(\log n)$ non-fixed points. This contrasts with a classic result by Müller [31] that shows that there are isomorphic subgraphs (where the isomorphism may contain an unbounded number of non-fixed points) of size $n/2$ but no isomorphic subgraphs of size $n(1 + \varepsilon)/2$ for any constant $\varepsilon > 0$.
2. **Ordered Model:** Our main result in the ordered model is that $\exp(\widetilde{O}(n^{1/3}))$ samples (i.e., a random principal submatrix of the adjacency matrix) suffice to reconstruct graphs of constant degeneracy, as long as the retention probability p is a constant. Recall that the *degeneracy* of a graph is the smallest $k \in \mathbb{N}$ such that every induced subgraph has a vertex of degree at most k.[2] One of the key technical steps in the first result is showing

[1] This follows by considering a graph that consists of $n/2$ vertex-disjoint edges. The edges of such a graph appear in a trace independently of one another. Hence, the probability that every edge appears in at least one of t traces is $(1 - (1 - p^2)^t)^{n/2}$ and this is at most $(1 - 2/n)^{n/2} \leq 1/e$ if $t < \log_{1-p^2}(2/n) = \Omega(\log(n)/p^2)$.

[2] Note that the degeneracy of a graph is constant factor related to the arboricity of the graph. It is a robust notion of the sparsity of a graph in that it ensures that the induced subgraph on any r nodes

that reconstruction of low degeneracy graphs can be reduced to learning a small number of moments of sets of the form $\{i-j : j < i, (i,j) \in E\}$ and $\{j-i : i < j, (i,j) \in E\}$. These moments can then be learned via an extension of methods from complex analysis that have been developed for the trace reconstruction problem. Our results represents a strong separation between sample complexity of reconstruction in the ordered and unordered models since 2-regular graphs are a special case of low degeneracy graphs. Finally, we show that any graph can be reconstructed with $\exp(\widetilde{O}(n^{1/2}))$ traces. The upper bound is established via a slight modification of a result by Krishnamurthy et al. [20]; they considered independent row/column deletions and the modification is required to deal with the fact that in our setting a row is deleted iff the corresponding column is deleted.

2 Reconstruction of Almost All Graphs in the Unordered Model

For reconstructing almost all graphs in the unordered model, the high level approach is:

1. Determine a *consistent* labeling of all nodes in the traces such that two nodes receive the same label iff they correspond to the same node in the unknown graph G. Determining this labelling will be the main technical challenge in our approach and it is especially challenging in the unordered model (compared to the ordered model) because there is no apparent ordering of the nodes that are observed in a trace.
2. If each pair of nodes of G appears together in some trace, we know whether or not there exists an edge between these two nodes. Note that

   $$T := 3p^{-2} \log n$$

   traces are sufficient to ensure this second condition with high probability.[3]

The natural question is how to determine a consistent labeling. We do this by considering all pairs of traces and for each node in the first trace of the pair we determine which node, if any, it corresponds to in the second trace. If we do this for all pairs of traces and every node of G appears in at least one trace, then for each node of G we identify all of its occurrences amongst the traces. Hence, the problem of finding a consistent labeling reduces to the problem of finding corresponding nodes in two traces.

How do we find and label corresponding nodes between two traces? Suppose one trace is the induced subgraph on a set of nodes A and the second trace is the induced subgraph on a set of nodes B. To find the corresponding nodes, a possible approach is the find the largest subgraph in $G[A]$ that is isomorphic to a subgraph in $G[B]$. If this subgraph were $G[A \cap B]$ *and* $G[A \cap B]$ were asymmetric, i.e., had no non-trivial automorphisms, then this would allow us to identify corresponding nodes. If G is random, there is indeed reason to hope that $G[A \cap B]$ is the largest common subgraph. In fact, if the probability p used in the generation of the traces is strictly greater than $1/\sqrt{2}$ we can show that this approach works exactly as stated, via a classic graph theory result by Müller [24]. His result establishes that for any constant $\varepsilon > 0$, for almost every graph G, the induced subgraphs with at least $(1+\varepsilon)n/2$ vertices have no nontrivial automorphisms and are pairwise non-isomorphic. We omit the details as we will instead prove a more general result that applies even when $p < 1/\sqrt{2}$. To

has $O(r)$ edges.

[3] Throughout this paper, we will mean a "high probability" bound to be one that holds with probability $1 - 1/\text{poly}(n)$. In this case, the high probability bound follows because the probability that there exists a pairs of nodes that does not appear up in the same trace is at most $\binom{n}{2}(1-p^2)^T \leq n^2 e^{-p^2 T}$.

prove the more general result, in the next section we will define a family of graphs called *distinctive graphs* and prove that a random graph is distinctive with high probability. In the following section, we show that it is possible to find corresponding nodes between two traces with high probability if the graph G is distinctive. This will require more than just finding the largest pair of isomorphic subgraphs, as described below.

2.1 Active Isomorphisms and Distinctive Graphs

Strengthening Müller's result to apply to subgraphs of size less than $n/2$ would give a possible approach to finding corresponding nodes when $p < 1/\sqrt{2}$. Unfortunately this is not possible: a random graph contains two isomorphic subgraphs on $n/2$ nodes with probability at least $1/2$. For example, for any pair of nodes u and v, $G[C \cup \{u\}] \cong G[C \cup \{v\}]$ where C consists of nodes that are neighbors of both u and v or neighbors of neither, and the expected size of $C \cup \{u\}$ is $n/2$. However, note that the isomorphism in this example will consist almost entirely of fixed points. And indeed, this turns out to be a constraint that we can leverage to our benefit: if we disallow isomorphisms with many fixed points, we can break the $n/2$ barrier in Müller's result.

▶ **Definition 1** (Active Isomorphisms). *Given a graph G and two subsets A, B of vertices, say A and B have an* active isomorphism *if there exists an isomorphism between $G[A]$ and $G[B]$ with no fixed points. Say G has an* active subgraph isomorphism of size M *if there exist vertex subsets A and B with $|A| = |B| = M$ with an active isomorphism.*

We next show that a random graph drawn from $\mathcal{G}(n, 1/2)$ is unlikely to have large active subgraph isomorphisms.

▶ **Theorem 2.** *For all sufficiently large n, with high probability, a random graph on n vertices has no active subgraph isomorphisms of size $M = 20 \log n$.*

Proof. Let G be a random graph drawn from $\mathcal{G}(n, 1/2)$, and let $A, B \subseteq V(G)$ be subsets of size M and let $\varphi : A \to B$ be a bijection that has no fixed points. Consider the pairs $\mathcal{A} := \{\{a_1, a_2\} : a_1, a_2 \in A, a_1 \neq a_2\}$ and $\mathcal{B} := \{\{b_1, b_2\} : b_1, b_2 \in B, b_1 \neq b_2\}$. Note that φ naturally induces a bijection φ' from $\mathcal{A}$ to $\mathcal{B}$. Each vertex pair $\mathcal{A}$ is either an edge or a non-edge; we think of these as the two *types* of pairs. The map φ is an isomorphism precisely when φ' preserves types. Note that while φ has no fixed points, φ' may have fixed points. In particular, $\{u, v\}$ is a fixed point of φ' iff $\varphi(u) = v$ and $\varphi(v) = u$. Hence, if t is the number of non-fixed points of φ', it follows that $t \geq \binom{M}{2} - M/2$. Define a directed graph $\mathcal{G}$ on the vertex set $\mathcal{A} \sqcup \mathcal{B}$ with:

- An arc from $\{u, u'\} \in \mathcal{A}$ to $\{v, v'\} \in \mathcal{B}$ if $\varphi'(\{u, u'\}) = \{v, v'\}$.
- An arc from $\{v, v'\} \in \mathcal{B}$ to $\{w, w'\} \in \mathcal{A}$ if $\{v, v'\} = \{w, w'\}$.

See Figure 1 for an example. Since each node has out-degree and in-degree at most 1 (where every node in $\mathcal{A}$ has out-degree exactly 1), $\mathcal{G}$ is a disjoint union of even cycles and odd paths that start in $\mathcal{A}$ and end in $\mathcal{B}$. For φ to be an isomorphism, all nodes in the same connected component of $\mathcal{G}$ must have the same type. If the component is a path with k nodes in $\mathcal{B}$, the probability all nodes of the component have the same type is $1/2^k$. If the component is a cycle with k nodes in $\mathcal{B}$, the probability is $1/2^{k-1}$. Hence, the probability of φ being an isomorphism is $1/2^{|\mathcal{B}|-c}$ where c is the number of cycles. Note that there are at most $|\mathcal{B}| - t$ cycles with exactly one node in $\mathcal{B}$ and the rest have at least two nodes in $\mathcal{B}$. Hence, $(|\mathcal{B}| - t) + 2(c - |\mathcal{B}| + t) \leq |\mathcal{B}|$ and this implies $|\mathcal{B}| - c \geq t/2$. Hence, the probability φ is an isomorphism is at most $1/2^{t/2}$.

The probability of A and B having an active isomorphism is bounded above by the total number of bijections from A to B with all points non-fixed, times the probability of such a bijection being an isomorphism, which is bounded above by $M! \cdot 2^{-t/2} \leq 2^{M \log M - t/2} \leq 2^{M \log M - \frac{1}{2}\binom{M}{2} + M/4}$. Taking the union bound over all subgraphs of size M still gives $\binom{n}{M}^2 \cdot 2^{M \log M - \frac{1}{2}\binom{M}{2} + M/4} \leq 2^{3M \log n + M/4 - \frac{1}{2}\binom{M}{2}} = 2^{-40 \log^2 n + 10 \log n} \leq 2^{-30 \log^2 n} = n^{-\Omega(\log n)}$. ◀

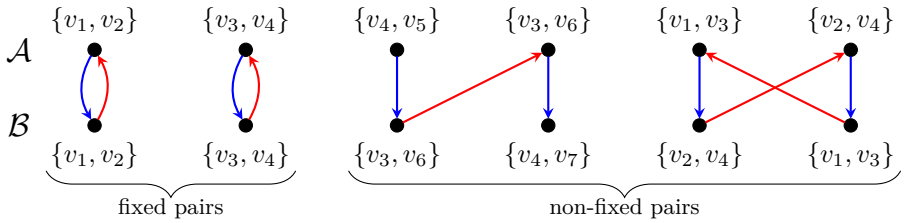


Figure 1 A construction from the proof of Theorem 2. Here, the blue arcs represent the isomorphism φ' between pairs in $\mathcal{A}$ and pairs in $\mathcal{B}$, while red arcs represent the *same* pair of vertices.

▶ **Corollary 3** (Extension to Müller). *For all sufficiently large n, with high probability, there are no two isomorphic subgraphs of a random n-vertex graph G for which the isomorphism has $20 \log n$ or more non-fixed points.*

Proof. Theorem 2 immediately implies there does not exist a subgraph isomorphism with more than M non-fixed points because a subgraph isomorphism with M non-fixed points implies the existence of a size M active subgraph isomorphism. ◀

Suppose a graph G has a subgraph H that has no nontrivial automorphisms. We can now fix a *canonical* ordering τ of $V(H)$, and define the *signature* of a vertex $v \in V(G)$ with respect to H as the length-$|V(H)|$ binary vector whose ith entry is 1 if and only if v is adjacent to the ith vertex of H in the ordering τ. For a fixed asymmetric subgraph H, we say vertices $u, v \in V(G)$ are *distinguishable with respect to H* if and only if they have distinct signatures with respect to H.

▶ **Definition 4** (Distinctive Graphs). *We say that a graph G on n vertices is* distinctive *if the following conditions are met:*

1. *Any subgraph of G on $200 \log n$ or more vertices is asymmetric.*
2. *For any two subsets $A, B \subseteq V(G)$ satisfying $G[A] \cong G[B]$, there are at most $200 \log n$ non-fixed points in the isomorphism between them.*
3. *For all but a $1/n$ fraction of subgraphs H of G with $|V(H)| \geq 200 \log n$, the vertices in $V(G)$ are pairwise distinguishable with respect to H.*

Note that the isomorphism in the second condition in the definition is well-defined and unique, because $G[A]$ and $G[B]$ of size more than $200 \log n$ are asymmetric by the first condition; similarly, the third condition is well-defined, because H is asymmetric, also by the first condition.

▶ **Theorem 5.** *Almost all graphs are distinctive.*

Proof. Consider a random graph G. The probability that any particular k-node random subgraph has a non-identity isomorphism is at most $2^{k \log k - k^2/100}$ (see e.g. Theorem 3.1 in [28]). Taking the union bound over all $\binom{n}{k}$ subgraphs of G and all possible sizes of the subgraph from k to n, the probability of G having a non-asymmetric subgraph on k or more vertices is at most $2^{\log n(1+2k)+k-k^2/100}$, which is $1/\text{poly}(n)$ for $k \geq 200 \log n$. The second follows directly from Corollary 3 above. The third is due to the following counting argument. Fix a random subgraph H of G of size at least $200 \log n$. For a fixed $v \in V(G)$, the probability of there being an edge to any given vertex of H is $1/2$, and these are independent for different vertices v. Therefore, the signature of a vertex v with respect to a fixed H corresponds to a uniformly random binary string if $v \notin H$. If $v \in H$, the string is uniformly random aside from the index corresponding to v, which is deterministically 0. The probability that two of these signatures match is at most $2^{-|V(H)|+1} \leq 2^{-200 \log n+1} \leq n^{-199}$. Therefore, by the union bound, the probability that two vertices in G have the same signature with respect to H is at most $\binom{n}{2}/n^{199} \leq 1/n^{197}$. Therefore, the expected fraction of random subgraphs H of size at least $200 \log n$ with the property that there are two distinct vertices that are indistinguishable with respect to H is at most $1/n^{197}$. The probability this fraction exceeds $1/n$ is at most $1/n^{196}$ by an application of the Markov bound. ◀

2.2 Reconstruction of Distinctive Graphs

Let G be a distinctive graph on n vertices. In this section, we will show an upper bound on the sample complexity of reconstructing G from uniformly random induced subgraphs, obtained by retaining each vertex independently with probability $p \geq 12n^{-1/6} \log^{2/3} n$. Recall that if we take $T := 3p^{-2} \log n$ samples uniformly at random, then we see each pair of vertices appear together in *some* trace.

Consider first just two such random induced subgraphs $G[A]$ and $G[B]$ where A and B are subsets of the nodes formed independently by sampling each node in G with probability p. Let H be the largest graph that appears as an induced subgraph of both $G[A]$ and $G[B]$. Let $A' \subseteq A$ and $B' \subseteq B$ be the nodes in A and B that induce H. Note that it could be that $A' = B' = A \cap B$, but while $G[A \cap B]$ is a subgraph of both $G[A]$ and $G[B]$, we do not know if it is the largest subgraph that appears in both. However, since $|A'| = |B'| \geq |A \cap B|$ and $\mathbb{E}[|A \cap B|] = p^2 n \gg 200 \log n$ by our choice of p we know $|A'| = |B'| \geq 200 \log n$ with high probability. Then, by Distinctive Property 1, we can assume that $G[A']$ and $G[B']$ are asymmetric and that there is a unique isomorphism φ between the copy of H in $G[A]$ and the copy of H in $G[B]$.

Let us now subsample the vertices in $G[A']$ with probability

$$\alpha := \frac{1}{1200T^2 \log n},$$

and call the resulting set of vertices $C' \subset A'$. Observe that the number of vertices not fixed by φ is at most $200 \log n$ by Distinctive Property 2. So by Markov's inequality, the probability that we subsample such a non-fixed point is at most $200\alpha \log n$. So with probability at least $1 - 200\alpha \log n$, the set C' consists entirely of fixed points in φ and is therefore entirely in the intersection $A \cap B$.

Let $\mathcal{D}_{C'}$ be the distribution of C', i.e., we sample A, find H, and then subsample the nodes of H. For the sake of analysis, suppose we can identify the nodes in $A \cap B$ and let C be the set formed by sampling from $A \cap B$ with probability α. Let $\mathcal{D}_C$ be the distribution of C. Both the definition of $\mathcal{D}_{C'}$ and $\mathcal{D}_C$ is with respect to some fixed A and B.

▶ **Theorem 6.** *The variational distance between $\mathcal{D}_C$ and $\mathcal{D}_{C'}$ is at most $200\alpha \log n$.*

Proof. Let $x, y \in \{0,1\}^{|(A\cap B)\cup A'|}$ be the characteristic vectors of C and C' respectively, where we have padded each of their domains up with zeros if necessary, for notational convenience. Let $\gamma_i(b) := \mathbb{P}(x_i = b)$ and $\beta_i(b) := \mathbb{P}(y_i = b)$ where $b \in \{0,1\}$. Define $S_1 := A \cap B$, and $S_2 := A'$. Then, $\ell_1(\mathcal{D}_C, \mathcal{D}_{C'})$ is

$$\begin{aligned}
&\sum_{z\in\{0,1\}^{|(A\cap B)\cup A'|}} \left|\prod_i \gamma_i(z_i) - \prod_i \beta_i(z_i)\right| \\
&\leq \sum_i \sum_{b\in\{0,1\}} \left|\gamma_i(b) - \beta_i(b)\right| \\
&= \sum_{i\in S_1\cap S_2} \sum_{b\in\{0,1\}} \left|\gamma_i(b) - \beta_i(b)\right| + \sum_{i\in S_1\setminus S_2} \sum_{b\in\{0,1\}} \left|\gamma_i(b) - \beta_i(b)\right| \\
&\quad + \sum_{i\in S_2\setminus S_1} \sum_{b\in\{0,1\}} \left|\gamma_i(b) - \beta_i(b)\right|.
\end{aligned}$$

where the first inequality follows from the fact the ℓ_1-distance between two product distributions is at most the sum of the ℓ_1 distance between the marginals.[4]

Of these, the first term vanishes, as the inner difference is zero, whereas the two other terms are bounded by 2α for each term, giving us $\ell_1(X, Y) \leq 2\alpha(|S_1 \setminus S_2| + |S_2 \setminus S_1|) \leq 2\alpha \cdot 200 \log n$. Since the variational distance is half the ℓ_1-distance, the stated result follows. ◀

▶ **Corollary 7.** *If $p \geq 12n^{-1/6}\log^{2/3} n$, then with probability at least $1 - 400\alpha \log n$ (where the probability is taken over the choice of A, B and the subsampling of A), C' satisfies the condition in distinctive property 3.*

Proof. Let S be the event that the graph we draw satisfies Distinctive Property 3. Then, by Theorem 6, we know $\mathbb{P}_{\mathcal{D}_C}(S) - \mathbb{P}_{\mathcal{D}_{C'}}(S) \leq \|\mathcal{D}_C - \mathcal{D}_{C'}\|_{TV} \leq 200\alpha \log n$. Recall that $T = 3p^{-2}\log n$ and $\alpha = 1/(1200T^2 \log n)$. Therefore,

$$\mathbb{E}[|C|] = p^2 n\alpha = \frac{p^2 n}{1200T^2 \log n} = \frac{p^2 n \cdot p^4}{1200 \cdot 9\log^2 n \cdot \log n} > 250 \log n$$

for $p \geq 12n^{-1/6}\log^{2/3} n$. By an application of the Chernoff bound, $|C| \geq 200 \log n$ with high probability. Because G is distinctive, this implies that C satisfies the condition in Distinctive Property 3. Note that for $p = \widetilde{\Omega}(n^{-1/6})$, we have $200\alpha \log n = \widetilde{\Omega}(n^{-2/3}) \gg 1/n$, and so it follows that $P_{\mathcal{D}_{C'}}(S)$ is at least $1 - 1/n - 200\alpha \log n \gg 1 - 400\alpha \log n$. So, with the stated probability, C' satisfies the property of condition 3. ◀

This gives rise to our main result, the following algorithm for reconstructing G.

▶ **Theorem 8.** *Let G be a distinctive graph on n vertices and $\delta > 0$. We can reconstruct G with probability at least $1 - \delta$ from $\Theta(p^{-2} \cdot \log n \cdot \log(1/\delta))$ traces, when the retention probability p satisfies $p = \widetilde{\Omega}(n^{-1/6})$.*

---

[4] This can be verified via induction of the number of marginals since by the triangle inequality:

$$\left|\prod_{i\geq 1}\gamma_i(z_i) - \prod_{i\geq 1}\beta_i(z_i)\right| \leq \left|\prod_{i\geq 1}\gamma_i(z_i) - \alpha_1(z_1)\prod_{i\geq 2}\beta_i(z_i)\right| + \left|\alpha_1(z_1)\prod_{i\geq 2}\beta_i(z_i) - \prod_{i\geq 1}\beta_i(z_i)\right|.$$

where the second term when summed over z equals the ℓ_1 distance between the first marginal and we apply the induction hypothesis to the first term.

Proof. The proof relies on the following observation. Suppose that for *any* two traces G_1 and G_2, and any vertices $x \in G_1$ and $y \in G_2$, we can identify whether or not x and y are the "same" in the sense of corresponding to the same original vertex in G. Now, if we have enough traces so that each vertex in the original graph G appears in at least one of them, then we can *consistently* cluster all the vertices in all the traces into n clusters, with the vertices in each cluster corresponding to the same vertex in G. These clusters give rise to a "labeling" of the nodes in the traces with labels 1 through n. If now, in addition, each *pair* of vertices in G appear together in some trace, we have a way of identifying whether there is an edge between the pair of nodes in G, and therefore we can recover the isomorphism class of G, which is equivalent to reconstructing G in the unordered setting. This corresponds to the approach highlighted in the high level description at the start of this section.

Formally, we have the following algorithm for reconstructing G given $T = 3p^{-2}\log n$ random induced subgraphs of G. For any two of these subgraphs, we can find the largest common subgraph to both of them, and then subsample from it with probability α as defined above. By Corollary 7, this subsampled subgraph satisfies the third distinctive property with high probability, and so we can use it to obtain the signatures of all vertices in $A \cup B$, enabling us to label the two sampled subgraphs consistently with respect to each other. With high probability, every pair of vertices will appear together in one of the sampled subgraphs. Therefore, as long as we have a consistent labeling of all vertices in these subgraphs with respect to each other, we will have a consistent labeling of the entire graph G, and therefore be able to reconstruct it.

The probability of this happening, by union bounding over the at most T^2 pairs of random subgraphs we generated, is at least $1 - 400T^2\alpha\log n = 1 - 400T^2\log n \cdot \frac{1}{1200T^2\log n} = 1 - \frac{1}{3} = 2/3$. We can now reduce the failure probability to δ by repeating the process $O(\log 1/\delta)$ times and taking the most commonly reconstructed graph. This requires an additional factor of $O(\log 1/\delta)$ in the number of traces. ◀

▶ **Corollary 9.** *Let n be a sufficiently large integer and $p = \widetilde{\Omega}(n^{-1/6})$. For almost all n-node graphs, $\Theta(p^{-2} \cdot \log n \cdot \log(1/\delta))$ traces suffice for reconstruction with probability at least $1 - \delta$.*

3 Reconstruction of Arbitrary Graphs in the Unordered Model

We next consider reconstructing arbitrary graphs and show that even distinguishing two fixed graphs may require $2^{\Omega(n)}$ random induced subgraphs, highlighting the vast gap between random graphs and arbitrary graphs. In fact the lower bound even applies to distinguishing between two graphs with maximum degree 2. This immediately implies a lower bound for reconstruction. Note that for any constant p, the entire graph is selected as a random subgraph with probability p^n and therefore $O(1/p^n) = 2^{O(n)}$ is a trivial upper bound on the sample complexity for full reconstruction. So the following lower bound establishes that this trivial upper bound is optimal.

▶ **Theorem 10.** *Distinguishing the cycle C_{2n} with high probability from two disjoint copies of the cycle C_n requires $2^{\Omega(n)}$ traces in the unordered model.*

Proof. Let D_1 be the distribution over subgraphs generated when the original graph is C_{2n}. Let D_1' be the distribution conditioned on the event A that we now define. Partition the vertices of C_{2n} into n pairs of "opposite" nodes (i.e. pairs of nodes at a distance exactly n from each other). Let A be the event that there exists an opposite pair in which both nodes are deleted. Note that $\Pr(A) = 1 - (1-q^2)^n$ where q is the deletion probability.

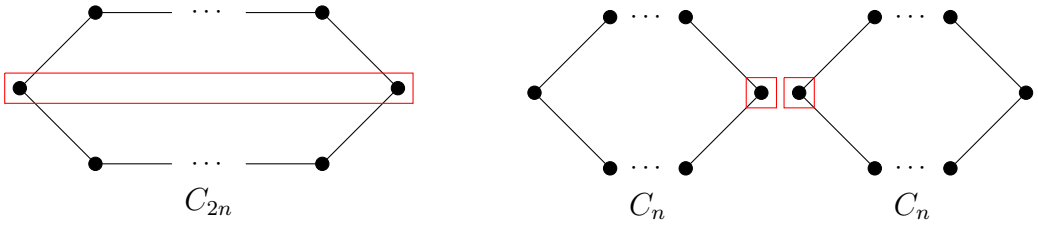


Figure 2 The construction for the proof of Theorem 10. The event A is the deletion of a pair of "opposite" nodes, such as the pair shown in red on the left; the even B is the deletion of a pair of nodes, one from each of the cycles, such as the pair shown in red on the right. Observe that these two events leave us with the same graph.

Now suppose the input graph is two copies of C_n. Let D_2 be the distribution over subgraphs generated. Let D_2' be the distribution conditioned on the following event B. Partition the vertices into pairs of nodes where each pair consists of exactly one node from each C_n. Let B be the event that there exists a pair in which both nodes are deleted. Note that $\Pr(B) = 1 - (1-q^2)^n$. Therefore, by the triangle inequality, we can bound the total variational distance between D_1 and D_2 as follows:

$$\|D_1 - D_2\|_{TV} \leq \|D_1 - D_1'\|_{TV} + \|D_1' - D_2'\|_{TV} + \|D_2' - D_2\|_{TV} = O(\Pr(\overline{A}) + \Pr(\overline{B})) = 2^{-\Omega(n)}$$

where we used the fact that $D_1' = D_2'$ and substituted $q = 1/2$. Hence, we need at least $2^{\Omega(n)}$ samples to distinguish between D_1 and D_2. ◀

It is worth remarking here that we needed degree-2 graphs for the example in Theorem 10, as evidenced by the following observation, which we state as a theorem.

▶ **Theorem 11.** *For any $\delta > 0$, a graph with maximum degree one can be reconstructed with probability at least $1 - \delta$ in $\Theta(n \log(1/\delta))$ samples.*

Proof. A graph with maximum degree one is a matching and some isolated vertices. It suffices, therefore, to learn the size $k \leq n/2$ of the matching. This size in the random subgraph is distributed as $\mathrm{Bin}(k, 1/4)$. But the unknown value k can be determined by taking the average matching observed over $t = O(n)$ traces. Specifically, let X be defined to be the average matching size. Then, because k is an integer, if we have $|X - k/4| < 1/8$, then $4X$ rounded to the nearest integer is exactly k. We have $\mathbb{E}[X] = k/4$ and $\mathbb{V}[X] \leq k/(4t)$. Hence, by Chebyshev's inequality, we have $\Pr[|X - k/4| \geq 1/8] \leq \frac{k/(4t)}{1/8^2} \leq 1/10$, where $t = cn$ for some sufficiently large constant c. We can boost the probability up as before for an additional $\log(1/\delta)$ factor. ◀

4 Reconstruction of Low Degeneracy Graphs in the Ordered Model

In this section, we turn our attention to the *ordered* model. We show that the sample complexity of reconstructing graphs with constant degeneracy is $\exp(\widetilde{O}(n^{1/3}))$, as long as the retention probability p is a constant. Recall that the *degeneracy* of an undirected graph is the smallest value d such that every induced subgraph has a node of degree at most d. Note that the degeneracy is within a factor 2 of the *arboricity*, i.e., the minimum number of forests into which its edges can be partitioned. Hence, the result applies to a natural and large class of graphs, that includes all trees, planar graphs, and indeed, all graphs of bounded treewidth.

4.1 Discussion of Challenges

Given a graph G and its adjacency matrix A, let us first consider what happens structurally to the matrix A when we pass it through a deletion channel. For instance, consider the degree sequence, viewed as the sequence of row weights of the matrix A. Of course, the number of terms in the degree sequence in general decreases (and we would expect about pn terms to survive in expectation), but observe that the terms that do survive may also change in *value*. In fact, each surviving vertex v now contributes a term to the degree sequence that is drawn from the distribution $\mathrm{Bin}(\deg(v), p)$. However, the *value* of a particular element of the degree sequence in the trace is not in general independent of its *position* within that trace. This is because the decrease in its value is not independent of the total number of neighbors that are deleted in the channel. The *shift* in position, on the other hand, is only dependent on the number of neighbors that appear *before* the corresponding row that are deleted. This is the main difficulty that our approach circumvents.

4.2 The Offset Method

We assume that we have a fixed n-vertex graph G with a fixed adjacency matrix A. For any vertex $i \in [n]$, we define the *backward* and *forward offsets*.

$$A_{\leftarrow}(i) = \{i - j : j < i, (i,j) \in E\} \qquad A_{\rightarrow}(i) = \{j - i : i < j, (i,j) \in E\}$$

For example, if

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

then we have backward offsets $A_{\leftarrow}(1) = \varnothing$, $A_{\leftarrow}(2) = \{1\}$, $A_{\leftarrow}(3) = \{1\}$, and $A_{\leftarrow}(4) = \{1, 2\}$, and forward offsets $A_{\rightarrow}(1) = \{1\}$, $A_{\rightarrow}(2) = \{1, 2\}$, $A_{\rightarrow}(3) = \{1\}$, and $A_{\rightarrow}(4) = \varnothing$. Of course, $A_{\leftarrow}(1) = A_{\rightarrow}(n) = \varnothing$ for any adjacency matrix.

Let $a_{i,k} = \sum_{x \in A_{\leftarrow}(i)} x^k$ and $b_{i,k} = \sum_{x \in A_{\rightarrow}(i)} x^k$, where by convention we set $0^0 = 0$ and $1^0 = 1$. These are the *offset moments of order* k for vertex i. Note that $a_{i,0} + b_{i,0}$ is the degree of vertex i.

We need the following result due to Krasikov and Roddity, whose proof follows from Corollaries 2.4 and 2.5 in [19].

▶ **Theorem 12** (Krasikov-Roddity, 1997). *Let $S = \{u_1, \ldots, u_d\}$ be any subset of $\{0, 1, \ldots, n-1\}$ of size d. Then, S is uniquely determined by the system*

$$u_1^r + \ldots + u_d^r = n_r, \quad r = 1, \ldots, d.$$

Now we state the core idea of our proof, which relies on the following observation about the quantities $\{a_{i,k}\}$ and $\{b_{i,k}\}$ that we just defined.

▶ **Theorem 13.** *Let G have degeneracy d. Then, the values $\{a_{i,k}\}_{i \in [n]}$ and $\{b_{i,k}\}_{i \in [n]}$ for $k = 0, \ldots, d$ uniquely determine G. In other words, reconstructing $\{a_{i,k}\}$ and $\{b_{i,k}\}$ with high probability suffices to reconstruct G with high probability.*

Proof. We use induction on the number of non-isolated nodes. The base case when the number of non-isolated nodes is 0 is trivial. Suppose the Theorem is true when there are up to t non-isolated nodes. Let G be a d-degenerate graph with $t + 1$ non-isolated nodes. There

exists a node i with degree at most d, with backward and forward offsets $A_{\leftarrow}(i)$ and $A_{\rightarrow}(i)$ respectively, and offset moments $a_{i,k}$ and $b_{i,k}$ for $k = 0, \ldots, d$. By Theorem 12, given $a_{i,k}$ and $b_{i,k}$ we can reconstruct $A_{\leftarrow}(i)$ and $A_{\rightarrow}(i)$.

The idea is to identify this vertex i and reconstruct its neighbors, and induct on the remaining graph with these edges between node i and its neighbors removed. Clearly, i is identifiable from the zeroth moment. Let G' be the graph formed by removing all edges incident to i. For all remaining vertices j, let $a'_{j,k}$ and $b'_{j,k}$ be the corresponding offset moments. Observe that $a'_{j,k}$ and $b'_{j,k}$ can be computed from i, $a_{j,k}$, $b_{j,k}$, $A_{\leftarrow}(i)$, and $A_{\rightarrow}(i)$; furthermore, G' itself is a d-degenerate graph with at most $t-1$ non-isolated nodes, and so by induction, there is no other such graph with $t-1$ non-isolated nodes with the same offset moments. We can now reconstruct G' precisely by induction, and add in the missing edges from vertex i using $A_{\leftarrow}(i)$ and $A_{\rightarrow}(i)$. ◀

▶ Remark 14. Note that the quantities $a_{i,k}$ and $b_{i,k}$ for $k = 0, 1, \ldots, d$ would suffice to learn the neighborhood of node i directly (given Theorem 12) if the degree of node i were at most d, and we could bypass induction altogether. But if the degree is strictly bigger than d, we would have to first reconstruct other parts of the graph so that, after doing so, there are at most d unknown edges incident to i. This is where the inductive argument becomes necessary.

We now state the main result of this section. We relegate the somewhat technical proof to the next subsection. This proof uses similar complex analytic techniques as in [27], which are now standard in the literature, which involve bounding the values of Littlewood polynomials on the unit circle in the complex plane. In our case, crucially, we need to understand how the moments behave, which requires additional work.

▶ **Theorem 15.** *$\{a_{i,k}\}_{i\in[n]}$ and $\{b_{i,k}\}_{i\in[n]}$ can be reconstructed with high probability using $\exp(\widetilde{O}(d^{2/3}n^{1/3}))$ traces.*

4.3 Computing Offset Moments: Proof of Theorem 15

Recall that p and q are the retention and deletion probability respectively, so that $p + q = 1$. In this subsection, we make a simplification: if $p \in (1/m, 1/(m-1)]$ for some integer m, we assume, without loss of generality, that $p = 1/m$. This makes the analysis easier. Of course, given a deletion channel corresponding to retention probability p, we can always manually simulate one with any lower retention probability, so this is a valid assumption.

Recall that the object of interest in this subsection is the kth moment, where k can go up to the degeneracy d, which suffices by Theorem 13. We denote by $\widetilde{a}_{j,k}$ and $\widetilde{b}_{j,k}$ the *observed* (i.e., sampled) values of $a_{j,k}$ and $b_{j,k}$ respectively from our traces. Consider the polynomial $\sum_{i\geq 1} b_{i,k}w^{i-1}$, and consider its expected value when we look at the trace from the deletion channel. We have, by linearity of expectation,

$$\mathbb{E}\left[\sum_{i\geq 1}\widetilde{b}_{i,k}w^{i-1}\right] = \sum_{i\geq 1} w^{i-1}\mathbb{E}[\widetilde{b}_{i,k}]. \tag{1}$$

Consider the event that the ith row of the trace comes from the jth row of the original adjacency matrix A, for some $i \leq j \leq n$. This happens precisely when exactly $i-1$ of the first $j-1$ rows are retained, and the jth row is also retained, which happens with probability $\binom{j-1}{i-1}p^i q^{j-i}$. However, the *value* of $b_{j,k}$ changes as well, which we need to account for.

To analyze this change, the first key thing to observe is that the shift in the eventual position of $b_{j,k}$ is independent of the change in its value; the former is a function of the rows *before* j that are deleted, while the latter is a function of the rows *after* j that are deleted, and we assert independence in the deletion probability of each row.

It remains to analyze the expected value of $b_{j,k}$ after A is passed through the deletion channel. Recall that $b_{j,k} = \sum_{x \in A_{\rightarrow}(j)} x^k$, the sum of kth powers of the forward offsets in row j. Now, for a given offset x in the row under consideration, it survives with probability p, and if so, ends up as the offset $1+y$, where y is a random variable that follows a $\mathrm{Bin}(x-1,p)$ distribution, since each of the $x-1$ columns between the diagonal and the original offset can be deleted with probability p. Therefore, the value of $b_{j,k}$ is

$$\sum_{x \in A_{\rightarrow}(j)} p(1+y)^k, \tag{2}$$

where $y \sim \mathrm{Bin}(x-1,p)$. Since each offset is bounded above by n, and there are at most n of them, this expression is bounded above by $O(n^{k+1})$.

We also need a lower bound: ignoring the extra factor of p, each of the $k+1$ terms in the expansion of (2) is an integer multiple of a power of y, and so the expected value of each such term is an integer multiple of m^{-k} (since the terms less than 1 are all products of the form $p^r q^s$, where $r+s \leq k$). Therefore, each nonzero term in the expectation is bounded away in absolute value from 0 by m^{-k-1}.

An exactly symmetric argument holds for the expected value of $a_{j,k}$ as well, by "indexing" in the opposite direction to $b_{j,k}$. Denote by $\Phi_a(j,k)$ and $\Phi_b(j,k)$ these expected values of $a_{j,k}$ and $b_{j,k}$ respectively, where $\Phi_a(j,k)$ and $\Phi_b(j,k)$ are between $\Omega(m^{-k-1})$ and $O(n^{k+1})$. The lower bound is necessary, as we do not want these expected values to be (exponentially) close to zero.

It follows from the arguments above that (1) reduces to (using backward offsets instead of forward ones, by a symmetric argument):

$$\begin{aligned}\mathbb{E}\left[\sum_{i\geq 1} \widetilde{a}_{i,k} w^{i-1}\right] &= \sum_{i\geq 1} w^{i-1} \sum_{j=i}^{n} \binom{j-1}{i-1} p^i q^{j-i} \Phi_a(j,k) \\ &= \sum_{j\geq 1} \Phi_a(j,k) \sum_{i=1}^{j} \binom{j-1}{i-1} p^i w^{i-1} q^{j-i}.\end{aligned}$$

With a change of variables, this becomes

$$\sum_{j\geq 1} \Phi_a(j,k) p \sum_{i'=0}^{j-1} \binom{j-1}{i'} (pw)^{i'} q^{j-1-i'} = \sum_{j\geq 1} \Phi_a(j,k) p (pw+q)^{j-1}$$

We write $pw+q=z$. To obtain a lower bound on the variational distance between two distributions coming from two different matrices, say A and A' with $\widetilde{a}_{i,k}$ and $\widetilde{a}'_{i,k}$ denoting the distribution of backward offsets from them respectively, we obtain

$$\mathbb{E}\left[\sum_{i\geq 1} (\widetilde{a}_{i,k} - \widetilde{a}'_{i,k}) w^{i-1}\right] = \sum_{j=1}^{n} (\Phi_a(j,k) - \Phi'_a(j,k)) p z^{j-1}. \tag{3}$$

The right side of equation (3) is a polynomial in z of degree less than n, where by the triangle inequality, each nonzero coefficient is upper bounded in absolute value by $O(n^{k+1})$, and lower bounded by $\Omega(m^{-k-1})$.

We now need a technical and non-trivial lemma. This is a generalization of Theorem 7 in [20], which in turns adapts a lemma in [27]. The arguments in those papers can be extended in a straightforward way to apply the same results to powers higher than 2, which is the core idea of this lemma. We use a localized lower bound on a generalized version of Littlewood polynomials, where the coefficients can be (up to) polynomially large instead of being in $\{-1, 0, 1\}$. We omit the proof here.

▶ **Lemma 16** (Generalized Littlewood polynomial bound)**.** *Let $G(z)$ be a nonzero complex polynomial in z with its degree bounded by n, integer coefficients, and each coefficient bounded in absolute value by $O(n^r)$. Then, for any fixed positive L, there is some $z^\star \in \{e^{i\theta} : -\pi/L \leq \theta \leq \pi/L\}$ such that $|G(z^\star)| \geq n^{(1-L)(r+1)}$.*

This lemma enables us to conclude the following.

▶ **Lemma 17.** *We have,*

$$\sum_{i\geq 1} \left|\mathbb{E}\left[\widetilde{a}_{i,k} - \widetilde{a}'_{i,k}\right]\right| \geq \exp\left(-\widetilde{\Omega}(k + k^{2/3}n^{1/3})\right).$$

Proof. Multiplying both sides of (3) by m^{k+1} yields a polynomial precisely of the kind described in Lemma 16 on the right hand side. Therefore, applying Lemma 16, we conclude that for some $z^\star \in \{e^{i\theta} : -\pi/L \leq \theta \leq \pi/L\}$,

$$\begin{aligned} m^{k+1}\sum_{i\geq 1} \left|\mathbb{E}\left[\widetilde{a}_{i,k} - \widetilde{a}'_{i,k}\right]\right| \cdot |(z^\star)^{i-1}| &\geq \left|\mathbb{E}\Big[\sum_{i\geq 1} m^{k+1}(\widetilde{a}_{i,k} - \widetilde{a}'_{i,k})(z^\star)^{i-1}\Big]\right| \\ &\geq n^{(1-L)(k+2+\frac{\log m}{\log n}(k+1))}. \end{aligned}$$

Noting that $|z^\star| < \exp(C_1/L^2)$ for some finite constant C_1, we now obtain

$$\begin{aligned} m^{k+1}\sum_{i\geq 1} \left|\mathbb{E}\left[\widetilde{a}_{i,k} - \widetilde{a}'_{i,k}\right]\right| &\geq n^{(1-L)(k+2+\frac{\log m}{\log n}(k+1))}\exp(-C_1 n/L^2) \\ &= \exp\left(-C_2 kL\log n - C_1 n/L^2\right) \end{aligned}$$

for some constant C_2. Here, C_2 is dependent on $\log m$, which is constant when the retention probability p is a constant. The right hand side of this equation is maximized when L is $\widetilde{O}(n^{1/3}/k^{1/3})$. We then conclude

$$\sum_{i\geq 1} \left|\mathbb{E}\left[\widetilde{a}_{i,k} - \widetilde{a}'_{i,k}\right]\right| \geq m^{-k-1}\cdot\exp\left(-\widetilde{\Omega}(k^{2/3}n^{1/3})\right) = \exp\left(-\widetilde{\Omega}(k + k^{2/3}n^{1/3})\right),$$

where $k = O(n)$ can be absorbed into the second term. ◀

To finish the proof of Theorem 15, we just need a standard union bound argument.

▶ **Theorem 18** (Folklore)**.** *Let $\mathcal{F}$ be a family of distributions where any two distributions $A, B \in \mathcal{F}$ have variational distance at least ε, for some $\varepsilon > 0$. Then, we can distinguish any member of $\mathcal{F}$ using $O(\log(|\mathcal{F}|)/\varepsilon^2)$ samples.*

Using Theorem 18 directly on the distributions defined in the proof of Theorem 15 proves that we can recover $\{a_{i,k}\}_{i\in[n]}$ in $\exp(\widetilde{O}(k^{2/3}n^{1/3}))$ traces. This holds for $\{b_{i,k}\}_{i\in[n]}$ as well, proving Theorem 15.

5 Reconstructing Arbitrary Graphs in the Ordered Model

In this section, we prove that arbitrary adjacency matrices can be reconstructed with high probability using $\exp(\widetilde{O}(n^{1/2}))$ samples in the ordered model. This is in contrast to the unordered model where we showed, in Theorem 10, that $\exp(\Omega(n))$ samples were necessary. The proof is a small modification of an existing result by Krishnamurthy et al. [20] on the problem on reconstructing arbitrary binary matrices when rows and columns are deleted independently.

▶ **Theorem 19.** *For graph reconstruction,* $\exp(O(n^{1/2}\sqrt{q\log n}/p))$ *traces suffice with high probability to recover an arbitrary adjacency matrix* $A \in \{0,1\}^{n\times n}$*, where* p *is the retention probability and* $q = 1-p$*.*

Proof Sketch. In our problem, the ith row is deleted iff the ith column is deleted, while the proof in [20] breaks when there are such dependencies. However, it is possible to make a small modification in the existing proof to handle this. The idea is to re-index the entries of the matrices such that the probability the entry in position (i',j') of the original matrix ends up in position (i,j) can be expressed conveniently in terms of two independent random variables. Let us formalize the change and sketch the rest of the approach.

For a matrix $A \in \{0,1\}^{n\times n}$, let $\widetilde{A}$ denote a matrix trace. Let us denote the (i,j)th entry of the matrix as $A_{i,j}$, for $i,j = 0,1,\ldots,n-1$, an indexing protocol we adhere to for every matrix. We restrict our attention to the entries above the diagonal, which suffices for reconstruction. For complex numbers $w_1, w_2 \in \mathbb{C}$, similar to the proof of Theorem 15, observe that

$$\begin{aligned}\mathbb{E}\left[\sum_{i,j=0}^{n-1}\widetilde{A}_{i,i+j}w_1^i w_2^j\right] &= p^2\sum_{i,j}w_1^i w_2^j \sum_{k_i\geq i,k_j\geq j} A_{k_i,k_i+k_j}\binom{k_i}{i}\binom{k_j-1}{j-1}p^i q^{k_i-i}p^{j-1}q^{k_j-j}\\ &= p^2\sum_{k_1=0,k_2=1}^{n-1} A_{k_1,k_1+k_2}(pw_1+q)^{k_1}(pw_2+q)^{k_2}\end{aligned}$$

Thus, for two adjacency matrices A, B, we have

$$\begin{aligned}\frac{1}{p^2}\mathbb{E}\left[\sum_{i,j=0}^{n-1}(\widetilde{A}_{i,i+j}-\widetilde{B}_{i,i+j})w_1^i w_2^j\right] &= \sum_{\substack{k_1=0\\k_2=1}}^{n-1}(A_{k_1,k_1+k_2}-B_{k_1,k_1+k_2})(pw_1+q)^{k_1}(pw_2+q)^{k_2}\\ &\triangleq f(z_1,z_2),\end{aligned}$$

where $z_1 = pw_1+q$ and $z_2 = pw_2+q$. The rest of the argument is identical to the proof of Krishnamurthy et al. [20]. Specifically, since all the coefficients of $f(z_1,z_2)$ are in $\{-1,0,1\}$, and the degree is $n-1$ in each variable it can be shown that for any $L>0$ there exist $z_1^\star, z_2^\star \in \{e^{i\theta} : |\theta| \leq \pi/L\}$ such that $|f(z_1^\star,z_2^\star)| \geq \exp(-C_1L^2\log n)$ for some constant C_1. If $z_1^\star = pw_1^\star+q$ and $z_2^\star = pw_2^\star+q$ then $|w_1^\star|, |w_2^\star| \leq \exp(C_2q/(Lp)^2)$ for some constant C_2.

Substituting these bounds and applying the triangle inequality gives,

$$\begin{aligned}\frac{1}{p^2}\sum_{i,j}\left|\mathbb{E}[\widetilde{A}_{i,i+j}-\widetilde{B}_{i,i+j}]\right| &\geq \frac{f(z_1^\star,z_2^\star)}{|w_1^\star|^n|w_2^\star|^n}\\ &\geq \exp\left(-C_1L^2\log n - \frac{2C_2qn}{L^2p^2}\right)\\ &\geq \exp\left(-C\frac{\sqrt{nq\log n}}{p}\right) \triangleq \varepsilon\end{aligned}$$

where the second inequality follows by optimizing for L, similar to our approach in the previous section. So if we estimate each $\mathbb{E}[\widetilde{A}_{i,i+j}]$ and $\mathbb{E}[\widetilde{B}_{i,i+j}]$ up to additive error bounded above by $p^2\varepsilon/(2n^2)$, we can distinguish between A and B. This immediately leads to the claimed bound via a union bound over all possible pairs and adjacency matrices. ◀

It seems difficult to construct lower bounds the sample complexity of reconstruction in the ordered model, beyond an obvious reduction from string trace reconstruction as follows. Given a binary string $\sigma = (x_1, \ldots, x_n)$, we can create the ordered graph G_σ on $n+1$ vertices $v_1, \ldots, v_{n+1}$ by adding edges of the form (v_i, v_{n+1}) if and only if $x_i = 1$. Clearly, there is an injective map from length-n binary strings to $(n+1)$-vertex ordered graphs. Furthermore, taking a trace of a binary string by passing it through a deletion channel with probability q of deletion corresponds exactly to taking a trace from the ordered subgraph conditioned on the vertex $n+1$ being preserved. This automatically implies a lower bound of $\widetilde{\Omega}(n^{3/2})$ samples for graph reconstruction in the ordered model due to [6]. On the surface, the ordered graph reconstruction problem seems to be fundamentally harder than the string trace reconstruction problem as well, but it is significantly harder to improve the lower bound.

6 Conclusion and Open Problems

We considered two natural graph reconstruction problems: reconstructing a graph from random induced subgraphs (the unordered model) and reconstructing an graph adjacency matrix via random symmetric submatrices (the ordered model). We showed that for almost all graphs G on n nodes, $\Theta(p^{-2}\log n)$ random induced subgraphs are necessary and sufficient to reconstruct G with high probability if each subgraph is formed by deleting each node with probability $1-p$. In contrast, we showed that there exist pairs of graphs that require $2^{\Omega(n)}$ random induced subgraphs to distinguish even when $p = 1/2$. We showed that $\exp(\widetilde{O}(n^{1/3})$ random symmetric submatrices are sufficient to construct sparse graphs (specifically, graphs with constant degeneracy or arboricity) and observed that $\exp(\widetilde{O}(n^{1/2}))$ random symmetric submatrices are sufficient to reconstruct arbitrary graphs.

Some Open Questions. In a fairly general sense, our results resolve the sample complexity of graph reconstruction in the unordered model. However, it may interesting to also consider time complexity. For example, the current approach requires isomorphism testing for an exponential number of pairs of subgraphs and it seems plausible that a more efficient approach could exist. For the ordered model, it is natural to ask whether the sample complexity of our upper bounds can be improved. Proving lower bounds for trace construction type problems is notoriously difficult and there is currently an exponential gap between the best lower and upper bounds. However, more tractable open questions include whether our results are optimal for mean-based algorithms [27], i.e., algorithms that only use the expected value of each bit in the trace. Another potential direction is whether it is possible to adapt ideas from Chase's recent work [7] to improve the exponential dependence on n or the degeneracy d. This may require substantial work in finding the two-dimensional extensions to several results. Of course, while d-degenerate graphs are a robust class of structures in themselves, it would be a natural next step to try to relax that condition altogether. Removing this constraint means denser rows of the adjacency matrix, which seems to require many more samples to effectively reconstruct, as well as novel ideas that go beyond the Krasikov-Roddity methods of set reconstruction from [19]. However, there may well be other classes of graphs that our current approach is suitable for and easily generalizable to. For instance, graphs with high girth are a natural candidate for trying to reconstruct using the techniques of this paper, as

small-diameter neighborhoods in an arbitrary high-girth graph are acyclic, and therefore the high-girth condition is akin to a "local" bounded-degeneracy condition. It seems natural to try and extend our approach to this class of graphs as well. Yet another direction to explore would be to consider lower values of p in the unordered model. Our current bounds are good enough for $p = \widetilde{\Omega}(1/n^{1/6})$, but we suspect this bound is an artifact of our approach rather than being inherent to the problem. While the majority of the related literature on similar problems typically concern themselves with constant p, which are covered by our work, exploring e.g., $p = O(\text{poly}\log n)/n$ may require new techniques.

References

1. Frank Ban, Xi Chen, Adam Freilich, Rocco A. Servedio, and Sandip Sinha. Beyond trace reconstruction: Population recovery from the deletion channel. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 745–768, 2019. doi:10.1109/FOCS.2019.00050.
2. Tugkan Batu, Sampath Kannan, Sanjeev Khanna, and Andrew McGregor. Reconstructing strings from random traces. In *Symposium on Discrete Algorithms*, 2004.
3. Bela Bollobas. Almost every graph has reconstruction number three. *Journal of Graph Theory*, 14(1):1–4, 1990.
4. Tatiana Brailovskaya and Miklós Z. Rácz. Tree trace reconstruction using subtraces. *CoRR*, abs/2102.01541, 2021. arXiv:2102.01541.
5. Joshua Brakensiek, Ray Li, and Bruce Spang. Coded trace reconstruction in a constant number of traces. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 482–493, 2020. doi:10.1109/FOCS46700.2020.00052.
6. Zachary Chase. New lower bounds for trace reconstruction. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 57(2):627–643, 2021. doi:10.1214/20-AIHP1089.
7. Zachary Chase. Separating words and trace reconstruction. In *STOC 2021: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, June 2021*, pages 21–31, 2021.
8. Xi Chen, Anindya De, Chin Ho Lee, Rocco A. Servedio, and Sandip Sinha. Polynomial-time trace reconstruction in the smoothed complexity model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 54–73. SIAM, 2021. doi:10.1137/1.9781611976465.5.
9. Mahdi Cheraghchi, Ryan Gabrys, Olgica Milenkovic, and Joao Ribeiro. Coded trace reconstruction. *IEEE Transactions on Information Theory*, PP:1–1, May 2020. doi:10.1109/TIT.2020.2996377.
10. Sami Davies, Miklos Z. Racz, and Cyrus Rashtchian. Reconstructing trees from traces. In Alina Beygelzimer and Daniel Hsu, editors, *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *Proceedings of Machine Learning Research*, pages 961–978, Phoenix, USA, 25–28 June 2019. PMLR. URL: http://proceedings.mlr.press/v99/davies19a.html.
11. Anindya De, Ryan O'Donnell, and Rocco A. Servedio. Optimal mean-based algorithms for trace reconstruction. In *Symposium on Theory of Computing*, 2017.
12. Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Birkhäuser, 2013. doi:10.1007/978-0-8176-4948-7.
13. Lisa Hartung, Nina Holden, and Yuval Peres. Trace reconstruction with varying deletion probabilities. In *Workshop on Analytic Algorithmics and Combinatorics*, 2018.
14. Nina Holden and Russell Lyons. Lower bounds for trace reconstruction. *The Annals of Applied Probability*, 30(2):503–525, 2020. doi:10.1214/19-AAP1506.
15. Nina Holden, Robin Pemantle, and Yuval Peres. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. In *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018.*, pages 1799–1840, 2018.

16 Thomas Holenstein, Michael Mitzenmacher, Rina Panigrahy, and Udi Wieder. Trace reconstruction with constant deletion probability and related results. In *Symposium on Discrete Algorithms*, 2008.

17 Paul J. Kelly. A congruence theorem for trees. *Pacific Journal of Mathematics*, 7(1):961–968, 1957. doi:pjm/1103043674.

18 Géza Kós, Péter Ligeti, and Péter Sziklai. Reconstruction of matrices from submatrices. *Mathematics of Computation*, 2009. doi:10.1090/S0025-5718-09-02210-8.

19 I. Krasikov and Y. Roditty. On a reconstruction problem for sequences. *Journal of Combinatorial Theory, Series A*, 1997.

20 Akshay Krishnamurthy, Arya Mazumdar, Andrew McGregor, and Soumyabrata Pal. Trace reconstruction: Generalized and parameterized. *IEEE Trans. Inf. Theory*, 67(6):3233–3250, 2021. doi:10.1109/TIT.2021.3066010.

21 Thomas Maranzatto and Lev Reyzin. Reconstructing arbitrary trees from traces in the tree edit distance model. *CoRR*, abs/2102.03173, 2021. arXiv:2102.03173.

22 Andrew McGregor, Eric Price, and Sofya Vorotnikova. Trace reconstruction revisited. In *European Symposium on Algorithms*, 2014.

23 Elchanan Mossel and Nathan Ross. Shotgun assembly of labeled graphs. *IEEE Transactions on Network Science and Engineering*, 6(2):145–157, 2019. doi:10.1109/TNSE.2017.2776913.

24 Vladimír Müller. Probabilistic reconstruction from subgraphs. *Commentationes Mathematicae Universitatis Carolinae*, 017(4):709–719, 1976. URL: http://eudml.org/doc/16787.

25 Shyam Narayanan. Improved algorithms for population recovery from the deletion channel. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1259–1278. SIAM, 2021. doi:10.1137/1.9781611976465.77.

26 Shyam Narayanan and Michael Ren. Circular Trace Reconstruction. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2021.18.

27 Fedor Nazarov and Yuval Peres. Trace reconstruction with $\exp(O(n^{1/3})$ samples. In *Symposium on Theory of Computing*, 2017.

28 Jaroslav Nesetril. Graph theory and combinatorics. *Fields Institute Summer Thematic Program on the Mathematics of Constraint Satisfaction*, 2011.

29 Yuval Peres and Alex Zhai. Average-case reconstruction for the deletion channel: Subpolynomially many traces suffice. In *Symposium on Foundations of Computer Science*, 2017.

30 Hannah Spinoza and Douglas West. Reconstruction from the deck of k-vertex induced subgraphs. *Journal of Graph Theory*, 90(4):497–522, 2019.

31 S. M. Ulam. *A collection of mathematical problems.* Interscience Tracts in Pure and Applied Mathematics, no. 8. Interscience Publishers, New York-London, 1960.

32 Krishnamurthy Viswanathan and Ram Swaminathan. Improved string reconstruction over insertion-deletion channels. In *Symposium on Discrete Algorithms*, 2008.

The SDP Value of Random 2CSPs

Amulya Musipatla
Carnegie Mellon University, Pittsburgh, PA, USA

Ryan O'Donnell
Carnegie Mellon University, Pittsburgh, PA, USA

Tselil Schramm
Stanford University, CA, USA

Xinyu Wu
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We consider a very wide class of models for sparse random Boolean 2CSPs; equivalently, degree-2 optimization problems over $\{\pm 1\}^n$. For each model $\mathcal{M}$, we identify the "high-probability value" $s^*_{\mathcal{M}}$ of the natural SDP relaxation (equivalently, the quantum value). That is, for all $\epsilon > 0$ we show that the SDP optimum of a random n-variable instance is (when normalized by n) in the range $(s^*_{\mathcal{M}} - \epsilon, s^*_{\mathcal{M}} + \epsilon)$ with high probability. Our class of models includes non-regular CSPs, and ones where the SDP relaxation value is strictly smaller than the spectral relaxation value.

2012 ACM Subject Classification Theory of computation → Random network models; Theory of computation → Approximation algorithms analysis

Keywords and phrases Random constraint satisfaction problems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.97

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2108.01038

Funding *Ryan O'Donnell, Xinyu Wu*: Supported by NSF grant FET-1909310.

1 Introduction

A large number of important algorithmic tasks can be construed as constraint satisfaction problems (CSPs): finding an assignment to Boolean variables to optimize the number of satisfied constraints. Almost every form of constraint optimization is NP-complete; thus one is led to questions of efficiently finding near-optimal solutions, or understanding the complexity of average-case rather than worst-case instances. Indeed, understanding the complexity of random sparse CSPs is of major importance not just in traditional algorithms theory, but also in, e.g., cryptography [25], statistical physics [27], and learning theory [14].

Suppose we fix the model $\mathcal{M}$ for a random sparse CSP on n variables (e.g., random k-SAT with a certain clause density). Then it is widely believed that there should be a constant $c^*_{\mathcal{M}}$ such that the optimal value of a random instance is $c^*_{\mathcal{M}} \pm o_{n\to\infty}(1)$ with high probability (whp). (Here we define the optimal value to mean the maximum number of simultaneously satisfiable constraints, divided by the number of variables.) Unfortunately, it is extremely difficult to prove this sort of result; indeed, it was considered a major breakthrough when Bayati, Gamarnik, and Tetali [6] established it for one of the simplest possible cases: Max-Cut on random d-regular graphs (which we will denote by $\mathcal{MC}_d$). Actually "identifying" the value $c^*_{\mathcal{M}}$ (beyond just proving its existence) is even more challenging. It is generally possible to estimate $c^*_{\mathcal{M}}$ using heuristic methods from statistical physics, but making these estimates


rigorous is beyond the reach of current methods. Taking again the example of Max-Cut on d-regular random graphs, it was only recently [16] that the value $c^*_{\mathcal{MC}_d}$ was determined up to a factor of $1 \pm o_{d\to\infty}(1)$. The value for any particular d, e.g. $c^*_{\mathcal{MC}_3}$, has yet to be established.

Returning to algorithmic questions, we can ask about the *computational feasibility* of optimally solving sparse random CSPs. There are two complementary questions to ask: given a random instance from model $\mathcal{M}$ (with presumed optimal value $c^*_{\mathcal{M}} \pm o_{n\to\infty}(1)$), can one efficiently *find* a solution achieving value $\gtrapprox c^*_{\mathcal{M}}$, and can one efficiently *certify* that every solution achieves value $\lessapprox c^*_{\mathcal{M}}$? The former question is seemingly a bit more tractable; for example, a very recent breakthrough of Montanari [29] gives an efficient algorithm for (whp) finding a cut in a random graph $\mathcal{G}(n,p)$ graph of value at least $(1-\epsilon)c^*_{\mathcal{G}(n,p)}$. On the other hand, we do not know any algorithm for efficiently certifying (whp) that a random instance has value at most $(1+\epsilon)c^*_{\mathcal{G}(n,p)}$. Indeed, it reasonable to conjecture that no such algorithm exists, leading to an example of a so-called "information-computation gap".

To bring evidence for this we can consider *semidefinite programming* (SDP), which provides efficient algorithms for certifying an upper bound on the optimal value of a CSP [20]. Indeed, it is known [35] that, under the Unique Games Conjecture, the basic SDP relaxation provides essentially optimal certificates for CSPs in the *worst case*. In this paper we in particular consider Boolean 2CSPs – more generally, optimizing a homogeneous degree-2 polynomial over the hypercube – as this is the setting where semidefinite programming is most natural. Again, for a fixed model $\mathcal{M}$ of random sparse Boolean 2CSPs, one expects there should exist a constant $s^*_{\mathcal{M}}$ such that the optimal SDP-value of an instance from $\mathcal{M}$ is whp $s^*_{\mathcal{M}} \pm o_{n\to\infty}(1)$. Philosophically, since semidefinite programming is doable in polynomial time, one may be more optimistic about proving this and explicitly identifying $s^*_{\mathcal{M}}$. Indeed, some results in this direction have recently been established.

1.1 Prior work on identifying high-probability SDP values

Let us consider the most basic case: $\mathcal{MC}_d$, Max-Cut on random d-regular graphs. For ease of notation, we will consider the equivalent problem of maximizing $\frac{1}{n}x^{\mathsf{T}}(-\boldsymbol{A})x$ over $x \in \{\pm 1\}^n$, where $\boldsymbol{A}$ is the adjacency matrix of a random n-vertex d-regular graph.[1] Although $s^*_{\mathcal{MC}_d}$, the high-probability SDP relaxation value, was pursued as early as 1987 [8] (see also [18]), it was not until 2015 that Montanari and Sen [30] established the precise result $s^*_{\mathcal{MC}_d} = 2\sqrt{d-1}$. That is, in a random d-regular graph, whp the basic SDP relaxation value [8, 37, 15, 34] for the size of the maximum cut is $(\frac{d}{4} + \sqrt{d-1} \pm o_{n\to\infty}(1))n$. Here the special number $2\sqrt{d-1}$ is the maximum eigenvalue of the d-regular infinite tree.

The proof of this result has two components: showing $\textsc{Sdp}(-\boldsymbol{A}) \geq 2\sqrt{d-1} - \epsilon$ whp, and showing $\textsc{Sdp-Dual}(-\boldsymbol{A}) \leq 2\sqrt{d-1} + \epsilon$ whp. Here $\textsc{Sdp}(A) = \max\{\langle \rho, A\rangle : \rho \succeq 0,\ \rho_{ii} = \frac{1}{n}\ \forall i\}$ denotes the "primal" SDP value on matrix A (commonly associated with the Goemans–Williamson rounding algorithm [23]), and $\textsc{Sdp-Dual}(A) = \min\{\lambda_{\max}(A+\mathrm{diag}(\zeta)) : \mathrm{avg}_i(\zeta_i) = 0\}$ denotes the (equal) "dual" SDP value on A. To show the latter bound, it is sufficient to observe that $\textsc{Sdp-Dual}(-\boldsymbol{A}) \leq \lambda_{\max}(-\boldsymbol{A})$, the "eigenvalue bound", and $\lambda_{\max}(-\boldsymbol{A}) \leq 2\sqrt{d-1} + o_n(1)$ whp by Friedman's Theorem [21]. As for lower-bounding $\textsc{Sdp}(-\boldsymbol{A})$, Montanari and Sen used the "Gaussian wave" method [19, 13, 24] to construct primal SDP solutions achieving at least $2\sqrt{d-1} - \epsilon$ (whp). The idea here is essentially to build the SDP solutions using an approximate eigenvector (of finite support) of the infinite d-regular tree achieving eigenvalue $2\sqrt{d-1} - \epsilon$; the fact that SDP constraint "$\rho_{ii} = \frac{1}{n}\ \forall i$" can be satisfied relies heavily on the regularity of the graph.

[1] Throughout this work, **boldface** is used to denote random variables.

▶ Remark 1. The Montanari–Sen result in passing establishes that the (high-probability) eigenvalue and SDP bounds coincide for random regular graphs. This is consistent with a known theme, that the two bounds tend to be the same (or nearly so) for graphs where "every vertex looks similar" (in particular, for regular graphs). This theme dates back to Delorme and Poljak [15], who showed that $\textsc{Sdp-Dual}(-A) = \lambda_{\max}(-A)$ whenever A is the adjacency matrix of a vertex-transitive graph.

Subsequently, the high-probability SDP value $s^*_{\mathcal{M}}$ was established for a few other models of random regular 2CSPs. Deshpande, Montanari, O'Donnell, Schramm, and Sen [17] showed that for $\mathcal{M} = \mathcal{NAE}3_c$ – meaning random regular instances of NAE-3SAT (not-all-equals 3Sat) with each variable participating in c clauses – we have $s^*_{\mathcal{M}} = \frac{9}{8} - \frac{3}{8} \cdot \frac{\sqrt{c-1}-\sqrt{2}}{c}$. We remark that NAE-3SAT is effectively a 2CSP, as the predicate $\mathrm{NAE}_3 : \{\pm 1\}^3 \to \{0,1\}$ may be expressed as $\frac{3}{4} - \frac{1}{4}(xy + yz + zx)$, supported on the "triangle" formed by variables x, y, z. The analysis in this paper is somewhat similar to that in [30], but with the infinite graph $\mathfrak{X} = K_3 \star K_3 \star \cdots \star K_3$ (c times) replacing the d-regular infinite tree. This $\mathfrak{X}$ is the $2c$-regular infinite "tree of triangles" depicted (partly, in the case $c = 3$) in Figure 1. More generally, [17] established the high-probability SDP value for large random (edge-signed) graphs that locally resemble $K_r \star K_r \star \cdots \star K_r$, the $(r-1)c$-regular infinite "tree of cliques K_r". (The $r = 2$ case essentially generalizes [30].) As in [30], $s^*_{\mathcal{M}}$ coincides with the (high-probability) eigenvalue bound. The upper bound on $s^*_{\mathcal{M}}$ is shown by using Bordenave's proof [9] of Friedman's Theorem for random (c, r)-biregular graphs. The lower bound on $s^*_{\mathcal{M}}$ is shown using the Gaussian wave technique, relying on the distance-regularity of the graphs $K_r \star K_r \star \cdots \star K_r$ (indeed, it is known that every infinite distance-regular graph is of this form).

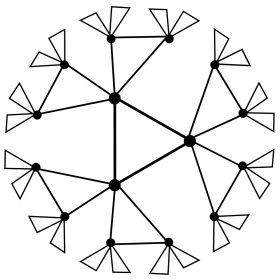


Figure 1 The 6-regular infinite graph $K_3 \star K_3 \star K_3$, modeling random 3-regular NAE3-SAT.

Mohanty, O'Donnell, and Paredes [28] generalized the preceding two results to the case of "two-eigenvalue" 2CSPs. Roughly speaking, these are 2CSPs formed by placing copies of a small weighted "constraint graph" $\mathcal{H}$ – required to have just two distinct eigenvalues – in a random regular fashion onto n vertices/variables. (This is indeed a generalization [17], as cliques have just two distinct eigenvalues.) As two-eigenvalue examples, [28] considered CSPs with the "CHSH constraint" – and its generalizations, the "Forrelation$_k$" constraints – which are important in quantum information theory [11, 1]. Here the SDP value of an instance is particularly relevant as it is precisely the optimal "quantum entangled value" of the 2CSP [12]. Once again, it is shown in [28] that the high-probability SDP and eigenvalue bounds coincide for these types of CSPs. The two-eigenvalue condition is used at a technical level in both the variant of Bordenave's theorem proven for the eigenvalue upper bound, and in the Gaussian wave construction in the SDP lower bound.

Most recently, O'Donnell and Wu [33] used the results of Bordenave and Collins [10] (a substantial generalization of [9]) to establish the high-probability *eigenvalue relaxation bound* for very wide range of random 2CSPs, encompassing all those previously mentioned: namely, any quadratic optimization problem defined by random "matrix polynomial lifts" over literals.

1.2 Our work

In this work, we establish the high-probability SDP value $s^*_{\mathcal{M}}$ for random instances of any 2CSP model $\mathcal{M}$ arising from lifting matrix polynomials (as in [33]). This generalizes all previously described work on SDP values, and covers many more cases, including random lifts of any base 2CSP (as used in certain community detection models) and random graphs modeled on any free/additive/amalgamated product. Such graphs have seen numerous applications within theoretical computer science, for example the zig-zag product in derandomization (e.g. [36]) and lifts of 2CSPs in the study of the stochastic block model (e.g [2]). At the end of this section we derive a corollary of our main theorem with applications to the latter. See Section 2 for more details and definitions, and see [33] for a thorough description of the kinds of random graph/2CSP models that can arise from matrix polynomials.

Very briefly, a matrix polynomial p is a small, explicit "recipe" for producing random n-vertex edge-weighted graphs, each of which "locally resembles" an associated *infinite* graph $\mathfrak{X}_p$. For example, $p_3(Y_1, Y_2, Y_3) \coloneqq Y_1 + Y_2 + Y_3$ is a recipe for random (edge-signed) 3-regular n-vertex graphs, and here $\mathfrak{X}_{p_3}$ is the infinite 3-regular tree. As another example, if $p_{333}(Z_{1,1}, \dots, Z_{3,3})$ denotes the following matrix polynomial –

$$\begin{pmatrix} 0 & Z_{1,1}Z^*_{1,2} + Z_{2,1}Z^*_{2,2} + Z_{3,1}Z^*_{3,2} & Z_{1,1}Z^*_{1,3} + Z_{2,1}Z^*_{2,3} + Z_{3,1}Z^*_{3,3} \\ Z_{1,2}Z^*_{1,1} + Z_{2,2}Z^*_{2,1} + Z_{3,2}Z^*_{3,1} & 0 & Z_{1,2}Z^*_{1,3} + Z_{2,2}Z^*_{2,3} + Z_{3,2}Z^*_{3,3} \\ Z_{1,3}Z^*_{1,1} + Z_{2,3}Z^*_{2,1} + Z_{3,3}Z^*_{3,1} & Z_{1,3}Z^*_{1,2} + Z_{2,3}Z^*_{2,2} + Z_{3,3}Z^*_{3,2} & 0 \end{pmatrix} \tag{1}$$

– then p_{333} is a recipe for random (edge-signed) 6-regular n-vertex graphs where every vertex participates in 3 triangles. In this case, $\mathfrak{X}_{p_{333}}$ is the infinite graph (partly) depicted in Figure 1. The Bordenave–Collins theorem [10] shows that if $\boldsymbol{A}$ is the adjacency matrix of a random *unsigned* n-vertex graph produced from a matrix polynomial p, then whp the "nontrivial" spectrum of $\boldsymbol{A}$ will be within ϵ (in Hausdorff distance) of the spectrum of $\mathfrak{X}_p$. In the course of derandomizing this theorem, O'Donnell and Wu [33] established that for random *edge-signed* graphs, the modifier "nontrivial" should be dropped. As a consequence, in the signed case one gets $\lambda_{\max}(\boldsymbol{A}) \approx \lambda_{\max}(\mathfrak{X}_p)$ up to an additive ϵ, whp; i.e., the high-probability eigenvalue bound for CSPs derived from p is precisely $\lambda_{\max}(\mathfrak{X}_p)$. We remark that for simple enough p there are formulas for $\lambda_{\max}(\mathfrak{X}_p)$; regarding our two example above, it is $2\sqrt{2}$ for $p = p_3$, and it is 5 for $p = p_{333}$. In particular, if p is a linear matrix polynomial, $\lambda_{\max}(\mathfrak{X}_p)$ may be determined numerically with the assistance of a formula of Lehner [26] (see also [22] for the case of standard random lifts of a fixed base graph).

In this paper we investigate the high-probability *SDP* value – denote it s^*_p – of a large random 2CSP (Boolean quadratic optimization problem) produced by a matrix polynomial p. Critically, our level of generality lets us consider *non-regular* random graph models, in contrast to all previous work. Because of this, we see cases in contrast to Remark 1, where (whp) the SDP value is strictly smaller than the eigenvalue relaxation bound. As a simple example, for random edge-signed $(2, 3)$-biregular graphs, the high-probability eigenvalue bound is $\sqrt{2-1} + \sqrt{3-1} = 1 + \sqrt{2} \approx 2.414$, but our work establishes that the high-probability SDP value is $\sqrt{\frac{13}{4} + 2\sqrt{2}} - \frac{1}{10} \approx 2.365$.

An essential part of our work is establishing the appropriate notion of the "SDP value" of an infinite graph $\mathfrak{X}_p$, with adjacency operator A_∞. While the eigenvalue bound $\lambda_{\max}(A_\infty)$ makes sense for the infinite-dimensional operator A_∞, the SDP relaxation does not. The definition $\textsc{Sdp}(A_\infty) = \max\{\langle \rho, A_\infty \rangle : \rho \succeq 0,\ \rho_{ii} = \frac{1}{n}\ \forall i\}$ does not make sense, since "n" is ∞. Alternatively, if one tries the normalization $\rho_{ii} = 1$, any such ρ will have infinite trace, and hence $\langle \rho, A_\infty \rangle$ may be ∞. Indeed, since the only control we have on A_∞'s "size" will be an operator norm ("∞-norm") bound, the expression $\langle \rho, A_\infty \rangle$ is only guaranteed to make sense if ρ is restricted to be trace-class (i.e., have finite "1-norm").

On the other hand, we know that the eigenvalue bound $\lambda_{\max}(A_\infty)$ is too weak, intuitively because it does not properly "rebalance" graphs $\mathfrak{X}_p$ that are not regular/vertex-transitive. The key to obtaining the correct bound is introducing a new notion, intermediate between the eigenvalue and SDP bounds, that is appropriate for graphs $\mathfrak{X}_p$ arising from matrix polynomial recipes. Although these graphs may be irregular, their definition also allows them to be viewed as *vertex-transitive* infinite graphs with $r \times r$ *matrix* edge-weights. In light of their vertex-transitivity, Remark 1 suggests that a "maximum eigenvalue"-type quantity – suitably defined for matrix-edge-weighted graphs – might serve as the sharp replacement for SDP value. We introduce such a quantity, calling it the *partitioned SDP* bound. Let G be an n-vertex graph with $r \times r$ matrices as edge-weights, and let A be its adjacency matrix, thought of as a Hermitian $n \times n$ matrix whose entries are $r \times r$ matrices. We will define

$$\textsc{PartSdp}(A) = \sup\{\langle \rho, A \rangle : \rho \succeq 0,\ \operatorname{tr}(\rho)_{ii} = \tfrac{1}{r}\}, \tag{2}$$

where here $\operatorname{tr}(\rho)$ refers to the $r \times r$ matrix obtained by summing the entries on A's main diagonal (themselves $r \times r$ matrices), and $\operatorname{tr}(\rho)_{ii}$ denotes the scalar in the (i,i)-position of $\operatorname{tr}(\rho)$. This partitioned SDP bound may indeed be regarded as intermediate between the maximum eigenvalue and the SDP value. On one hand, given a scalar-edge-weighted n-vertex graph with adjacency matrix A, we may take $r = 1$ and then it is easily seen that $\textsc{PartSdp}(A)$ coincides with $\lambda_{\max}(A)$. On the other hand, if we regard A as a 1×1 matrix and take $r = n$ (so that we a have single vertex with a self-loop weighted by all of A), then $\textsc{PartSdp}(A) = \textsc{Sdp}(A)$.

In the full version, we define $\textsc{PartSdp}(A)$ even for bounded-degree infinite graphs with $r \times r$ edge-weights. Furthermore, it has the following SDP dual:

$$\textsc{PartSdp-Dual}(A) = \inf\{\lambda_{\max}(A + \mathbb{1}_{n\times n} \otimes \operatorname{diag}(\zeta)) : \operatorname{avg}(\zeta_1, \ldots, \zeta_r) = 0\}.$$

We show in the full version that there is no SDP duality gap between $\textsc{PartSdp}(A)$ and $\textsc{PartSdp-Dual}(A)$, even in the case of infinite graphs. It is precisely the common value of $\textsc{PartSdp}(\mathfrak{X}_p)$ and $\textsc{PartSdp-Dual}(\mathfrak{X}_p)$ that is the high-probability SDP value of large random 2CSPs produced from p; our main theorem is the following:

▶ **Theorem 2.** *Let p be a matrix polynomial with $r \times r$ coefficients. Let A_∞ be the adjacency operator (with $r \times r$ entries) of the associated infinite lift $\mathfrak{X}_p$, and write $s_p^* = \textsc{PartSdp}(A_\infty) = \textsc{PartSdp-Dual}(A_\infty)$. Then for any $\epsilon, \beta > 0$ and sufficiently large n, if $\boldsymbol{A}_n$ is the adjacency matrix of a random edge-signed n-lift of p, it holds that $s_p^* - \epsilon \le \textsc{Sdp}(\boldsymbol{A}_n) \le s_p^* + \epsilon$ except with probability at most β.*

Note that $\textsc{PartSdp}(A_\infty)$ is a fixed value only dependent on the polynomial p, a finitary object.

The upper bound $\textsc{Sdp}(\boldsymbol{A}_n) \le \textsc{PartSdp-Dual}(A_\infty) + \epsilon$ in this theorem can be derived from the results of [10, 33]. Our main work is to prove the lower bound $\textsc{Sdp}(\boldsymbol{A}_n) \ge \textsc{PartSdp}(A_\infty) - \epsilon$. For this, our approach is inspired by the Gaussian Wave construction

of [30, 17] for d-regular graphs (in the random lifts model), which can be viewed as constructing a feasible $\textsc{Sdp}(\boldsymbol{A}_n)$ solution of value $\lambda_{\max}(A_\infty) - \epsilon$ using a truncated eigenfunction of A_∞. Since local neighborhoods in A_∞ look like local neighborhoods in $\boldsymbol{A}_n$ with high probability, the eigenfunction can be "pasted" almost everywhere into the graph $\boldsymbol{G}_n$, which gives an SDP solution of value near $\lambda_{\max}(A_\infty)$.

This approach runs into clear obstacles in our setting. Indeed, the raw eigenfunctions are of no use to us, as *the SDP value may be smaller than the spectral relaxation value.* Instead, we first show that there is a ρ_0 with only finitely many nonzero entries that achieves the sup in Equation (2) up to ϵ. This is effectively a finite $r \times r$ matrix-edge-weighted graph. We then show that this ρ_0 can (just as in the regular case) whp be "pasted" almost everywhere into the graph $\boldsymbol{G}_n$ defined by $\boldsymbol{A}_n$, which gives an SDP solution of value close to $\textsc{PartSdp}(A_\infty)$. The fact that $\mathfrak{X}_p$ and $\boldsymbol{G}_n$ are regarded as regular tree-like graphs with matrix edge-weights (rather than as irregular graphs with scalar edges-weights) is crucially used to show that the "pasted solution" satisfies the finite SDP's constraints "$\rho_{ii} = \frac{1}{n}\ \forall i$".

1.2.1 Application to hypothesis testing in block models

Montanari and Sen's aforementioned work [30] was motivated in part by the following question: can the basic SDP value serve as a *hypothesis test* for distinguishing Erdős-Rényi block models from random graphs with a planted partition/max-cut structure? Though polynomial-time hypothesis tests are known to exist whenever the task is information-theoretically possible [31] (even for the robust version of the problem, using a non-basic SDP [4]), the question of whether the canonical basic (Goemans–Williamson) SDP value can be used to hypothesis test is still interesting. Montanari and Sen show that the answer is affirmative for average degree $d = \Theta(1)$ large enough, but for small constant d their question remains open.

Using our results, we resolve the analogue of the question of Montanari and Sen in the large-planted-cut regime of the random-lift version of the block model, the so-called "equitable block model" [2]. The n-vertex equitable 2-community block model with internal degree a and external degree b is the distribution $\textsc{Block}(n, a, b)$ defined as a random $n/2$-lift of the 2-vertex graph with b parallel edges and a self-loops on each vertex.

▶ **Corollary 3.** *For any $\epsilon, \beta > 0$ and sufficiently large n, if $\boldsymbol{G}_n \sim \textsc{Block}(n, a, b)$ and $\boldsymbol{A}_n$ is the adjacency matrix of $\boldsymbol{G}_n$, then*

$$\left|\textsc{Sdp}(-\boldsymbol{A}_n) - \max(b - a,\ 2\sqrt{a + b - 1})\right| \leq \epsilon$$

except with probability at most β.

The bound $b - a$ is achieved by the integral cut which partitions the vertices according to their preimage in the 2-vertex graph. The upper bound $\textsc{Sdp}(-\boldsymbol{A}_n) \leq \max(b - a, 2\sqrt{a + b - 1}) + \epsilon$ follows from the eigenvalue bound on $-\boldsymbol{A}_n$. As above, what is new is our lower bound which shows that $\textsc{Sdp}(-\boldsymbol{A}_n) \geq 2\sqrt{a + b - 1} - \epsilon$.

In an $(a + b)$-regular random graph, the result of Montanari and Sen [30] (see also [3]) implies that $\textsc{Sdp}(-\boldsymbol{A}_n) = 2\sqrt{a + b - 1} \pm \epsilon$ with high probability. Hence, Corollary 3 proves that the basic (Goemans–Williamson) SDP value is the same in both models and so does not furnish a hypothesis testing algorithm when $b - a < 2\sqrt{a + b - 1}$. This is consistent with the prediction of [2], who conjecture that when $b - a < 2\sqrt{a + b - 1}$, no polynomial-time algorithm can robustly (under the addition of random noise) hypothesis test between random regular graphs and the equitable block model.[2] In [2], the authors give a lower bound for a different SDP relaxation, but they do not characterize the basic SDP value.

[2] We remark that in the Erdős-Rényi version of the 2-community block model, the threshold for information-

2 Preliminaries

To preface the following definitions and concepts, we remark that our "real" interest is in graphs/2CSPs with real scalar weights. The introduction of matrix edge-weights facilitates two things: helping us define a wide variety of interesting scalar-weighted graphs via matrix polynomial lifts; and, facilitating the definition of PartSdp$(\cdot)$, which we use to bound the SDP relaxation value of the associated 2CSPs. Our use of potentially complex matrices is also not really essential; we kept them because prior work that we utilize ([10], the tools in Section 3) is stated in terms of complex matrices. However the reader will not lose anything by assuming that all Hermitian matrices are in fact symmetric real matrices.

2.1 Matrix-weighted graphs

In the following definitions, we'll restrict attention to graphs with at-most-countable vertex sets V and bounded degree. We also often use bra-ket notation, with $(|v\rangle)_{v\in V}$ denoting the standard orthonormal basis for the complex vector space $\ell_2(V)$.

▶ **Definition 4** (Matrix-weighted graph). *Fix any $r \in \mathbb{N}^+$. A* matrix-weighted graph *will refer to a directed simple graph $G = (V, E)$ with self-loops allowed, in which each directed edge e has an associated* weight $a_e \in \mathbb{C}^{r\times r}$. *If $(v, w) \in E \implies (w, v) \in E$ and $a_{(w,v)} = a^*_{(v,w)}$, we say that G is an* undirected *matrix-weighted graph. The* adjacency matrix *of G is the operator A, acting on $\ell_2(V) \otimes \mathbb{C}^r$, given by*

$$\sum_{(v,w)\in E} |w\rangle\langle v| \otimes a_{(v,w)}.$$

It can be helpful to think to think of A in matrix form, as a $|V| \times |V|$ matrix whose entries are themselves $r \times r$ edge-weight matrices. Note that if G is undirected if and only if A is self-adjoint, $A = A^*$.

▶ **Definition 5** (Extension of a matrix-weighted adjacency matrix). *Given a $|V| \times |V|$ matrix A with $r \times r$ entries, we may also view it as a $|V|r \times |V|r$ matrix with scalar entries. When we wish to explicitly call attention to the distinction, we will call the latter matrix the* extension *of A, and denote it by $\widetilde{A}$.*

2.2 Matrix polynomials

▶ **Definition 6** (Matrix polynomial). *Let $Y_1, \dots, Y_d$ be formal indeterminates that are their own inverses, and let $Z_1, \dots, Z_e$ be formal indeterminates with formal inverses $Z_1^*, \dots, Z_e^*$. For a fixed r, we define a* matrix polynomial *to be a formal noncommutative polynomial p over the indeterminates $Y_1, \dots, Z_e^*$, with coefficients in $\mathbb{C}^{r\times r}$. In particular, we may write*

$$p = \sum_w a_w w,$$

where the sum is over words w on the alphabet of indeterminates, each a_w is in $\mathbb{C}^{r\times r}$, and only finitely many a_w are nonzero. Here we call a word reduced *if it has no adjacent Y_iY_i or $Z_iZ_i^*$ pairs. We will denote the empty word by $\mathbb{1}$.*

theoretic and computational distinguishability coincide. However, in the equitable case, the models are always information-theoretically distinguishable, since in Block(n, a, b) there is always an integral solution of value exactly $b - a$, and this will not occur in the d-regular case with high probability.

As we will shortly describe, we will always be considering substituting unitary operators for the Z_i's, and unitary involution operators for the Y_i's. Thus we can think of Z_i^* as both the inverse *and* the "adjoint" of indeterminate Z_i, and similarly we think of $Y_i^* = Y_i$.

▶ **Definition 7** (Adjoint of a polynomial)**.** *Given a matrix polynomial $p = \sum_w a_w w$ as above, we define its* adjoint *to be*

$$p^* = \sum_w a_w^* w^*,$$

where a^* is the usual adjoint of $a \in \mathbb{C}^{r\times r}$, and w^* is the adjointed reverse of w. That is, if $w = w_1 \cdots w_k$ then $w^* = w_k^* \cdots w_1^*$, where $\mathbb{1}^* = \mathbb{1}$, $Y_i^* = Y_i$, and $Z_i^{**} = Z_i$. We say p is self-adjoint *if $p^* = p$ formally.*

Note that in any self-adjoint polynomial, some terms will be self-adjoint, and others will come in self-adjoint pairs. In this work, we will only be considering self-adjoint polynomials.

2.3 Lifts of matrix polynomials

▶ **Definition 8** (n-lift)**.** *Given a matrix polynomial over the indeterminates $Y_1, \ldots, Z_e^*$, we define an n-*lift *to be a sequence $\mathcal{L} = (M_1, \ldots, M_d, P_1, \ldots, P_e)$ of $n \times n$ matrices, where each P_i is a signed permutation matrix and each M_i is a signed matching matrix.*[3] *A* random n-lift $\boldsymbol{\mathcal{L}} = (\boldsymbol{M}_1, \ldots, \boldsymbol{M}_d, \boldsymbol{P}_1, \ldots, \boldsymbol{P}_e)$ *is one where the matrices are chosen independently and uniformly at random. A* random unsigned n-lift $\boldsymbol{\mathcal{L}} = (\boldsymbol{M}_1, \ldots, \boldsymbol{M}_d, \boldsymbol{P}_1, \ldots, \boldsymbol{P}_e)$ *is one where the permutation and matching matrices are chosen independently and uniformly at random, and all the signs are $+1$.*

▶ **Definition 9** (Evaluation/substitution of lifts.)**.** *Given an n-lift $\mathcal{L}$ and a word w, we define $\mathcal{L}^w$ to be the $n \times n$ operator obtained by substituting appropriately into w: namely, $Y_i = M_i$, $Z_i = P_i$, and $Z_i^* = P_i^*$ for each i (and substituting the empty word with the $n \times n$ identity operator). Given also a matrix polynomial $p = \sum_w a_w w$, we define the* evaluation of p at $\mathcal{L}$ *to be the following operator on $\mathbb{C}^n \otimes \mathbb{C}^r$:*[4]

$$p(\mathcal{L}) = \sum_w \mathcal{L}^w \otimes a_w.$$

▶ Remark 10. Note that each P_i is unitary and each M_i a unitary involution (as promised), so $p^*(\mathcal{L}) = p(\mathcal{L})^*$. Thus $p(\mathcal{L})$ is a self-adjoint operator whenever p^* is a self-adjoint polynomial. In this case we also have that $p(\mathcal{L})$ may be viewed as the adjacency matrix of an undirected graph on vertex set $[n]$ with $r \times r$ edge-weights.

Note that the evaluation $p(\mathcal{L})$ of a matrix polynomial may be viewed as the adjacency matrix of a undirected graph on $[n]$ with $r \times r$ edge-weights; or, its extension may be viewed as the adjacency matrix of an undirected graph on $[n] \times [r]$ with scalar edge-weights. In this way, each fixed matrix polynomial p, when applied to a random lift, gives rise to a random (undirected, scalar-weighted) graph model.

---

[3] A signed matching matrix is the adjacency matrix of a perfect matching with ± 1 edge-signs. If $d > 0$ then we must restrict to even n.

[4] Note that coefficients a_w are written on the left in p, as is conventional, but we take the tensor/Kronecker product on the right so that the matrix form of $p(\mathcal{L})$ may be more naturally regarded as an $n \times n$ matrix with $r \times r$ entries.

▶ **Example 11.** *A simple example is the matrix polynomial*

$$p(Y_1, Y_2, Y_3) = Y_1 + Y_2 + Y_3.$$

Here $r = 1$ and each coefficient is just the scalar 1. This p gives rise to a model of random edge-signed 3-regular graphs on $[n]$.

By moving to actual matrix coefficients with $r > 1$, one can get the random (signed) graph model given by randomly n-lifting any base r-vertex graph H.

▶ **Example 12.** *As a simple example,*

$$p(Z_1, Z_2, Z_3) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} Z_1 + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} Z_1^* + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} Z_2 + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} Z_2^* + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} Z_3 + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} Z_3^*$$

is the recipe for random 3-regular (edge-signed) $(n + n)$-vertex bipartite graphs. The reader may like to view this as a 2×2 matrix of polynomials,

$$p(Z_1, Z_2, Z_3) = \begin{pmatrix} 0 & Z_1 + Z_2 + Z_3 \\ Z_1^* + Z_2^* + Z_3^* & 0 \end{pmatrix},$$

but recall that we actually Kronecker-product the coefficient matrices "on the other side". So rather than as a 2×2 block-matrix with $n \times n$ blocks, we think of the resulting adjacency matrix as an $n \times n$ block-matrix with 2×2 blocks; equivalently, an n-vertex graph with 2×2 matrix edge-weights.

▶ **Example 13.** *The matrix polynomial p_{333} mentioned in (1) gives an example of a* nonlinear *polynomial with matrix coefficients. Again, we wrote it there as a 3×3 matrix of polynomials for compactness, but for analysis purposes we will view it as a degree-2 polynomial with 3×3 coefficients.*

▶ **Definition 14** (∞-lift)**.** *Formally, we extend Definition 8 to the case of $n = \infty$ as follows. Let V_∞ denote the free product of groups $\mathbb{Z}_2^{\star d} \star \mathbb{Z}^{\star e}$, with its components generated by $g_1, \dots, g_d$, $h_1, h_1^{-1}, \dots, h_e, h_e^{-1}$. Thus the elements of V_∞ are in one-to-one correspondence with the reduced words over indeterminates $Y_1, \dots, Z_e^*$. The generators $g_1, \dots, g_d, h_1, \dots, h_e$ act as permutations on V_∞ by left-multiplication, with the first d in fact being matchings. We write $\sigma_1, \dots, \sigma_{d+e}$ for these permutations, and we also identify them with their associated permutation operators on $\ell_2(V_\infty)$. Finally, we write $\mathfrak{L}_\infty = (\sigma_0, \dots, \sigma_{d+2e})$ for* "the" ∞-lift *associated to p. (Note that this lift is "unsigned".)*

▶ **Definition 15** (Evaluation at the ∞-lift, and $\mathfrak{X}_p$.)**.** *The evaluation of a matrix polynomial p at the infinity lift $\mathcal{L}_\infty$ is now defined just as in Definition 9; the resulting operator $p(\mathcal{L}_\infty)$ operates on $\ell_2(V_\infty) \otimes \mathbb{C}^r$. We may think of the result as a matrix-weighted graph on vertex set V_∞, and we will sometimes denote this graph by $\mathfrak{X}_p$. When p is understood, we often write $A_\infty = p(\mathcal{L}_\infty)$ for the adjacency operator of $\mathfrak{X}_p$, which can be thought of as an infinite matrix with rows/columns indexed by V_∞ and entries from $\mathbb{C}^{r \times r}$, or as its "extension" $\widetilde{A}_\infty$, an infinite matrix with rows/columns indexed by $V_\infty \times [r]$ and scalar entries.*

▶ **Example 16.** *For the polynomial $p = Y_1 + \cdots + Y_d$, the corresponding graph $\mathfrak{X}_p$ is the infinite (unweighted) d-regular tree.*

We may now state a theorem which is essentially the main result ("Theorem 2") of [10]. The small difference is that our notion of random n-lifts, which includes ± 1 *signs* on the matchings/permutations, lets one eliminate mention of "trivial" eigenvalues (see [33, Thms. 1.9, 10.10]).

▶ **Theorem 17.** *Let p be a self-adjoint matrix polynomial with coefficients from $\mathbb{C}^{r\times r}$ on indeterminates $Y_1, \dots, Z_e^*$. Then for all $\epsilon, \beta > 0$ and sufficiently large n, the following holds:*

Let $\boldsymbol{A}_n = p(\boldsymbol{\mathcal{L}}_n)$, where $\boldsymbol{\mathcal{L}}_n$ is a random n-lift, and let $A_\infty = p(\mathcal{L}_\infty)$. Then except with probability at most β, the spectra $\operatorname{spec}(\boldsymbol{A}_n)$ and $\operatorname{spec}(A_\infty)$ are at Hausdorff distance at most ϵ

2.4 Random lifts as optimization problems

Given a Hermitian (i.e., self-adjoint) matrix $A \in \mathbb{C}^{n\times n}$, we are interested in the task of maximizing $x^\top Ax$ over all Boolean vectors $x \in \{\pm 1\}^n$. (Since A is Hermitian, the quantity $x^\top Ax$ is always real, so this maximization problem makes sense.) This is the same as maximizing the homogeneous degree-2 (commutative) polynomial $\sum_{i,j} A_{ij}x_ix_j$ over $x \in \{\pm 1\}^n$, and it is also essentially the same task as the Max-Cut problem on (scalar-)weighted undirected graphs. More precisely, if G is a weighted graph on vertex set $[n]$ with adjacency matrix A, then G's maximum cut is indicated by the $x \in \{\pm 1\}^n$ that maximizes $x^\top(-A)x$. For the sake of scaling we will also include a factor of $\frac{1}{n}$ in this optimization problem, leading to the following definition:

▶ **Definition 18** (Optimal value)**.** *Given a Hermitian matrix $A \in \mathbb{C}^{n\times n}$, we define*

$$\textsc{Opt}(A) = \sup_{x\in\{\pm1\}^n} \left\{\tfrac{1}{n} x^{\mathsf{T}} Ax\right\} = \sup_{x\in\left\{\pm\frac{1}{\sqrt{n}}\right\}^n} \left\{x^{\mathsf{T}} Ax\right\}.$$

(For finite-dimensional A, the sup*s and* inf*s mentioned in this section are all achieved.)*

We remark that

$$x^{\mathsf{T}} Ax = \operatorname{tr}(x^{\mathsf{T}} Ax) = \operatorname{tr}(xx^\top A) = \langle xx^\top, A\rangle,$$

where we use the notation $\langle B, C\rangle = \operatorname{tr}(BC)$. Thus we also have

$$\textsc{Opt}(A) = \sup_{\rho\in \mathrm{Cut}_n} \left\{\tfrac{1}{n}\langle \rho, A\rangle\right\},$$

where Cut_n is the "cut polytope", the convex hull of all matrices of the form xx^{T} for $x \in \{\pm 1\}^n$. (Since $\langle \rho, A\rangle$ is linear in ρ, maximizing over the convex hull is the same as maximizing over the extreme points, which are just those matrices of the form xx^{T}.)

The above optimization problem has a natural relaxation: maximizing $\frac{1}{n}x^{\mathsf{T}} Ax$ over all *unit* vectors x. This leads to the following efficiently computable upper bound on $\textsc{Opt}(A)$:

▶ **Definition 19** (Eigenvalue bound)**.** *Given a Hermitian matrix $A \in \mathbb{C}^{n\times n}$, we define the* eigenvalue bound *to be*

$$\textsc{Eig}(A) = \sup\{\langle \rho, A\rangle : \rho \succeq 0,\ \operatorname{tr}(\rho) = 1\},$$

where here $\rho \succeq 0$ denotes that ρ is (Hermitian and) positive semidefinite.

The matrices ρ being optimized over in $\textsc{Eig}(A)$ are known as *density matrices*; i.e., $\textsc{Eig}(A)$ is the maximal inner product between A and any density matrix. Note that if $\varrho \in \mathrm{Cut}_n$, then $\rho = \frac{1}{n}\varrho$ is a density matrix. Thus, $\textsc{Eig}(A)$ is a relaxation of $\textsc{Opt}(A)$, or in other words, $\textsc{Opt}(A) \le \textsc{Eig}(A)$.

The set of density matrices is convex, and it's well known that its extreme points are all the rank-1 density matrices; i.e., those ρ of the form xx^{T} for $x \in \mathbb{C}^n$ with $\|x\|_2^2 = 1$. Thus in $\textsc{Eig}(A)$ it is equivalent to just maximize over these extreme points:

$$\textsc{Eig}(A) = \sup_{\substack{x\in\mathbb{C}^n\\ \|x\|_2^2=1}} \left\{\langle xx^{\mathsf{T}}, A\rangle\right\} = \sup_{\substack{x\in\mathbb{C}^n\\ \|x\|_2^2=1}} \left\{x^{\mathsf{T}} Ax\right\}.$$

From this formula we see that $\textsc{Eig}(A)$ is also equal to $\lambda_{\max}(A)$, the maximum eigenvalue of A; hence the terminology "eigenvalue bound". One may also think of $\textsc{Eig}(A)$ and $\lambda_{\max}(A)$ as SDP duals of one another.

We now mention another well known, tighter, upper bound on $\textsc{Opt}(A)$.

▶ **Definition 20** (Basic SDP bound)**.** *Given a Hermitian matrix $A \in \mathbb{C}^{n\times n}$, the* basic SDP bound *is defined to be*

$$\textsc{Sdp}(A) = \sup\{\langle \rho, A\rangle : \rho \succeq 0,\ \rho_{ii} = \tfrac{1}{n}, \forall i\}.$$

Recall that an $n \times n$ matrix ϱ is a *correlation matrix* [38] if it is PSD and has all diagonal entries equal to 1. Thus $\textsc{Sdp}(A)$ is equivalently maximizing $\frac{1}{n}\langle \varrho, A\rangle$ over all correlation matrices ϱ. We also note that any cut matrix is a correlation matrix, and any correlation matrix is a density matrix, hence so

$$\textsc{Opt}(A) \le \textsc{Sdp}(A) \le \textsc{Eig}(A).$$

▶ **Definition 21** (Dual SDP bound)**.** *The semidefinite dual of $\textsc{Sdp}(A)$ is the following [15]:*

$$\textsc{Sdp-Dual}(A) = \inf_{\substack{\zeta\in\mathbb{R}^n \\ \mathrm{avg}(\zeta_1,\dots,\zeta_n)=0}} \{\lambda_{\max}(A + \mathrm{diag}(\zeta))\}.$$

Despite the fact that the usual "Slater condition" for strong SDP duality fails in this case (because the set of correlation matrices isn't full-dimensional), one can still show [34] that $\textsc{Sdp}(A) = \textsc{Sdp-Dual}(A)$ indeed holds for finite-dimensional A.

▶ Remark 22. In this work we frequently consider matrix-weighted graphs with adjacency matrices A, thought of as $n\times n$ matrices with entries from $\mathbb{C}^{r\times r}$. For such matrices, whenever we write $\textsc{Opt}(A)$, we mean $\textsc{Opt}(\widetilde{A})$ for the $nr \times nr$ "extension" matrix $\widetilde{A}$ (see Definition 5), and similarly for $\textsc{Eig}(A)$, $\lambda_{\max}(A)$, $\textsc{Sdp}(A)$, $\textsc{Sdp-Dual}(A)$.

As mentioned in Section 1, the eigenvalue bound $\lambda_{\max}(A)$ makes sense when A is the adjacency matrix of an infinite graph (with bounded degree). However $\textsc{Sdp}(A)$ does not extend to the infinite case, as the number "n" appearing in its definition is not finite. On the other hand, we now introduce a new, intermediate, "maximum eigenvalue-like" bound that is appropriate for matrix-weighted graphs. This is the "partitioned SDP bound" appearing in the statement of our main Theorem 2. In the following Section 3, we will show that it generalizes well to the case of infinite graphs.

▶ **Definition 23** (Partitioned SDP bound)**.** *Let A be an $n \times n$ Hermitian matrix with entries from $\mathbb{C}^{r\times r}$. We define its* partitioned SDP bound *to be*

$$\textsc{PartSdp}(A) = \sup\{\langle \rho, A\rangle : \rho \succeq 0,\ \mathrm{tr}(\rho) \in \tfrac{1}{r}\mathrm{Corr}_r\},$$

where:

- *the matrices ρ are also thought of as $n \times n$ matrices with entries from $\mathbb{C}^{r\times r}$;*
- *$\langle \rho, A\rangle$ is interpreted as $\langle \widetilde{\rho}, \widetilde{A}\rangle$;*
- *$\mathrm{tr}(\rho)$ denotes the sum of the diagonal entries of ρ, which is an $r \times r$ matrix;*
- *Corr_r is the set of $r \times r$ correlation matrices;*
- *in other words, the final condition is that $\mathrm{tr}(\rho)_{ii} = \frac{1}{r}$ for all $i \in [r]$.*

▶ Remark 24. As mentioned, the partitioned SDP bound can be viewed as "intermediate" between the eigenvalue bound and the SDP bound. To explain this, suppose A is an $n \times n$ Hermitian matrix. On one hand, we can regard A as an $n \times n$ matrix with 1×1 matrix entries ($r = 1$); in this viewpoint, $\textsc{PartSdp}(A) = \textsc{Eig}(A)$. On the other hand, we can regard A as a 1×1 matrix with a single $n \times n$ matrix entry ($r = n$); in this viewpoint, $\textsc{PartSdp}(A) = \textsc{Sdp}(A)$.

It is easy to see that the partitioned SDP bound indeed has an SDP formulation, and we now state its SDP dual:

▶ **Definition 25.** *The SDP dual of* $\textsc{PartSdp}(A)$ *is the following:*

$$\textsc{PartSdp-Dual}(A) = \inf_{\substack{\zeta \in \mathbb{R}^r \\ \mathrm{avg}(\zeta_1,\ldots,\zeta_r)=0}} \{\lambda_{\max}(A + \mathbb{1}_{n\times n} \otimes \mathrm{diag}(\zeta))\}.$$

Weak SDP duality, $\textsc{PartSdp}(A) \le \textsc{PartSdp-Dual}(A)$, holds as always, but again it is not obvious that strong SDP duality holds. In fact, not only does strong duality hold, it even holds in the case of *infinite* matrices A. This fact is crucial for our work, and proving it the subject of the upcoming technical section.

3 The infinite SDPs

This technical section is deferred to the full version and we only state some results here. First, we show that strong duality $\textsc{PartSdp}(A) = \textsc{PartSdp-Dual}(A)$ holds, even for infinite matrices A with $r \times r$ entries. Even in the finite case this is not trivial, as the feasible region for the SDP $\textsc{PartSdp}(A)$ is not full-dimensional, and hence the Slater condition ensuring strong duality does not apply. The infinite case involves some additional technical considerations, needed so that we may eventually apply the strong duality theorem for conic linear programming of Bonnans and Shapiro [7, Thm. 2.187]. Second, we show that in the optimization problem $\textsc{PartSdp}(A)$, values arbitrarily close to the optimum can be achieved by matrices ρ of finite support (i.e., with only finitely many nonzero entries). Indeed (though we don't need this fact), these finite-rank ρ need only have rank at most r. This fact is familiar from the case of $r = 1$, where the optimizer in the eigenvalue bound Definition 19 is achieved by a ρ of rank 1 (namely $|\psi\rangle\langle\psi|$ for any maximum eigenvector $|\psi\rangle$). Finally, we consolidate all these results into a theorem statement suitable for use with graphs produced by infinite lifts of matrix polynomials.

3.1 SDP duality for matrix edge-weighted graphs

Let V_∞ be a countable set of nodes and let G_∞ be a bounded-degree graph on V_∞ with matrix edge-weights from $\mathbb{C}^{r\times r}$. Let A_∞ be the adjacency operator for G_∞, acting on $\ell_2(V_\infty) \otimes \mathbb{C}^r$ and assumed self-adjoint; we may think of it as an infinite matrix with rows and columns indexed by V_∞, and with entries from $\mathbb{C}^{r\times r}$.

For any $\epsilon > 0$, there exists:

- $\hat{\zeta} \in \mathbb{R}^r$ with $\mathrm{avg}_j(\hat{\zeta}_j) = 0$;
- a finite subset $F \subset V_\infty$;
- a PSD matrix ρ with rows/columns indexed by V_∞ and entries from $\mathbb{C}^{r\times r}$, supported on the rows/columns F, with

$$\mathrm{tr}(\rho)_{jj} = \tfrac{1}{r}, \quad j = 1 \ldots r;$$

such that for

$$\hat{A} = \widetilde{A}_\infty + \mathbb{1}_{V_\infty} \otimes \mathrm{diag}(\hat{\zeta}),$$

we have

$$s^* := \lambda_{\max}(\hat{A}) = \textsc{PartSdp-Dual}(A_\infty) = \textsc{PartSdp}(A_\infty) \ge \langle \rho, A_\infty \rangle = \langle \rho_F, A_F \rangle \ge s^* - \epsilon, \tag{3}$$

where ρ_F, A_F denote ρ, A_∞ (respectively) restricted to the rows/columns F.

4 The SDP value of random matrix polynomial lifts

In this section we prove our main Theorem 2. To that end, let p be any self-adjoint matrix polynomial over indeterminates $Y_1, \dots, Y_d, Z_1, \dots, Z_e^*$ with $r \times r$ coefficients. Let $A_\infty = p(\mathcal{L}_\infty)$ denote the adjacency operator of the infinite lift $\mathfrak{X}_p$, and write $s^* = \textsc{PartSdp}(A_\infty) = \textsc{PartSdp-Dual}(A_\infty)$ as in Equation (3). Fix any $\epsilon, \beta > 0$, and let $\boldsymbol{A}_n = p(\boldsymbol{\mathcal{L}})$ denote the adjacency matrix of a corresponding n-lift, formed from $\boldsymbol{\mathcal{L}} = (\boldsymbol{M}_1, \dots, \boldsymbol{M}_d, \boldsymbol{P}_1, \dots, \boldsymbol{P}_e)$. Our goal is to show that except with probability at most β (assuming n is sufficiently large),

$$s^* - \epsilon \le \textsc{Sdp}(\boldsymbol{A}_n) = \textsc{Sdp-Dual}(\boldsymbol{A}_n) \le s^* + \epsilon.$$

Given our setup, the upper bound follows easily from prior work, namely Theorem 17. Let $\hat{\zeta}$ and $\hat{A}$ be as in Section 3.1, and consider the matrix polynomial p' defined by

$$p' = p + \mathrm{diag}(\hat{\zeta})\mathbb{1}.$$

Then on one hand, the ∞-lift of p' has adjacency operator precisely $\hat{A}$; on the other hand,

$$\boldsymbol{A}'_n \coloneqq p'(\boldsymbol{\mathcal{L}}) = \boldsymbol{A}_n + \mathbb{1}_{n\times n} \otimes \mathrm{diag}(\hat{\zeta}).$$

Thus Theorem 17 tells us that except with probability $\beta/2$ (provided n is large enough), the spectra $\mathrm{spec}(\boldsymbol{A}'_n)$ and $\mathrm{spec}(\hat{A})$ are at Hausdorff distance at most ϵ, from which it follows that

$$\lambda_{\max}(\boldsymbol{A}'_n) \le \lambda_{\max}(\hat{A}) + \epsilon = s^* + \epsilon.$$

But this indeed proves $\textsc{Sdp-Dual}(\boldsymbol{A}_n) \le s^* + \epsilon$, because $\hat{\zeta}$ has $\mathrm{avg}(\hat{\zeta}) = 0$ and hence is feasible for $\textsc{Sdp-Dual}(\boldsymbol{A}_n)$.

It therefore remains to prove $\textsc{Sdp}(\boldsymbol{A}_n) \ge s^* - \epsilon$.

4.1 A lower bound on the basic SDP value

In this section we complete the proof of our main theorem by showing that $\textsc{Sdp}(\boldsymbol{A}_n) \ge s^* - \epsilon$ except with probability at most $o(1) = o_{n\to\infty}(1)$ (which is at most $\beta/2$ as needed, provided n is large enough).

Let F, ρ, ρ_F, A_F be as in Section 3.1, except with that section's "ϵ" replaced by $\epsilon/2$, so that $\langle \rho_F, A_F \rangle \ge s^* - \epsilon/2$. Adding finitely many vertices to F if necessary, we may assume that it consists of all reduced words over $Y_1, \dots, Y_d, Z_1, \dots, Z_e^*$ of length at most some finite f_0. We also make the following definition:

▶ **Definition 26** (Cycle in a lift). *Given an n-lift $\mathcal{L}$, a* cycle of length $\ell > 0$ *is a pair (i, w), where $i \in [n]$ and w is a reduced word of length ℓ such that $\mathcal{L}^w |i\rangle = \pm |i\rangle$, and $\langle i|\mathcal{L}^{w'}|i\rangle = 0$ (i.e., $\mathcal{L}^{w'}|i\rangle \neq \pm|i\rangle$) for all proper prefixes w' of w.*

We will employ the following basic random graph result, [10, Lem. 23], stated in our language:

▶ **Lemma 27.** *For the random n-lift $\boldsymbol{\mathcal{L}}$, the expected number of cycles of length ℓ is $O(\ell(d + 2e - 1)^\ell)$.*

Applying this for all $\ell \le f \coloneqq 2f_0 + \deg(p)$ and using Markov's inequality, we conclude:

▶ **Corollary 28.** *Except with probability at most $n^{-.99}$, the random n-lift $\boldsymbol{\mathcal{L}}$ has at most $O(n^{.99})$ cycles of length at most f. In this case, we can exclude a set of "bad" vertices $B \subseteq [n]$ with $|B|/n \le O(n^{-.01}) = o(1)$ so that:*

$$\forall i \notin B, \quad \forall \text{ reduced words } w \text{ with } 0 < |w| \le 2f_0 + \deg(p), \qquad \langle i|\boldsymbol{\mathcal{L}}^w|i\rangle = 0.$$

We henceforth fix an outcome $\boldsymbol{\mathcal{L}} = \mathcal{L}$ (and hence $\boldsymbol{A}_n = A_n$) such that the conclusion of Corollary 28 holds, accruing our $o(1)$ probability of failure. Under this assumption, we will show $\textsc{Sdp}(A_n) \geq s^* - \epsilon/2 - o(1)$, which is sufficient to complete the proof.

▶ Remark 29. Note that our choice of $\boldsymbol{\mathcal{L}}$ depends only on the structure (i.e. the non-zero entries of A_n), not the signs. Therefore, we can take the signs in $\boldsymbol{\mathcal{L}}$ to be arbitrary; in particular, $\boldsymbol{\mathcal{L}}$ can be unsigned.

Our plan will be to first construct a "provisional" near-feasible PSD solution σ for $\textsc{Sdp}(A_n)$, with rows/columns indexed by $[n]$ and with $r \times r$ entries, such that:

- $|\langle \sigma, A_n \rangle - \langle \rho_F, A_F \rangle| \leq o(1)$, and hence $\langle \sigma, A_n \rangle \geq s^* - \epsilon/2 - o(1)$;
- for $i \notin B$, the $r \times r$ matrix σ_{ii} has diagonal entries $\frac{1}{nr}$.

Then, we will show how to "fix" σ to a some σ' that is truly feasible for $\textsc{Sdp}(A_n)$, while still having $\langle \sigma', A_n \rangle \geq \langle \sigma, A_n \rangle - o(1) \geq s^* - \epsilon/2 - o(1)$.

4.1.1 Constructing a near-feasible solution

▶ **Definition 30.** *For each $i \in [n]$, we define a linear operator $\Phi_i : \mathbb{C}^F \to \mathbb{C}^n$ by*

$$\Phi_i = \sum_{v \in F} \mathcal{L}^v \, |i\rangle\langle v| \,,$$

and also $\widetilde{\Phi}_i = \Phi_i \otimes \mathbb{1}_{r \times r}$. We furthermore define σ_i to be the $n \times n$ matrix with $r \times r$ entries whose extension $\widetilde{\sigma}_i$ is

$$\widetilde{\sigma}_i = \widetilde{\Phi}_i \cdot \widetilde{\rho}_F \cdot \widetilde{\Phi}_i^\mathsf{T}.$$

▶ Remark 31. $\widetilde{\sigma}_i$ is PSD, being the conjugation by $\widetilde{\Phi}_i$ of the PSD operator $\widetilde{\rho}_F$.

▶ **Proposition 32.** *If $i \notin B$, then $\widetilde{\Phi}_i^\mathsf{T} \widetilde{A}_n \widetilde{\Phi}_i = \widetilde{A}_F$.*

Proof. Thinking of $\widetilde{\Phi}_i^\mathsf{T} \widetilde{A}_n \widetilde{\Phi}_i$ as an $F \times F$ matrix of $r \times r$ matrices, it follows that its (u, v) entry is given by

$$\sum_{\text{term } a_w w \text{ in } p} \langle i | \mathcal{L}^{v^* w u} | i \rangle \, a_w.$$

On the other hand, the (u, v) entry of A_F is by definition

$$\sum_{\text{term } a_w w \text{ in } p} 1[v^* w u = \mathbb{1}] a_w,$$

where "$v^* w u = \mathbb{1}$" denotes that the reduced form of word $v^* w u$ is the empty word. We therefore have equality for all u, v provided $\langle i | \mathcal{L}^{v^* w u} | i \rangle = 0$ whenever $v^* w u \neq \emptyset$. But Corollary 28 tells us this indeed holds for $i \notin B$, because $|v^* w u| \leq 2 f_0 + \deg(p)$. ◀

▶ **Corollary 33.** *For $i \notin B$ we have $\langle \sigma_i, A_n \rangle = \langle \rho_F, A_F \rangle$.*

Proof. When $i \notin B$,

$$\langle \sigma_i, A_n \rangle = \operatorname{tr}(\widetilde{\sigma}_i \widetilde{A}_n) = \operatorname{tr}(\widetilde{\Phi}_i \widetilde{\rho}_F \widetilde{\Phi}_i^\mathsf{T} \widetilde{A}_n) = \operatorname{tr}(\widetilde{\rho}_F \widetilde{\Phi}_i^\mathsf{T} \widetilde{A}_n \widetilde{\Phi}_i) = \operatorname{tr}(\widetilde{\rho}_F \widetilde{A}_F) = \langle \rho_F, A_F \rangle,$$

where the last equality used Proposition 32 ◀

We now define our "provisional" SDP solution σ via

$$\sigma = \operatorname*{avg}_{i\in[n]}\{\sigma_i\};$$

this is indeed PSD, being the average of PSD operators. Using Corollary 33, $|B|/n = o(1)$, and the fact that $|\langle\sigma_i, A_n\rangle| \le O(1)$ for every i (since σ_i only has $O(1)$ nonzero entries, each bounded in magnitude by $O(1)$), we conclude:

▶ **Proposition 34.** *$|\langle\sigma, A_n\rangle - \langle\rho_F, A_F\rangle| \le o(1)$, and hence $\langle\sigma, A_n\rangle \ge s^* - \epsilon/2 - o(1)$.*

Now similar to Proposition 35 we have the following:

▶ **Proposition 35.** *If $j \notin B$, then the (j,j) entry of σ is $\frac{1}{n}\operatorname{tr}(\rho_F)$ (and hence is an $r \times r$ matrix with diagonal entries equal to $\frac{1}{nr}$).*

Proof. By definition, the (j,j) entry of σ is

$$\begin{aligned}
&= \operatorname*{avg}_{i\in[n]} \sum_{u,v\in F} \langle j|\mathcal{L}^u|i\rangle \langle u|\rho_F|v\rangle \langle i|\mathcal{L}^{v^*}|j\rangle \\
&= \tfrac{1}{n} \sum_{u,v\in F} \langle u|\rho_F|v\rangle \cdot \sum_{i\in[n]} \langle j|\mathcal{L}^u|i\rangle \langle i|\mathcal{L}^{v^*}|j\rangle \\
&= \tfrac{1}{n} \sum_{u,v\in F} \langle u|\rho_F|v\rangle \cdot \langle j|\mathcal{L}^{uv^*}|j\rangle \qquad \text{(since } \textstyle\sum_i |i\rangle\langle i| = \mathbb{1}\text{)}
\end{aligned}$$

where we are writing $\langle u|\rho_F|v\rangle$ for the $r \times r$ matrix at the (u,v) entry of ρ_F. Now when $j \notin B$, we have that $\langle j|\mathcal{L}^{vw^*}|j\rangle = 1[vw^* = \emptyset]$ by Corollary 28, since $|vw^*| \le 2f_0$. Thus all summands above drop out, except for the ones with $u = v$; this indeed gives $\frac{1}{n}\operatorname{tr}(\rho_F)$. ◀

4.1.2 Fixing σ

Finally, we slightly fix σ to make it truly feasible for $\textsc{Sdp}(A)$. Let σ' be the $n \times n$ matrix, with entries from $\mathbb{C}^{r\times r}$, defined as follows:

$$\sigma'_{ij} = \begin{cases} \sigma_{ij} & \text{if } i,j \notin B, \\ \frac{1}{nr}\mathbb{1}_{r\times r} & \text{if } i = j \in B, \\ 0 & \text{else.} \end{cases}$$

This σ' is easily seen to be PSD, being a principal submatrix of the PSD matrix σ, direct-summed with the PSD matrix $\frac{1}{nr}\mathbb{1}_{r\times r}$. As well, the $nr \times nr$ extension matrix $\widetilde{\sigma}'$ has all diagonal entries equal to $\frac{1}{nr}$, by Proposition 35. Thus $\widetilde{\sigma}'$ is feasible for $\textsc{Sdp}(A_n) = \textsc{Sdp}(\widetilde{A}_n)$, and it remains for us to show that

$$\langle\sigma, A_n\rangle - \langle\sigma', A_n\rangle \le o(1); \tag{4}$$

this will imply $\langle\sigma', A_n\rangle \ge s^* - \epsilon/2 - o(1)$ by Proposition 34, and hence $\textsc{Sdp}(A_n) \ge s^* - \epsilon$ (for sufficiently large n), as desired.

We have

$$\langle\sigma, A_n\rangle - \langle\sigma', A_n\rangle = \langle\sigma - \sigma', A_n\rangle \le \|\widetilde{\sigma} - \widetilde{\sigma}'\|_1 \|\widetilde{A}_n\|_\infty.$$

Next,

$$\|\widetilde{A}_n\|_\infty = \|p(\mathcal{L})\|_\infty = \|\sum_w \mathcal{L}^w \otimes a_w\|_\infty \le \sum_w \|\mathcal{L}^w\|_\infty \cdot \|a_w\|_\infty = \sum_w \|a_w\|_\infty \le O(1).$$

Here the final equality is because each $\mathcal{L}^w$ is a signed permutation matrix (hence has $\|\mathcal{L}^w\|_\infty = 1$), and the final inequality is because p has only constantly many coefficients, of constant size. Thus to establish Inequality (4), it remains to show $\|\widetilde{\sigma} - \widetilde{\sigma}'\|_1 \leq o(1)$.

Define the orthogonal projection matrices $\Pi_B = \sum_{i \in B} |i\rangle\langle i| \otimes \mathbb{1}_{r \times r}$ and similarly $\Pi_{\overline{B}}$, where $\overline{B} = [n] \setminus B$. Observe that

$$\sigma' = \tfrac{1}{nr}\Pi_B + \Pi_{\overline{B}}\sigma\Pi_{\overline{B}},$$

and thus

$$\|\widetilde{\sigma} - \widetilde{\sigma}'\|_1 = \|\widetilde{\sigma} - \widetilde{\Pi}_{\overline{B}}\widetilde{\sigma}\widetilde{\Pi}_{\overline{B}} - \tfrac{1}{nr}\widetilde{\Pi}_B\|_1 \leq \|\widetilde{\sigma} - \widetilde{\Pi}_{\overline{B}}\widetilde{\sigma}\widetilde{\Pi}_{\overline{B}}\|_1 + \tfrac{1}{nr}\|\widetilde{\Pi}_B\|_1.$$

But $\frac{1}{nr}\|\widetilde{\Pi}_B\|_1 = \frac{|B|}{n} = o(1)$, so it remains to show

$$\|\widetilde{\sigma} - \widetilde{\Pi}_{\overline{B}}\widetilde{\sigma}\widetilde{\Pi}_{\overline{B}}\|_1 \leq o(1).$$

Note that $\widetilde{\sigma} \in \mathbb{C}^{nr \times nr}$ is nearly a density matrix: it is PSD, and all but a $1 - o(1)$ fraction of its diagonal entries are $\frac{1}{nr}$, with the remaining ones being bounded in magnitude by $O(\frac{1}{n})$. Thus $\mathrm{tr}(\widetilde{\sigma}) = 1 \pm o(1)$, and we can therefore scale $\widetilde{\sigma}$ by a $1 \pm o(1)$ factor to produce a true density matrix $\hat{\sigma}$. Clearly it now suffices to show

$$\|\hat{\sigma} - \widetilde{\Pi}_{\overline{B}}\hat{\sigma}\widetilde{\Pi}_{\overline{B}}\|_1 \leq o(1).$$

But this follows from Winter's Gentle Measurement Lemma [39, Lem. 9], which bounds the quantity on the left by $\sqrt{8\lambda}$, where $\lambda = 1 - \mathrm{tr}(\hat{\sigma}\widetilde{\Pi}_{\overline{B}}) = o(1)$. This completes the proof.

5 Application to block models

In this section we will describe the application of our results to the hypothesis testing problem in block models; that is, using the (Goemans–Williamson) SDP value of the negated adjacency matrix $\textsc{Sdp}(-A)$ to distinguish between uniformly regular d-regular graphs and the "equitable stochastic block model" described by Bandeira et al and others [32, 5, 2].

In the n-vertex equitable 2-community block model, the n vertices are divided into two groups, and each vertex has a edges to vertices of the same group, and b edges to vertices of the other group (and $a + b = d$). A random graph of this form is generated by taking a random unsigned $n/2$-lift of the 2-vertex graph which has b parallel edges, and a self-loops on each vertex. The corresponding polynomial which describes the lift is

$$\begin{aligned} p(Y_1, \ldots, Y_{2a}, Z_1, \ldots, Z_b) = &\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \left(\sum_{i=1}^{a} Y_i \right) + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \left(\sum_{i=a+1}^{2a} Y_i \right) \\ &+ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \left(\sum_{i=1}^{b} Z_i \right) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \left(\sum_{i=1}^{a} Z_i^* \right). \end{aligned} \tag{5}$$

By evaluating p at a random unsigned $n/2$-lift $\boldsymbol{\mathcal{L}} = (\boldsymbol{M}_1, \ldots, \boldsymbol{M}_{2a}, \boldsymbol{P}_1, \ldots, \boldsymbol{P}_b)$ we obtain (the adjacency matrix of) a random graph $\boldsymbol{A}_n$ in the n-vertex equitable 2-community block model. We note that the infinite lift $p(\mathcal{L}_\infty)$ is (multiple copies of) the infinite d-regular graph; thus in particular we have $\mathrm{spec}(A_\infty) = [-2\sqrt{d-1}, 2\sqrt{d-1}]$.

To upper bound $\textsc{Sdp}(-\boldsymbol{A}_n)$, we use the eigenvalue bound $\textsc{Sdp}(-\boldsymbol{A}_n) \leq \lambda_{\max}(-\boldsymbol{A}_n)$. In contrast to the previous sections of our paper, the finite random lift here is unsigned, hence we need to take some care with the "trivial eigenvalues" of the resulting lift.

▶ **Definition 36** (Trivial eigenvalues). *Given a matrix polynomial p with coefficients from $\mathbb{C}^{r\times r}$, the associated* trivial eigenvalues *are simply the (multiset of) eigenvalues of $p(1,\dots,1) \in \mathbb{C}^{r\times r}$.*

Note that the trivial eigenvalues of our polynomial in Equation (5) are those of $\begin{pmatrix} a & b \\ b & a \end{pmatrix}$, namely $a+b$ and $a-b$. In the case of unsigned matrix polynomial lifts, Bordenave and Collins [10] show the following:

▶ **Theorem 37.** *Let p be a self-adjoint matrix polynomial with coefficients from $\mathbb{C}^{r\times r}$ on indeterminates $Y_1,\dots,Z_e^*$. Write T for the trivial eigenvalues of p. Then for all $\epsilon,\beta>0$ and sufficiently large n, the following holds:*

Let $\boldsymbol{A}_n = p(\boldsymbol{\mathcal{L}}_n)$, where $\boldsymbol{\mathcal{L}}_n$ is a random unsigned n-lift, and let $A_\infty = p(\mathcal{L}_\infty)$. Then except with probability at most β, the spectra $\operatorname{spec}(\boldsymbol{A}_{n,\perp}) \setminus T$ and $\operatorname{spec}(A_\infty)$ are at Hausdorff distance at most ϵ.

Applying this to the equitable stochastic block model, it means that the nontrivial eigenvalues of graphs in the model are between $-2\sqrt{d-1}-\epsilon$ and $2\sqrt{d-1}+\epsilon$ with high probability (i.e., the same range as for random d-regular graphs). Recalling that the trivial eigenvalues of $-\boldsymbol{A}_n$ are $-(a+b)$ and $-(a-b)$, we indeed get $\lambda_{\max}(\boldsymbol{A}_n) \le \max(b-a,\ 2\sqrt{a+b-1})+\epsilon$ with high probability, implying the upper bound in Corollary 3

As for the lower bound in Corollary 3, we refer to Remark 29 to see that the proof in Section 4.1 holds regardless of signs on the lift used to construct the SDP solution. Thus Section 4.1 also establishes a lower bound of $\textsc{Sdp}(-\boldsymbol{A}_n) \ge 2\sqrt{a+b-1}-\epsilon$. Meanwhile, in the $\textsc{Block}(n,a,b)$ we can always take an integral solution that is the ±1-indicator for the two communities in the partition; this will have value exactly $b-a$ and hence we also always have $\textsc{Sdp}(-\boldsymbol{A}_n) \ge b-a$. This completes the proof of the lower bound in Corollary 3.

References

1. Scott Aaronson and Andris Ambainis. Forrelation: a problem that optimally separates quantum from classical computing. *SIAM J. Comput.*, 47(3):982–1038, 2018. doi:10.1137/15M1050902.
2. Afonso S Bandeira, Jess Banks, Dmitriy Kunisky, Christopher Moore, and Alex Wein. Spectral planting and the hardness of refuting cuts, colorability, and communities in random graphs. In *Proceedings of the 34th Annual Conference on Learning Theory*, pages 410–473. PMLR, 2021.
3. Jess Banks, Robert Kleinberg, and Cristopher Moore. The lovász theta function for random regular graphs and community detection in the hard regime. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2017.
4. Jess Banks, Sidhanth Mohanty, and Prasad Raghavendra. Local statistics, semidefinite programming, and community detection. In *soda21*, pages 1298–1316. SIAM, 2021.
5. Paolo Barucca. Spectral partitioning in equitable graphs. *Physical Review E*, 95(6):062310, 2017.
6. Mohsen Bayati, David Gamarnik, and Prasad Tetali. Combinatorial approach to the interpolation method and scaling limits in sparse random graphs. *Annals of Probability*, 41(6):4080–4115, 2013.
7. J. Frédéric Bonnans and Alexander Shapiro. *Perturbation Analysis of Optimization Problems*. Springer Series in Operations Research. Springer-Verlag, New York, 2000. doi:10.1007/978-1-4612-1394-9.
8. Ravi Boppana. Eigenvalues and graph bisection: An average-case analysis. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 280–285, 1987.
9. Charles Bordenave. A new proof of Friedman's second eigenvalue theorem and its extension to random lifts. *Ann. Sci. Éc. Norm. Supér. (4)*, 53(6):1393–1439, 2020. doi:10.24033/asens.245.

10 Charles Bordenave and Benoît Collins. Eigenvalues of random lifts and polynomials of random permutation matrices. *Annals of Mathematics*, 190(3):811–875, 2019. doi:10.4007/annals.2019.190.3.3.

11 John Clauser, Michael Horne, Abner Shimony, and Richard Holt. Proposed experiment to test local hidden-variable theories. *Physical Review Letters*, 23(15):880–884, 1969.

12 Richard Cleve, Peter Høyer, Benjamin Toner, and John Watrous. Consequences and limits of nonlocal strategies. In *Proceedings of the 19th Annual Computational Complexity Conference*, pages 236–249. IEEE, 2004.

13 Endre Csóka, Balázs Gerencsér, Viktor Harangi, and Bálint Virág. Invariant Gaussian processes and independent sets on regular graphs of large girth. *Random Structures Algorithms*, 47(2):284–303, 2015. doi:10.1002/rsa.20547.

14 Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From average case complexity to improper learning complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 441–448, 2014.

15 Charles Delorme and Svatopluk Poljak. Laplacian eigenvalues and the maximum cut problem. *Mathematical Programming*, 62:557–574, 1993.

16 Amir Dembo, Andrea Montanari, Subhabrata Sen, et al. Extremal cuts of sparse random graphs. *Annals of Probability*, 45(2):1190–1217, 2017.

17 Yash Deshpande, Andrea Montanari, Ryan O'Donnell, Tselil Schramm, and Subhabrata Sen. The threshold for SDP-refutation of random regular NAE-3SAT. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2305–2321. SIAM, 2019.

18 Wilm Donath and Alan Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Develop.*, 17:420–425, 1973. doi:10.1147/rd.175.0420.

19 Yehonatan Elon. Gaussian waves on the regular tree. Technical report, arXiv, 2009. arXiv:0907.5065.

20 Uriel Feige and László Lovász. Two-prover one-round proof systems: Their power and their problems. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 733–744, 1992.

21 Joel Friedman. A proof of Alon's second eigenvalue conjecture and related problems. *Memoirs of the American Mathematical Society*, 195(910):viii+100, 2008.

22 Jorge Garza-Vargas and Archit Kulkarni. Spectra of infinite graphs via freeness with amalgamation. Technical report, arXiv, 2021. arXiv:1912.10137.

23 Michel Goemans and David Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995. doi:10.1145/227683.227684.

24 Viktor Harangi and Bálint Virág. Independence ratio and random eigenvectors in transitive graphs. *Ann. Probab.*, 43(5):2810–2840, 2015. doi:10.1214/14-AOP952.

25 Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability Obfuscation from well-founded assumptions. Technical report, arXiv, 2020. arXiv:2008.09317.

26 Franz Lehner. Computing norms of free operators with matrix coefficients. *Amer. J. Math.*, 121(3):453–486, 1999. URL: http://muse.jhu.edu/journals/american_journal_of_mathematics/v121/121.3lehner.pdf.

27 Marc Mézard and Andrea Montanari. *Information, physics, and computation*. Oxford Graduate Texts. Oxford University Press, Oxford, 2009. doi:10.1093/acprof:oso/9780198570837.001.0001.

28 Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. The SDP value for random two-eigenvalue CSPs. In Christophe Paul and Markus Bläser, editors, *Proceedings of the 37th Annual Symposium on Theoretical Aspects of Computer Science*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:45, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2020.50.

29 Andrea Montanari. Optimization of the Sherrington–Kirkpatrick hamiltonian. *SIAM Journal on Computing*, FOCS19, 2021.

30 Andrea Montanari and Subhabrata Sen. Semidefinite programs on sparse random graphs and their application to community detection. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*, pages 814–827. ACM, New York, 2016. doi:10.1145/2897518.2897548.

31 Elchanan Mossel, Joe Neeman, and Allan Sly. Reconstruction and estimation in the planted partition model. *Probability Theory and Related Fields*, 162(3):431–461, 2015.

32 MEJ Newman and Travis Martin. Equitable random graphs. *Physical Review E*, 90(5):052824, 2014.

33 Ryan O'Donnell and Xinyu Wu. Explicit near-fully X-Ramanujan graphs. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*, pages 1045–1056, 2020.

34 Svatopluk Poljak and Franz Rendl. Nonpolyhedral relaxations of graph-bisection problems. *SIAM Journal on Optimization*, 5(3):467–487, 1995.

35 Prasad Raghavendra. *Approximating NP-hard problems: efficient algorithms and their limits*. PhD thesis, University of Washington, 2009.

36 Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Ann. of Math. (2)*, 155(1):157–187, 2002. doi:10.2307/3062153.

37 Franz Rendl and Henry Wolkowicz. A projection technique for partitioning the nodes of a graph. *Ann. Oper. Res.*, 58:155–179, 1995. Applied mathematical programming and modeling, II (APMOD 93) (Budapest, 1993). doi:10.1007/BF02032130.

38 George Styan. Hadamard products and multivariate statistical analysis. *Linear Algebra Appl.*, 6:217–240, 1973. doi:10.1016/0024-3795(73)90023-2.

39 Andreas Winter. Coding theorem and strong converse for quantum channels. *Institute of Electrical and Electronics Engineers. Transactions on Information Theory*, 45(7):2481–2485, 1999. doi:10.1109/18.796385.

Strongly Sublinear Algorithms for Testing Pattern Freeness

Ilan Newman
Department of Computer Science, University of Haifa, Israel

Nithin Varma
Chennai Mathematical Institute, India

Abstract

For a permutation $\pi : [k] \to [k]$, a function $f : [n] \to \mathbb{R}$ contains a π-appearance if there exists $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that for all $s, t \in [k]$, $f(i_s) < f(i_t)$ if and only if $\pi(s) < \pi(t)$. The function is π-free if it has no π-appearances. In this paper, we investigate the problem of testing whether an input function f is π-free or whether f differs on at least εn values from every π-free function. This is a generalization of the well-studied monotonicity testing and was first studied by Newman, Rabinovich, Rajendraprasad and Sohler [28]. We show that for all constants $k \in \mathbb{N}$, $\varepsilon \in (0, 1)$, and permutation $\pi : [k] \to [k]$, there is a one-sided error ε-testing algorithm for π-freeness of functions $f : [n] \to \mathbb{R}$ that makes $\tilde{O}(n^{o(1)})$ queries. We improve significantly upon the previous best upper bound $O(n^{1-1/(k-1)})$ by Ben-Eliezer and Canonne [7]. Our algorithm is adaptive, while the earlier best upper bound is known to be tight for nonadaptive algorithms.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Property testing, Pattern freeness, Sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.98

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2106.04856 [30]

Funding *Ilan Newman*: The author was supported by the Israel Science Foundation, grant number 379/21.

1 Introduction

Given a permutation $\pi : [k] \to [k]$, a function $f : [n] \to \mathbb{R}$ contains a π-appearance if there exists $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that for all $s, t \in [k]$ it holds that $f(i_s) < f(i_t)$ if and only if $\pi(s) < \pi(t)$. In other words, the function restricted to the indices $\{i_1, \ldots, i_k\}$ respects the ordering in π. The function is π-free if it has no π-appearance. For instance, the set of all real-valued monotone non-decreasing functions over $[n]$ is $(2, 1)$-free. The notion of π-freeness is well-studied in combinatorics, where the famous Stanley-Wilf conjecture about the bound on the number of π-free permutations $f : [n] \to [n]$ has spawned a lot of work [13, 14, 5, 25, 3], ultimately culminating in a proof by Marcus and Tardos [26]. The problem of designing algorithms to determine whether a given permutation $f : [n] \to [n]$ is π-free is an active area of research [2, 1, 10], with linear time algorithms for constant k [23, 20]. Apart from the theoretical interest, practical motivations to study π-freeness include the study of motifs and patterns in time series analysis [11, 32, 24].

In this paper, we study property testing of π-freeness, first studied by Newman, Rabinovich, Rajendraprasad and Sohler [28]. Specifically, given $\varepsilon \in (0, 1)$, an ε-testing algorithm for π-freeness accepts an input function f that is π-free, and rejects if at least εn values of

f need to be changed for it to become π-free[1]. This problem is a generalization of the well-studied monotonicity testing on the line ($(2,1)$-freeness), which was one of the first works in combinatorial property testing, and is still being studied actively [17, 18, 12, 15, 6, 16, 31].

Newman, Rabinovich, Rajendraprasad and Sohler [28] showed that for a general permutation π of length k, the problem of π-freeness can be ε-tested using a nonadaptive[2] algorithm of query complexity $O_{k,\varepsilon}(n^{1-1/k})$.[3] Additionally, they showed that, for nonadaptive algorithms, one cannot obtain a significant improvement on this upper bound for $k \geq 4$. In a subsequent work, Ben-Eliezer and Canonne [7] improved this upper bound to $O_{k,\varepsilon}(n^{1-1/(k-1)})$, which they showed to be tight for nonadaptive algorithms. For monotone permutations π of length k, namely, either $(1, 2, \ldots, k)$ or $(k, k-1, \ldots, 1)$, [28] presented an algorithm with query complexity $(\varepsilon^{-1} \log n)^{O(k^2)}$ to ε-test π-freeness. This was improved, in a sequence of works [8, 9], to $O_{k,\varepsilon}(\log n)$, which is optimal for constant ε even for the special case of testing $(2,1)$-freeness [19].

Despite the extensive study and advances in testing freeness of monotone permutations, improving the complexity of testing freeness of arbitrary permutations has remained open all this while. For arbitrary permutations of length at most 3, [28] gave an adaptive algorithm for testing freeness with query complexity $(\varepsilon^{-1} \log n)^{O(1)}$. However, the case of general $k > 3$ has remained elusive. In particular, the techniques of [28] for $k = 3$ do not seem to generalize even for $k = 4$.

As remarked above, optimal *nonadaptive* algorithms are known for any k [7], but, their complexity tends to be linear in the input length as k grows. For the special case of $(2,1)$-freeness, it is well-known that adaptivity does not help at all in improving the complexity of testing [18, 19]. Adaptivity is known to help somewhat for the case of testing freeness of monotone permutations of length k, where, every nonadaptive algorithm has query complexity $\Omega((\log n)^{\log k})$ [8], and the $O_{k,\varepsilon}(\log n)$-query algorithm of Ben-Eliezer, Letzter, and Waingarten [9] is adaptive. Adaptivity significantly helps in testing freeness of arbitrary permutations of length 3 as shown by [28] and [7].

Our results. In this work, we give adaptive ε-testing algorithms for π-freeness of permutations π of arbitrary constant length k with complexity $\tilde{O}_{k,\varepsilon}(n^{o(1)})$. Hence, testing π-freeness has quite efficient sublinear algorithms even for relatively large patterns. Our result shows a strong separation between adaptive and nonadaptive algorithms for testing pattern freeness.

▶ **Theorem 1.1.** *Let $\varepsilon \in (0,1)$, $k \in \mathbb{N}$ and $\pi : [k] \to [k]$ be a permutation. There exists an ε-tester for π-freeness of functions $f : [n] \to \mathbb{R}$ with query complexity $\tilde{O}_{k,\varepsilon}(n^{O(1/\log\log\log n)})$.*

Discussion of our techniques. The algorithm that we design has one-sided error and rejects only if it finds a π-appearance in the input function $f : [n] \to \mathbb{R}$. In the following paragraphs, we present some ideas behind a $\tilde{O}(\sqrt{n})$-query algorithm for detecting a π-appearance in a function f that is ε-far from π-free, for a permutation π of length 4. The case of length-4 permutations is not much different from the general case (where, we additionally recurse on problems of smaller length patterns). The $\tilde{O}(\sqrt{n})$ queries algorithm, however, is much

[1] Algorithms in this area are typically randomized, and the decisions to accept and reject are with high constant probability. See [33, 22] for definitions of property testing.

[2] An algorithm whose queries do not depend on the answers to previous queries is a nonadaptive algorithm. It is adaptive otherwise.

[3] Throughout this work, we are interested in the parameter regime of constant $\varepsilon \in (0,1)$ and k. The notation $O_{k,\varepsilon}(\cdot)$ hides a factor that is an arbitrary function of these parameters.

simpler than the general one, but it outlines many of the ideas involved in the latter one. A more detailed description appears in Section 3. The formal description of the general algorithm is given in Section 4. The correctness proof and query complexity analysis of the general algorithm can be found in the full version [30].

For a parameter $\varepsilon \in (0,1)$, a function f is ε-far from π-free if at least an εn of its values needs to be changed in order to make it π-free. In other words, the Hamming distance of f from the closest π-free real-valued function over $[n]$ is at least εn. A folklore fact is that the Hamming distance and the deletion distance of f to π-freeness are equal, where the deletion distance of f to π-freeness is the cardinality of the smallest set $S \subseteq [n]$ such that f restricted to $[n] \setminus S$ is π-free. By virtue of this equality, a function that is ε-far from π-free has a matching of π-appearances of cardinality at least $\varepsilon n/4$. This observation facilitates our algorithm and all previous algorithms on testing π-freeness, including monotonicity testers.

The basic ingredient in our algorithms is the use of a natural representation of $f : [n] \to \mathbb{R}$ by a Boolean function over a grid $[n] \times R(f)$, where $R(f)$ denotes the range of f. Specifically, we visualize the function as a grid of n points in $\mathbb{R}^2$, such that for each $i \in [n]$, the point $(i, f(i))$ is a marked point of the grid. We denote this grid with marked points as G_n. This view has been useful in the design of approximation algorithms for the related and fundamental problem of estimating the length of Longest Increasing Subsequence (LIS) in a real-valued array [35, 34, 27, 29]. Adopting this view, for any permutation π, a π-appearance at places $(i_1, \ldots, i_k)$ in f corresponds naturally to a k-tuple of points $\{i_j, f(i_j)\},\ j = 1 \ldots k$ in G_n, for which their relative order (in G_n) forms a π-appearance. The converse is also true: every π-appearance in the Boolean grid G_n corresponds to a π-appearance in f.

We note that the grid G_n is neither known to nor directly accessible by the algorithm, and in particular, $R(f)$ is not assumed to be known. A main first step in our algorithm is to approximate the grid G_n by a coarser $m \times m$ grid of boxes, $G_{m,m}$, for $m << n$, a parameter that will determine the query complexity. The grid $G_{m,m}$ is defined as follows. Suppose that we have a partition of $R(f)$ into m disjoint contiguous intervals of increasing values, referred here as "layers", $I_1, \ldots, I_m$, and let $S_1, \ldots, S_m$ be a partition of $[n]$ into m contiguous intervals of equal length, referred to as "stripes". These two partitions decompose G_n and the f-points in it into m^2 boxes which form the grid of cells $G_{m,m}$. The (i,j)-th cell of this grid is the Cartesian product $S_i \times I_j$, and is denoted $\mathsf{box}(S_i, I_j)$. We view the non-empty cells in $G_{m,m}$ as a coarse approximation of G_n (and of the input function, equivalently). The grid $G_{m,m}$ has a natural order on its boxes (viewed as points in $[m] \times [m]$).

While $G_{m,m}$ is also not directly accessible to the algorithm, it can be well-approximated very efficiently. Fixing m, we use sampling by $\tilde{O}(m)$ queries to identify and *mark* the boxes in $G_{m,m}$ that contain a non-negligible density of points of G_n. This provides a good enough approximation of the grid $G_{m,m}$. For the rest of this top-level explanation, assume that we have fixed $m << n$, and we know $G_{m,m}$; that is, we know the the number of points of G_n belonging to each box in $G_{m,m}$, but not necessarily the points themselves.

If we find k nonempty boxes in $G_{m,m}$ that form a π-appearance when viewed as points in the $[m] \times [m]$ grid, then G_n (and hence f) contains a π-appearance for any set of k points that is formed by selecting one point from each of the corresponding boxes. See Figure 1(A) for such a situation, for $\pi = (3,2,1,4)$. We first detect such π-appearances by our knowledge of $G_{m,m}$. However, the converse is not true: it could be that G_n contains many π-appearances, where the corresponding points, called "legs", are in boxes that share layers or stripes, and hence do not form π-appearances in $G_{m,m}$. See e.g., Figure 1(B) for such an appearance for $\pi = (3,2,1,4)$. Thus, assuming that the function is far from being π-free, and no π-appearances are detected in $G_{m,m}$, there must be many π-appearances in

which some legs share a layer or a stripe in $G_{m,m}$. In this case, the seminal result of Marcus and Tardos [26], implies that only $O(m)$ of the boxes in $G_{m,m}$ are non-empty. An averaging argument implies that if f is ε-far from being π-free, then after deleting layers or stripes in $G_{m,m}$ with $\omega(1)$ dense boxes, we are still left with a partial function (on the undeleted points) that is ε'-far from being π-free, for a large enough ε'.

Now, to be specific, consider $\pi = (3, 2, 1, 4)$ although all the following ideas work for any specific 4-length permutation. Any π-appearance has its four legs spread over at most 4 marked boxes. This implies that there are only constantly many non-isomorphic ways of arranging the marked boxes containing any particular π-appearance (in terms of the order relation among the marked boxes, and the way the legs of the π-appearance are included in them). These constantly many ways are called "configurations" in the sequel. Thus any π-appearance is consistent with a certain configuration. Additionally, in case multiple points in a π-appearance share some marked boxes, this appearance induces the appearances of permutations of length smaller than 4 in each box (which are sub-permutations ν of π). If a constant fraction of the π-appearances are spread across multiple marked boxes, there will be many such ν-appearances in the marked boxes in the coarse grid. Hence, one phase of our algorithm will run tests for ν-appearances for smaller patterns ν (which can be done in $\operatorname{polylog} n$ queries using known testers for patterns of length at most 3) on each marked box, and combine these ν-appearances to detect a π-appearance, if any. This phase, while seemingly simple will require extra care, as combining sub-patterns appearances into a global π-appearance is not always possible. This is a major issue in the general case for $k > 4$.

The simpler case is when there is a constant fraction of π-appearances such that all 4 points of each such appearance belong to a single marked box. This can be solved by randomly sampling a few marked boxes and querying all the points in them to see if there are any π-appearances. A special treatment has to be made in the case a constant fraction of the π-appearances have their legs belonging to the same layer or the same stripe. But this will be an easy extension of the "one-box" case.

To obtain the desired query complexity, consider first setting $m = \tilde{O}(\sqrt{n})$. Getting a good enough estimate of $G_{m,m}$ as described above take $\tilde{O}(m) = \tilde{O}(\sqrt{n})$ queries. Then, testing each box for ν-freeness, for smaller permutations ν takes $\operatorname{polylog} n$ per test, but since this is done for all marked boxes, this step also takes $\tilde{O}(m) = \tilde{O}(\sqrt{n})$. Finally, in the last simpler case, we may just query all indices in a sampled box that contains at most $n/m = \sqrt{n}$ indices, by our setting of m. This results in a $\tilde{O}(\sqrt{n})$-query tester for π-freeness.

To obtain a better complexity, we reduce the value of m, and, in the last step, we randomly sample a few marked boxes and run the algorithm recursively. This is so, since, in the last step, we are in the case that for a constant fraction of the π-appearances, all four legs of each π-appearance belong to a single marked box (or a constant number of marked boxes sharing a layer or stripe). The depth of recursion depends monotonically on n/m and the larger it is the smaller is the query complexity. The bound we describe in this article is $n^{O(1/\log\log n \log n)}$ which is due to the exponential deterioration of the distance parameter ε in each recursive call. Our algorithm for permutations of length $k > 4$ uses, in addition to the self-recursion, a recursion on k too.

Finally, even though it was not explicitly mentioned, we call ν-freeness or π-freeness algorithms on marked boxes (or a collection of constantly many marked boxes sharing a layer or stripe) and not the entire grid. Since we do not know which points belong to the marked boxes, but only know that their density is significant, we can access points in them only via sampling and treating points that fall outside the desired box as being *erased*. This necessitates the use of erasure-resilient testers [16]. Such testers are known for all permutation patterns of length at most 3 [16, 29, 28]. In addition, the basic tester we design is also erasure-resilient, which makes it possible for it to be called recursively.

Some additional complications we had to overcome. In the recursive algorithm for k-length permutation freeness, $k \geq 4$, we need to find ν-appearances that are restricted to appear in specific configurations, for smaller length permutations ν. To exemplify this notion, consider testing $\nu = (1, 3, 2)$-freeness. In the usual (unrestricted) case, $f : [n] \mapsto \mathbb{R}$ has a ν-appearance if the values at any three indices have a ν-consistent order. In a restricted case, we may ask ourselves whether f is free of ν-appearances where the indices corresponding to the $1, 3$-legs of a ν-appearance are at most $n/2$ (that is in the first half of $[n]$), while the index corresponding to the 2-leg is larger than $n/2$. This latter property seems at least as hard to test as the unrestricted one. In particular, for the ν-appearance as described above, it could be that while f is far from being ν-free in the usual sense, it is still free of having restricted ν-appearances. In our algorithm, we need to test (at lower recursion levels) freeness from such restricted appearances. The extra restriction is discussed in Section 3 and Section 4.

Open questions. The major open question is to determine the exact (asymptotic) complexity of testing π-freeness of arbitrary permutations $\pi : [k] \to [k]$, $k \geq 3$. While the gaps for $k = 3$ are relatively small (within polylog n range), the gaps are yet much larger for $k \geq 4$. We do not have any reason to think that the upper bound obtained in this draft is tight. We did not try to optimize the exponent of n in the $\tilde{O}(n^{o(1)})$ expression, but the current methods do not seem to bring down the query complexity to polylog n. We conjecture, however, that the query complexity is polylog n for all constant k. Another open question is whether the complexity of a two-sided error testing might be lower than of one-sided error testing.

Finally, Newman and Varma [29] used lower bounds on testing pattern freeness of monotone patterns of length $k \geq 3$ (for nonadaptive algorithms), to obtain lower bounds on the query complexity of nonadaptive algorithms for LIS estimation. Proving any lower bound better than $\Omega(\log n)$ for adaptively testing freeness, for arbitrary permutations of length k for $k \geq 3$, may translate in a similar way to lower bounds on adaptive algorithms for LIS estimation.

Other definitions of π-freeness. In the definition of π-freeness, we required strict inequalities on function values to have an occurrence of the pattern. A natural variant is to allow weak inequalities, that is – for a set indices $1 \leq i_1 < i_2 \cdots < i_k \leq n$ a *weak*-π appearance is when for all $s, t \in [k]$ it holds that $f(i_s) \leq f(i_t)$ if and only if $\pi(s) < \pi(t)$. Such a relaxed requirement would mean that having a collection of k or more equal values is already a π-appearance for any pattern π. For monotone patterns of length k, the deletion distance equals to the Hamming distance, for any k, for this relaxed definition as well. We do not know if this is true for larger k for non-monotone patterns in general, although we suspect that the Hamming distance is never larger than the deletion distance by more than a constant factor. Proving this will be enough to make our results true for testing freeness of any constant size forbidden permutation, even with the relaxed definition. We show that the Hamming distance is equal to the deletion distance for patterns of length at most 4. Hence, Theorem 1.1 also holds for weak-π-freeness for $k \leq 4$.

Another variant that may seem related is when the forbidden order pattern is not necessarily a permutation (that is, arbitrary function from $[k]$ to $[k]$ which is not one-to-one). For example, for the 4-pattern $\alpha = (1, 2, 3, 1)$, an α-appearance in f at indices $i_1 < i_2 < i_3 < i_4$ is when $f(i_1) < f(i_2) < f(i_3)$ and $f(i_4) = f(i_1)$, as dictated by the order in α. For testing freeness of such patterns, $\Omega(\sqrt{n})$ adaptive lower bounds exist (due to a simple probabilistic argument) even for the very simple case of $(1, 1)$-freeness, which corresponds to the property of being a one-to-one function.

An interesting point to mention, in this context, is that for testing freeness of forbidden permutations, a major tool that we use is the Marcus-Tardos bound. Namely, that the number of 1's in an $m \times m$ Boolean matrix that does not contain a specific permutation matrix of order k is $O(m)$. For non-permutation patterns, similar bounds are not true in general anymore, but do hold in many cases (or hold in a weak sense, e.g., only slightly more than linear). In such cases, the Marcus-Tardos bound could have allowed relatively efficient testing. However, the lower bounds hinted above for the $(1,1)$-pattern makes the testing problem completely different from that of testing forbidden permutation patterns.

Another area where we have significant gaps in our knowledge is about testing for pattern freeness for functions of bounded or restricted range (for the special case of $(2,1)$-freeness, such a study was initiated by Pallavoor, Raskhodnikova and Varma [31] and followed upon by others [6, 29]). We do know that in the very extreme case, that is, for functions from the line $[n]$ to a constant-sized range, pattern freeness is testable in constant time even for much more general class of forbidden patterns [4].

Lastly, if we restrict our attention to functions $f : [n] \to [n]$ that are themselves permutations, Fox and Wei [21] argued that for some special types of distance measures such as the rectangular-distance and Kendall's tau distance, testing π-freeness can be done in constant query complexity. Testing π-freeness w.r.t. the Hamming or deletion distances is very different, and still remains open for this setting.

Organization. Section 2 contains the notation, important definitions, and a discussion of some key concepts related to testing π-freeness. Section 3 contains a high level overview of an $\tilde{O}(\sqrt{n})$-query algorithm for patterns of length 4. The formal description of our π-freeness tester for permutations π of length $k \geq 4$ and the proof ideas for a special case appear in Section 4. All the missing proofs can be found in the full version [30].

2 Preliminaries and discussion

For a function $f : [n] \to \mathbb{R}$, we denote by $R(f)$ the image of f. We often refer to the elements of the domain $[n]$ as *indices*, and the elements of $R(f)$ as *values*. For $S \subseteq [n]$, $f|_S$ denotes the restriction of f to S. Throughout, n will denote the domain size of the function f.

We often refer to events in a probability space. For ease of representation, we will say that an event E occurs with high probability, denoted "w.h.p.", if $\Pr(E) > 1 - n^{-\log n}$, to avoid specifying accurate constants.

Let $\mathcal{S}_k$ denote the set of all permutations of length k. We view $\pi = (a_1, \ldots, a_k) \in \mathcal{S}_k$ as a function (and not as a cyclus), that is, where $\pi(i) = a_i,\ i \in [k]$. We refer to a_i as the ith value in π, and as the a_i-leg of π. Thus e.g., for $\pi = (4,1,2,3)$, the first value is 4, and the third is 2, while the 4-leg of π is at the first place and its 1-leg is at the second place. We often refer to $\pi \in \mathcal{S}_k$ as a k-pattern.

2.1 Deletion distance vs. Hamming distance

Let $f : [n] \to \mathbb{R}$. The deletion distance of f from being π-free is $\mathrm{Ddist}_\pi(f) = \min\{|S| :\ S \subseteq [n],\ f|_{[n]\setminus S}$ is π-free$\}$. Namely, it is the cardinality of the smallest set $S \subseteq [n]$ that intersects each π-appearance in f. The Hamming distance of f from being π-free, $\mathrm{Hdist}_\pi(f)$ is the minimum of $\mathrm{dist}(f, f') = |\{i :\ i \in [n],\ f(i) \neq f(i')\}|$ over all functions $f' : [n] \to \mathbb{R}$ that are π-free. For $0 \leq \varepsilon < 1$ we say that f is ε-far from π-freeness in deletion distance, or Hamming distance, if $dist_\pi(f) \geq \varepsilon n$, and otherwise we say that f is ε-close to π-freeness, where $dist_\pi(f)$ is the corresponding distance.

▷ Claim 2.1. $\text{Ddist}_\pi(f) = \text{Hdist}_\pi(f)$

Claim 2.1 is extremely important for testing π-freeness, and is what gives rise to *all* testers of monotonicity, as well as π-freeness that are known. This is due to the fact that the tests are really designed for the deletion distance, rather then the Hamming distance. The folklore observation made in Claim 2.2 facilitates such tests, and Claim 2.1 makes the tests work also for the Hamming distance. Due to Claim 2.1, we say that a function f is ε-far from π-free without specifying the distance measure.

Let $\pi \in \mathcal{S}_k$ and $f : [n] \to \mathbb{R}$. A matching of π-appearances in f is a collection of π-appearances that are pairwise disjoint as sets of indices in $[n]$. The following claim is folklore and immediate from the fact that the size of a minimum vertex cover of a k-uniform hypergraph is at most k times the cardinality of a maximal matching.

▷ Claim 2.2. Let $\pi \in \mathcal{S}_k$. If $f : [n] \to \mathbb{R}$ is ε-far from being π-free, then there exists a matching of π-tuples of size at least $\varepsilon n/k$.

All our algorithms have one-sided error, i.e., they always accept functions that are π-free. For functions that are far from being π-free, using Claim 2.2, our algorithms aim to detect some π-appearance, providing a witness for the function to not be π-free. Hence, in the description below, and throughout the analysis of the algorithms, the input function is assumed to be ε-far from π-free.

2.2 Viewing a function as a grid of points

Let $f : [n] \to \mathbb{R}$. We view f as points in an $n \times |R(f)|$ grid G_n. The horizontal axis of G_n is labeled with the indices in $[n]$. The vertical axis of G_n represents the image $R(f)$ and is labeled with the distinct values in $R(f)$ in increasing order, $r_1 < r_2 < \ldots < r_{n'}$, where $|R(f)| = n' \leq n$. We refer to an index-value pair $(i, f(i)), i \in [n]$ in the grid as a *point*. The grid has n points, to which our algorithms do not have direct access. In particular, we do not assume that $R(f)$ is known. The function is one-to-one if $|R(f)| = n$.

Note that if M is a matching of π-appearances in f, then M defines a corresponding matching of π-appearances in G_n. We will always consider this alternative view, where the matching M is a set of disjoint π-appearances in the grid G_n.

2.2.1 Coarse grid of boxes

For a pair of subsets (S, I), where $S \subseteq [n]$ and $I \subseteq R(f)$, we denote by $\mathsf{box}(S, I)$, the subgrid $S \times I$ of G_n with the set of points in G_n (corresponding to f) in this subgrid. In most cases, S and I will be intervals in $[n]$ and $R(f)$, respectively, and hence the name *box*. The *length* of $\mathsf{box}(S, I)$ is defined to be $|S|$. A box is *nonempty* if it contains at least one point and is *empty* otherwise.

Consider an arbitrary collection of pairwise disjoint contiguous value intervals $\mathcal{L} = \{I_1, \ldots I_m\}$, such that $I \subseteq \cup_{i\in[m]} I_i$. The set $\mathcal{L}$ naturally defines a partition of the points in $\mathsf{box}(S, I)$ into m horizontal *layers*, $L_i = \{(j, f(j)) : j \in S, f(j) \in I_i\},\ i \in [m]$. A layer is *multi-valued* if it has two points with different values. It is said to be *single-valued* otherwise.

Assume that, in addition to a set of layers $\mathcal{L}$, we have a partition of S into disjoint intervals $S = \cup_1^m S_i$ where $S_i = [a_i, b_i]$, and $b_i < a_{i+1},\ i = 1, \ldots m-1$, then $\mathcal{S} = (S_1, \ldots S_m)$ partitions $\mathsf{box}(S, I)$ and the points in it, into m vertical *stripes* $\{\text{St}(S)\}_{S\in\mathcal{S}}$ where $\text{St}(S) = \mathsf{box}(S, I)$ contains the points $\{(i, f(i)) : i \in S\}$. The layering $\mathcal{L}$ together with the stripes $\mathcal{S}$ partition $\mathsf{box}(S, I)$ into a coarse grid $G_{m,m}$ of boxes $\{\mathsf{box}(S_i, I_j)\}_{i,j\in[m]}$ that is isomorphic to the

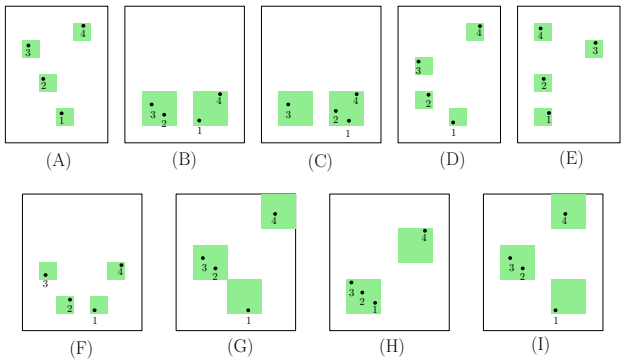


Figure 1 Each rectangle represents a different grid G_n, where the green shaded boxes correspond to some nonempty boxes in those grids. Each figure represents a different configuration type with respect to the appearance of some 4-length pattern. The dots and the numbers indicate possible splittings of the 4 legs of π. *Figure (E) represents the pattern* $(4, 2, 1, 3)$ *and all others represent the pattern* $(3, 2, 1, 4)$.

grid $[m] \times [m]$. Note that $\mathsf{box}(S, I)$ could even be the entire grid G_n. Given such a grid $G_{m,m}$, the layer of $\mathsf{box}(S_i, I_j)$, denoted $L(\mathsf{box}(S_i, I_j))$, is $\mathsf{box}(S, I_j)$ and its stripe, denoted $\mathrm{St}(\mathsf{box}(S_i, I_j))$, is $\mathsf{box}(S_i, I)$.

We say that *layer L is below layer L'*, and write $L < L'$, if the largest value of a point in L is less than the smallest value of a point in L'. For stripes $\mathrm{St}(S), \mathrm{St}(S')$, we write $\mathrm{St}(S) < \mathrm{St}(S')$ if the largest index in S is smaller than the smallest index in S'. For the grid $G_{m,m}$ and two boxes B_1, B_2 in it, $B_1 < B_2$ if $L(B_1) < L(B_2)$ and $\mathrm{St}(B_1) < \mathrm{St}(B_2)$.

2.2.2 Patterns among and within nonempty boxes

Consider a coarse grid of boxes, $G_{m,m}$, defined as above on the grid of points G_n. There is a natural homomorphism from the points in G_n to the nonempty boxes in $G_{m,m}$ where those points fall. For f and a grid of boxes $G_{m,m}$ as above, we refer to this homomorphism implicitly. This homomorphism defines when $G_{m,m}$ contains a π-appearance in a natural way. For example, consider the permutation $\pi = (3, 2, 1, 4) \in \mathcal{S}_4$. We say that $G_{m,m}$ *contains* π if there are nonempty boxes B_1, B_2, B_3, B_4 such that $\mathrm{St}(B_1) < \mathrm{St}(B_2) < \mathrm{St}(B_3) < \mathrm{St}(B_4)$ and $L(B_3) < L(B_2) < L(B_1) < L(B_4)$ (see Figure 1(A)).

▶ **Observation 2.3.** *Let $\mathcal{L}, \mathcal{S}$ be a partition of G_n into layers and stripes as above, with $|\mathcal{L}| = m$, $|\mathcal{S}| = m$ then if $G_{m,m}$ contains π then G_n (and equivalently f) has a π-appearance.*

The converse of Observation 2.3 is not true; G_n may contain a π-appearance while $G_{m,m}$ does not. This happens when some of the boxes that contain the π-appearance share a layer or a stripe. Two boxes are *directly-connected* if they share a layer or a stripe. The transitive closure of the relation *directly-connected* is called *connected*. An arrangement of boxes where every two boxes are connected is called a *connected component*, or simply, a component. The size of a connected component is the number of boxes in it.

For $\pi \in \mathcal{S}_k$, a π-appearance in G_n implies that the k points corresponding to such a π-appearance are in $i \leq k$ distinct components in $G_{m,m}$, where the jth component C_j may contain b_j boxes each containing at least one point of the corresponding π-appearance. We refer to the π-values in the corresponding boxes of the components as *legs*. For example, for $\pi = (3, 2, 1, 4)$, the π-appearance shown in Figure 1(B) is contained in two boxes that share the same layer, and hence form one component. The left box contains the $3, 2$ legs of the π-appearance and other contains the $1, 4$ legs. A different 1-component 2-boxed appearance in the same two boxes has 3 appearing in B_1 and all the other legs in B_2 as in Figure 1(C).

Examples for $\pi = (3, 2, 1, 4)$-appearances with two components C_1, C_2 are illustrated in Figure 1(F) and Figure 1(H). In the first, C_1, C_2 contain 2 boxes each, where C_1 contains the $(3, 4)$ legs of the appearance, each in one box, and C_2 contains the $(1, 2)$ legs. In the second, each component is 1-boxed, where the first contains the $(3, 2, 1)$-legs and the other contains the 4-leg of the appearance. Figure 1(A) contains a $(3, 2, 1, 4)$-appearance in 4 components. Some other possible appearances with 1 component and 3 components are illustrated in Figure 1(B), Figure 1(C), Figure 1(D) and Figure 1(G).

To sum up, each π-appearance in G_n defines an arrangement of nonempty boxes in $G_{m,m}$ that contain the legs of that appearance. This arrangement is defined by the relative order of the layers and stripes among the boxes, and has at most k components. Such a box-arrangement that can contain the legs of a π-appearance is called a *configuration*. Note that there may be many different π-appearances in distinct boxes, all having the same configuration $\mathcal{C}$. Namely, in which, the arrangements of the boxes in terms of the relative order of layers and stripes are identical. So, a configuration is just a set of boxes (or points) in the $k \times k$ grid. An actual set of boxes in $G_{m,m}$ forming a specific type of configuration is referred to as a *copy* of that configuration.

Let $c(k)$ be the number of all possible configurations that are consistent with a π-appearance, for $\pi \in \mathcal{S}_k$. For any fixed π, the number $c(k)$ of distinct types of configurations is constant as shown in the following observation.

▶ **Observation 2.4.** $c(k) \leq 2^{O(k \log k)}$

A configuration $\mathcal{C}$ does not fully specify the way in which a π-appearance can be present. It is necessary to also specify the way the k legs of the π-appearance are partitioned among the boxes in a copy of $\mathcal{C}$. Let $\mathcal{B}$ denote a set of boxes forming the configuration $\mathcal{C}$. Let $\phi : [k] \to \mathcal{B}$ denote the mapping of the legs of the π-appearance to boxes in $\mathcal{B}$, where $\phi(j), j \in [k]$ denotes the box in $\mathcal{B}$ containing the j-th leg of the π-appearance. We say that the copy of $\mathcal{C}$ formed by the boxes in $\mathcal{B}$ contains a ϕ-legged π-appearance.

A configuration $\mathcal{C}$ in which the boxes form $p \geq 2$ components, and that is consistent with a π-appearance, defines $\nu_1, \ldots, \nu_p$ -appearances, respectively, in the p components of $\mathcal{C}$, where ν_j for $j \in [p]$ is the subpermutation of π that is defined by the restriction of π to the j-th component. In addition, $\mathcal{C}$ defines the corresponding mappings $\phi_j, j = 1, \ldots p$, of the corresponding legs of each ν_j to the corresponding boxes in the jth component. For example, consider $\pi = (3, 2, 1, 4)$ and the box arrangement shown in Figure 1(F). That arrangement has two connected components: one that contains B_1, B_4 and the other that contains B_2, B_3, where we number the boxes from left to right (by increasing stripe order). Further, the (only) consistent partition of the legs of π into these boxes is $\pi(i) \in B_i,\ i \in [4]$. In particular, it means that the component formed by B_1, B_4 contains the $3, 4$ legs of π and the component formed by B_2, B_3 contains the $2, 1$ legs of π. Thus, in terms of the discussion above, the component formed by B_1, B_4 has a $\nu_1 = (1, 2)$-appearance (corresponding to the $3, 4$ legs of π), with leg mapping ϕ_1 mapping the 1-leg into B_1 and the 2-leg into B_4. Similarly, the component formed by B_2, B_3 has a $\nu_2 = (2, 1)$-appearance (corresponding to the $2, 1$ legs of π) with corresponding leg mapping ϕ_2 that maps the 2-leg into B_2 and the 1-leg into B_3. Note that the converse is also true: every ν_1 appearance in the component $B_1 \cup B_4$, with a leg-mapping ϕ_1 (that is, in which the $1, 2$ legs are in B_1, B_4 respectively), in addition to a ν_2 appearance in $B_2 \cup B_3$ with the leg-mapping ϕ_2, results in a π-appearance in $G_{m,m}$.

This latter comment leads to the crucial observation that if π defines the corresponding $\nu_1, \ldots, \nu_p$ appearances in the p components of the configuration $\mathcal{C}$, then, *any* $\nu_1, \ldots, \nu_p$-appearances in the p components of any copy of $\mathcal{C}$ with consistent leg-mappings is a π-appearance in $\mathcal{C}$. This is formally stated below.

▶ **Definition 2.5.** *Let $\nu \in \mathcal{S}_r$. Let $B_1, \ldots, B_p$ be a set of boxes forming* one *component C and $\phi : [r] \mapsto \{B_1, \ldots, B_p\}$ be an arbitrary mapping of the legs of a ν-appearance to boxes. We say that C has a ϕ-legged ν-appearance if there is a ν-appearance in $\cup_{j=1}^{p} B_j$ in which for each $i \in [r]$, the i-th leg of ν appears in the box $B_{\phi(i)}$.*

▶ **Observation 2.6.** *Let $\pi \in \mathcal{S}_k$ and assume that there exists a π-appearance in G_n that in the grid of boxes $G_{m,m}$ forms a configuration $\mathcal{C}$ that contains t components $C_1, \ldots, C_t$, with C_j having r_j boxes, $j = 1, \ldots, t$ respectively. Let the restriction of this π-appearance to $C_1, \ldots, C_t$ define the permutation patterns $\nu_1, \ldots \nu_t$ in $C_1, \ldots, C_t$, with leg mappings $\phi_1, \ldots \phi_t$, respectively.*

Then, any collection of configuration copies of C_j, $j = 1, \ldots t$, for which $\cup_1^t C_j$ is a copy of $\mathcal{C}$, and ϕ_j-legged ν_j-appearances in C_i, $i = 1, \ldots, t$, defines a π-appearance in $\cup_1^t C_j$. ⌟

2.3 Erasure-resilient testing

Erasure-resilient (ER) testing, introduced by Dixit, Raskhodnikova, Thakurta and Varma [16], is a generalization of property testing. In this model, algorithms get oracle access to functions for which the values of at most α fraction of the points in the domain are erased by an adversary, for $\alpha \in [0, 1)$. As part of our algorithm for testing π-freeness for $\pi \in \mathcal{S}_k$ for $k \geq 4$, we call testers for smaller subpatterns on sub-regions of the grid G_n which may be defined by, say, $\mathsf{box}(S, I)$ for some $S \subseteq [n], I \subseteq R(f)$. In this case, the only access to points in $\mathsf{box}(S, I)$ is by sampling indices from S and checking whether their values fall in I. If the values do not fall in I, we can treat them as erasures. Given the assurance that the number of points falling in $\mathsf{box}(S, I)$ is a constant fraction of $|S|$, we can simply run ER testers on $f|_S$ to test for these smaller subpatterns.

3 High level description of the basic algorithm for $\pi \in \mathcal{S}_4$

Many of the high level ideas in the design of our π-freeness tester of complexity $\tilde{O}(n^{o(1)})$ are described in this section. For simplicity, we describe first the ideas behind a $\tilde{O}(\sqrt{n})$-query tester for π-freeness of $\pi \in \mathcal{S}_4$. Towards the end of this section, we briefly describe how to generalize these ideas to obtain the query complexity of $\tilde{O}(n^{o(1)})$ and for longer constant-length permutations. For simplicity, we assume in what follows that the input function $f : [n] \to \mathbb{R}$ is one-to-one. The algorithm for functions that are not one-to-one differs in a few places and these are explained in Section 4.1.

For the purposes of this high level description, we fix the forbidden permutation $\pi = (3, 2, 1, 4)$. The same algorithm works for any $\pi \in \mathcal{S}_4$. We view f as an (implicitly given) $n \times |R(f)|$ grid G_n consisting of points $(i, f(i))$ for $i \in [n]$, where, in particular, $R(f)$ is neither known nor bounded. Our first goal is to approximate G_n by a coarse grid of boxes $G_{m,m}$, as described above, for $m = \sqrt{n}$. This is done by querying f on $\tilde{\Theta}(m)$ random indices, after which we obtain a partition $\mathcal{L}$ of $R(f)$ into $m' = \Theta(m)$ horizontal layers (value intervals). Then we partition the index set $[n]$ into m' contiguous intervals $\{S_i\}_{i=1}^{m'}$ of equal length. This results in a grid $G_{m',m'}$ in which we estimate the density of each box as the number of sampled points falling in that box, normalized by n/m'. A box $\mathsf{box}(S_i, I_j), i, j \in [m']$ will be tagged as *dense* if it contains $\Omega(1)$ fraction of sampled points. All of the above takes $\tilde{O}(m) = \tilde{O}(\sqrt{n})$ queries, for the above choice of m. It satisfies the following properties with high probability:

- Each layer, that is $\mathsf{box}([n], I_j)$, $j \in [m']$, has approximately the same number of points.
- It is either the case that the dense boxes contain all but an insignificant fraction of the points in G_n, or the total number of marked boxes is larger than $m' \log n$.

Next, we use the following lemma of Marcus and Tardos.

▶ **Lemma 3.1** ([26]). *For any $\pi \in \mathcal{S}_k$, $k \in \mathbb{N}$, there is a constant $\kappa(k) \in \mathbb{N}$ such that for any $r \in \mathbb{N}$, if a grid $G_{r,r}$ contains at least $\kappa(k) \cdot r$ marked points, then it contains a π-appearance among the marked points.*

Let $\kappa = \kappa(4)$. Using Lemma 3.1, we may assume that there are at most $\kappa \cdot m'$ non-empty boxes in $G_{m',m'}$, as otherwise, we already would have found a π-appearance in $G_{m',m'}$, which by Observation 2.3, implies a π-appearance in G_n and in f as well. Hence, as a result of the gridding, if we do not see a π-appearance among the sampled points, the second item above implies that there are $\Theta(m')$ dense boxes in $G_{m',m'}$ and that these boxes cover a large fraction of the points of G_n.

An averaging argument implies that, for an appropriate constant $d = d(\varepsilon)$, only a small constant fraction of layers (or stripes) contain more than d nonempty boxes. Therefore, since the grid G_n is ε-far from being π-free, the restriction of G_n to the layers and stripes that contain at most d boxes each, is also ε'-far from π-free for a large enough constant $\varepsilon' < \varepsilon$. This implies that G_n restricted to the points in dense boxes that belong to layers and stripes containing at most d dense boxes each, has a matching M of π-appearances of size at least $\varepsilon' n/4$. We assume in what follows that this is indeed the situation.

An important note at this point, is that every dense box B is contained in $O(d^3)$ (that is, constantly many) 1-component configurations with at most 4 dense boxes. This implies that there are $O(m)$ such copies of 1-component configurations in $G_{m',m'}$.

Recall that every π-appearance in M defines a configuration of at most 4 components in $G_{m',m'}$. Hence, the matching M of size $|M| = \Omega(n)$ can be partitioned into 4 sub-matchings $M = M_1 \cup M_2 \cup M_3 \cup M_4$, where M_i, $i = 1, \ldots, 4$ consists of the π-appearances participating in configurations having exactly i components. Since $|M| = \Omega(n)$ it follows that at least one of M_i, $i = 1, 2, 3, 4$ is of linear size. Now, any π-appearance in M_4 is an appearance in 4 distinct dense boxes in $G_{m',m'}$, where no two share a layer or a stripe. In that case, such an appearance can be directly detected from the tagged $G_{m',m'}$ with no further queries.

The description of the rest of the algorithm can be viewed as a treatment of several independent cases regarding which of the constantly many configuration types contributes the larger mass out of the $\Omega(n)$ π-appearances in $M_1 \cup M_2 \cup M_3$. There are only two significant cases, but to ease the reader, we split these two cases into the more natural larger number of cases, and observe at the end that most cases can be treated conceptually in the same way.

Case 1: Assume that $|M_1| \geq \varepsilon' n/3$, and further, for simplicity, that a constant fraction of the π-appearances in M_1 are in a single-box component. Then, on average, a dense box, out of the $\Theta(m')$ dense boxes, is expected to contain at least $\Theta(n/m') = \Theta(m') = \Theta(\sqrt{n})$ many π-appearances. Thus a random dense box B is likely to have $\Theta(\sqrt{n})$ many π-appearances, and hence, making queries to all points of such a box will enable us to find one such π-appearance. This takes an additional $n/m' = \Theta(\sqrt{n})$ queries, which is within the query budget.

Next, consider the case that a constant fraction of the π-appearances in M_1 belong to a configuration $\mathcal{C}$ that has more than one dense box (but only one connected component). By a similar argument, a random dense box is expected to participate in at least $\Theta(n/m')$ many π-appearances of copies of configuration-type $\mathcal{C}$. Since each dense box is part of at most $O(d^3)$ (constantly many) connected components of at most 4 dense boxes, sampling a random dense box B and querying all the indices in each of the components that contain at most 4 dense boxes and involve B, is likely to find a π-appearance with high probability. Each connected component is over at most $4n/m'$ indices, resulting in $O(n/m')$ queries.

Case 2: $|M_3| \geq \varepsilon' n/3$, and assume first that a constant fraction of the members in M_3 belong to copies of a configuration $\mathcal{C}$ of 3 components B_1, B_2, B_3, where each one is a single box. For our current working example, $\pi = (3,2,1,4)$, assume further that B_1 contains the $3,2$ legs of a π-appearance and B_2, B_3 contain its 1 and 4 legs, respectively (see Figure 1(G) for an example). In this case B_1 is not $(2,1)$-free (as B_1 contains the $(3,2)$-subpattern of π).

By an averaging argument, it follows that there is a dense box B for which: (a) B is far from $(2,1)$-free, and (b) there are corresponding dense boxes B_2, B_3 that, together with B, form a copy of the configuration $\mathcal{C}$. Now, a test follows easily. We test every dense box for $(2,1)$-freeness, which can be done in $O(\log n)$ queries per box, and hence in $\tilde{O}(m)$ in total. Then, by the guarantee above we will find the corresponding B and B_2, B_3 and a π-appearance in it (by Observation 2.6 with the trivial mapping).

A similar argument holds for a 3-component configuration $\mathcal{C}'$ in which one component contains more than one box, and for any configuration of 3 components.

Case 3: Assume now that $|M_2| \geq \varepsilon' n/3$, and that the corresponding configurations of the π-appearances in M_2 contain two single-box components B_1, B_2, where B_1 holds the first 3 legs of π and B_2 holds the 4-th leg. E.g., For $\pi = (3,2,1,4)$, the configuration $\mathcal{C}$ contains two boxes B_1, B_2 where B_1 contains the subpattern $(3,2,1)$ and B_2 is any nonempty box such that $B_1 < B_2$, (see Figure 1(H) for an illustration). An averaging argument, as made in Case 2, shows that there is a dense box B_1 for which (a) B_1 is far from $(3,2,1)$-free, and (b) there is a corresponding dense box B_2 that, together with B_1, forms a copy of the configuration $\mathcal{C}$. This suggests a test that is conceptually similar to the test in Cases 1 and 2. We test each box for being $(3,2,1)$-free. This can be done in $\operatorname{polylog} n$ queries (e.g., [8]). Then once finding a $(3,2,1)$ in B_1 for which (a) and (b) hold, $B_1 \cup B_2$ contains a π-appearance.

We note here that for the example above, we ended by testing for $(3,2,1)$-freeness which is relatively easy. For a different configuration or π, we might need to test B_1 for a different $\nu \in \mathcal{S}_3$, but this can be done for any $\nu \in \mathcal{S}_3$ using $O(\operatorname{polylog} n)$ queries [28]. Hence the same argument and complexity guarantee hold for any 2-component configuration $\mathcal{C}$ as above.

Case 4: A more complicated situation arises when $|M_2| \geq \varepsilon' n/3$, and the corresponding configurations of the π-appearances in M_2 are formed of two components D, B, with D holding 3 legs of π in 2 or 3 boxes (rather than in one box as in Case 3). E.g., $\pi = (4,2,1,3)$, and the configuration $\mathcal{C}$ as illustrated in Figure 1(E).

By a similar averaging argument to that made in Case 2, it follows that there is a dense box B_1 for which (a) there are dense boxes B_2, B_3 forming a copy D' of D with B_1, and a dense box B such that the configuration formed by D', B is a copy of $\mathcal{C}$, and (b) there are $\Omega(n/m) = \Omega(\sqrt{n})$ ϕ-legged $(3,2,1)$-appearances in D', where ϕ is consistent with the leg mapping that is induced by the configuration $\mathcal{C}$. This implies a conceptually similar test to that of the simpler Case 3 above - we test each of the $O(m)$ components D for $(3,2,1)$-freeness, and then with the existence of the corresponding box B we find a π-appearance. However, this is not perfectly accurate: the algorithm for finding $\nu = (3,2,1)$ in D', although efficient, might find a $(3,2,1)$-appearance where the 3 legs appear in B_1 or in $B_1 \cup B_2$. But this does not extend with B to form a π-appearance, as the leg mapping is not consistent with the one that is induced by $\mathcal{C}$. Namely, unlike before, we do not only need to find a ν-appearance in D but rather a ϕ-legged ν-appearance with respect to a fixed mapping ϕ (that in this case maps each leg to a different box in the component D').

To resolve the problem we need to efficiently test ϕ-legged ν-appearances in multi-boxed components. This, however, we currently do not know how to do. Instead, we design a test that either finds a ϕ-legged ν-appearance, or finds the original π-appearance. This is done using the procedure $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ that will be described in Section 4.

Case 5: The last case that we did not consider yet is when most of the π-appearances are in a configuration containing more than one component, with at least two components containing two (or more) legs each. For $\pi \in \mathcal{S}_4$ the only such case is when the configuration $\mathcal{C}$ contains exactly two components, each containing exactly two legs of π. Returning to our working example with $\pi = (3, 2, 1, 4)$, such examples are depicted e.g., in Figure 1(B) or 1(F). For the explanation below, we will discuss the case that the configuration $\mathcal{C}$ is as in Figure 1(F). Namely, it contains components D_1 that is above D_2, with two boxes each $D_1 = \{B_1, B_4\}$ and $D_2 = \{B_2, B_3\}$, and so that every box contains exactly one leg of π (boxes are numbered by order from left to right in $G_{m',m'}$). Our goal is to find two copies D'_1, D'_2 of the components D_1, D_2 respectively, that form a copy of $\mathcal{C}$, and to find a ϕ_1-legged appearance of $(1, 2)$ in D'_1, and a ϕ_2-legged appearance of $(2, 1)$ in D'_2, so that Observation 2.6 will implies that these two appearances form together a π appearance. Indeed, an averaging argument shows that there are D'_1, D'_2 as above, with D'_i containing $\Omega(n/m)$ ϕ_i-legged appearances of ν_i for $i = 1, 2$. However, we do not know that sampling a pair D'_1, D'_2 in some way, will result in such a good pair. Rather, we are only assured of the existence of only one such pair! Hence, in this case we need to test *every* component D', and for every $\nu \in \mathcal{S}_2$, and for every leg mapping ϕ, for a ϕ-legged ν-appearance in D' in order to find such an asserted pair of components. Such restricted ν-appearances can be tested in $O(\log n)$ queries per component. Therefore, this takes $\tilde{O}(m)$ queries in total. The same argument holds for any $\pi \in \mathcal{S}_4$, and for every configuration that is consistent with Case 5.

Concluding remarks

- At some places in the algorithm above, we had to test for ν-appearances (or restricted ν-appearances) in "dense" subgrids of G_n. For this, we need that all our algorithms are ER, which will be implicitly clear from the description. We also need to take care of reducing the total error when we do non-constant number of by tests, or want to guarantee a large success probability for a large number of events - this is done by trivial amplification that results in a multiplicative $\operatorname{polylog} n$ factor.
- In Case 1, we reduced the problem of finding a π-appearance in G_n, that is assumed to be ε-far from π-free, to the same problem on a subrange of the indices (formed by a small component) of size $\Theta(n/m)$ (with a smaller but constant distance parameter $\varepsilon' < \varepsilon$). For the setting of $m = \sqrt{n}$, solving the problem on the reduced domain was trivially done by querying all indices in the subrange. In the general algorithm, where our goal is a query complexity of $n^{o(1)}$, we set $m = n^\delta$ for an appropriately small δ and apply self-recursion in Case 1.
- In Cases 2, 3, 4 we end up testing ν-freeness for $\nu \in \mathcal{S}_2 \cup \mathcal{S}_3$ in dense boxes, or ϕ-legged ν-freeness of such ν in components of multiple dense boxes. An average argument shows that this can simply be done by sampling one box or component, and making queries to all indices therein. This however, is true only for $\pi \in \mathcal{S}_4$.
  Case 5 is different: here sampling a small number of components does not guarantee an expected large number of the corresponding appearances. This is the reason that we need to test *all* components with at most 2 dense boxes, for ϕ-legged ν-freeness, and for every $\nu \in \mathcal{S}_2$ and leg mapping ϕ. Algorithm $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ can do this for any

$\nu \in \mathcal{S}_2 \cup \mathcal{S}_3$ in n^δ queries for an arbitrary small constant δ. Since we have to do it in Case 5, we may do the same in cases 2, 3, 4 as well! As a result, the algorithm above will contain only two cases: Case 1 where we reduce the problem to the same problem but on a smaller domain, and the new Case 2 where we test *every* small component for ϕ-legged ν-appearance for every $\nu \in \mathcal{S}_2 \cup \mathcal{S}_3$ and every leg mapping ϕ – namely a case in which we reduce the problem to testing (restricted appearances) for smaller patterns.

- In view of the comment above, the idea behind improving the complexity to n^δ for constant $0 < \delta < 1$ is obvious: Choosing $m = n^{\delta/2}$ will result in an $m \times m$ grid, where Layering can be done in $\tilde{O}(n^{\delta/2})$ queries. Then, Case 2 will be done in an additional n^δ queries by setting a query complexity for $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ to be $n^{\delta/2}$ per component. The self-recursion in Case 1 will result in the same problem over a range of n/m. For the fixed $m = n^{\delta/2}$, this will result in a recursion depth of $2/\delta$, after which the domain size will drop down to m and allow making queries to all corresponding indices. This results in a total of $\tilde{O}(n^\delta)$ queries, including the amplification needed to account for the accumulation of errors and deterioration of the distance parameter at lower recursion levels.
- **Generalized testing and testing beyond $k = 4$.** Applying the same ideas to $\pi \in \mathcal{S}_k$, $k \geq 5$ works essentially the same way, provided we can test for ϕ-legged ν-freeness of $\nu \in \mathcal{S}_r$ for $r < k$. This we know how to do for $\nu \in \mathcal{S}_3$ but not beyond. In particular, one difficulty is that after gridding, a superlinear number of non-empty boxes do not guarantee such appearance, as Lemma 3.1 does not apply. However, for our goal of testing π-freeness for $\pi \in \mathcal{S}_k$, we can relax the task of finding ϕ-legged ν-freeness of $\nu \in \mathcal{S}_r, r \leq k$ to the following problem which we call "generalized-testing ν w.r.t. π", denoted $\mathsf{GeneralizedTesting}_\pi(\nu, D, \phi)$: The inputs are a permutation $\nu \in \mathcal{S}_r$, a component D, and a leg mapping ϕ. Our goal is to find either a ϕ-legged ν-appearance **OR** a π-appearance in D. The way we solve this generalized problem is very similar, conceptually, to the way we solve the unrestricted problem. This will be defined formally in the next section.

4 Generalized testing of forbidden patterns

In this section, we formally define the problem of testing (or deciding) freeness from ν-appearances with a certain leg-mapping. We then provide an algorithm for a relaxation of this testing problem, which we call $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$. This will imply, in turn, our algorithm for testing π-freeness.

Recall that G_n denotes the $n \times n$ grid that represents the input function $f : [n] \to \mathbb{R}$. Let $G_{\ell,\ell}$ be a partition of G_n into a grid of boxes for arbitrary $\ell \geq 1$, and D be a connected component in $G_{\ell,\ell}$ containing t boxes $B_1, \ldots, B_t$. Let $\nu \in \mathcal{S}_r$, and let $\phi : [r] \mapsto [t]$ be an arbitrary mapping of the legs of ν into the boxes of D, where $t \leq r$. We say that $1 \leq i_1 \leq \ldots \leq i_r \leq n$ is a ϕ-legged ν-appearance if $(i_1, \ldots, i_r)$ forms a ν-appearance in G_n such that $(i_j, f(i_j)) \in B_x$ for $x = \phi(j)$, $j = 1, \ldots, r$. That is, the corresponding legs of the ν-appearance are mapped into the boxes given by ϕ. For example, consider Figure 1(B), $\nu = (3, 2, 1, 4)$, and D the component formed by the two boxes in the same layer. The function ϕ maps the 3 and 2-legs of the appearance to the left box and the 1 and 4-legs to the right box.

For ϕ, ν as above, the property of containing a ϕ-legged ν-appearance is a generalization of the problem of pattern-freeness; taking $\ell = 1$, $G_{\ell,\ell}$ is just G_n itself viewed as one single box D. Any ν-appearance in G_n is a ϕ-legged ν-appearance for the constant function $\phi \equiv D$.

The complexity of testing ϕ-legged ν-freeness is not clear and was not previously explicitly studied. For permutations of length 2 as well as for longer monotone permutations, its complexity is identical to unrestricted testing and can be done in $\text{polylog}\, n$ queries. For length 3 non-monotone permutations it can be done in $\text{polylog}\, n$ queries using similar ideas as in [28]. For larger $k \geq 4$ the complexity is open.

While testing ϕ-legged ν-freeness is interesting in its own, we encounter it only as a sub-problem in the testing of standard π-freeness. This motivates the following definition.

Let $\pi \in \mathcal{S}_k$, G_n be fixed. Further let $G_{\ell,\ell}$ be a decomposition of G_n into boxes and D be a t-boxed *one component* over boxes $(B_1, \ldots, B_t)$, $t \leq r$, in $G_{\ell,\ell}$. For inputs $\nu \in \mathcal{S}_r$, and $\phi : [r] \mapsto \{B_1, \ldots, B_t\}$, the problem $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$, is to find a ϕ-legged ν-appearance in D **OR** to find any (unrestricted) π-appearance. The distance of D from being ϕ-legged ν-free is defined naturally. The distance for the generalized testing w.r.t π, $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$, is defined as the distance from being free of ϕ-legged ν-appearance regardless of the π-appearances.

Our algorithm for $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$ is called $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ (presented in Algorithm 1), where $\pi \in \mathcal{S}_k$ is fixed. The algorithm has inputs: $\nu \in \mathcal{S}_r$, $r \leq k$, a component D in a decomposition $G_{\ell,\ell}$ of G_n that contain boxes $B_1, \ldots, B_t$, $t \leq r$, and a function $\phi : [r] \to \{B_1, \ldots B_t\}$. In addition, it gets a distance parameter ε, and a free parameter m that that is used to control the query complexity which will be $\tilde{O}(m^r)$ for $m \geq n^{\Omega(1/\log\log\log n)}$. With high probability, the algorithm either finds a π-appearance or a ϕ-legged ν-appearance in D, if D is ε-far from being free of ϕ-legged ν-appearances.

The algorithm is recursive. A recursion is done by reducing ν to smaller length patterns, and/or self-reduction to the same ν but on a smaller length box D'. The base cases are when the length of D is small enough to allow queries to all indices in D, or when $\nu \in \mathcal{S}_2$, in which case the algorithm is reduced to testing monotonicity.

The permutation $\pi \in \mathcal{S}_k$ is fixed and hardwired into the algorithm. G_n is assumed to fixed and not part of the recursion. The actual input is the t-boxed component D in a grid of boxes $G_{\ell,\ell}$ over G_n (that is, a sub-function of the original function f).

In general, the complexity of testing ϕ-legged ν-appearances may depend on ϕ and ν. Our algorithm does not use any structure of π. The only role of π in the algorithm is to ensure that after gridding D, the resulting grid $D_{m,m}$ contains only a linear in m number of marked boxes (as otherwise, by Lemma 3.1, a π-appearance is guaranteed).

Finally, and as we already pointed out, the algorithm for $\mathsf{GeneralizedTesting}_\pi(\nu, \phi, D)$ will allow us to test for π-freeness in G_n by calling $\mathsf{AlgTest}_\pi(\pi, \phi, G_n, m, \varepsilon)$, where ϕ is the constant function that maps each of the k legs to the single box G_n.

The following theorem asserts the correctness of $\mathsf{AlgTest}_\pi(\pi, \phi, G_n, m, \varepsilon)$ and the corresponding query complexity. We assume that ℓ and a corresponding component D in the grid of boxes $G_{\ell,\ell}$, inside G_n, is given.

▶ **Theorem 4.1.** *Let $\varepsilon \in (0,1)$ and $\nu \in \mathcal{S}_r$, $r \leq k$. Let $D = \mathsf{box}(S, I)$ be a connected component in $G_{\ell,\ell}$, composed of boxes $B_1, \ldots, B_t$, $t \leq r$ and $\phi : [r] \to \{B_1, \ldots B_t\}$. If D is ε-far from being free of ϕ-legged ν-appearances, then $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ finds, either a ϕ-legged ν-appearance or a π-appearance, with probability at least $1 - o(1)$. Its query complexity is $\tilde{O}(n^{\eta r})$, for $\eta \in (0,1)$ and $m = n^\eta$.*

We note that since $\mathsf{AlgTest}_\pi(\nu, \phi, D, m, \varepsilon)$ either finds π or a ϕ-legged ν-appearance in D, then if D is free of ϕ-legged ν-appearance, the algorithm will never return such an appearance. As a result, the corollary below follows by calling $\mathsf{AlgTest}_\pi(\pi, \phi, G_n, m, \varepsilon)$.

Algorithm 1 $\mathsf{AlgTest}_{\pi}(\nu, \phi, D, m, \varepsilon)$.

Input: pattern $\nu \in \mathcal{S}_r$; D is a component in $G_{\ell,\ell}$ containing boxes $B_1, \ldots, B_t$ for $1 \leq t \leq r$; the function $\phi : [r] \mapsto \{B_1, \ldots, B_t\}$ is a leg-mapping of ν into the boxes of D.

Goal: Find a ϕ-legged ν-appearance **or** an unrestricted π-appearance, in D.

Let $S = \cup_{i\in[r]} \mathrm{St}(B_{\phi(i)})$ and $I = \cup_{i\in[r]} L(B_{\phi(i)})$ define $\mathsf{box}(S, I)$ in $G_{\ell,\ell}$ that contains D (or the subcomponent of D where a ν-appearance should be found).

Base cases:

1. If $|S| \leq m$ query all indices in S. **Output** "π-appearance is found" or "ϕ-legged ν-appearance is found" if one of these is found, otherwise **output** "not found".
2. If $r \leq 2$, use the test for restricted appearance of 2-patterns. This is easy and will not be described in this manuscript. If $r = 1$, **output** "ϕ-legged ν-appearance is found" if $\phi(1)$ contains a point. If $k = 1$, **output** "π is found" if D contains a point.
3. If the sampled points in D already contains a ϕ-legged ν-appearance, or contains a π-appearance then **output**: "found ϕ-legged ν-appearance" or "found π-appearance" respectively.

Gridding D: We set $\beta = \frac{\varepsilon}{200k\kappa(k)}$. Call Gridding on $\mathsf{box}(S, I)$ with parameters: (S, I, m, β). The details on the procedure Gridding can be found in the full version [30]. As a result, we obtain a decomposition of $\mathsf{box}(S, I)$ (and D) into an $m' \times m'$ grid of sub-boxes $D_{m',m'}$, $m \leq m' \leq 2m$, where a subset of boxes are *marked* and a subset of the marked boxes are *dense*.

Simple case: If $D_{m',m'}$ contains more than $\kappa(k) \cdot m'$ marked sub-boxes then **output** "found π-appearance".

Sparsification: Delete every stripe and layer in $D_{m',m'}$ that contains more than $d = 100k\kappa(k)/\varepsilon$ marked sub-boxes. Delete all non-dense sub-boxes.

Multi-component configurations: For every possible configuration $\mathcal{C}$ of sub-boxes that is consistent with ϕ such that $\mathcal{C}$ forms components $\mathcal{C}_1, \ldots \mathcal{C}_p$, $p > 1$, the pair (ν, ϕ) define sub-permutations of ν : $\nu_1, \ldots, \nu_p$ and subfunctions of ϕ : $\phi_1, \ldots, \phi_p$ on $\mathcal{C}_1, \ldots \mathcal{C}_p$ respectively.

Let c denote the number of distinct configurations with at most r components.

Recursively **call** $\mathsf{AlgTest}_{\pi}(\nu_i, \phi_i, D_i, m, \varepsilon')$ with distance parameter $\varepsilon' = \frac{9\varepsilon}{10kcr^2 \cdot r! \cdot (2d)^r}$ for *every* component D_i, where D_i is a copy of $\mathcal{C}_i$ in $D_{m',m'}$, and is contained in D. Note that the recursive call is done for smaller length patterns ν_is.

Output "found ϕ-legged ν-appearance" if for a copy $(D_1, \ldots, D_p)$ of $(\mathcal{C}_1, \ldots, \mathcal{C}_p)$, the region D_i contains a ϕ_i-legged ν_i-appearance for each $i = 1, \ldots, p$. Or **output** "found π-appearance" if a π-appearance is found among the sampled points.

One component configurations: Let $\mathcal{A}$ be the set of all possible copies of configurations $\mathcal{C}$ in $D_{m',m'}$ for which $\mathcal{C}$ forms *one* component, and that are contained in D. Note that $\mathcal{A}$ contains $O(m)$ such copies.

loop $\frac{\log^3 n}{\varepsilon^r}$ times:

Sample a member D' from $\mathcal{A}$. For each ϕ-consistent mapping ϕ', recursively **call** $\mathsf{AlgTest}_{\pi}(\nu, \phi', D', m, \varepsilon'')$ with $\varepsilon'' = \frac{9\varepsilon}{20k \cdot (2d)^r \cdot (r-1)! \cdot r^r}$.

▷ A mapping ϕ' from the legs of a ν-appearance to the sub-boxes in $D_{m',m'}$, is ϕ-consistent if for each i-leg, $i = 1, \ldots, r$, the sub-box $\phi'(i)$ is contained in the box $\phi(i)$.

end loop

If no output is declared in any of the previous steps, **output** "not-found".

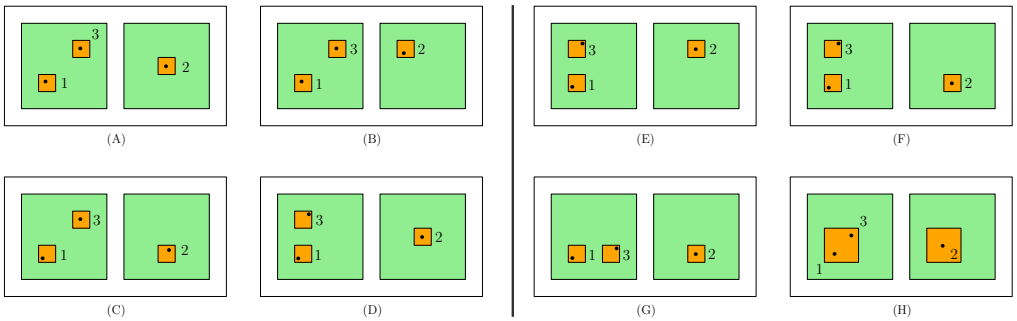


Figure 2 Different configurations for (1,3,2)-appearances.

▶ **Corollary 4.2.** *There is a 1-sided error test for π-freeness, for every $\pi \in \mathcal{S}_k$ of query-complexity $\tilde{O}(n^{\delta k})$ for arbitrary $\delta > 1/\log\log\log n$.*

4.1 Proof of Correctness

In this section, we give a description of the algorithm and the proof sketch for the first non-base case of testing ϕ-legged ν-freeness of $\nu \in \mathcal{S}_r$, $r = 3$, with respect to an arbitrary $\pi \in \mathcal{S}_k$ and for some fixed $k \geq 4$. The full proof of Theorem 4.1 can be found in the full version [30].

4.1.1 An example for $\nu \in \mathcal{S}_3$

For this exposition, we fix $\nu = (1, 3, 2)$, and D being composed of 2 boxes B_1 B_2, in the same layer, where B_1 is to the left of B_2, and ϕ maps the $1, 3$ legs of ν to B_1, and the 2-leg to B_2. See Figure 2(D) for illustration of one such case. In the figure, the green boxes represent B_1 and B_2. The orange boxes indicate the subboxes in the finer grid formed when gridding is called on the green boxes.

We note that Figure 2(D) illustrates the hardest case for $\nu \in \mathcal{S}_3$. There are additional one-component configurations in which the boxes are in the same stripe or layer, but these turn out to be much easier. We will set $m = m(n)$ to be defined later and express the complexity as a function of m. We do not specify π since, as explained above, π is only needed at Step 4 when the number of marked boxes is superlinear in m' in some recursive call. The same argument holds for any $\pi \in \mathcal{S}_k$, $k \geq 4$.

Algorithm for $\nu = (1, 3, 2)$ and B_1, B_2 as above:

1. We assume that B_1, B_2 have at most $s \leq n$ indices each, and that the distance of $B_1 \cup B_2$ from being free from ϕ-legged ν-appearance is at least $\varepsilon = \Omega(1)$. In particular B_1, B_2 are dense. We start in Step 3 of Algorithm 1 where we do gridding of the union of B_1 and B_2 into a $m' \times m'$ grid, $D_{m',m'}$, of sub-boxes (each having roughly at most s/m' indices), where $m \leq m' \leq 2m$. We either find a π-appearance among the sampled points or we may assume, after Steps 4, 5 that there are $O(m')$ dense sub-boxes in $B_1 \cup B_2$ and that each layer and each stripe contains $O(1)$ dense boxes. This is obtained by an averaging argument and is described in the formal proof in the full version. It shows that if $B_1 \cup B_2$ contains a large matching of ϕ-legged ν-appearances, then so does the restricted domain after deleting points from non-dense boxes, and deleting layers and stripes that contain too many dense boxes. This steps takes $\tilde{O}(m)$ queries (the complexity of gridding).
2. A ϕ-legged ν-appearance in $B_1 \cup B_2$ can be in 8 possible configurations in the grid $D_{m',m'}$, as depicted in Figure 2. Consider first $\mathcal{C}_1, \ldots, \mathcal{C}_4$ as in Figure 2(A)-(D), that form 2 or 3 components each. For these, a ϕ-legged ν-appearance in $B_1 \cup B_2$ decomposes into two

or three subpatterns, and for which any restricted appearances in the corresponding components results in a ϕ-legged ν-appearance. E.g., in Figure 2(B) the configuration $\mathcal{C}_2$ contains one component $D_1 = (B_{1,3}, B_{2,2})$, where $B_{1,3} \in B_1, B_{2,2} \in B_2$, and another single boxed component $B_{1,1} \in B_1$, where $B_{i,j}$ is the orange subbox contained within the green box B_i and such that the j-th leg belongs to $B_{i,j}$ for $i \in [2], j \in [3]$.

In Step 6, we test each of the $O(m)$ many copies of D_1 for a ϕ'-legged $(2,1)$-appearance for which $\phi'(2) = B_{1,3}$ and $\phi'(1) = B_{2,2}$. Then for any such D_1-copy in which a such ϕ'-legged $(2,1)$-appearance is found, any nonempty dense box $B_{1,1}$ forming with D_1 a copy of $\mathcal{C}_2$ results in a ϕ-legged ν-appearance.

Since this is a reduction to generalized 2-pattern appearance, the recursion stops here with $O(\log n)$-complexity per copy of D_1. Hence, altogether this will contribute a total of $\tilde{O}(m)$ queries. The same argument holds for any of $\mathcal{C}_i,\ i = 1,2,3,4$.

If a desired appearance is found, then clearly a correct output is produced.

Finally, if indeed (B_1, B_2) contains $\Omega(s)$ (that is, linear in the length of $B_1 \cup B_2$) many ϕ-legged ν-appearances that are consistent with one of the configurations $\mathcal{C}_i, i = 1,2,3,4$, then there will be such a D_1 and corresponding $B_{1,1}$ that together contribute $\Omega(s/m)$ (that is – linear in the domain size of D_1) such subpattern appearances by an averaging argument.

We note that for the more general case of $r > 3$, the reduction will be done in higher complexity per component (that is dependent on m rather than just $O(\log n)$).

3. Consider now $\mathcal{C}_i, i = 5,6,7,8$ that form 1-component each (with 2 or 3 orange subboxes). In these cases, if such appearances contribute ε' to the total distance, then a simple averaging argument shows that for a uniformly sampled component, its distance will be linear from being free from ϕ-legged ν-appearances. Hence in Step 9, sampling such a component will enable us to recursively find a ϕ-legged ν-appearance with high probability. Since the length of a component on which the recursive call is made is $\Theta(s/m)$, the complexity of this step is $\tilde{O}(q(s/m))$, where $q(*)$ is the complexity of the algorithm, for the case of $\nu \in \mathcal{S}_3$, in terms of the length of D.

The correctness of the algorithm follows from the fact that if D is indeed far from being ϕ-legged ν-free, then it must be that there are linearly many ϕ-legged ν-appearances in at least one of the 8 configurations discussed above, and for each case, either a π-appearance or a ϕ-legged ν-appearance is found, by induction.

The base case is for $m = n$ for which the trivial algorithm that queries all indices is obviously correct, and has complexity n. To understand the query complexity for general m, let a be the smallest integer for which $m^a \geq n$. We express the query complexity for functions over a domain of length n, and parameter m as $q(m,a)$ for a as defined above. We get that, omitting polylogarithmic factors, $q(m,1) = m = s$ (that is the base case above), and $q(m,a) = m + m + q(m, a-1)$ where the first summand is the number of queries made by the gridding, the second is the number of queries made by Step 2 above (corresponding to Step 6 in AlgTest), and the last is the query complexity of the recursive call on subbox of length s/m with the same m, for which the corresponding $a' = a - 1$.

The recursion equation implies that $q(a,m) = \tilde{O}(am)$, which implies a query complexity n^δ by choosing $m = n^\delta$. We note that this is true as long as m (and hence δ) is large enough, as the recursion depth is a and there is an exponential deterioration of the distance parameter at each recursive call in Steps 2 and 3 above (corresponding to Steps 6 and 9 in AlgTest). However, for arbitrary constant $\delta < 1$ we can achieve complexity n^δ and this is true even for $\delta = 1/\log\log\log n$ for which the complexity becomes $n^{o(1)}$.

A final note that is due here, is that for $\nu \in \mathcal{S}_r$ with $r \geq 4$, the complexity of Step 2 above is not $O(\log n)$, and thus the complexity dependence on m becomes important.

References

1. Shlomo Ahal and Yuri Rabinovich. On complexity of the subpattern problem. *SIAM J. Discret. Math.*, 22(2):629–649, 2008. doi:10.1137/S0895480104444776.
2. Michael H. Albert, Robert E. L. Aldred, Mike D. Atkinson, and Derek A. Holton. Algorithms for pattern involvement in permutations. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation, 12th International Symposium, ISAAC 2001, Proceedings*, volume 2223 of *Lecture Notes in Computer Science*, pages 355–366. Springer, 2001.
3. Noga Alon and Ehud Friedgut. On the number of permutations avoiding a given pattern. *J. Comb. Theory, Ser. A*, 89(1):133–140, 2000. doi:10.1006/jcta.1999.3002.
4. Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2000. doi:10.1137/S0097539700366528.
5. Richard Arratia. On the Stanley-Wilf conjecture for the number of permutations avoiding a given pattern. *The Electronic Journal of Combinatorics*, 6:1–4, 1999.
6. Aleksandrs Belovs. Adaptive lower bound for testing monotonicity on the line. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018*, pages 31:1–31:10, 2018.
7. Omri Ben-Eliezer and Clément L. Canonne. Improved bounds for testing forbidden order patterns. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 2093–2112. SIAM, 2018.
8. Omri Ben-Eliezer, Clément L. Canonne, Shoham Letzter, and Erik Waingarten. Finding monotone patterns in sublinear time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1469–1494. IEEE Computer Society, 2019.
9. Omri Ben-Eliezer, Shoham Letzter, and Erik Waingarten. Optimal adaptive detection of monotone patterns. *CoRR*, abs/1911.01169, 2019. arXiv:1911.01169.
10. Benjamin Aram Berendsohn, László Kozma, and Dániel Marx. Finding and counting permutations via CSPs. *Algorithmica*, pages 1–26, 2021.
11. Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, 1994. Technical Report WS-94-03*, pages 359–370. AAAI Press, 1994.
12. Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David P. Woodruff. Transitive-closure spanners. *SIAM Journal on Computing*, 41(6):1380–1425, 2012. doi:10.1137/110826655.
13. Miklós Bóna. *Exact and asymptotic enumeration of permutations with subsequence conditions*. PhD thesis, Massachusetts Institute of Technology, 1997.
14. Miklós Bóna. The solution of a conjecture of Stanley and Wilf for all layered patterns. *J. Comb. Theory, Ser. A*, 85(1):96–104, 1999. doi:10.1006/jcta.1998.2908.
15. Deeparnab Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Proceedings of the ACM Symposium on Theory of Computing (STOC) 2013*, pages 419–428, 2013.
16. Kashyap Dixit, Sofya Raskhodnikova, Abhradeep Thakurta, and Nithin Varma. Erasure-resilient property testing. *SIAM J. Comput.*, 47(2):295–329, 2018. doi:10.1137/16M1075661.
17. Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *RANDOM-APPROX, 1999, Proceedings*, pages 97–108, 1999.

18 Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000. doi:10.1006/jcss.1999.1692.

19 Eldar Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004. doi:10.1016/j.ic.2003.09.003.

20 Jacob Fox. Stanley-Wilf limits are typically exponential. *CoRR*, abs/1310.8378, 2013. arXiv:1310.8378.

21 Jacob Fox and Fan Wei. Fast property testing and metrics for permutations. *Comb. Probab. Comput.*, 27(4):539–579, 2018. doi:10.1017/S096354831800024X.

22 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

23 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 82–101. SIAM, 2014.

24 Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002*, pages 550–556. ACM, 2002.

25 Martin Klazar. The Füredi-Hajnal conjecture implies the Stanley-Wilf conjecture. In *Formal power series and algebraic combinatorics*, pages 250–255. Springer, 2000.

26 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. doi:10.1016/j.jcta.2004.04.002.

27 Michael Mitzenmacher and Saeed Seddighin. Improved sublinear time algorithm for longest increasing subsequence. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1934–1947. SIAM, 2021.

28 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for forbidden order patterns in an array. *Random Struct. Algorithms*, 55(2):402–426, 2019. doi:10.1002/rsa.20840.

29 Ilan Newman and Nithin Varma. New sublinear algorithms and lower bounds for LIS estimation. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 100:1–100:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

30 Ilan Newman and Nithin Varma. Strongly sublinear algorithms for testing pattern freeness. *CoRR*, abs/2106.04856, 2021. arXiv:2106.04856.

31 Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, and Nithin Varma. Parameterized property testing of functions. *ACM Trans. Comput. Theory*, 9(4):17:1–17:19, 2018. doi:10.1145/3155296.

32 Pranav Patel, Eamonn J. Keogh, Jessica Lin, and Stefano Lonardi. Mining motifs in massive time series databases. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 370–377. IEEE Computer Society, 2002.

33 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

34 Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 1121–1145, 2019.

35 Michael E. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. *SIAM Journal on Computing*, 46(2):774–823, 2017.

An Optimal-Time RLBWT Construction in BWT-Runs Bounded Space

Takaaki Nishimoto
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Shunsuke Kanda
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Yasuo Tabei
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Abstract

The compression of highly repetitive strings (i.e., strings with many repetitions) has been a central research topic in string processing, and quite a few compression methods for these strings have been proposed thus far. Among them, an efficient compression format gathering increasing attention is the run-length Burrows–Wheeler transform (RLBWT), which is a run-length encoded BWT as a reversible permutation of an input string on the lexicographical order of suffixes. State-of-the-art construction algorithms of RLBWT have a serious issue with respect to (i) non-optimal computation time or (ii) a working space that is linearly proportional to the length of an input string. In this paper, we present *r-comp*, the first optimal-time construction algorithm of RLBWT in BWT-runs bounded space. That is, the computational complexity of r-comp is $O(n + r \log r)$ time and $O(r \log n)$ bits of working space for the length n of an input string and the number r of equal-letter runs in BWT. The computation time is optimal (i.e., $O(n)$) for strings with the property $r = O(n/\log n)$, which holds for most highly repetitive strings. Experiments using a real-world dataset of highly repetitive strings show the effectiveness of r-comp with respect to computation time and space.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases lossless data compression, Burrows–Wheeler transform, highly repetitive text collections

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.99

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2202.07885`

Supplementary Material *Software (Source Code)*: `https://github.com/kampersanda/rcomp`

1 Introduction

Highly repetitive strings (i.e., strings including many repetitions) have become common in research and industry. For instance, the 1000 Genomes Project [32] was established for the purpose of building a detailed catalogue of human genetic variation, and it has sequenced a large number of human genomes. Nowadays, approximately 60 billion pages are said to exist on the Internet, and large sections of those pages (e.g., version-controlled documents) are highly repetitive. There is therefore a growing demand to develop scalable data compression for efficiently storing, processing, and analyzing a gigantic number of highly repetitive strings.

To fulfill this demand, quite a few data compression methods for highly repetitive strings have been developed. Examples are LZ77 [34], grammar compression [19, 12, 31, 14], block trees [2], and many others [25, 22, 10]. Among them, an efficient compression format gathering increased attention is the *run-length Burrows–Wheeler transform* (RLBWT), which is a run-length encoded BWT [6] as a reversible permutation of an input string on the lexicographical order of suffixes. Recently, researchers have focused on developing string

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 99; pp. 99:1–99:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Summary of state-of-the-art RLBWT construction algorithms. The update time in the rightmost column is the time needed to construct a new RLBWT from the current RLBWT for a character newly added to the string. The update time of r-comp is amortized. T is an input string of alphabet size σ and length n; $|\mathsf{PFP}|$ is the size of the dictionary and factorization created by the prefix-free parsing of T [5].

Method	Type	Running time	Working space (bits)	Update time
D. Belazzougui+ [4]	indirect	$O(n)$	$O(n \log \sigma)$	Unsupported
J. Munro+ [20]	indirect	$O(n)$	$O(n \log \sigma)$	Unsupported
D.Kempa [15]	indirect	$O(n/\log_\sigma n + r \log^7 n)$	$O(n \log \sigma + r \log^6 n)$	Unsupported
D.Kempa+[16]	indirect	$O(n \log \sigma / \sqrt{\log n})$	$O(n \log \sigma)$	Unsupported
Big-BWT [5]	indirect	$O(n)$	$O(\|\mathsf{PFP}\| \log n)$	Unsupported
KK method [17, 23]	direct	$O(n(\log \log n)^2 + r \log^8 n)$	$O(r \operatorname{polylog} n)$	Unsupported
PP method [30]	direct	$O(n \log r)$	$O(r \log n)$	$O(\log r)$
Faster-PP method [28]	direct	$O(n \log r)$	$O(r \log n)$	$O(\log r)$
r-comp (this study)	direct	$O(n + r \log r)$	$O(r \log n)$	$O(1 + (r \log r)/n)$

processing methods such as locate query [13, 1, 3, 26], document listing [7], and substring enumeration [27] on RLBWT. Although several algorithms for constructing the RLBWT from an input string have been proposed thus far, there is no prior work that achieves the computational complexity of optimal time (i.e., time linearly proportional to the length of the input string) and BWT-runs bounded space (i.e., a working space linearly proportional to the number of equal-letter runs in the BWT and logarithmically proportional to the length of the input string).

Contribution. We present *r-comp*, the first construction algorithm of RLBWT that achieves optimal time and BWT-runs bounded space. R-comp directly constructs the RLBWT of an input string. It reads one character of an input string at a time from the reversed string and gradually builds the RLBWT corresponding to the suffixes read so far. The state-of-the-art online construction methods [30, 28] use inefficient data structures such as dynamic wavelet trees and B-trees for inserting each character into the current RLBWT at an insertion position, which is the most time-consuming part in an RLBWT construction. We present a new *divided BWT (DBWT)* representation of BWT and a new bipartite graph representation on DBWT called *LF-interval graph* to speed up the construction of RLBWT. The DBWT and LF-interval graph are efficiently built while reading each character one by one, and they enable us to quickly compute an appropriate position for inserting each character into the current RLBWT of the string. Another remarkable property of r-comp is the ability to extend the RLBWT for a newly added character without rebuilding the data structures used in r-comp from the beginning.

As a result, the computational complexity of r-comp is $O(n + r \log r)$ time and $O(r \log n)$ bits of working space for the length n of an input string and the number r of equal-letter runs in BWT. In particular, the computational complexity is optimal (i.e., $O(n)$) for strings with the property $r = O(n/\log n)$, which holds for most highly repetitive strings. We experimentally tested the ability of r-comp to compress various highly repetitive strings, and we show that r-comp performs better than other methods with respect to computation time and space.

2 Related work

There are two types of methods for indirectly or directly constructing the RLBWT of a string (see Table 1 for a summary of state-of-the-art construction algorithms of RLBWT). In the indirect constructions of RLBWT, the BWT of an input string is first built and then the

BWT is encoded into the RLBWT by run-length encoding. Several efficient algorithms for constructing the BWT of a given string have been proposed [4, 16, 21, 8, 16, 20]. Let T be a string of length n with an alphabet of size σ, and let r be the number of equal-letter runs in its BWT. Kempa [15] proposed a RAM-optimal time construction of the BWT of string T with compression ratio $n/r = \Omega(\mathrm{polylog}\, n)$. The algorithm runs in $O(n/\log_\sigma n)$ time with $O(n \log \sigma)$ bits of working space. Kempa and Kociumaka also proposed a BWT construction in $O(n \log \sigma)$ bits of working space [16]. This algorithm runs in $O(n \log \sigma / \sqrt{\log n})$ time, which is bounded by $o(n)$ time for a string with $\log \sigma = o(\sqrt{\log n})$. These algorithms are not space efficient for highly repetitive strings in that their working space is linearly proportional to the length of the input string.

Big-BWT [5] is a practical algorithm for constructing the BWT of a huge string using *prefix-free parsing*, which constructs a dictionary of strings and a factorization from string T. Although Big-BWT runs in optimal time (i.e., $O(n)$) with $O(|\mathsf{PFP}| \log n)$ bits of working space for the sum $|\mathsf{PFP}|$ of (i) the lengths of all the strings in the dictionary and (ii) the number of strings in the factorization, Big-BWT is not space efficient for highly repetitive strings in the worst case, because $|\mathsf{PFP}|$ can be $\sqrt{n}$ times larger than r, resulting in $\Omega(r\sqrt{n} \log n)$ bits of working space (see the full version of the paper [24] for the proof). Even worse, several data structures used in these indirect constructions cannot be updated. Thus, one needs to rebuild the data structures from scratch for a newly added character, which reduces the usability of indirect constructions of RLBWT.

In the direct constructions of RLBWT, Policriti and Prezza [30] proposed an algorithm for the construction of RLBWT, which we call *PP method*. The PP method reads an input string in reverse by one character, and it gradually builds the RLBWT corresponding to the suffix that was just read, where an inefficient dynamic wavelet tree is used for inserting a character into the RLBWT at an appropriate position, limiting the scalability of the PP method in practice. Ohno et al. [28] proposed a faster method, which we call *Faster-PP method*, by replacing the dynamic wavelet tree used in the PP method by a B-tree. Whereas both the PP method and Faster-PP method run with the same time and space complexities – $O(n \log r)$ time and $O(r \log n)$ bits of working space – the time complexity is not the optimal time for most highly repetitive strings.

Kempa and Kociumaka [17] proposed a conversion algorithm, which is referred to as *KK method*, from the LZ77 parsing [34] of T to the RLBWT in $O(z \log^7 n)$ time with $O(z\, \mathrm{polylog}\, n)$ bits of space, where z is the number of phrases in the parsing. Theoretically, we can compute the RLBWT of an input string by combining the KK method with an algorithm for computing the LZ77 parsing (e.g., [23]), and the working space of their conversion is bounded by $O(r\, \mathrm{polylog}\, n)$ bits because $z = O(r \log n)$ [22]. Kempa and Langmead [18] proposed a practical algorithm for constructing a compressed grammar from an input string in $\Omega(n)$ time using an approximate LZ77 parsing. Because these methods use several static data structures that cannot be updated, the data structures must be rebuilt from scratch when a new character is added.

Although there are several algorithms for indirectly or directly constructing the RLBWT, no previous work has been able to achieve optimal time (i.e., $O(n)$ time) with BWT-runs bounded space (i.e., $O(r \log n)$ bits). We present *r-comp*, the first direct construction of RLBWT that achieves optimal time with BWT-runs bounded space for most highly repetitive strings. Details of r-comp are presented in the following sections.

This paper is organized as follows. Section 3 introduces basic notions used in this paper, and a DBWT representation of BWT is presented in Section 4. Section 5 presents an LF-interval graph representation of DBWT and a fast update operation on LF-interval

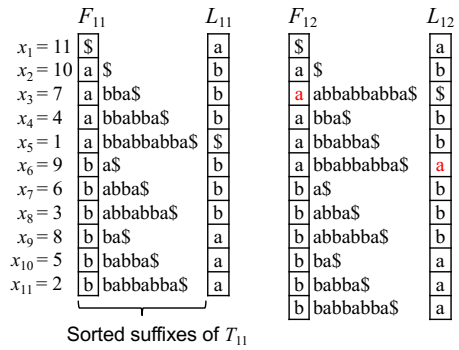


Figure 1 (Left) Sorted suffixes of $T_{11} = abbabbabba\$$, F_{11}, and L_{11}. (Right) Sorted suffixes of $T_{12} = aabbabbabba\$$, F_{12}, and L_{12}.

graphs. The r-comp algorithm is presented in Section 6. Section 7 presents the experimental results using the r-comp algorithm on benchmark and real-world datasets of highly repetitive strings.

3 Preliminaries

Basic notation. An *interval* $[b, e]$ for two integers b and e ($b \leq e$) represents the set $\{b, b+1, \ldots, e\}$. Let T be a string of length n over an alphabet $\Sigma = \{1, 2, \ldots, n^{O(1)}\}$ of size σ, and $|T|$ be the length of T (i.e., $|T| = n$). Let $T[i]$ be the i-th character of T (i.e., $T = T[1], T[2], \ldots, T[n]$) and $T[i..j]$ be the substring of T that begins at position i and ends at position j. Let T_δ be the suffix of T of length δ ($1 \leq \delta \leq n$), i.e., $T_\delta = T[(n-\delta+1)..n]$. A *rank query* $\mathsf{rank}(T, c, i)$ on a string T returns the number of occurrences of character c in $T[1..i]$, i.e., $\mathsf{rank}(T, c, i) = |\{j \mid T[j] = c, 1 \leq j \leq i\}|$.

For a string P, $P[i] < P[j]$ means that the i-th character of P is smaller than the j-th character of P. Moreover, $T \prec P$ means that T is lexicographically smaller than P. Formally, $T \prec P$ if and only if either of the following two conditions holds: (i) there exists an integer i such that $T[1..i-1] = P[1..i-1]$ and $T[i] < P[i]$; (ii) T is a prefix of P (i.e., $T = P[1..|T|]$) and $|T| < |P|$. Here, $\mathsf{occ}_<(T, c)$ denotes the number of characters smaller than character c in string T (i.e., $\mathsf{occ}_<(T, c) = |\{j \mid j \in \{1, 2, \ldots, n\} \text{ s.t. } T[j] < c\}|$). Special character \$ is the smallest character in Σ. Throughout this paper, we assume that special character \$ only appears at the end of T (i.e., $T[n] = \$$ and $T[i] \neq \$$ for all $\{1, 2, \ldots, n-1\}$).

A *run* is defined as the maximal repetition of the same character. Formally, a substring $T[i..j]$ of T is a *run* of the same character c if it satisfies the following three conditions: (i) $T[i..j]$ is a repetition of the same character c (i.e., $T[i] = T[i+1] = \cdots = T[j] = c$); (ii) $i = 1$ or $T[i-1] \neq c$; (iii) $j = n$ or $T[j+1] \neq c$.

We use base-2 logarithm throughout this paper. Our computation model is a unit-cost word RAM with a machine word size of $\Theta(\log n)$ bits. We evaluate the space complexity in terms of the number of machine words. A bitwise evaluation of space complexity can be obtained with a $\log n$ multiplicative factor.

BWT, LF function, and RLBWT. The BWT [6] of a suffix T_δ is a permuted string L_δ of T_δ, and it is constructed as follows: all the suffixes of T_δ are sorted in the lexicographical order and the character preceding each suffix is taken. Formally, let $x_1, x_2, \ldots, x_\delta$ be the starting positions of the sorted suffixes of T_δ (i.e., $x_1, x_2, \ldots, x_\delta$ are a permutation

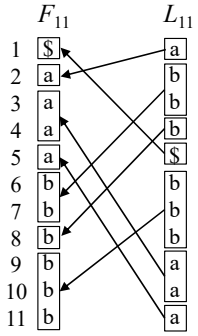


Figure 2 DBWT-repetitions and their corresponding F-intervals on F_{11} for a DBWT $D_{11} = a, bb, b, \$, bbb, aa, a$ of BWT L_{11} in Figure 1. Each rectangle on L_{11} represents a DBWT-repetition, and each rectangle on F_{11} represents an F-interval on F_{11}. Each directed arrow indicates the F-interval corresponding to the DBWT-repetition on L_{11}.

of sequence $1, 2, \ldots, \delta$ such that $T_\delta[x_1..\delta] \prec T_\delta[x_2..\delta] \prec \cdots \prec T_\delta[x_n..\delta]$). Then, $L_\delta = T_\delta[x_1 - 1], T_\delta[x_2 - 1], \ldots, T_\delta[x_\delta - 1])$, where $T_\delta[0]$ is defined as the last character of T_δ (i.e., $T_\delta[0] = T_\delta[\delta] = \$$). Similarly, the permuted string F_δ of suffix T_δ consists of the first characters of the sorted suffixes of T_δ, i.e., $F_\delta = T_\delta[x_1], T_\delta[x_2], \ldots, T_\delta[x_\delta]$.

Figure 1 illustrates the sorted suffixes of T_{11} and T_{12} for $T = aabbabbabba\$$. Here, $x_1, x_2, \ldots, x_{11}$ are the starting positions of the sorted suffixes of T_{11}. Moreover, $L_{11} = abbb\$bbbaaa$ and $F_{11} = \$aaaabbbbbb$. The BWT of T is $L_{12} = ab\$bbabbbaaa$.

There is a one-to-one correspondence between L_δ and F_δ because the two strings are permutations of T_δ. Formally, for two integers $i, j \in \{1, 2, \ldots, \delta\}$, $L_\delta[i]$ corresponds to $F_\delta[j]$ if and only if either of the following two conditions holds: (i) $x_i - 1 = x_j$ or (ii) $x_i = 1$ and $x_j = \delta$. *LF function* LF_δ is a bijective function from L_δ to F_δ [11]. Function $\mathsf{LF}_\delta(i) = j$ for two integers $i, j \in \{1, 2, \ldots, \delta\}$ if and only if $L_\delta[i]$ corresponds to $F_\delta[j]$. *LF formula* [11] is a well-known property of LF function, and it enables us to compute the corresponding position in F_δ from a position in L_δ. Namely, $\mathsf{LF}_\delta(i)$ is equal to the summation of (i) the number of characters in L_δ smaller than $L_\delta[i]$ and (ii) the number of $L_\delta[i]$ in the prefix $L_\delta[1..i]$, i.e., $\mathsf{LF}_\delta(i) = \mathsf{occ}_<(L_\delta, L_\delta[i]) + \mathsf{rank}(L_\delta, i, L_\delta[i])$.

BWT can be separated into all the runs of the same character. We call each run BWT-run. For BWT L_δ, r BWT-runs $P_1, P_2, \ldots, P_r$ satisfy (i) $L_\delta = P_1, P_2, \ldots, P_r$ and (ii) each P_i $(i = 1, 2, \ldots, r)$ is a run of the same character in L_δ. The RLBWT of a suffix T_δ is defined as a sequence of r pairs $(P_1[1], |P_1|)$, $(P_2[1], |P_2|)$, …, $(P_r[1], |P_r|)$. The RLBWT can be stored in $r(\log n + \log \sigma)$ bits, and we can recover T_δ from the RLBWT using LF function (e.g., [26]). Throughout this paper, r denotes the number of BWT-runs in the BWT of T.

In Figure 1, the BWT-runs in the BWT L_{11} of T_{11} are $a, bbb, \$, bbb$, and aaa. The RLBWT of T_{11} is $(a, 1)$, $(b, 3)$, $(\$, 1)$, $(b, 3)$, and $(a, 3)$.

4 DBWT

The divided BWT (DBWT) is a general concept in the RLBWT and is the foundation of the LF-interval graph. Formally, the DBWT D_δ of BWT L_δ is defined as a sequence $L_\delta[p_1..(p_2 - 1)], L_\delta[p_2..(p_3 - 1)], \ldots, L_\delta[p_k..(p_{k+1} - 1)]$ for $p_1 = 1 < p_2 < \cdots < p_k < p_{k+1} = n + 1$, where $L_\delta[p_i..(p_{i+1} - 1)]$ for each $i = 1, 2, \ldots, k$ is a repetition of the same character. We call each repetition of the same character in the DBWT *DBWT-repetition*. A DBWT-repetition is not necessarily a run. DBWT D_δ is equal to the RLBWT of T_δ if and only if $L_\delta[p_i..(p_i - 1)]$ for each $i \in \{1, 2, \ldots, k\}$ is a run.

In Figure 2, sequence $D_{11} = a, bb, b, \$, bbb, aa, a$ of equal-letter repetitions is a DBWT for BWT L_{11} of string T_{11}. The DBWT-repetitions in D_{11} are the strings enclosed by the rectangles on L_{11}.

The DBWT of a BWT is not unique because a BWT can be divided by various criteria. We present a criterion for DBWT in the following in order to efficiently build RLBWT. The LF function maps each DBWT-repetition $L_\delta[p_i..(p_{i+1}-1)]$ into the consecutive characters on interval $[\mathsf{LF}_\delta(p_i), \mathsf{LF}_\delta(p_{i+1}-1)]$, which is called an *F-interval* on F_δ. The LF formula enables us to compute $\mathsf{LF}_\delta(j)$ for each position $j \in [p_i, (p_{i+1}-1)]$ on the i-th DBWT-repetition in $O(1)$ time using the starting position p_i of the DBWT-repetition and the F-interval $[\mathsf{LF}_\delta(p_i), \mathsf{LF}_\delta(p_{i+1}-1)]$ corresponding to the DBWT-repetition as follows: $\mathsf{LF}_\delta(j) = \mathsf{LF}_\delta(p_i) + j - p_i$.

In Figure 2, each F-interval on F_{11} corresponding to a DBWT-repetition on L_{11} is enclosed by a rectangle. The F-intervals on F_{11} are $[1,1]$, $[2,2]$, $[3,4]$, $[5,5]$, $[6,7]$, $[8,8]$, and $[9,11]$. The F-interval corresponding to the second DBWT-repetition bb is $[6,7]$.

Let α be a user-defined parameter no less than 2 (i.e., $\alpha \geq 2$). DBWT-repetition $L_\delta[p_i..(p_{i+1}-1)]$ is said to *cover* the starting position $\mathsf{LF}_\delta(p_j)$ of an F-interval $[\mathsf{LF}_\delta(p_j), \mathsf{LF}_\delta(p_{j+1})]$ on F_δ if interval $[p_i, (p_{i+1}-1)]$ on F_δ contains the position $\mathsf{LF}_\delta(p_j)$ (i.e., $\mathsf{LF}_\delta(p_j) \in [p_i, (p_{i+1}-1)]$). The DBWT-repetition is said to be α-*heavy* if it covers at least α starting positions of the F-intervals on F_δ for parameter $\alpha \geq 2$. Similarly, F-interval $[\mathsf{LF}_\delta(p_i), \mathsf{LF}_\delta(p_{i+1}-1)]$ on F_δ is said to cover the starting position p_j of a DBWT-repetition $L_\delta[p_j..(p_{j+1}-1)]$ if interval $[\mathsf{LF}_\delta(p_i), \mathsf{LF}_\delta(p_{i+1}-1)]$ on L_δ contains the position p_j (i.e., $p_j \in [\mathsf{LF}_\delta(p_i), \mathsf{LF}_\delta(p_{i+1}-1)]$). An F-interval is said to be α-heavy if it covers at least α starting positions of DBWT-repetitions for parameter $\alpha \geq 2$. A DBWT is said to be α-*balanced* if the DBWT includes neither α-heavy DBWT-repetitions nor F-intervals, and D_δ^α denotes an α-balanced DBWT of BWT L_δ.

In Figure 2 with $\alpha = 3$, the fifth DBWT-repetition bbb of D_{11} covers two starting positions of F-intervals $[6,7]$ and $[8,8]$ on F_{11}, and the DBWT-repetition is not 3-heavy. The F-interval $[9,11]$ of the fifth DBWT-repetition covers the starting positions of two DBWT-repetitions aa and a. Moreover, the F-interval of the fifth DBWT-repetition is not 3-heavy. Thus, D_{11} is 3-balanced because D_{11} includes neither 3-heavy DBWT-repetitions nor F-intervals.

In the next section, the α-balanced DBWT is used to derive the time needed to update an LF-interval graph.

5 LF-interval graph

An LF-interval graph is a bipartite graph that represents both (i) the correspondence between each pair of elements in L_δ and F_δ according to the LF function and (ii) a covering relationship between DBWT-repetitions and F-intervals on a DBWT. The LF-interval graph $\mathsf{Grp}(D_\delta)$ for DBWT D_δ of k DBWT-repetitions $L_\delta[p_1..(p_2-1)], L_\delta[p_2..(p_3-1)], \ldots, L_\delta[p_k..(p_{k+1}-1)]$ is defined as 4-tuple $(U \cup V,\ E_{LF} \cup E_L \cup E_F,\ B_U \cup B_V,\ B_L \cup B_F)$, as detailed in the following.

Set $U = \{u_1, u_2, \ldots, u_k\}$ is a set of nodes, and u_i for each $i \in \{1, 2, \ldots, k\}$ represents the i-th DBWT-repetition $L_\delta[p_i..(p_{i+1}-1)]$ on DBWT D_δ. Moreover, set $V = \{v_1, v_2, \ldots, v_k\}$ is a set of nodes, and v_i for each $i \in \{1, 2, \ldots, k\}$ represents the F-interval $[\mathsf{LF}_\delta(p_i), \mathsf{LF}_\delta(p_{i+1}-1)]$ mapped from the i-th DBWT-repetition represented as u_i on DBWT D_δ by the LF function.

The set E_{LF} of undirected edges in LF-interval graph $\mathsf{Grp}(D_\delta)$ represents the correspondence between DBWT-repetitions on DBWT D_δ and F-intervals on F_δ according to the LF function. Formally, $E_{LF} \subseteq (U \times V)$ is a set of undirected edges between U and V, and $(u_i, v_j) \in E_{LF}$ holds if and only if the i-th DBWT-repetition $L_\delta[p_i..(p_{i+1}-1)]$ represented as u_i is mapped to the j-th F-interval $[\mathsf{LF}_\delta(p_j), \mathsf{LF}_\delta(p_{j+1}-1)]$ represented as v_j. Namely, $E_{LF} = \{(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)\}$.

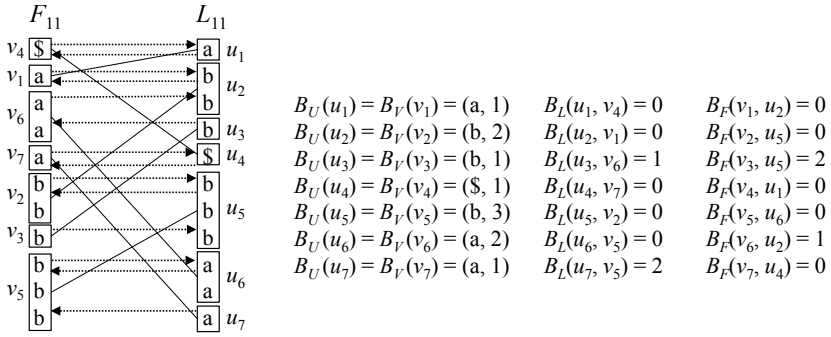


Figure 3 LF-interval graph $\mathsf{Grp}(D_{11})$ for DBWT D_{11} in Figure 2.

Two sets E_L and E_F of directed edges represent the covering relationship between DBWT-repetitions and F-intervals on DBWT D_δ. Set $E_L \subseteq (U \times V)$ is a set of directed edges from U to V, and $(u_i, v_j) \in E_L$ holds if and only if F-interval $[\mathsf{LF}_\delta(p_j), \mathsf{LF}_\delta(p_{j+1} - 1)]$, represented as v_j, covers the starting position p_i of DBWT-repetition $L_\delta[p_i..(p_{i+1} - 1)]$, represented as u_i. Formally, $E_L = \{(u_i, v_j) \mid 1 \leq i, j \leq k \text{ s.t. } p_i \in [\mathsf{LF}_\delta(p_j), \mathsf{LF}_\delta(p_{j+1} - 1)]\}$. Similarly, $E_F \subseteq (V \times U)$ is a set of directed edges from V to U, and $(v_j, u_i) \in E_F$ holds if and only if DBWT-repetition $L_\delta[p_i..(p_{i+1} - 1)]$, represented as u_i, covers the starting position $\mathsf{LF}_\delta(p_j)$ of F-interval $[\mathsf{LF}_\delta(p_j), \mathsf{LF}_\delta(p_{j+1} - 1)]$, represented as v_j. Formally, $E_F = \{(v_j, u_i) \mid 1 \leq i, j \leq k \text{ s.t. } \mathsf{LF}_\delta(p_j) \in [p_i, (p_{i+1} - 1)]\}$.

Function $B_U : U \to (\Sigma, \mathbb{N})$ is a label function for the set U of nodes, and it maps each node $u_i \in U$ to a pair consisting of the character in Σ and the length in $\mathbb{N} = \{1, 2, \ldots\}$ for the i-th DBWT-repetition represented by u_i. Namely, $B_U(u_i) = (L_\delta[p_i], p_{i+1} - p_i)$. Similarly, $B_V : V \to (\Sigma, \mathbb{N})$ is a label function for the set V of nodes, and it maps each node $v_i \in V$ to a pair consisting of the character in Σ and the length in $\mathbb{N}$ for the repetition $F_\delta[\mathsf{LF}_\delta(p_i)..\mathsf{LF}_\delta(p_{i+1} - 1)]$ of the same character on the F-interval represented by v_i. Namely, $B_V(v_i) = (F_\delta[\mathsf{LF}_\delta(p_i)], \mathsf{LF}_\delta(p_{i+1} - 1) - \mathsf{LF}_\delta(p_i) + 1)$. For all $i \in \{1, 2, \ldots, k\}$, $B_U(u_i) = B_V(v_i)$ holds by the LF formula.

Function $B_L : E_L \to \mathbb{N}$ is a label function for the set E_L of directed edges, and it maps each edge $(u_i, v_j) \in E_L$ to an integer value representing the difference between the starting position p_i of DBWT-repetition $L_\delta[p_i..(p_{i+1} - 1)]$, represented as u_i, and the starting position $\mathsf{LF}_\delta(p_j)$ of F-interval $[\mathsf{LF}_\delta(p_j), \mathsf{LF}_\delta(p_{j+1} - 1)]$, represented as v_j. Namely, $B_L(u_i, v_j) = p_i - \mathsf{LF}_\delta(p_j)$.

Similarly, $B_F : E_F \to \mathbb{N}$ is a label function for the set E_F of directed edges, and it maps each edge $(v_j, u_i) \in E_F$ to an integer value representing the difference between the starting position $\mathsf{LF}_\delta(p_j)$ of F-interval $[\mathsf{LF}_\delta(p_j), \mathsf{LF}_\delta(p_{j+1} - 1)]$, represented as v_j, and the starting position p_i of DBWT-repetition $L_\delta[p_i..(p_{i+1} - 1)]$, represented as u_i. Namely, $B_F(v_j, u_i) = \mathsf{LF}_\delta(p_j) - p_i$.

Figure 3 illustrates LF-interval graph $\mathsf{Grp}(D_{11})$ for DBWT D_{11} in Figure 2. For $U = \{u_1, u_2, \ldots, u_7\}$ and $V = \{v_1, v_2, \ldots, v_7\}$, each node $u_i \in U$ (respectively, $v_j \in V$) is enclosed by a rectangle on L_{11} (respectively, F_{11}). We have $E_{LF} = \{(u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4), (u_5, v_5), (u_6, v_6), (u_7, v_7)\}$. We depict each undirected edge in set E_{LF} by solid lines. Moreover, $E_L = \{(u_1, v_4), (u_2, v_1), (u_3, v_6), (u_4, v_7), (u_5, v_2), (u_6, v_5), (u_7, v_5)\}$, and $E_F = \{(v_1, u_2), (v_2, u_5), (v_3, u_5), (v_4, u_1), (v_5, u_6), (v_6, u_2), (v_7, u_4)\}$. Each directed edge in the two sets E_L and E_F is depicted by a dotted arrow. The four label functions B_U, B_V, B_L, and B_F are listed in Figure 3.

5.1 Dynamic data structures for the LF-interval graph

Several dynamic data structures are used for efficiently updating LF-interval graph $\mathsf{Grp}(D_\delta)$. Two doubly linked lists are used for supporting the insertions and deletions of nodes in U and V. The nodes in U should be totally ordered with respect to the starting position of the DBWT-repetition, which is represented as a node in U. Namely, $u_1 < u_2 < \cdots < u_k$. The nodes in set U are stored in a doubly linked list, where each node $u_i \in U$ has previous and next pointers connecting to the previous and next nodes, respectively, in the total order of nodes in U. Similarly, the nodes in V should be totally ordered with respect to the starting position of the F-interval, which is represented as a node in V. Namely, $v_{\pi_1} < v_{\pi_2} < \ldots < v_{\pi_k}$ holds for permutation $\pi_1, \pi_2, \ldots, \pi_k$ of sequence $1, 2, \ldots, k$ such that $\mathsf{LF}_\delta(p_{\pi_1}) < \mathsf{LF}_\delta(p_{\pi_2}) < \ldots < \mathsf{LF}_\delta(p_{\pi_k})$. The nodes in V are stored in another doubly linked list, where each node $v_i \in V$ has previous and next pointers connecting to the previous and next nodes in the increasing order of nodes in V, respectively. The space of two doubly linked lists storing nodes in U and V is $O(k \log n)$ bits.

All the nodes corresponding to α-heavy DBWT-repetitions in U are stored in an array data structure in any order. Similarly, all the nodes corresponding to α-heavy F-intervals in V are stored in another array data structure in any order. The two arrays take $O(k \log n)$ bits of space. Each array stores nothing if D_δ is α-balanced.

An *order maintenance data structure* [9] is used for comparing two nodes in U with the total order of U, and the data structure supports the following three operations: (i) the order operation determines whether or not node $u_i \in U$ precedes node $u_j \in U$ in the total order of U; (ii) the insertion operation inserts node $u_i \in U$ right after node $u_j \in U$ in the total order of U; (iii) the deletion operation deletes node $u_i \in U$ from U. The data structure supports these three operations in $O(1)$ time with $O(k \log n)$ bits of space, and it is used with a B-tree that stores the nodes in V, as explained below.

A B-tree (a type of self-balancing search tree) is built on the set V of nodes using the combination of the order maintenance data structure, where each node v_i in V is totally ordered with respect to (i) the total order of the node u_i in U that is connected to v_i by an edge in E_{LF} (i.e., $(u_i, v_i) \in E_{LF}$) and (ii) the first character $L_\delta[p_i]$ of the DBWT-repetition is represented as u_i. For a node $v_i \in V$, the B-tree stores a pair $(u_i, L_\delta[p_i])$ as the key of the node v_i. Nodes in V are totally ordered using the key, and $v_i \in V$ precedes $v_j \in V$ if and only if either of the following conditions holds: (i) $L_\delta[p_i] < L_\delta[p_j]$ or (ii) $L_\delta[p_i] = L_\delta[p_j]$ and u_i precedes u_j in the total order of U (i.e., $i < j$). Condition (ii) is efficiently computed in $O(1)$ time by the order maintenance data structure of U. According to the following lemma, the order of keys in the B-tree is the same as that of the nodes stored in the doubly linked list of V (i.e., the order of the nodes in the B-tree is $v_{\pi_1} < v_{\pi_2} < \ldots < v_{\pi_k}$).

▶ **Lemma 1.** *For two distinct nodes $v_i, v_j \in V$, the key of v_i precedes that of v_j in the B-tree of V if and only if v_i precedes v_j in the doubly linked list of V (i.e., $\mathsf{LF}_\delta(p_i) < \mathsf{LF}_\delta(p_j)$).*

Proof. See the full version of the paper [24]. ◀

The B-tree with the order maintenance data structure supports the three operations of search, insertion, and deletion for any node in V in $O(\log k)$ time with $O(k \log n)$ bits of space.

5.2 Extension of BWT ([30, 28])

The BWT of a suffix can be extended from the BWT of a shorter suffix [30, 28]. In this section, we review the extension of BWT, which is used for updating the LF-interval graph. The BWT $L_{\delta+1}$ of a suffix $T_{\delta+1}$ of length $\delta + 1$ can be computed from the BWT L_δ of the

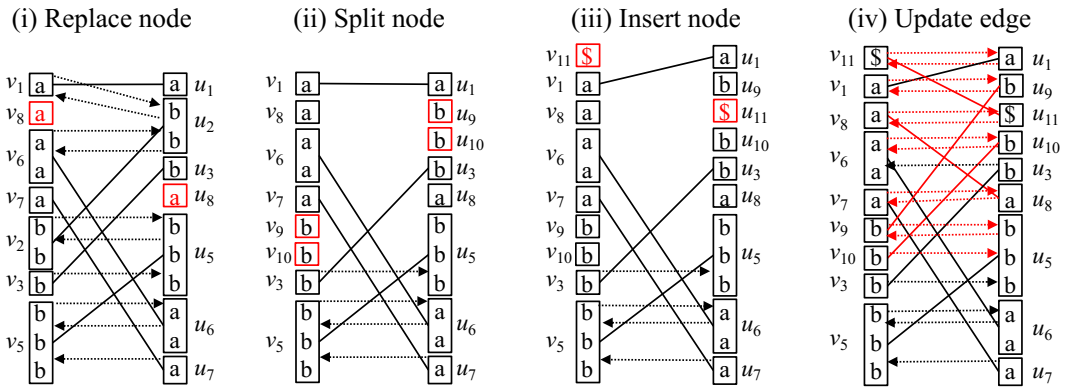


Figure 4 Each step in the update operation for the LF-interval graph in Figure 3. New nodes and edges created in each step are colored in red.

suffix T_δ of length δ using the following two steps: (i) special character \$ in L_δ is replaced with the first character c of $T_{\delta+1}$ (i.e., $c = T[n-\delta]$); (ii) special character \$ is inserted into L_δ at a position ins. Here, ins is computed by the LF formula as follows: for the position rep of special character \$ in L_δ (i.e., $L_\delta[\mathsf{rep}] = \$$), $\mathsf{ins} = \mathsf{occ}_<(L_\delta, c) + \mathsf{rank}(L_\delta, \mathsf{rep}, c) + 1$.

In Figure 1, BWT L_{12} of suffix T_{12} can be extended from L_{11} of suffix T_{11}. The first character c of T_{12} is a, and special character \$ is replaced with a at $\mathsf{rep} = 5$ on L_{11}. The insertion position ins for L_{11} is 3 because $\mathsf{occ}_<(L_{11}, a) + \mathsf{rank}(L_{11}, \mathsf{rep}, a) + 1 = 3$.

5.3 Foundation of updates of the LF-interval graph

Given the first character c in suffix $T_{\delta+1}$ of length $\delta+1$, an update operation of LF-interval graph $\mathsf{Grp}(D^\alpha_\delta)$ for an α-balanced DBWT $D^\alpha_\delta = L_\delta[p_1..(p_2-1)], L_\delta[p_2..(p_3-1)], \ldots, L_\delta[p_k..(p_{k+1}-1)]$ of T_δ updates $\mathsf{Grp}(D^\alpha_\delta)$ to $\mathsf{Grp}(D^{2\alpha+1}_{\delta+1})$ for a $(2\alpha+1)$-balanced DBWT $D^{2\alpha+1}_{\delta+1}$ of $T_{\delta+1}$. The update operation updates the given LF-interval graph according to the extension of BWT. This operation consists of four main steps: (I) *replace node*, (II) *split node*, (III) *insert node*, and (IV) *update edge*. Note that the update operation presented in this section is a foundation for the ones presented in the following two subsections, where several modifications are made to the foundation for faster operation.

(I) Replace node. This step replaces the node $u_i \in U$ labeled $(\$, 1)$ with a new one $u_{i'}$ labeled $(c, 1)$, and it updates V according to the replacement of the node in U. Node u_i can be found in $O(1)$ time by keeping track of it on U. The doubly linked list of U is updated according to the replacement. The node $v_i \in V$ connected to u_i by edge $(u_i, v_i) \in E_{LF}$ is removed from V, and a new node $v_{i'}$ labeled $(c, 1)$ is inserted into V at the position next to the most backward node v_g of the nodes whose keys are smaller than key (u_i, c). Node v_g can be found in $O(\log k)$ time using the B-tree of V. This step takes $O(\log k)$ time in total.

Figure 4-(I) shows an example of the replace-node step for the LF-interval graph $\mathsf{Grp}(D_{11})$ in Figure 3. Node $u_4 \in U$ labeled $(\$, 1)$ on $\mathsf{Grp}(D_{11})$ is replaced with node u_8 labeled $(a, 1)$. Node $v_4 \in V$, which is connected to u_4 by edge $(v_4, u_4) \in E_{LF}$, is removed from V, and edge (v_4, u_4) is removed from E_{LF}. A new node v_8 with label $(a, 1)$ is inserted into V. This node is inserted into the doubly linked list of V at the position next to v_1 (i.e., $v_g = v_1$).

(II) Split node. The insert-node step (as the next step) inserts a new node representing special character \$ into U. However, before the insert-node step, the split-node step splits a node $u_j \in U$ into two new nodes at an appropriate position on the doubly linked list of

U. This step is executed for inserting the new node representing special character \$ into U at a appropriate position in the insert-node step. Following the extension of BWT in Section 5.2, node $u_j \in U$ has label $(L_\delta[p_j], p_{j+1} - p_j)$ for the two starting positions p_j and p_{j+1} satisfying $p_j < \mathsf{ins} < p_{j+1}$ for the insertion position ins of special character \$. Such a node u_j exists if and only if (i) the BWT $L_{\delta+1}$ of $T_{\delta+1}$ does not have special character \$ as the last character (i.e., $\mathsf{ins} \neq \delta + 1$) and (ii) $p_i \neq \mathsf{ins}$ for all $i \in \{1, 2, \ldots, k\}$. This is because $p_1 < p_2 < \ldots < p_{k+1}$ and $p_{k+1} = \delta + 1$ hold.

If node u_j does not exist in U, this step does not split nodes. Otherwise, u_j is replaced with two new nodes $u_{j'}$ and $u_{j'+1}$ in the doubly linked list of U, where $u_{j'}$ is previous to $u_{j'+1}$. The new nodes $u_{j'}$ and $u_{j'+1}$ are labeled as $(L_\delta[p_j], \mathsf{ins} - p_j)$ and $(L_\delta[p_j], p_{j+1} - \mathsf{ins})$ using insertion position ins, respectively.

Although we do not know position ins in the split-node step, we can find node u_j. This is because (i) set V contains node v_{gnext} representing the F-interval starting at position ins unless $\mathsf{ins} = \delta + 1$, and (ii) v_{gnext} is next to v_g in the doubly linked list of V for the node v_g searched for in the replace-node step. The following lemma ensures that we can find u_j and compute the labels of the new nodes in $O(1)$ time.

▶ **Lemma 2.** *The following two statements hold after executing the replace-node step: (i) we can check whether u_j exists or not in $O(1)$ time; (ii) we can find u_j and compute the labels of two nodes $u_{j'}$ and $u_{j'+1}$ in $O(1)$ time.*

Proof. See the full version of the paper [24]. ◀

Next, set V is updated according to the replacement of nodes in U, i.e., for undirected edge $(u_j, v_j) \in E_{LF}$, node v_j is replaced with two new nodes $v_{j'}$ and $v_{j'+1}$ in the doubly linked list of V, where $v_{j'}$ is previous to $v_{j'+1}$. Nodes $v_{j'}$ and $v_{j'+1}$ have the same labels of $u_{j'}$ and $u_{j'+1}$, respectively. Therefore, this step takes $O(1)$ time.

Figure 4-(II) illustrates an example of the split-node step. In this example, $\mathsf{ins} = 3$, $v_{\mathsf{gnext}} = v_6$, $v_j = v_2$, $v_{j'} = v_9$, and $v_{j'+1} = v_{10}$ hold. In Figure 3, the directed edge starting at node v_6 is labeled as integer 1 by function $B_F(v_6, u_2)$, and the directed edge points to node u_2 with label $(b, 2)$. Hence, node u_2 is replaced with two nodes u_9 and u_{10}. Two nodes u_9 and u_{10} are labeled with pairs $(b, 1)$ and $(b, 1)$, respectively. Node v_2 is connected to u_2 by edge $(v_2, u_2) \in E_{LF}$, and v_2 is replaced with two nodes v_9 and v_{10}. Here, v_9 and v_{10} are labeled with pairs $(b, 1)$ and $(b, 1)$, respectively.

(III) Insert node. This step inserts a new node $u_{x'}$ labeled $(\$, 1)$ into U, and it updates V according to the insertion of U. Analogous to the extension of BWT described in Section 5.2, the position for inserting the new node in the doubly linked list of U is determined according to the following three cases: (i) Node $u_j \in U$ was found and it was split into two nodes $u_{j'}$ and $u_{j'+1}$ in the split-node step. In this case, node $u_{x'}$ is inserted at the position next to $u_{j'} \in U$ on the doubly linked list of U. (ii) Node u_j was not found, and new node $v_{i'}$ is inserted at the position next to the last element on the doubly linked list of V in the replace-node step. In this case, $u_{x'}$ is inserted at the position next to the last element on the doubly linked list of U. (iii) Node u_j was not found, and $v_{i'}$ is inserted at the position previous to a node $v_{\mathsf{gnext}} \in V$ on the doubly linked list of V. In this case, v_{gnext} is connected to a node $u_x \in U$ by a directed edge in E_F, and $u_{x'}$ is inserted into the doubly linked list of U at the position previous to u_x.

Next, this step creates a new node $v_{x'}$ labeled $(\$, 1)$, and it is inserted into the doubly linked list of V. The new node is inserted at the top of the list, because the new label includes special character \$. This step takes $O(1)$ time.

Figure 4-(III) illustrates an example of the insert-node step. Because the split-node step replaced node u_2 with two nodes u_9 and u_{10}, the insert-node step inserts node u_{11} labeled $(\$, 1)$ at the position next to u_9 on the doubly linked list of U. On the other hand, the step inserts node v_{11} labeled $(\$, 1)$ in the doubly linked list of V at the position previous to node v_1.

(IV) Update edge. This step updates the set E_{LF} of undirected edges and two sets E_L and E_F of directed edges according to the two sets U and V, which were updated in the previous steps. This step consists of two phases: (i) for the nodes of u_i, v_i, u_j, and v_j removed in the replace-node and split-node steps, the edges connected to these nodes are removed from E_{LF}, E_L, and E_F; (ii) new edges connecting new nodes (i.e., $u_{j'}$, $u_{j'+1}$, $v_{i'}$, and $v_{j'+1}$) are added to E_{LF}, E_L, and E_F. The labels of new directed edges are computed at the second phase. The number of removed edges and new edges can be bounded by $O(\alpha)$ because (i) every node in the LF-interval graph for an $O(\alpha)$-balanced DBWT is connected to $O(\alpha)$ edges, (ii) the DBWT represented by the given LF-interval graph $\mathsf{Grp}(D_\delta^\alpha)$ is α-balanced, and (iii) the LF-interval graph $\mathsf{Grp}(D_{\delta+1}^{2\alpha+1})$ outputted by this update operation represents a $(2\alpha+1)$-balanced DBWT. Because the number of updated edges is small, this step can be performed in $O(\alpha)$ time. See the full version of the paper [24] for the details of the update-edge step. Formally, we obtain the following lemma.

▶ **Lemma 3.** *The update-edge step takes $O(\alpha)$ time.*

Proof. See the full version of the paper [24]. ◀

In Figure 4-(IV), four edges (u_8, v_8), (u_9, v_9), (u_{10}, v_{10}), and (u_{11}, v_{11}) connecting new nodes are added to E_{LF}. Six directed edges (u_1, v_{11}), (u_9, v_1), (u_{11}, v_8), (u_{10}, v_6), (u_8, v_7), and (u_5, v_9) are added to E_L. Similarly, seven directed edges (v_{11}, u_1), (v_1, u_9), (v_8, u_{11}), (v_6, u_{10}), (v_7, u_8), (v_9, u_5), and (v_{10}, u_5) are added to E_F.

Update of the data structures. Similar to the update-edge step, the four data structures in the LF-interval graph (i.e., the order maintenance data structure, the B-tree of set V, and two arrays that store nodes representing α-heavy DBWT-repetitions and F-intervals) are updated according to the removed nodes and new nodes.

See the full version of the paper [24] for the details of the algorithm updating the four data structures. The following lemma concerning the update time of the four data structures.

▶ **Lemma 4.** *Updating the four data structures takes $O(\alpha + \log k)$ time.*

Proof. See the full version of the paper [24]. ◀

The update operation takes $O(\alpha + \log k)$ time in total. The following lemma concerning the theoretical results on this update operation holds.

▶ **Theorem 5.** *The following two statements hold: (i) the update operation takes $O(\alpha + \log k)$ time; (ii) the update operation takes as input the LF-interval graph for an α-balanced DBWT D_δ^α of BWT L_δ, and it outputs the LF-interval graph for a $(2\alpha+1)$-balanced DBWT $D_{\delta+1}^{2\alpha+1}$ of BWT $L_{\delta+1}$ with at most two α-heavy DBWT-repetitions and at most two α-heavy F-intervals.*

Proof. See the full version of the paper [24]. ◀

From Theorem 5, the update operation outputs the LF-interval graph for a $(2\alpha+1)$-balanced DBWT $D_{\delta+1}^{2\alpha+1}$ of BWT $L_{\delta+1}$, which is not α-balanced. The output DBWT $D_{\delta+1}^{2\alpha+1}$ is balanced into an α-balanced DBWT $D_{\delta+1}^{\alpha}$ by the balancing operation presented in Section 5.6. The update of the LF-interval graph presented in this section takes $O(\alpha + \log k)$ time, which results in an $O(\alpha n + n \log k)$-time construction of the RLBWT from an input string of length n. The following two sections (Section 5.4 and Section 5.5) present two modified updates of the LF-interval graph in $O(\alpha + \log k)$-time and $O(\alpha)$-time, respectively, in order to achieve the $O(\alpha n + r \log r)$-time construction of an RLBWT with $O(r \log n)$ bits of working space.

5.4 $O(\alpha + \log k)$-time update of LF-interval graph

This section presents the update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$ taking an LF-interval graph $\mathsf{Grp}(D_\delta^\alpha)$ and the first character c of suffix $T_{\delta+1}$ as input and running in $O(\alpha + \log k)$ time by modifying the foundation of the update operation presented in Section 5.3. For the node u_{i-1} previous to the node u_i representing special character \$ in the doubly linked list of U and the node u_{i+1} next to u_i, update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$ is applied if neither u_{i-1} nor u_{i+1} has labels including character c.

We first present a modified update of the B-tree of V in $O(\log k)$ time. This update replaces the original update of the B-tree. The following lemma holds with respect to nodes searched for using the B-tree of V in the replace-node step of this update operation.

▶ **Lemma 6.** *For the node $v_g \in V$ searched for using the B-tree of V in the replace-node step of the update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$, v_g satisfies any one of the following three properties: (i) for undirected edge $(u_g, v_g) \in E_{LF}$ and node $u_{g+1} \in U$ next to u_g in the doubly linked list of U, the two consecutive nodes u_g and u_{g+1} have labels including different characters, and the label of u_{g+1} does not include special character \$; (ii) for the node $u_{g+2} \in U$ next to u_{g+1} in the doubly linked list of U, the two nodes u_g and u_{g+2} have labels including different characters, and the label of u_{g+1} includes special character \$; (iii) the label of u_g includes special character \$.*

Proof. See the full version of the paper [24]. ◀

Thus, the B-tree of V stores only the nodes in V satisfying one of the three conditions of Lemma 6, because Lemma 6 ensures only such nodes are searched for in the replace-node step of this update operation.

The target nodes inserted into the B-tree of V (respectively, the target nodes deleted from the B-tree of V) are limited to four (respectively, three) according to the following lemma.

▶ **Lemma 7.** *Nodes $u_i \in U$ and $v_i \in V$ are the nodes removed from U and V by the replace-node steps, respectively. Node $u_{i-1} \in U$ is the node previous to node u_i in the doubly linked list of U, and $v_{i-1} \in V$ is the node connected to u_{i-1} by undirected edge $(u_{i-1}, v_{i-1}) \in E_{LF}$. Node $v_j \in V$ is the node removed from V by the split-node step, and $v_{j'}, v_{j'+1} \in V$ are the nodes newly created by the same step. Similarly, $v_{i'} \in V$ and $v_{x'} \in V$ are the nodes created by the replace-node and insert-node steps, respectively. Then, the following two statements hold for the update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$: (i) targets inserted into the B-tree of V can be limited to only four nodes $v_{i-1}, v_{i'}, v_{x'}$, and $v_{j'+1}$; (ii) the targets deleted from the B-tree of V can be limited to only three nodes, v_{i-1}, v_i, and v_j.*

Proof. See the full version of the paper [24]. ◀

Thus, all the nodes inserted into the B-tree of V can be found by searching for only four nodes $v_{i-1}, v_{i'}, v_{x'}$, and $v_{j'+1}$ and checking whether or not each one, $v_{i-1}, v_{i'}, v_{x'}$, and $v_{j'+1}$, satisfies one of the three conditions presented in Lemma 6. Similarly, all the nodes deleted from the B-tree of V can be found by searching for only three nodes v_{i-1}, v_i, and v_j and checking whether or not each of these nodes satisfies one of the three conditions. This modified update of the B-tree takes $O(\log k)$ time in total.

The algorithm of update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$ is the same as that of the original update operation presented in Section 5.3 except for the algorithm updating the B-tree of V. Hence, update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$ takes $O(\alpha + \log k)$ time in total. The following lemma concerning the theoretical results on this update operation holds.

▶ **Lemma 8.** *Assume that the B-tree of V in LF-interval graph $\mathsf{Grp}(D_\delta^\alpha)$ contains only nodes satisfying one of the three conditions of Lemma 6: (i) update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$ runs in $O(\alpha+\log k)$ time; (ii) the update operation outputs the LF-interval graph $\mathsf{Grp}(D_{\delta+1}^{2\alpha+1})$ for a $(2\alpha+1)$-balanced DBWT $D_{\delta+1}^{2\alpha+1}$ of BWT $L_{\delta+1}$ with at most two α-heavy DBWT-repetitions and at most two α-heavy F-intervals; (iii) the B-tree of V in the outputted LF-interval graph contains only nodes satisfying one of the three conditions of Lemma 6.*

In the next subsection, the second modified update operation of LF-interval graph achieves $O(\alpha)$ time using the B-tree of V containing only nodes satisfying one of the three conditions of Lemma 6.

5.5 $O(\alpha)$-time update of LF-interval graph

This section presents fast update operation $\mathsf{fastUpdate}(\mathsf{Grp}(D_\delta^\alpha), c)$, which takes an LF-interval graph $\mathsf{Grp}(D_\delta^\alpha)$ and the first character c of suffix $T_{\delta+1}$ as input and runs in $O(\alpha)$ time. This time is achieved by modifying the foundation of the update operation presented in Section 5.3, and the B-tree of V needs to contain only nodes of satisfying one of the three conditions of Lemma 6 in the input and output LF-interval graphs, similar to the update operation in Section 5.4 (Lemma 8).

For node $u_{i-1} \in U$ previous to node $u_i \in U$ that represents special character \$ in the doubly linked list of U and node $u_{i+1} \in U$ next to u_i, the fast update operation is applied if either or both u_{i-1} and u_{i+1} have a label including character c; the update operation in Section 5.4 is applied otherwise. The large computational demand of the update operation on LF-interval graphs presented in Section 5.3 derives from the access and update of the B-tree of V in $O(\log k)$ time, resulting in an $O(\alpha + \log k)$ time update of LF-interval graphs. We present two improvements to the foundation of the update operation: (i) deletion and insertion operations of the B-tree of V in $O(1)$ time and (ii) the replace-node step in $O(1)$ time without using the B-tree of V. The details of the fast operation are presented in the full version of the paper [24].

Deletion and insertion operations of B-tree in constant time. Generally, inserting/deleting a key into/from the B-tree of V takes $O(\log k)$ time. We present $O(1)$-time deletion and insertion operations of a specific node in the B-tree of V without the need for heavyweight operations to maintain the balance of the B-tree.

Recall that (i) $u_{i'} \in U$ and $v_{i'} \in V$ are the nodes created by the replace-node step, (ii) $v_j \in V$ is the node removed from V by the split-node step, and (iii) $u_{x'} \in U$ and $v_{x'} \in V$ are the nodes created by the insert-node step. Let $u_{i'-1} \in U$ (respectively, $u_{i'+1} \in U$) be the node previous to node $u_{i'}$ (respectively, the node next to node $u_{i'}$) in the doubly linked list of U after the insert-node step has been executed. Then, there exist two nodes $v_{i'-1}$ and $v_{i'+1} \in V$ such that $(u_{i'-1}, v_{i'-1}), (u_{i'+1}, v_{i'+1}) \in E_{LF}$.

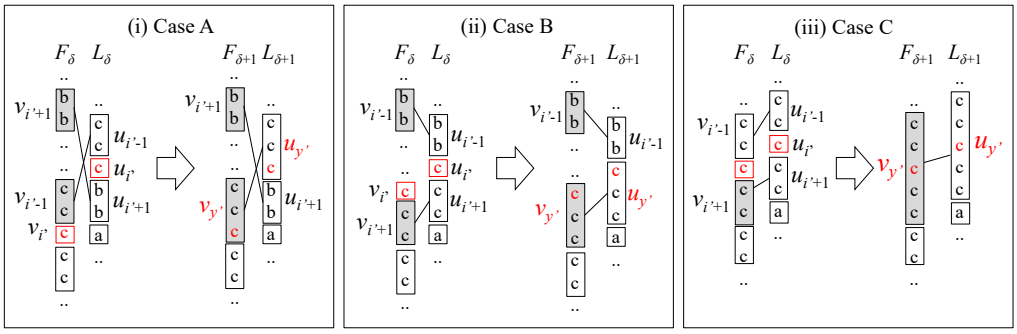


Figure 5 Three cases A, B, and C for preprocessing deletions/insertions in the B-tree. Gray nodes are stored in the B-tree.

In the fast update operation, the target nodes deleted from the B-tree are limited to at most four nodes $v_i, v_j, v_{i'-1}$, and $v_{i'+1}$, which is similar to Lemma 7-(ii). In the doubly linked list of V, node v_i is replaced with the new node $v_{x'}$ created in the insert-node step of the update operation. Because the two nodes v_i and $v_{x'}$ represent special character \$, both satisfy the third condition of Lemma 6. Thus, both nodes v_i and $v_{x'}$ are placed on the root of the B-tree. Hence, v_i can be deleted and $v_{x'}$ can be inserted into the B-tree in $O(1)$ time without balancing the B-tree.

Next, in the doubly linked list of V, node v_j is replaced with the two nodes $v_{j'}$ and $v_{j'+1} \in V$ that were created in the split-node step of the update operation. Node $v_{j'}$ satisfies none of the three conditions of Lemma 6. Thus, v_j is deleted from the B-tree of V, and $v_{j'+1}$ (that is, not $v_{j'}$) is only inserted into the B-tree of V if v_j is contained in the B-tree of V because of the next lemma.

▶ **Lemma 9.** *The following two statements hold: (i) $v_{j'}$ satisfies none of the three conditions of Lemma 6; (ii) for the node $u_j (\neq u_{i-1})$ connected to v_j by undirected edge $(u_j, v_j) \in E_{LF}$, $v_{j'+1}$ satisfies any one of the three conditions of Lemma 6 if and only if v_j satisfies any one of the three conditions.*

Proof. See the full version of the paper [24]. ◀

Because v_j is replaced with $v_{j'+1}$ in the doubly linked list of V, the element representing v_j can be replaced with the element representing $v_{j'+1}$ in the B-tree from Lemma 1. Hence, v_j is deleted from the B-tree of V and $v_{j'+1}$ is inserted into the B-tree of V in $O(1)$ time without balancing the B-tree. Illustrations of the replacement of two nodes v_i and v_j in the doubly linked list of V can be found in the full version of the paper [24].

The above procedure inserts $v_{j'+1}$ into the B-tree of V even if the node satisfies none of the three conditions of Lemma 6. This is because Lemma 9-(ii) assumes that $u_j \neq u_{i-1}$ holds, but $u_j \neq u_{i-1}$ is not always true. If the assumption holds, $v_{j'+1}$ is appropriately inserted into the B-tree of V. Otherwise (i.e., $v_{j'+1} = v_{i'-1}$ holds), node $v_{j'+1}$ is appropriately deleted from the B-tree of V in $O(1)$ time, which is explained next.

Two nodes $v_{i'-1}$ and $v_{i'+1}$ are appropriately deleted from the B-tree of V, and new nodes are inserted into the B-tree of V in the update operation of the LF-interval graph. For updating the B-tree of V in $O(1)$ time, we merge at most three nodes $v_{i'}$, $v_{i'-1}$, and $v_{i'+1}$ into a new node in a prepossessing step. The merging of nodes and update of the B-tree are performed according to the following three cases.

Case A: $v_{i'-1}$ has a label including character c and $v_{i'+1}$ does not have a label including character c (Figure 5-(i)). First, the two consecutive nodes $u_{i'-1}$ with label (c, ℓ) and $u_{i'}$ with label $(c, 1)$ are merged into a new one $u_{y'}$ with label $(c, \ell + 1)$ in the doubly linked list of U. Then, set V is updated according to the merge of the two nodes in U, i.e., the two consecutive nodes $v_{i'-1}$ and $v_{i'}$ are merged into a new one $v_{y'}$ with label $(c, \ell + 1)$ in the doubly linked list of V.

Node $v_{i'+1}$ is kept in the B-tree of V (if the node is contained in the B-tree). Node $v_{i'-1}$ is deleted from the B-tree of V and new node $v_{y'}$ is inserted only if $v_{i'-1}$ is contained in the B-tree of V. Because $v_{i'-1}$ is replaced by $v_{y'}$ in the doubly linked list of V, $v_{i'-1}$ can be deleted from the B-tree of V and $v_{y'}$ can be inserted in $O(1)$ time from Lemma 1.

Case B: $v_{i'-1}$ does not have a label including character c, and $v_{i'+1}$ has a label including character c (Figure 5-(ii)). First, the two consecutive nodes $u_{i'}$ and $u_{i'+1}$ with label (c, ℓ') are merged into a new node $u_{y'}$ with label $(c, \ell' + 1)$ in the doubly linked list of U. Then, set V is updated according to the merge of the two nodes in U.

In this case, $v_{i'-1}$ is contained in the B-tree of V, and the node is kept. Node $v_{i'+1}$ is deleted from the B-tree of V and new node $v_{y'}$ is inserted only if $v_{i'+1}$ is contained in the B-tree of V. Because $v_{i'+1}$ is replaced with $v_{y'}$ in the doubly linked list of V, $v_{i'+1}$ is deleted from the B-tree of V and $v_{y'}$ is inserted in $O(1)$ time from Lemma 1.

Case C: both $v_{i'-1}$ and $v_{i'+1}$ have labels including the same character c (Figure 5-(iii)). First, the three consecutive nodes $u_{i'-1}$, $u_{i'}$ and $u_{i'+1}$ are merged into a new node $u_{y'}$ with label $(c, \ell + \ell' + 1)$ including the same character c in the doubly linked list of U. Then, set V is updated according to the merge of the three nodes in U.

In this case, $v_{i'-1}$ is not contained in the B-tree of V. Node $v_{i'+1}$ is deleted from the B-tree and the new node $v_{y'}$ is inserted if $v_{i'+1}$ is contained in the tree; otherwise, $v_{y'}$ is not inserted into the tree. Because $v_{i'+1}$ is replaced by $v_{y'}$ in the doubly linked list of V, $v_{i'+1}$ is deleted and $v_{y'}$ is inserted into the B-tree of V in $O(1)$ time from Lemma 1.

One of the three cases always holds (see the full version of the paper [24]). The preprocessing of the B-tree of V (i.e., the merging of nodes) does not affect Lemma 9. Hence, updating the B-tree takes $O(1)$ time.

Replace-node step in constant time. The replace-node step is performed in $O(1)$ time by finding the position for inserting a new node $v_{i'}$ with label $(c, 1)$ into the doubly linked list of V without accessing the B-tree of set V. This is made possible if either or both u_{i-1} and u_{i+1} have labels including the same character c. The following lemma holds.

▶ **Lemma 10.** *If either or both u_{i-1} and u_{i+1} have labels including the same character c, the position for inserting the new node $v_{i'}$ into the doubly linked list of V can be found in $O(1)$ time.*

Proof. See the full version of the paper [24]. ◀

The following lemma concerning the conclusion of the fast update operation holds.

▶ **Lemma 11.** *Assume that the B-tree of V in LF-interval graph $\mathsf{Grp}(D^{\alpha}_{\delta})$ contains only the nodes that satisfy one of the three conditions of Lemma 6: (i) fast update operation $\mathsf{fastUpdate}(\mathsf{Grp}(D^{\alpha}_{\delta}), c)$ runs in $O(\alpha)$ time; (ii) the fast update operation outputs the LF-interval graph for a $(2\alpha+1)$-balanced DBWT $D^{2\alpha+1}_{\delta+1}$ of BWT $L_{\delta+1}$ with at most two α-heavy DBWT-repetitions and at most two α-heavy F-intervals; (iii) the B-tree of V in the outputted LF-interval graph contains only nodes satisfying one of the three conditions of Lemma 6.*

Proof. See the full version of the paper [24]. ◀

Both update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$ and fast update operation $\mathsf{fastUpdate}(\mathsf{Grp}(D_\delta^\alpha), c)$ output the LF-interval graph for a $(2\alpha+1)$-balanced DBWT $D_{\delta+1}^{2\alpha+1}$ of BWT $L_{\delta+1}$ with at most two α-heavy DBWT-repetitions and at most two α-heavy F-intervals. The outputs are balanced by the balancing operation presented in the next subsection such that the LF-interval graph represents an α-balanced DBWT $D_{\delta+1}^\alpha$ of BWT $L_{\delta+1}$.

5.6 Balancing operation of the LF-interval graph

Balancing operation $\mathsf{balance}(\mathsf{Grp}(D_\delta))$ takes the LF-interval graph $\mathsf{Grp}(D_\delta)$ for a DBWT D_δ of BWT L_δ as input, and it outputs the LF-interval graph $\mathsf{Grp}(D_\delta^\alpha)$ for an α-balanced DBWT D_δ^α of the same BWT. The basic idea behind the balancing operation is to iteratively remove each of nodes representing α-heavy DBWT repetitions and α-heavy F-intervals from the given LF-interval graph by splitting the chosen node into two nodes. The balancing operation repeats this process until it obtains the LF-interval graph for an α-balanced DBWT.

We explain the algorithm of the balancing operation. At each iteration, the balancing operation processes the LF-interval graph for an $O(\alpha)$-balanced DBWT with $O(\alpha)$ α-heavy DBWT-repetitions and $O(\alpha)$ α-heavy F-intervals. Let $(u_i, v_i) \in E_{LF}$ be an undirected edge such that node $u_i \in U$ represents an α-heavy DBWT-repetition, or node $v_i \in V$ represents an α-heavy F-interval. At the iteration, node u_i is split into two nodes, and v_i is split into two nodes according to the splitting of u_i. The LF-interval graph $\mathsf{Grp}(D_\delta^{O(\alpha)})$ is updated according to the splitting of u_i and v_i. One iteration of the balancing operation takes $O(\alpha)$ time. See the full version of the paper [24] for the details of the balancing operation.

The following lemma concerning the theoretical results on the balancing operation holds.

▶ **Lemma 12.** *For the LF-interval graph $\mathsf{Grp}(D_\delta^{2\alpha+1})$ for a $(2\alpha+1)$-balanced DBWT including at most two α-heavy DBWT-repetitions and at most two α-heavy F-intervals, we assume that the B-tree of V in the LF-interval graph contains only nodes satisfying one of the three conditions of Lemma 6. Then, balancing operation $\mathsf{balance}(\mathsf{Grp}(D_\delta^{2\alpha+1}))$ takes $O(\alpha)$ time per iteration for all $\alpha \geq 4$, and the B-tree of V in the outputted LF-interval graph contains only nodes satisfying one of the three conditions of Lemma 6.*

Proof. See the full version of the paper [24]. ◀

6 R-comp algorithm

In this section, we present the r-comp algorithm, and we also present its space and time complexities. The r-comp algorithm takes input string T and parameter $\alpha \geq 2$, and it outputs the RLBWT of T. The pseudo-code of r-comp is given in Algorithm 1.

The algorithm reads T from its end to its beginning (i.e., $T[n], T[n-1], \ldots, T[1]$), and it gradually builds the LF-interval graph $\mathsf{Grp}(D_n^\alpha)$ for an α-balanced DBWT D_n^α of BWT L_n of T. At each $\delta \in \{1, 2, \ldots, n-1\}$, LF-interval graph $\mathsf{Grp}(D_\delta^\alpha)$ for an α-balanced DBWT D_δ^α of BWT L_δ of $T[(n-\delta+1)..n]$ (i.e., suffix T_δ) is built. For the node $u_i \in U$ representing special character \$ in the doubly linked list of set U of nodes in LF-interval graph $\mathsf{Grp}(D_\delta^\alpha)$, two nodes $u_{i-1} \in U$ and $u_{i+1} \in U$ are previous and next to node u_i, respectively, in the list. If neither the label of u_{i-1} nor the label of u_{i+1} includes the $(n-\delta)$-th character c of T (i.e., the first character c of $T_{\delta+1}$), the update operation $\mathsf{update}(\mathsf{Grp}(D_\delta^\alpha), c)$ described in Section 5.4 is applied; otherwise the fast update operation $\mathsf{fastUpdate}(\mathsf{Grp}(D_\delta^\alpha), c)$ described in Section 5.5

Algorithm 1 R-comp algorithm. The algorithm takes string T and parameter $\alpha \geq 2$ as input, and it outputs the RLBWT of T. n : length of T; D_δ^α : α-balanced DBWT of BWT L_δ of $T[(n-\delta+1)..n]$; $\mathsf{Grp}(D_\delta^\alpha)$: the LF-interval graph of α-balanced DBWT D_δ^α.

```
1: function R-COMP(T, α)
2:     D_1^α ← $                                  ▷ Initialize D_1^α as special character $
3:     build Grp(D_1^α)
4:     for δ = 1, 2, ..., n − 1 do
5:         c ← T[n − δ]                           ▷ Read character c from T[n − δ]
6:         if neither the label of u_{i−1} nor the label of u_{i+1} includes character c then
7:             Grp(D_{δ+1}^{2α+1}) ← update(Grp(D_δ^α), c)      ▷ The update operation in Sec. 5.4
8:         else
9:             Grp(D_{δ+1}^{2α+1}) ← fastUpdate(Grp(D_δ^α), c) ▷ The fast update operation in Sec. 5.5
10:        end if
11:        Grp(D_{δ+1}^α) ← balance(Grp(D_{δ+1}^{2α+1}))      ▷ The balancing operation in Sec. 5.6
12:    end for
13:    Recover D_n^α from Grp(D_n^α)
14:    Convert D_n^α into the RLBWT of T
15:    Return the RLBWT
16: end function
```

is applied. Both the update operation and the fast update operation output an LF-interval graph $\mathsf{Grp}(D_{\delta+1}^{2\alpha+1})$ for DBWT $D_{\delta+1}^{2\alpha+1}$ of BWT $L_{\delta+1}$ that is not α-balanced. Thus, the r-comp algorithm balances the LF-interval graph such that it represents an α-balanced DBWT $D_{\delta+1}^\alpha$ using the balancing operation in Section 5.6 as $\mathsf{Grp}(D_{\delta+1}^\alpha)$. After $(n-1)$ iterations of those steps, the LF-interval graph $\mathsf{Grp}(D_n^\alpha)$ for α-balanced DBWT D_n^α of BWT L_n is obtained; it is then converted into D_n^α using the doubly linked list of U. Finally, D_n^α is converted into the RLBWT of T.

6.1 Space and time complexities

Space complexity. The r-comp algorithm requires $O(k \log n)$ bits of space for k DBWT-repetitions in the DBWT D_n^α because the LF-interval graph $\mathsf{Grp}(D_n^\alpha)$ for the DBWT requires $O(k \log n)$ bits of space. The value of k depends on the number of executions of update, fast update, and balancing operations executed by the r-comp algorithm. The following lemma ensures that k can be bounded by $O(r)$.

▶ **Lemma 13.** *We modify the fast update operation. Then, the following three statements hold: (i) this modification does not affect Lemma 11; (ii)* $k \leq r + k_{\mathsf{split}}$*; (iii)* $k_{\mathsf{split}} \leq \frac{2r}{\lceil \alpha/2 \rceil - 7}$ *holds for any constant* $\alpha \geq 16$*.*

Proof. See the full version of the paper [24]. ◀

Because $k = O(r)$ by Lemma 13, the following theorem is obtained.

▶ **Theorem 14.** *The r-comp algorithm takes* $O(r \log n)$ *bits of working space for* $\alpha \geq 16$*.*

Time complexity. The bottleneck of r-comp is the update operation of LF-interval graph with $O(\alpha + \log k)$ time in Section 5.4. The number of executions of the update operation can be bounded by $O(r)$. This fact indicates that we can bound the running time of r-comp by $O(\alpha n + r \log k)$, i.e., $O(\alpha n + r \log r)$. Finally, we obtain the following theorem.

▶ **Theorem 15.** *R-comp runs in $O(\alpha n + r \log r)$ time for $\alpha \geq 16$.*

Proof. See the full version of the paper [24]. ◀

7 Experiments

Setup. We empirically tested the performance of the r-comp algorithm on strings from four datasets with different types of highly repetitive strings: (i) nine strings from the Pizza&Chili repetitive corpus [29]; (ii) three strings (boost, samtools, and sdsl) of the latest revisions of Git repositories, each of which is 1GB in size; (iii) a 37GB string (enwiki) of English Wikipedia articles with a complete edit history [33]; and (iv) a 59GB string (chr19.1000) obtained by concatenating chromosome 19 from 1,000 human genomes in the 1000 Genomes Project [32]. See the full version of the paper [24] for the relevant statistics in the datasets. The ratio n/r of string length n to the number r of BWT-runs in BWT is the compression ratio of each string. The strings with high compression ratios are versions of Wikipedia articles (einstein.de.txt and einstein.en.txt), revisions of Git repositories (boost, samtools, and sdsl), and 1000 human genomes (chr19.1000).

We implemented two versions of the r-comp algorithm (i.e., r-comp and r-comp$_{\text{saving}}$). Here, r-comp is the straightforward implementation of the r-comp algorithm presented in Section 6; r-comp$_{\text{saving}}$ is a space-saving implementation of the r-comp algorithm. This version is more space efficient than r-comp because it uses a grouping technique with parameter $g = 16$. We compared r-comp and r-comp$_{\text{saving}}$ with three state-of-the-art algorithms, one indirect construction algorithm of RLBWT (Big-BWT) and two direct construction algorithms of RLBWT (the PP and Faster-PP methods), which were reviewed in Section 2 and summarized in Table 1. The comparisons were performed using a single thread on one core of a CPU. The source code of the r-comp algorithm is available at `https://github.com/kampersanda/rcomp`. See the full version of the paper [24] for the details of the setup.

Results. A comparison of the r-comp variants (r-comp and r-comp$_{\text{saving}}$) shows that the working space of r-comp$_{\text{saving}}$ was 3.8–4.4 times smaller than that of r-comp, whereas the construction time of r-comp$_{\text{saving}}$ was at most only 2.0 times slower and at most 2.0 times faster on world_leaders and samtools. These results show r-comp$_{\text{saving}}$ has a high compression performance when compared with r-comp. Comparisons of the experimental results of r-comp$_{\text{saving}}$ and the other methods are presented below.

r-comp$_{\text{saving}}$ was the fastest in the comparison to the direct RLBWT constructions of the PP and Faster-PP methods. Especially for strings with a large ratio n/r, r-comp$_{\text{saving}}$ was 81–118 times faster than the PP method and 3.8–5.1 times faster than the Faster-PP method; the working space of r-comp$_{\text{saving}}$ was 2.4–6.1 times larger than that of the PP method and 2.8–2.9 times larger than that of the Faster-PP method, which shows that the working space of r-comp$_{\text{saving}}$ is reasonable considering its construction time. For the large dataset enwiki, r-comp$_{\text{saving}}$ finished the construction in 6.8 hours, whereas the Faster-PP method took 15.4 hours. In addition, the PP method did not finish within 24 hours. For the 1000 human genomes chr19.1000, r-comp$_{\text{saving}}$ finished the construction in 11 hours, whereas the PP and Faster-PP methods did not finish within 24 hours.

In the comparison between r-comp and Big-BWT, r-comp was more space efficient than Big-BWT on most strings. Because r was much smaller than $|\mathsf{PFP}|$, the results are consistent with the theoretical bound $O(r \log n)$ of the working space of the r-comp algorithm (Theorem 14). On strings with large values of $|\mathsf{PFP}|/r$, whereas r-comp$_{\text{saving}}$ was slower

than Big-BWT, the difference in the construction times between r-comp$_{\text{saving}}$ and Big-BWT were reasonable if one considers the space efficiency of r-comp$_{\text{saving}}$. For example, for boost, r-comp was 26 times more space efficient and only 2.7 times slower; for world_leaders, r-comp was 3.9 times more space efficient and only 1.5 times slower.

On the large datasets (i.e., enwiki and chr19.1000), r-comp was more space-efficient but slower than Big-BWT; r-comp was 2.3 times space-efficient but 10 times slower than Big-BWT on enwiki; r-comp was 2.5 times space-efficient but 11 times slower than Big-BWT on chr19.1000.

Overall, r-comp$_{\text{saving}}$ was the fastest RLBWT construction in $O(r \log n)$ bits of space. Although Big-BWT was faster than r-comp$_{\text{saving}}$, it was not space efficient for several strings (e.g., boost). On most datasets, r-comp$_{\text{saving}}$ achieved a better tradeoff between construction time and working space. Furthermore, r-comp has a huge advantage in that it supports the insertion of a new character into RLBWT, while BigBWT does not support such an insertion operation.

8 Conclusion

We presented r-comp, the first optimal-time construction algorithm of RLBWT in $O(n)$ time with $O(r \log n)$ bits of working space for highly repetitive strings with $r = O(n/\log n)$. Experimental results using benchmark and real-world datasets of highly repetitive strings demonstrated the superior performance of the r-comp algorithm.

The idea behind the DBWT presented in this paper has a wide variety of applications, and it is applicable to the construction of various types of data structures. Therefore, a future task is to develop optimal-time constructions of various data structures for fast queries.

References

1. Hideo Bannai, Travis Gagie, and Tomohiro I. Refining the r-index. *Theor. Comput. Sci.*, 812:96–108, 2020.
2. Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Block trees. *J. Comput. Syst. Sci.*, 117:1–22, 2021.
3. Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. Composite repetition-aware data structures. In *Proceedings of CPM*, pages 26–39, 2015.
4. Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Trans. Algor.*, 16:17:1–17:54, 2020.
5. Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms Mol. Biol.*, 14:13:1–13:15, 2019.
6. Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *Technical report*, 1994.
7. Dustin Cobas, Veli Mäkinen, and Massimiliano Rossi. Tailoring r-index for document listing towards metagenomics applications. In *Proceedings of SPIRE*, pages 291–306, 2020.
8. Maxime Crochemore, Roberto Grossi, Juha Kärkkäinen, and Gad M. Landau. Computing the Burrows-Wheeler transform in place and in small space. *J. Disc. Algor.*, pages 44–52, 2015.
9. Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In *Proceedings of STOC*, pages 365–372, 1987.
10. Patrick Dinklage, Jonas Ellert, Johannes Fischer, Dominik Köppl, and Manuel Penschuck. Bidirectional text compression in external memory. In *Proceedings of ESA*, pages 41:1–41:16, 2019.
11. Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of FOCS*, pages 390–398, 2000.

12 Isamu Furuya, Takuya Takagi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Takuya Kida. Practical grammar compression based on maximal repeats. *Algorithms*, 13:103, 2020.

13 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67, 2020.

14 Artur Jez. A really simple approximation of smallest grammar. *Theor. Comput. Sci.*, 616:141–150, 2016.

15 Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In *Proceedings of SODA*, pages 1344–1357, 2019.

16 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *Proceedings of STOC*, pages 756–767, 2019.

17 Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. In *Proceedings of FOCS*, pages 1002–1013, 2020.

18 Dominik Kempa and Ben Langmead. Fast and space-efficient construction of AVL grammars from the LZ77 parsing. In *Proceedings of ESA*, 2021.

19 N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Proceedings of DCC*, pages 296–305, 1999.

20 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Proceedings of SODA*, pages 408–424, 2017.

21 Gonzalo Navarro and Yakov Nekrich. Optimal dynamic sequence representations. *SIAM J. Comput.*, 43:1781–1806, 2014.

22 Gonzalo Navarro, Carlos Ochoa, and Nicola Prezza. On the approximation ratio of ordered parsings. *IEEE Trans. Inform. Theory*, 67:1008–1026, 2021.

23 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Discret. Appl. Math.*, 274:116–129, 2020.

24 Takaaki Nishimoto, Shunsuke Kanda, and Yasuo Tabei. An optimal-time RLBWT construction in BWT-runs bounded space. *CoRR*, abs/2202.07885, 2022.

25 Takaaki Nishimoto and Yasuo Tabei. LZRR: LZ77 parsing with right reference. *Inf. Comput.*, page 104859, 2021.

26 Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on BWT-runs bounded space. In *Proceedings of ICALP*, pages 101:1–101:15, 2021.

27 Takaaki Nishimoto and Yasuo Tabei. R-enum: Enumeration of characteristic substrings in BWT-runs bounded space. In *Proceedings of CPM*, pages 21:1–21:21, 2021.

28 Tatsuya Ohno, Kensuke Sakai, Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. A faster implementation of online RLBWT and its application to LZ77 parsing. *J. Disc. Algor.*, 52-53:18–28, 2018.

29 Pizza&Chili repetitive corpus. `http://pizzachili.dcc.uchile.cl/repcorpus.html`.

30 Alberto Policriti and Nicola Prezza. LZ77 computation based on the run-length encoded BWT. *Algorithmica*, 80:1986–2011, 2018.

31 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302:211–222, 2003.

32 The 1000 Genomes Project Consortium. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491:56–65, 2012.

33 Enwiki dump progress on 20210401: All pages with complete edit history (`xml-p1p873`). `https://dumps.wikimedia.org/enwiki/`.

34 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, 23:337–343, 1977.

Space Characterizations of Complexity Measures and Size-Space Trade-Offs in Propositional Proof Systems

Theodoros Papamakarios
Department of Computer Science, University of Chicago, IL, USA

Alexander Razborov
University of Chicago, IL, USA
Steklov Mathematical Institute, Moscow, Russia

Abstract

We identify two new big clusters of proof complexity measures equivalent up to polynomial and $\log n$ factors. The first cluster contains, among others, the logarithm of tree-like resolution size, regularized (that is, multiplied by the logarithm of proof length) clause and monomial space, and clause space, both ordinary and regularized, in regular and tree-like resolution. As a consequence, separating clause or monomial space from the (logarithm of) tree-like resolution size is the same as showing a strong trade-off between clause or monomial space and proof length, and is the same as showing a super-critical trade-off between clause space and depth. The second cluster contains width, Σ_2 space (a generalization of clause space to depth 2 Frege systems), both ordinary and regularized, as well as the logarithm of tree-like size in the system $R(\log)$. As an application of some of these simulations, we improve a known size-space trade-off for polynomial calculus with resolution. In terms of lower bounds, we show a quadratic lower bound on tree-like resolution size for formulas refutable in clause space 4. We introduce on our way yet another proof complexity measure intermediate between depth and the logarithm of tree-like size that might be of independent interest.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof Complexity, Resolution, Size-Space Trade-offs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.100

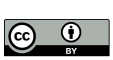
Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2021/074/ [33]

1 Introduction

With the rise of computer science, questions like "can we solve a problem?" got a quantitative counterpart: "how hard is it to solve a problem?". Proof complexity deals with the quantitative version of "can we prove a theorem?", namely, the question "how hard is it to prove a theorem?". The systematic study of the latter question for propositional proof systems started with Cook and Reckhow [13], where its fundamental role in complexity theory was identified.

The most natural, arguably also the most important, measure of the complexity of a proof is its size, and indeed, much of the research in propositional proof complexity has concentrated on proof size lower bounds. But given in particular their role in proof systems of practical significance, several other natural complexity measures have been considered, and that has led to a considerable line of study about relations between them (*simulations*), lack of relations thereof (*separations*) and the inherent impossibility of optimizing two different measures at once (*trade-offs*). To aid further discussion, let us review those measures and previous results that are most pertinent to this work.




A measure that directly emerged from the study of proof size lower bounds is width; the width of a resolution proof is the number of literals in the largest clause occurring in the proof. Its importance was accentuated by Ben-Sasson and Wigderson [8], who, building on the earlier works of Clegg et al. [12] and Impagliazzo et al. [23] showing an analogous result for polynomial calculus, showed that a short resolution proof can be transformed into a narrow one. Namely, we have

$$W(F \vdash \bot) \leq \log S_T(F \vdash \bot) + W(F), \tag{1}$$

$$W(F \vdash \bot) \leq O\left(\sqrt{n \log S_R(F \vdash \bot)}\right) + W(F). \tag{2}$$

Here $W(F \vdash \bot)$, $S_T(F \vdash \bot)$ and $S_R(F \vdash \bot)$ stand for the minimum width, tree-like size and DAG-like size respectively of refuting an unsatisfiable CNF F in resolution; similar notation is employed throughout the paper. $W(F)$ is the maximum width of a clause in F.

Space complexity for propositional proofs was introduced in [16, 1]. Esteban and Torán [16] showed that a short tree-like resolution proof can be transformed into a resolution proof of small clause space:

$$\mathrm{CSpace}(F \vdash \bot) \leq \log S_T(F \vdash \bot). \tag{3}$$

Atserias and Dalmau [2] demonstrated the first instance of the relationship between space and width, showing that a resolution proof having small clause space can be transformed into a narrow one:

$$W(F \vdash \bot) \leq \mathrm{CSpace}(F \vdash \bot) + W(F). \tag{4}$$

Constructive versions of their result were given by Filmus et al. [18] and Razborov (unpublished), see also Krajíček [26, Theorem 5.5.5]. It is worth noting that (3) and (4) taken together provide a refinement of (1) and that, viewed this way, we relate two *sequential* measures (tree-like size and width) with a *space* measure as an intermediate. We will see more examples of such an interplay in this paper.

More recently, Bonacina [9] showed that for total space in resolution (measured as the sum of widths of clauses in a configuration) we have

$$W(F \vdash \bot) \leq O\left(\sqrt{\mathrm{TSpace}(F \vdash \bot)}\right) + W(F), \tag{5}$$

and Galesi et al. [19] showed a weakened version of (4), but for the analogue of clause space in stronger proof systems operating with polynomials (or in fact even arbitrary Boolean functions of monomials):

$$W(F \vdash \bot) \leq O\left(\left(\mathrm{MSpace}(F \vdash \bot)\right)^2\right) + W(F). \tag{6}$$

Regularized[1] versions μ^* of space complexity measures are defined by multiplying the measure in question μ by the logarithm of the proof length; these were considered e.g. by Ben-Sasson [4] and Razborov [37]. The latter paper also contains a suggestion that the "right" level of precision when comparing measures of this kind are up to polynomial and $\log n$ factors;[2] we will henceforth call two measures *equivalent* if they simulate each other

[1] The paper [37] used the word "amortized" but Sam Buss pointed out to us that it is somewhat misleading in this context.

[2] Note that the size/length measures appear in this set-up under a logarithm. Hence this corresponds to *quasi*-polynomial simulations in the Cook-Reckhow framework.

in this sense. The paper [37] identified a big cluster of ordinary and regularized space complexity measures, including total and variable space, that are all equivalent to proof depth in resolution. One notable measure that defied this classification was (regularized) clause space.[3]

Our contributions

In this paper we identify two other big clusters of equivalent complexity measures not covered by the results in [37]. The cumulative picture combining both previously known and new results is summarized in Figure 1. There, arrows are to be interpreted as inequalities, and $\approx$

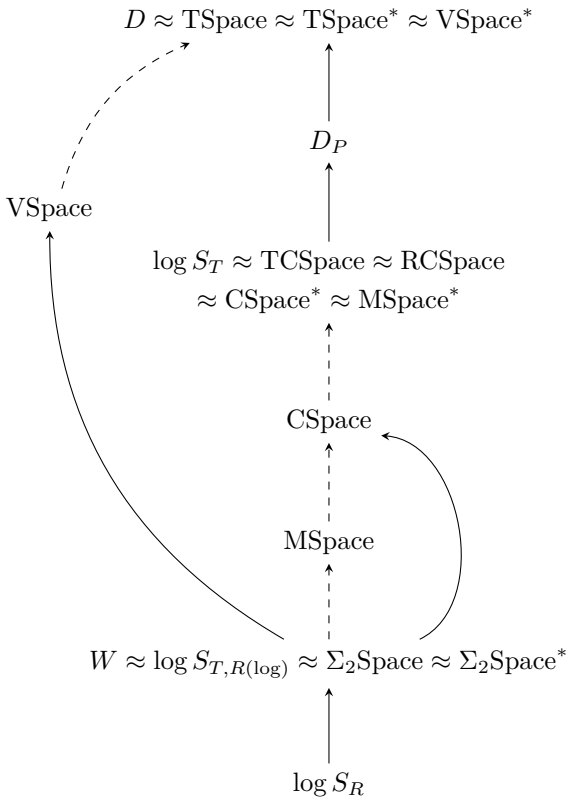


Figure 1 Simulations.

as equality, both up to polynomial and $\log n$ factors. A solid arrow from μ_1 to μ_2 indicates that a separation between μ_1 and μ_2 is known, that is, it additionally indicates that there exists a sequence $\{F_n\}$ of unsatisfiable CNFs such that $\mu_2(F_n \vdash \bot) \geq (\mu_1(F_n \vdash \bot)+\log n)^{\omega(1)}$. To improve readability, we have omitted from Figure 1 the argument $F \vdash \bot$.

Let us briefly explain this picture. The first new cluster is centered around the logarithm of tree-like resolution size. Given the proof method of the simulation (3) in [16], it can be obviously strengthened in two directions: by replacing the left-hand side with clause space in *tree-like* resolution or by replacing it with *regularized* clause space. Tree-like clause space in resolution was shown to be equivalent to the logarithm of tree-like size in the same paper [16, Corollary 5.1]; in other words, after this replacement in the left-hand side, the bound (3) becomes tight, within the precision we are tolerating.

[3] A technical remark: [37, Theorem 3.2] does not apply to clause space as it is not bounded from below by the number of variables.

We show that the second variant, that is *regularized* clause space, is also equivalent to the logarithm of tree-like resolution size, and this result extends to also include regularized *monomial* space to the same cluster. Given that [16, Corollary 5.1] also holds for (ordinary) clause space in *regular* resolution [16, Corollary 4.2], this means that all these space measures turn out to be equivalent to each other and to the log of tree-like resolution size. We also remark (given the results above, this readily follows from definitions) that *regularized* versions of the clause space in tree-like or regular resolution are also in this cluster.

The question of whether (ordinary) clause space also belongs here is what we consider to be a major, and most likely very difficult, open problem. But since it has turned out to be closely related to several other threads in proof complexity, we prefer to keep the momentum and defer further discussion to the concluding Section 5.

Our second cluster is presided by resolution width. First, we introduce a natural analogue of clause space in DNF resolution that we call Σ_2 *space*. This can be seen as an extension of clause space to depth 2 Frege systems; indeed, the restriction of Σ_2 space to depth 1 Frege is precisely clause space, and its restriction to k-DNF resolution, for constant k, coincides, up to a constant factor, with the concept of space that has been studied before for such systems (see e.g. [15, 7]). In our model, configurations are *arbitrary* sets of DNFs, and we charge k for every individual k-DNF in the memory. Clearly, $\Sigma_2\mathrm{Space} \leq \mathrm{CSpace}$ and $\Sigma_2\mathrm{Space}^* \leq \mathrm{CSpace}^*$. Then we strengthen the Atserias-Dalmau bound (4) by replacing CSpace with Σ_2Space and continue to show that *both* ordinary and regularized versions of Σ_2 space are actually equivalent to resolution width.

Thus, remarkably, the difficult open question on whether we have a strong trade-off between space and length for clause space gets a relatively easy negative solution for a stronger proof system. We have also been able to locate in this cluster another interesting size measure: the size of tree-like proofs in the system $R(\log)$, which gives a somewhat unexpected generalization of (1). We have not been able to retrieve the equivalence of width and tree-like size in $R(\log)$ from the literature in exactly this form but it is implicit in Lauria [27] and, with a bit of effort, can be traced back as far as Krajíček [24].

It is worth noting that some of the simulations in this cluster work only in the syntactical setting. This comes in contrast with what happens with the other two clusters: all simulations involving clause, monomial, variable and total space, also work in a purely semantic setting. For example, in case of monomial space we can allow arbitrary Boolean functions of monomials as memory configurations and allow any number of sound inferences to be performed at once in each step.

We use (some of) these simulations to prove:

1. *There are unsatisfiable CNFs F of size $O(n)$ with $S(F \vdash \bot) \leq O(n)$, $W(F \vdash \bot) \leq O(1)$ and* $\mathrm{MSpace}^*(F \vdash \bot) \geq \Omega(n/\log n)$ (Theorem 4.1).
   This is an improvement on the previously known bounds $\mathrm{MSpace}^*(F \vdash \bot) \geq \Omega(n^{2/11})$ [3], $\mathrm{MSpace}^*(F \vdash \bot) \geq \Omega(n^{1/4})$ [22] and $\mathrm{MSpace}^*(F \vdash \bot) \geq n^{1/2}/(\log n)^{O(1)}$ [21]. Unlike these previous results, our proof is remarkably simple.
2. *There are unsatisfiable CNFs F of size $O(n)$ with* $\mathrm{CSpace}(F \vdash \bot) = 4$ *and* $S_T(F \vdash \bot) \geq \Omega(n^2/\log n)$ (Theorem 4.7).
   This is a first, admittedly modest, step toward separating clause space and, say, tree-like size; as we already said, we will discuss this question in more details in Section 5. It is for this proof that we need the last unexplained entry D_P on Figure 1: it stands for *positive depth*, and it is a one-sided version of depth. We also remark that the space bound in this result is optimal. More precisely, we make a relatively simple observation (Theorem 4.2) that $\mathrm{CSpace}(F \vdash \bot) \leq 3$ if and only if F is "essentially Horn" in which case it will possess a linear size tree-like resolution refutation.

Finally, let us briefly summarize what is known (to the best of our knowledge) in terms of separating the measures in Figure 1. Let us start with "true" separations, i.e. separations that work modulo polynomial overheads and $\log n$ factors. From now on, for proof complexity measures μ_1, μ_2 we will use the notation $\mu_1 \preceq \mu_2$ to stand for $\mu_1(F \vdash \bot) \leq (\mu_2(F \vdash \bot) \log n)^{O(1)}$ for any CNF F in n variables; $\mu_1 \approx \mu_2$ is the same as $\mu_1 \preceq \mu_2 \wedge \mu_2 \preceq \mu_1$. Clearly $\preceq$ is transitive, and this implies that $\approx$ is an equivalence relation and $\preceq$ imposes a partial order on its equivalence classes.

Bonet and Galesi [10] proved that $W \not\preceq \log S_R$. More precisely, there are constant width formulas F of size $O(n^3)$ such that $S_R(F \vdash \bot) \leq O(n^3)$ and $W(F \vdash \bot) \geq \Omega(n)$. Ben-Sasson [4] proved that VSpace $\not\preceq$ CSpace, and after negating the variables in his formulas, this works two more levels up on Figure 1. Namely, there are constant-width formulas F of size $O(n)$ such that VSpace$(F \vdash \bot) \geq \Omega(n/\log n)$ while $D_P(F \vdash \bot) \leq O(1)$. This also provides a separation between D_P and D that, though, is much easier to prove directly [38, Theorem 4.6]. Without negating the variables, it is easy to see that Ben-Sasson's proof actually gives $D_P(F \vdash \bot) \geq \Omega(n/\log n)$, thus separating D_P from $\log S_T$ and hence from the whole middle cluster. Again, it is also easy to see this directly. Ben-Sasson, Håstad and Nordström [31, 6] separated clause space from width; while it is believed that their formulas should also have large monomial space complexity, the questions of separating clause space from monomial space, as well as monomial space from width are widely open.

Separating space complexity measures from their own regularized versions appear to be a very daunting task in general. As follows from Figure 1, for variable space this is equivalent to separating it from depth [38]. A quadratic separation between VSpace and VSpace* was proved in [37, Section 6], with a disappointingly elaborate proof. Nothing is known in terms of separating CSpace from (the cluster of) CSpace*: Theorem 4.7 makes a progress in that direction, but it is admittedly rather modest. Nothing seems to be known for CSpace vs. MSpace, and our structural picture provides a good heuristic explanation of the difficulty of this question: it would also separate MSpace from MSpace*. Finally, in [32] a quadratic separation between width and monomial space has been established using methods very different from those in [6].

The paper is organized as follows. After giving the necessary definitions in Section 2, in Section 3 we refine (many simulations do not actually involve a polynomial overhead or extra $\log n$ factors) and prove the relations of Figure 1. In Section 4 we prove items 1 and 2 above. The paper is concluded with a few remarks and open problems in Section 5.

2 Preliminaries

A *literal* is a propositional variable x or its negation $\overline{x}$. We let $\overline{\overline{x}} \stackrel{\text{def}}{=} x$. A *clause* is a disjunction (possibly empty) of literals over distinct variables, and a *term* is a conjunction (possibly empty) of such literals. For a clause $C = \ell_1 \vee \cdots \vee \ell_w$, we define the term $\overline{C} \stackrel{\text{def}}{=} \overline{\ell_1} \wedge \cdots \wedge \overline{\ell_w}$; similarly for a term $t = \ell_1 \wedge \cdots \wedge \ell_w$, $\overline{t} \stackrel{\text{def}}{=} \overline{\ell_1} \vee \cdots \vee \overline{\ell_w}$. The *width* of a clause or a term is the number of literals it contains. A *CNF formula* is a conjunction of clauses, and a *DNF formula* is a disjunction of terms. The *width*, $W(F)$, of a CNF or DNF formula F is the width of the largest clause or term it contains. A CNF or DNF formula of width at most w is called *w-CNF* or *w-DNF* respectively. Clauses may be alternatively viewed as 1-DNFs, but the latter class is slightly larger as tautological 1-DNFs like $x \vee \overline{x}$ *are* allowed.

A *partial (truth) assignment* (often called *restriction*) is a mapping from a subset V of all propositional variables to $\{0, 1\}$; it is naturally extended to the negations of the variables in V by $\alpha(\overline{x}) \stackrel{\text{def}}{=} \overline{\alpha(x)}$. The result of applying a partial assignment α to a CNF formula F is

another CNF formula $F|_\alpha$, obtained by deleting from F all literals ℓ such that $\alpha(\ell) = 0$ and deleting all clauses containing a literal ℓ such that $\alpha(\ell) = 1$. Similarly for DNF formulas. $F|_\alpha$ is called the *restriction* of F to α. For a formula F, we write $\alpha \models F$ if every total extension of α satisfies F or, in other words, if $F|_\alpha$ is *semantically* equal to 1. For a set of formulas S, $\alpha \models S$ means $\alpha \models \bigwedge_{F \in S} F$, and for two sets of formulas S and T, we write $S \models T$ if all total assignments satisfying every formula in S also satisfy every formula in T. For a clause C, we denote by α_C the minimal partial assignment such that $\alpha_C \models \overline{C}$.

Resolution is a proof system operating with clauses. Its inference rules are:

$$\frac{C}{C \vee D}, \quad \frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}. \tag{7}$$

The leftmost one is called the *weakening rule*; the rightmost one is called the *resolution rule*. We refer to the variable x in an application of the resolution rule as the variable being *resolved*. One of the reasons to include the (redundant) weakening rule is that it makes resolution proofs closed under restricting by a partial assignment.

The *width* $W(\pi)$ of a resolution proof π is defined as the maximum width of a clause in it, and the width $W(F \vdash \bot)$ is usually defined as the minimum width $W(\pi)$ of a resolution refutation π of F. This definition, however, is ill-suited for those CNFs that themselves have large width, like the pigeonhole principle. We have found it way more natural and convenient to work with its slightly modified version used in [20] that we will denote by $W(\vdash_F \bot)$. It is defined as follows.

Instead of just allowing the clauses C of F as axioms, we allow them to participate in the form of the following more general *F-cut rule*:

$$\frac{D \vee \overline{\ell_1} \qquad \dots \qquad D \vee \overline{\ell_r}}{D}, \tag{8}$$

where $\ell_1 \vee \ldots \vee \ell_r$ is a clause of F. In case some $D \vee \overline{\ell_j}$ contains contradictory literals, it is removed from the premises. In particular, when $D = C$, the list of premises becomes empty so the clauses of F are still available as axioms.

It is easy to see that $W(\vdash_F \bot) \le W(F \vdash \bot) \le W(\vdash_F \bot) + W(F) - 1$, hence the difference between the standard definition and ours becomes immaterial when $W(F)$ is small, and it does not have any noticeable impact on the *size* of a refutation.

One immediate advantage of this definition is that if we replace $W(F \vdash \bot)$ with $W(\vdash_F \bot)$ in (1), (2), (4), (5) or (6), we need not keep the annoying terms $W(F)$ any more, they just disappear. Simulations on Figure 1 will work without any restrictions on the width of the refuted CNF. More advantages of a similar flavor will become clear later, see Theorems 3.4 and 4.2 in particular.

Let us also remark that resolution with the F-cut rule is nothing else but Gentzen's sequent calculus with only atomic cuts, restricted to proving sequents of the form $C_1, \ldots, C_m \to$, where $C_1, \ldots, C_m$ are clauses (see [32]).

DNF resolution, or depth 2 Frege, is the straightforward extension of resolution where we allow, apart from variables in the resolution rule, also formulas of depth 1[4] to be resolved. DNF resolution operates with DNF formulas. Its axioms and inference rules are:

$$\frac{}{x \vee \overline{x}}, \quad \frac{G}{G \vee H}, \quad \frac{G \vee t_1 \qquad H \vee t_2}{G \vee H \vee (t_1 \wedge t_2)}, \quad \frac{G \vee t \qquad H \vee \overline{t}}{G \vee H},$$

[4] For this reason, some authors use the term "depth 1 Frege" for DNF resolution; we prefer to stick to the convention under which depth refers to *lines* in a Hilbert-style proof.

where G and H are DNF formulas and t, t_1, t_2 and $t_1 \wedge t_2$ are terms. The leftmost rule is the weakening rule in this context, and the rightmost rule is called the *cut rule*. The remaining rule allows us to deal with $\wedge$ connectives, and is called $\wedge$*-introduction*.

For a non-decreasing function $f : \mathbb{N} \to \mathbb{N}$, $R(f)$ is the subsystem of DNF resolution where each DNF in a proof of size s is required to have width at most $f(s)$. $R(k)$ for k a constant is usually denoted by $\mathrm{Res}(k)$ (thus, resolution is $\mathrm{Res}(1)$). DNF resolution and $R(f)$ were first introduced in [25].

Next, we would like to consider systems for manipulating terms. The syntactic details of such systems will not matter for our results, but for concreteness, let us present a prominent system of algebraic flavor originally introduced in [12]. We will actually use an extension, proposed in [1], called *polynomial calculus with resolution* and abbreviated as *PCR*. PCR works with a fixed field $\mathbb{F}$. Clauses/terms are represented as *monomials*. The syntactic objects PCR operates with are *polynomials* in $\mathbb{F}[x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}]$, represented as linear combinations over $\mathbb{F}$ of monomials, and a proof line P is to be interpreted as asserting that $P = 0$. The axioms and inference rules of the system are:

$$\frac{}{\ell^2 - \ell}, \quad \frac{}{\ell + \overline{\ell} - 1}, \quad \frac{P \qquad Q}{\alpha P + \beta Q}, \quad \frac{P}{\ell P},$$

where $\ell \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$, $P, Q \in \mathbb{F}[x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}]$ and $\alpha, \beta \in \mathbb{F}$.

In each of the above systems, non-logical axioms are given as a set of clauses S, viewed as a CNF formula F (in PCR, a clause $C = \ell_1 \vee \cdots \vee \ell_k \in S$ is represented as the monomial $\overline{\ell_1} \ldots \overline{\ell_k}$). A proof of the unsatisfiability of F, or a *refutation* of F, is a derivation of a syntactic contradiction, denoted by $\perp$, from the clauses of F. In resolution and DNF resolution $\perp$ is the empty clause; in PCR, it is the polynomial 1.

We can view proofs as DAGs, by drawing edges from premises to conclusions in applications of the inference rules. If a proof DAG is a tree, that is every formula or polynomial in it is used as a premise at most once, then we say that the proof is *tree-like*. The *size* of a tree-like proof is the number of its leaves, and its *depth* is the length of its longest root-to-leaf path. We will also consider a one-sided version of depth, which we call *positive depth*. (The analogue of this notion in the context of computational complexity was recently defined in [29].) The positive depth of a tree-like resolution proof is the maximum number of *negative* literals introduced along a root-to-leaf path. We denote tree-like size, depth and positive depth by S_T, D and D_P respectively.

To define space complexity measures, we need to consider a different topology, namely view a proof as a sequence of *memory configurations* [16, 1]. A memory configuration will be a set of clauses in resolution, a set of DNF formulas in DNF resolution, or a set of polynomials in PCR. In a proof from a CNF F then, to go from a memory configuration to the next we may do one of the following:

Axiom Download: add a clause of the formula F, or a logical axiom of the system we are working with;

Erasure: delete a clause/DNF formula/polynomial, or

Inference: add the result of applying an inference rule to formulas in the current configuration.

A proof in configurational form is said to be *tree-like* if, whenever a formula is used as a premise in an inference rule, it is immediately erased from the memory.

The *clause space* of a configuration in resolution is the number of clauses it contains, its *variable space* the number of *distinct* variables it contains, and its *total space* the total number of literals, counting repetitions, it contains. For DNF resolution, we will be interested in what we call Σ_2 *space* of a configuration. The Σ_2 space of a configuration $\mathcal{M} = \{G_1, \ldots, G_s\}$ is defined as the sum of widths: $\Sigma_2\mathrm{Space}(\mathcal{M}) \stackrel{\mathrm{def}}{=} W(G_1) + \ldots + W(G_s)$. For PCR, we will consider the *monomial space*, which is the number of *distinct* monomials in a configuration.

For a space measure μ on configurations and a proof $\pi = \mathcal{M}_1, \ldots, \mathcal{M}_t$, we naturally let $\mu(\pi) \stackrel{\text{def}}{=} \max\{\mu(\mathcal{M}_i) \mid 1 \leq i \leq t\}$. As in [37], we will also consider *regularized* versions μ^* defined as $\mu^*(\pi) \stackrel{\text{def}}{=} \mu(\pi) \cdot \log|\pi|$, where $|\pi| \stackrel{\text{def}}{=} t$ is the *length* of π. All logarithms in this paper have base 2.

Finally, for a complexity measure μ on proofs, we write $\mu(F \vdash G)$ for the minimum value of $\mu(\pi)$, taken over all proofs of G from F; if such a proof does not exist, we set $\mu(F \vdash G)$ to be ∞. In most cases, the measure μ clearly suggests what the underlying proof system should be. For example, $W(F \vdash \bot)$ is the minimum width of a resolution refutation of F, and $\text{MSpace}^*(F \vdash \bot)$ is the minimum regularized monomial space of a PCR refutation (in configurational form) of F. $S_T(F \vdash \bot)$ shall mean the minimum size of a tree-like resolution refutation of F. We shall use the notation $S_{T,R(f)}(F \vdash \bot)$ to mean the minimum size of a tree-like $R(f)$-refutation of F. $\text{TCSpace}(F \vdash \bot)$ is the minimum clause space taken over all tree-like configurational refutations of F in resolution. Likewise, $\text{RCSpace}(F \vdash \bot)$ stands for the clause space in *regular* resolution, i.e. the subsystem of resolution where we require that a variable cannot be resolved more than once on any path in (the DAG resulting from the expansion of) the configurational proof π.

3 Simulations

3.1 Tree-like resolution size and regularized monomial space

First we show that $\log S_T$ in resolution, TCSpace, RCSpace, CSpace^* and MSpace^*, are all equivalent. Our main new contribution is the following simulation.

▶ **Theorem 3.1.** *For any unsatisfiable CNF formula F over n variables,*

$$\log S_T(F \vdash \bot) \leq 2\text{MSpace}^*(F \vdash \bot)\log(n+1),$$
$$\text{TCSpace}(F \vdash \bot) \leq 2\left(\text{MSpace}^*(F \vdash \bot) + 1\right).$$

Proof. The proof is analogous to the construction in [37] showing that depth is upper bounded by regularized variable space. Let $\mathcal{M}_1, \ldots, \mathcal{M}_t$ be a refutation of F in configurational form, of monomial space s. We show, by induction on d, that for every interval $[i..j] \subseteq [1..t]$ with $j > i$, $j - i \leq 2^d$, and for every clause D such that $\alpha_D \models \mathcal{M}_i$ and $\alpha_D \models \neg\mathcal{M}_j$, it holds that $S_T(F \vdash D) \leq (n+1)^{ds}$ and, moreover, the assumed tree-like resolution proof can be carried out in clause space at most $ds + 2$. The theorem follows by taking $[i..j] \coloneqq [1..t]$, $d \coloneqq \lceil \log t \rceil$ and $D \coloneqq \bot$.

Suppose that $d = 0$, so that $j = i + 1$. The statement is vacuously true except when the step consists in downloading an axiom C from F, simply because in all other cases we have $\mathcal{M}_i \models \mathcal{M}_{i+1}$ and hence D with the specified properties does not even exist. Let D be a clause for which $\alpha_D \models \mathcal{M}_i$ and $\alpha_D \models \neg(\mathcal{M}_i \cup \{C\})$. Then we necessarily must have $\alpha_D \models \neg C$, which is equivalent to saying that D is a weakening of C.

For the inductive step, suppose that $d > 0$, let $[i..j] \subseteq [1..t]$ be any interval with $j - i \leq 2^d$, $j > i + 1$, and let D be a clause such that $\alpha_D \models \mathcal{M}_i$ and $\alpha_D \models \neg\mathcal{M}_j$. Set $k \coloneqq i + \lceil (j-i)/2 \rceil$, so that $k - i \leq 2^{d-1}$ and $j - k \leq 2^{d-1}$. Let the list $m_1, \ldots, m_s$ contain all monomials occurring in $\mathcal{M}_k$. For a clause A and a monomial $m = \ell_1 \ldots \ell_r$, consider the following derivation of A:

$$\dfrac{\dfrac{A \vee \ell_1 \qquad \dfrac{\dfrac{A \vee \ell_r \qquad A \vee \overline{\ell_1} \vee \ldots \vee \overline{\ell_r}}{A \vee \overline{\ell_1} \vee \ldots \vee \overline{\ell_{r-1}}}}{\vdots}}{A \vee \overline{\ell_1} \vee \overline{\ell_2}} \qquad \dfrac{A \vee \ell_2}{A \vee \overline{\ell_1}}}{A}$$

Call this derivation tree $\mathbf{T}_{A;m}$. For the required tree-like resolution proof of D, start with $\mathbf{T}_{D;m_1}$. To every leaf of $\mathbf{T}_{D;m_1}$ labelled by a clause D', append the tree $\mathbf{T}_{D';m_2}$. Continue this process for all $m_1, \ldots, m_s$. If at any point during this construction, a forbidden disjunction containing a variable and its negation occurs, then we delete that node and contract at its parent. The resulting tree $\mathbf{T}$ has at most $(n+1)^s$ leaves, and each of its leaves is labelled by a clause E such that $\alpha_E \models \mathcal{M}_k$ or $\alpha_E \models \neg\mathcal{M}_k$. From the induction hypothesis, there are tree-like resolution proofs of all those clauses from F, of size $(n+1)^{(d-1)s}$. Therefore, there is a tree-like resolution proof of D from F of size $(n+1)^{ds}$.

To see that this proof can be carried out in clause space at most $ds+2$, notice that the proof designated by $\mathbf{T}$ can be carried out in clause space $s+2$. Proceed with this proof, and whenever a clause at its leaves is downloaded, keep all current clauses in memory (there are at most s of them – the maximum clause space is hit when the parent of two leaves is brought to memory), and derive it in clause space at most $(d-1)s+2$. The fact that such a derivation exists is guaranteed by the induction hypothesis. The resulting proof has clause space at most $s+(d-1)s+2 = ds+2$. ◀

For the rest of the relations, we claim that for an unstatisfiable CNF F in n variables,

$$\begin{aligned}&\mathrm{RCSpace}(F \vdash \bot) \le \mathrm{TCSpace}(F \vdash \bot) \le \log S_T(F \vdash \bot) + 2\\ &\le 2\mathrm{MSpace}^*(F \vdash \bot)\log(n+1) + 2 \le 2\mathrm{CSpace}^*(F \vdash \bot)\log(n+1) + 2\\ &\le 2\mathrm{RCSpace}^*(F \vdash \bot)\log(n+1) + 2 \le 2\left(\mathrm{RCSpace}(F \vdash \bot)\right)^2 \log(n+1)\log(2n) + 2.\end{aligned}$$

The first inequality follows from the observation that every tree-like refutation can be pruned to the regular form, and this operation does not increase its space. The second inequality is [16, Theorem 2.1], and the third is Theorem 3.1. The fourth and the fifth inequalities are obvious. Finally, the last inequality follows from [16, Corollary 4.2].

As a byproduct, we get that TCSpace $\approx$ RCSpace. This comes in sharp contrast with the situation for size, where there is an exponential separation between tree-like and regular resolution [5].

We also see from [16, Corollary 4.2] that, somewhat surprisingly, instead of regularizing clause space by multiplying it by the logarithm of *size*, we could have as well used a much weaker regularization multiplying by the logarithm (!) of depth, and the resulting measure would still be in this cluster. This allows us to re-cast the main open problem of whether CSpace $\approx$ CSpace* in terms of the existence of a *super-critical* (in the sense of [36]) trade-off between clause space and depth.

The remaining (non-trivial) simulation on Figure 1 involving this cluster is:

▶ **Theorem 3.2.** *For any unsatisfiable CNF formula F,* $\mathrm{TCSpace}(F \vdash \bot) \le D_P(F \vdash \bot) + 2$.

Proof. The argument is a refinement of the argument in [16] showing that tree-like clause space is bounded by depth. We show, by induction on $\mathbf{T}$, that if $\mathbf{T}$ is a tree-like resolution proof of a clause E from F of positive depth d, then there is a tree-like resolution proof, in configurational form, of E from F of clause space at most $d+2$.

If $\mathbf{T}$ has size at most 2, then $d \leq 1$, and $\mathrm{TCSpace}(F \vdash \bot) \leq 3$. Otherwise, let $\mathbf{T}_1$ and $\mathbf{T}_2$ be the subproofs of $\mathbf{T}$ proving the two clauses E_1 and E_2 respectively from which E is derived via an application of the resolution rule and possibly applications of the weakening rule. One of $\mathbf{T}_1$ and $\mathbf{T}_2$, say $\mathbf{T}_1$, must have positive depth at most $d-1$. From the induction hypothesis, there is a tree-like proof π_1 of E_1 of clause space at most $d+1$, and a tree-like proof π_2 of E_2 of clause space at most $d+2$. Deriving first E_2 using π_2, and then, keeping E_2 in memory, deriving E_1 using π_1, we get a proof of E of clause space at most $d+2$. ◀

3.2 Resolution width and Σ_2 space

The simulations for our second cluster will depend upon the following "locality" property of DNF resolution.

▶ **Lemma 3.3.** *Let α be a partial assignment. For each of the inference rules of DNF resolution, if both premises contain a term satisfied by α, then α satisfies some term in the conclusion.*

The main theorem of this section says that as long as we transition from depth 1 Frege to depth 2 Frege, then not only width continues to be smaller than space, but in fact it becomes (almost) equal to it. As a historical remark, an extension of the Atserias-Dalmau bound (4) for the case of $\mathrm{Res}(k)$ is sketched in [18], and, although it is not stated explicitly, it is also apparent in [15].

▶ **Theorem 3.4.** *For any unsatisfiable CNF formula F,*

$$\frac{1}{5}\Sigma_2\mathrm{Space}(F \vdash \bot) \leq W(\vdash_F \bot) \leq \Sigma_2\mathrm{Space}(F \vdash \bot).$$

Proof. Let $\mathcal{M}_1, \ldots, \mathcal{M}_t$ be a DNF resolution refutation of F, of Σ_2 space s. We will construct a sequence $\mathbf{T}_1, \ldots, \mathbf{T}_t$ of derivations in the system "resolution plus the F-cut rule (8)". The property we are going to maintain is that for every clause D labelling a leaf of $\mathbf{T}_i$, either D is a weakening of a clause C in F (call such a leaf an *axiom leaf*) or the following hold:

1. for every $G \in \mathcal{M}_i$, α_D satisfies some term of G;
2. $W(D) \leq \Sigma_2\mathrm{Space}(\mathcal{M}_i)$.

$\mathbf{T}_1$ has one vertex labelled by the empty clause. Now suppose we have constructed $\mathbf{T}_{i-1}$ such that 1 and 2 hold for all non-axiom leaves. For every such leaf v labelled by a clause D, do the following.

- *Axiom Download*: Suppose that $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$, where $C = \ell_1 \vee \cdots \vee \ell_r$ is either a clause of F (viewed as a 1-DNF) or a logical axiom $x \vee \overline{x}$. If C and D contain conflicting literals, then item 1 is automatically satisfied and we do nothing at this leaf. Next, $C \subseteq D$ would have implied that C is a clause of F which is impossible since we have assumed that the leaf is non-axiom. Thus, there exists at least one $j \in [r]$ such that $\ell_j \notin D$, and for any such j we add to v a child labelled by $D \vee \overline{\ell_j}$. This will be an application of the F-cut rule if C is a clause or of the resolution rule if C is $x \vee \overline{x}$.
- *Erasure*: Suppose that $\mathcal{M}_i \subseteq \mathcal{M}_{i-1}$. Add to v a single child labelled by a clause $E \subseteq D$ such that $W(E) \leq \Sigma_2\mathrm{Space}(\mathcal{M}_i)$ and for every $G \in \mathcal{M}_i$, α_E satisfies some term of G.
- The case of an *inference* is immediately taken care of by Lemma 3.3, D does not change.

Since $\bot \in \mathcal{M}_t$, $\mathbf{T}_t$ may not contain any non-axiom leaves and hence defines a refutation. Also, it is clear from the construction and property 2 above that any clause D appearing in it must satisfy $W(D) \leq \max_{1 \leq i \leq t} \Sigma_2\mathrm{Space}(\mathcal{M}_i) = s$. Hence $W(\vdash_F \bot) \leq s$.

For the converse inequality, suppose that $C_1, \ldots, C_t$ is a refutation in the system "resolution plus the F-cut rule", of width w. For every $i \in [t]$, set $G_i := \bigvee_{j=1}^{i} \overline{C_j}$. Each G_i is a w-DNF. For our small space refutation, we will first derive G_t and $G_{t-1} \vee C_t$, then cutting on C_t derive from these formulas G_{t-1}, then derive $G_{t-2} \vee C_{t-1}$, and continue this way until we get the empty clause. Notice that $G_{i-1} \vee C_i$ is either a tautology with an obvious derivation in DNF resolution, or C_i is a clause of F. In the latter case, we can immediately derive $G_{i-1} \vee C_i$. Otherwise, C_i will be the result of applying either the resolution rule or the weakening rule or F-cut rule to some clauses among $C_1, \ldots, C_{i-1}$. In either case, it can be checked that $G_{i-1} \vee C_i$ has a tree-like proof of Σ_2 space at most $3w$, and therefore the proof above can be carried in Σ_2 space at most $5w$. ◀

▶ Remark 3.5. The second part of this proof implies that *a posteriori*, DNF resolution will retain its power in terms of space even if we restrict the *formula space* (the maximum number of DNFs in a configuration) to a constant. This in turn immediately implies, also a posteriori, that we can balance our definition of Σ_2 space replacing in it $W(G_1) + \ldots + W(G_s)$ with $s \cdot \max(W(G_1), \ldots, W(G_s))$, and the resulting measure will still be equivalent to Σ_2 space. We are not aware of a direct proof of this simulation by-passing width.

We get from Theorem 3.4 that strong length-space trade-offs conjectured for variable, clause and monomial space, are ruled out for DNF resolution. In particular, we get:

▶ **Corollary 3.6.** *For any unsatisfiable CNF formula F with n variables,*

$$\Sigma_2\mathrm{Space}^*(F \vdash \bot) \leq O\left((\Sigma_2\mathrm{Space}(F \vdash \bot))^2 \log n\right).$$

Proof. Let $s := \Sigma_2\mathrm{Space}(F \vdash \bot)$. By the first part of Theorem 3.4, F has a width $O(s)$ resolution refutation with the additional F-cut rule. We apply to this refutation the construction from the second part of Theorem 3.4 in which we can clearly assume $t \leq n^{O(s)}$ (since all clauses in the sequence can be assumed to be different). By an easy inspection, the length of the resulting refutation will still be $n^{O(s)}$. Therefore,

$$\Sigma_2\mathrm{Space}^*(F \vdash \bot) \leq O(s^2 \log n).$$ ◀

▶ **Corollary 3.7.** *If F has a constant Σ_2 space refutation, then it has a refutation of constant Σ_2 space and polynomial length.*

Proof. The refutation constructed in the proof of Corollary 3.6 will in our case also have constant Σ_2 space. ◀

Let us finally deal with the remaining measure, tree-like proofs in $R(\log)$.

▶ **Theorem 3.8.** *Let F be an unsatisfiable CNF formula over n variables. Then*

$$\Sigma_2\mathrm{Space}(F \vdash \bot)^{1/2} \leq \log S_{T,R(\log)}(F \vdash \bot) \leq O(W(\vdash_F \bot) \log n).$$

Proof. For the upper bound, let π be a resolution refutation of F of width $w := W(\vdash_F \bot)$. Apply to it the construction in the second part of the proof of Theorem 3.4 once again. By inspection (cf. the proof of Corollary 3.6), this refutation is tree-like, has size $n^{O(w)}$ and every term occurring in it has width at most w. Padding it with dummy formulas if necessary, we can assume that it has size $\geq 2^w$ which makes it into a tree-like $R(\log)$ refutation of the required size.

For the lower bound, the argument is an adaptation of the argument in [16] showing (3). Namely, by pebbling, a tree-like proof $\mathbf{T}$ of size $s > 1$ can be turned into a proof in configurational form, where each configuration contains at most $\log s$ formulas occurring in $\mathbf{T}$. If $\mathbf{T}$ is a refutation in $R(\log)$, then all terms occuring in $\mathbf{T}$ have width at most $\log s$, so the resulting refutation has Σ_2 space $(\log s)^2$. ◀

▶ Remark 3.9. For the more conventional system Res($\log n$), the subsystem of DNF resolution where each DNF in a refutation of F is required to have width $O(\log n)$, n the number of variables of F, the second inequality in Theorem 3.8 is false (see Figure 2). This follows from an easy adaptation of the proof of [15, Corollary 14].

4 Size-space trade-offs and tree-like size lower bounds

4.1 A lower bound on regularized monomial space

One application of the results of the previous section is that they easily allow us to show trade-offs[5] between regularized clause or monomial space and size.

It is known [22, 21] that there are formulas F of size $\Theta(n)$ that have a resolution refutation of size $O(n)$ (and thus a $O(n)$ refutation in the stronger system PCR), but $\mathrm{MSpace}^*(F \vdash \bot) \geq n^{1/2}/(\log n)^{O(1)}$. Theorem 3.1, combined with the lower bounds of [5] and [17] on $\log S_T$ and TCSpace immediately gives the following improvement.

▶ **Theorem 4.1.** *For every $n \geq 0$, there is a formula F of size $\Theta(n)$ that has a resolution refutation of size $O(n)$, width $O(1)$, and such that* $\mathrm{MSpace}^*(F \vdash \bot) \geq \Omega\left(n/\log n\right)$.

Proof. [5] demonstrates the existence of an $O(1)$-CNF F that has resolution refutations of size $O(n)$, width $O(1)$, and such that $\log S_T(F \vdash \bot) \geq \Omega(n/\log n)$. In fact, [5] shows that $\Omega(n/\log n)$ is also the lower bound on the number of points the Delayer can score in the Prover-Delayer game of [35] played on F. Now, it is proved in [17] that this number of points is precisely equal to $\mathrm{TCSpace}(F \vdash \bot)$ and then the result immediately follows from the second inequality in Theorem 3.1. ◀

4.2 Trade-offs between positive depth and tree-like size for Horn formulas and tree-like size lower bounds

We would like next to focus on tree-like size lower bounds for resolution attained for formulas with small clause space. We will show that a tree-like resolution refutation of a Horn formula actually describes a pebbling strategy, the space and time of the strategy being the positive depth and size respectively of the proof. This gives a more transparent version of the result of [5] used in the proof of Theorem 4.1, which moreover has a natural generalization allowing us to prove some tree-like lower bounds for formulas of small clause space.

[5] We would like to stress that, following the (perhaps, unfortunate) convention established in the previous papers, we mean *potential* trade-offs. In other words, we prove lower bounds on the regularized space and we only know that *our method fails* to extend them to the ordinary monomial space. As we explained in Section 1 and will further elaborate in Section 5, proving *actual* trade-offs in this setting is a major and difficult open problem.

4.2.1 Horn formulas – basics

Horn formulas, that include pebbling formulas, have seen a plethora of applications in proof complexity over the past two decades, including separating resolution size from tree-like resolution size [5], separating width from variable space and clause space [4, 6, 7], separating depth from tree-like clause space [38], and giving trade-offs [4, 7, 22, 3], to name a few.

A CNF formula is called *Horn* if every clause in it has at most one non-negated variable. Equivalently, a Horn formula is a set of implications involving variables, with at most one variable at the right hand side of the implication. An implication of the form $x_1, \dots, x_k \to y$ is asserting that if all the x_i's are true, then y is true; $x_1, \dots, x_k \to$ is asserting that one of the x_i's is false, $\to y$ is asserting that y is true, and $\to$ is a contradiction.

The following result states that Horn formulas make up, in a certain sense, the easiest class of formulas for proof complexity. For its purposes, it is convenient to define a slightly modified version $\mathrm{CSpace}(\vdash_F \bot)$ of the clause space, in the same vein we defined $W(\vdash_F \bot)$ above. Namely, we replace the three standard rules with the following

Three-in-one rule: from a configuration $\mathcal{M}$, infer any configuration $\mathcal{M}^* \subseteq \mathcal{M} \cup F \cup \{C\}$, where C is obtained from clauses in $\mathcal{M}, F$ via the resolution or weakening rule.

▶ **Theorem 4.2.** *Let F be a CNF formula. The following are equivalent:*

1. *F contains an unsatisfiable CNF sub-formula resulting from a Horn formula by negating some of its variables;*
2. $\mathrm{CSpace}(F \vdash \bot) \le 3$*;*
3. $\mathrm{CSpace}(\vdash_F \bot) \le 1$*;*
4. $W(\vdash_F \bot) \le 1$.

Proof. For $1 \implies 2$, we can w.l.o.g. assume that F itself is an unsatisfiable Horn formula. We show, by induction on the number of variables n, that it can be refuted in clause space 3. The base case is trivial. Now, suppose that $n > 0$, and let y be a variable such that F contains the clause $\to y$. Such a clause must exist, for if every clause contained a negated variable, then we could satisfy F by setting every variable to false. Setting $y := 1$, we get an unsatisfiable Horn formula $F|_{y:=1}$ with $n-1$ variables. From the induction hypothesis, there is a clause space 3 refutation of $F|_{y=1}$. Weakening every clause in it by $\overline{y}$ gives us a space 3 proof of $\overline{y}$ from F. Now we only have to download y and infer $\bot$.

For $2 \implies 3$, let $\mathcal{M}_1, \dots, \mathcal{M}_t$ be a space 3 refutation of the formula F; we can assume w.l.o.g. that it does not contain weakening rules. Consider a path in the corresponding refutation tree of *maximum* possible length, say $C_i \in \mathcal{M}_{t_i}$ $(0 \le i \le h)$ are such that $t_0 < \dots < t_h = t$, C_0 is an axiom, $C_h = \bot$ and for $i \ge 1$, C_i is obtained by resolving C_{i-1} with some $D_{i-1} \in \mathcal{M}_{t_i-1}$. It remains to show that D_{i-1} is actually an axiom for any $i \ge 1$. For $i = 1$ this follows from the maximality of the chosen path. For $i \ge 2$, we have $\mathcal{M}_{t_{i-1}} = \{C_{i-2}, D_{i-2}, C_{i-1}\}$. Therefore C_{i-1} is consistent (and hence not resolvable) with the two other clauses in $\mathcal{M}_{t_{i-1}}$. All clauses that may have been inferred in $\mathcal{M}_{t_{i-1}+1}, \dots, \mathcal{M}_{t_i}$ must have C_{i-1} as one of their premises and, as a consequence, are also not resolvable with C_{i-1}. Hence the only clauses in those configurations that may be resolvable with C_{i-1} (in particular, D_{i-2}) are the axioms.

The implication $3 \implies 4$ is proven by an argument similar to the first part of the proof of Theorem 3.4. Namely, a space 1 refutation of minimum length in the three-in-one model must necessarily be a sequence $\{C_1\}, \dots, \{C_t\}$, where C_{i+1} is obtained by resolving C_i with a clause in F. The non-axiom leaves of the tree $\mathbf{T}_i$ will simply be all those literals among $\overline{\ell_{i,1}}, \dots, \overline{\ell_{i,r_i}}$, where $C_i = \ell_{i,1} \vee \dots \vee \ell_{i,r_i}$, that are not axioms of F. It can routinely be

checked that, as in the proof of Theorem 3.4, $\mathbf{T}_i$ will be a resolution derivation using only the F-cut rule (notice that to keep the width 1, the weakening rule has to be incorporated into the F-cut rule).

Finally, for $4 \implies 1$, we again proceed by induction on the number of variables n of F. The base case is trivial. Suppose that $n > 0$. The fact that there is a width 1 refutation of F, forces F to have a one literal clause (since the refutation must start somewhere), say ℓ. Setting $\ell := 1$, we get a width 1 refutation of $F|_{\ell:=1}$. From the induction hypothesis, a sub-formula G of $F|_{\ell:=1}$ is unsatisfiable Horn up to negating some variables. Let $\widehat{G}$ be the corresponding sub-formula of F; $\widehat{G}$ is obtained from G by restoring $\overline{\ell}$ to some of its clauses. Then $\widehat{G} \wedge \ell$ is an unsatisfiable Horn sub-formula of F. ◀

4.2.2 Tree-like resolution proofs as pebbling strategies

The paper [4] shows that a configurational resolution refutation π of the so-called *pebbling contradiction* Peb_G on a graph G defines a pebbling strategy on G, of time at most $|\pi|$ and space equal to the variable space $\mathrm{VSpace}(\pi)$. These are strategies in the so-called black-white game of [14]. We shall show that a tree-like resolution proof $\mathbf{T}$ of any Horn formula H defines a pebbling strategy of time equal to the size of $\mathbf{T}$ and space essentially equal to the *positive* depth of $\mathbf{T}$. These are strategies in the more basic black-only pebbling game that in the case $H = \mathrm{Peb}_G$ corresponds to the black-only pebbling game on G. Urquhart [38] showed how to relate them to ordinary depth. In a sense, our Proposition 4.3 below can be viewed as a (far-reaching) refinement of his result.

The rules of the black-only pebbling game, played on a Horn formula H, are as follows. There is a limited amount of pebbles. Pebbles are placed on the variables of H according to the rules:

1. A pebble can be placed on a variable y if $x_1, \ldots, x_k \to y$ is a clause of H, and all $x_1, \ldots, x_k$ have pebbles on them. In particular, a pebble can be always placed on any variable y such that $\to y$ is a clause of H.
2. A pebble can be removed from a variable at any time.

A *configuration* of the pebbling game is a set of the variables of H. A *pebbling strategy* is a sequence of configurations, each resulting from the previous one by one of the rules above. We say that a pebbling strategy *refutes* H if it ends with a configuration where for some clause $x_1, \ldots, x_k \to$ of H, all variables $x_1, \ldots, x_k$ are pebbled. Note that if H is unsatisfiable, then such a clause must exist.

▶ **Proposition 4.3.** *Let H be an unsatisfiable Horn formula. A tree-like resolution refutation $\mathbf{T}$ of H of size s and positive depth d can be converted into a pebbling strategy that, starting with the empty configuration, refutes H in at most s steps and using at most $d + 1$ pebbles.*

Proof. We begin with a slight modification of our refutation. Namely, viewing $\mathbf{T}$ as a decision tree, its nodes naturally correspond to partial assignments, and for the clause C sitting at the node α, we have $\alpha \models \neg C$. Let us replace C with the *maximal* clause satisfying this property. This will give us a refutation, of the same size and positive depth, in which the resolution rule (7) is reduced to

$$\frac{C \vee x \qquad C \vee \overline{x}}{C} \tag{9}$$

and leaves are labelled by weakenings of axioms in H.

This refutation need not necessarily consist of Horn formulas even if the original one did so. Nonetheless we will still represent clauses in the sequential form $S \to T$, where S, T are disjoint sets of variables, like at the beginning of Section 4.2.1. Note that $|S| \leq d$ for any clause $S \to T$ appearing in $\mathbf{T}$.

We shall now show by induction that every subtree of $\mathbf{T}$ deriving a clause $S \to T$, leads to a pebbling strategy that, starting with pebbles on all variables of S and using at most $d+1$ pebbles, either refutes H, or ends with a configuration which has pebbles on all variables of S and on one variable of T. Thus, if T is empty then the former must occur and, in particular, the strategy corresponding to the empty sequent $\to$ will start with no pebbles on the variables of H and will refute H.

Suppose that $S \to T$ is at a leaf. If there are variables $x_1, \ldots, x_k$ in S such that $x_1, \ldots, x_k \to$ is a clause of H, then that leaf describes a strategy that, starting with pebbles on all variables in S, immediately refutes H. Otherwise, there must be variables $x_1, \ldots, x_k$ in S and a variable y in T such that $x_1, \ldots, x_k \to y$ is a clause of H. Then the strategy of that leaf is to put a pebble on y. Since $|S| \leq d$, the number of pebbles used is at most $d+1$, as required.

If $S \to T$ is not at a leaf, then consider its left and right subtrees $\mathbf{T}_1$ and $\mathbf{T}_2$ proving $S, x \to T$ and $S \to T, x$ respectively (cf. (9)). The strategy corresponding to $S \to T$ is defined as follows. First follow $\mathbf{T}_2$'s strategy. If that strategy either refutes H or places a pebble on one of T's variables, then we are done. Otherwise, when the strategy of $\mathbf{T}_2$ is concluded, there are pebbles on S and x. Remove all other pebbles and follow the strategy of $\mathbf{T}_1$. The bound $d+1$ on the number of pebbles used at any moment follows from the same bound for $\mathbf{T}_1$ and $\mathbf{T}_2$.

Clearly, the number of steps of the pebbling strategy corresponding to $\to$ is at most the size of $\mathbf{T}$, and the required bound on the number of pebbles was already noticed. ◀

▶ Remark 4.4. The proof of Proposition 4.3 relies on an intuitionistic interpretation of the resolution rule. In the intuitionistic tradition, the denotational view of assigning truth values is, philosophically, nonsense. A proposition is "true" if it is provable, and a proof of e.g. a formula $S \to T$ is a construction that given proofs of all the elements of S produces a proof of some element in T. What Proposition 4.3 says is that a tree-like resolution derivation of $S \to T$ precisely describes such a construction, assuming that proofs of all the clauses of H are known. Moreover this construction will be economical in the number of steps and memory if the size and the positive depth respectively of the proof are small. Let us further notice, that although Proposition 4.3 is stated for Horn formulas, it really is general; it could be stated, with minimal modifications, for arbitrary CNFs.

4.2.3 Tree-like size lower bounds

The following theorem turns pebbling time-space trade-offs for a Horn formula H into tree-like size lower bounds for its substituted version $H[\vee_2]$. We formulate it in a somewhat general form, to account for various kinds of pebbling trade-offs in the literature. The substituted version $F[\vee_2]$ of a CNF $F(x_1, \ldots, x_n)$ is defined by replacing x_i with $y_i \vee z_i$ for mutually distinct variables $\{y_1, z_1, \ldots, y_n, z_n\}$, followed by converting the result back to the CNF form in the straightforward way.

▶ **Theorem 4.5.** *Let H be an unsatisfiable Horn formula on n variables. Suppose that every pebbling strategy that refutes H in s steps and using d pebbles, starting with no pebbles on its variables, satisfies $(d-1) \cdot f(s) \geq g(n)$ for non-decreasing positive functions f, g. Then $f(t) \log t \geq g(n)$, where $t \stackrel{\text{def}}{=} S_T(H[\vee_2] \vdash 0)$.*

Proof. Create a probability space of partial assignments by choosing independently for every variable x of H, which was substituted by $y \vee z$, one of y and z with probability $1/2$ and setting it to zero. Note that for any α from this space, $H[\vee_2]|_\alpha$ is identical to H up to

renaming its variables and hence $\mathbf{T}|_\alpha$ is a refutation of H, again up to renaming variables. Let $\mathbf{T}$ be an arbitrary tree-like resolution refutation of $H[\vee_2]$ of size t represented as in the proof of Proposition 4.3. That is weakenings are omitted from the resolution rule, and appear at the leaves only. Let $D_1, \ldots, D_k$ be all clauses of positive depth $g(n)/f(t)$ occurring in $\mathbf{T}$. We have that

$$P\left[\bigvee_{i=1}^{k}(D_i|_\alpha \neq 1)\right] \leq \sum_{i=1}^{k} P[D_i|_\alpha \neq 1] \leq k2^{-g(n)/f(t)} \leq t2^{-g(n)/f(t)}.$$

If $f(t)\log t < g(n)$, then the above probability is smaller than 1, which means that there is a point α in our sample space such that $\mathbf{T}|_\alpha$ is a tree-like resolution refutation of size at most t and positive depth $\leq g(n)/f(t)$. This, from Proposition 4.3, gives a pebbling strategy that refutes H in t steps using d pebbles, where $(d-1)\cdot f(t) < g(n)$. ◀

Recall that for a DAG G, the pebbling contradiction Peb_G is defined as the Horn formula consisting of all clauses $S \to x$, where $x \in V(G)$ and S is the set of all its immediate predecessors, as well as the clauses $x \to$ for any sink x. Plugging into Theorem 4.5 various DAGs from the literature with known bounds on their pebbling complexity and various functions f, we can get several corollaries. The first is a simplified proof of the separation by Ben-Sasson et al.

▶ **Corollary 4.6** [5]. *There are formulas of size $O(n)$ having DAG-like resolution refutations of size $O(n)$, every tree-like resolution refutation of which requires size* $\exp(\Omega(n/\log n))$.

Proof. This is by setting $f := 1$ in Theorem 4.5, and using the graphs of [34] having constant in-degree and requiring $\Omega(n/\log n)$ pebbles to pebble. ◀

The next result was promised in the introduction. It should be compared with Theorem 4.2.

▶ **Theorem 4.7.** *There are formulas of size $O(n)$ having tree-like resolution refutations of clause space 4, every tree-like resolution refutation of which has size $\Omega(n^2/\log n)$.*

Proof sketch. This is by setting $f(t) := t$ in Theorem 4.5, and using the graphs of [28, Theorem 2.3.2] having linear size and exhibiting a $dt \geq \Omega(n^2)$ trade-off. These graphs can be pebbled using 3 pebbles, and that immediately gives that $\mathrm{CSpace}(\mathrm{Peb}_{G_n}[\vee_2] \vdash \bot) \leq O(1)$. By being more careful, it is possible to bring the space down to the minimum possible value, namely 4, for which a super-linear lower bound on tree-like resolution size is possible. ◀

By using the construction from [28, Theorem 4.2.6], Theorem 4.7 can be further generalized to a lower bound $(n/\log n)^{\Omega(k)}$ on the tree-like resolution size of refuting formulas with clause space k. Let us further notice that the fact that the space 4 refutation in Theorem 4.7 is tree-like might be interesting, as typically tree-like resolution size lower bounds have been proven in the literature based on the prover-delayer game of [35], which also gives a lower bound for the clause space of tree-like resolution refutations (cf. Theorem 4.1).

5 Concluding remarks

We showed that $\log S_T$, CSpace^* and MSpace^* are equivalent up to polynomial and $\log n$ factors, demonstrating a picture perfectly analogous to the picture involving D, VSpace^* and TSpace^* in [37]. The most important question remains (widely) open:

▶ Problem 5.1. Is it true that $\mathrm{CSpace} \approx \log S_T$ or $\mathrm{MSpace} \approx \log S_T$? Recall for comparison that $\log S_T \approx \mathrm{CSpace}^* \approx \mathrm{MSpace}^*$.

Equivalently, do there exist strong trade-offs between clause (or monomial) space and length? It should be contrasted with trade-offs results in e.g. [7, 3], and it is a perfect analogue of Urquhart's question [38] about variable space vs. depth studied in [37, Section 6]. Let us make a few more remarks about this problem.

Firstly, for very small space essentially this question was already asked in the literature before. Namely (see e.g. [30, Open Problem 16]), are there formulas having constant clause space refutations and with the property that any such refutation must necessarily have super-polynomial length? Suitably adjusting it to our framework:

▶ Problem 5.2 (small space variant). Are there formulas that have $(\log n)^{O(1)}$ clause or monomial space refutations and with the property that any such refutation must be of super-quasi-polynomial length $\exp((\log n)^{\omega(1)})$? Equivalently, any tree-like resolution refutation must have super-quasi-polynomial length.

In terms of the perceived difficulty, we do not discern too much of a difference between Problems 5.1, 5.2 and Nordström's question. In fact, we would like to cautiously conjecture that there are formulas F with $\mathrm{CSpace}(F \vdash \bot) \leq 5$ and $\mathrm{CSpace}^*(F \vdash \bot) \geq \exp\left(n^{\Omega(1)}\right)$. But the only result we were able to prove in that direction is the rather weak Theorem 4.7.

Secondly, as suggested by Figure 1, any strong separation between monomial space and clause space would immediately solve Problem 5.1 for monomial space. As we consider the latter to be most likely very difficult, we take it as a good heuristic explanation of why we have not seen any progress on the former problem as well. But let us ask this, and one obviously relevant question, explicitly anyway:

▶ Problem 5.3. Is it true that $\mathrm{CSpace} \approx \mathrm{MSpace}$? Is it true that $\mathrm{MSpace} \approx W$?

We note that by the result from [31, 6], at least one of these must be false. A quadratic separation between width and monomial space has been recently proved by the first author (manuscript in preparation). For a discussion on related topics, see also [11, Section 7.5.5].

Finally, while all these conjectured trade-offs are very strong, they are still not super-critical in the sense of [36] (the required lower bound on length never exceeds 2^n). However, as we pointed out in Section 3.1 in all these questions refutation length can be replaced with depth. Since the depth, as a stand-alone measure, is always bounded by n, these actually *are* questions about the existence of a *super-critical* trade-off between clause space and *depth*.

We have (somewhat surprisingly) proved that DNF resolution behaves very differently from resolution with respect to space. Intermediate systems based on $\mathrm{Res}(k)$ for a constant k were studied in a similar context before, and it is very natural to wonder what is the situation for those systems.

Let us first remark that for $\mathrm{Res}(k)$-refutations, the definition of space from [15, 7] (formula space) coincides with ours up to a factor of k so we need not distinguish between the two. Then Theorem 3.1 readily generalizes to this regime and gives $\log S_{T,\mathrm{Res}(k)} \approx \mathrm{Res}(k)\mathrm{Space}^*$, extending the bottom half of Figure 1 as shown in Figure 2. The proof of Corollary 3.6, however, fails for a constant k as badly as it fails for $k = 1$. Hence we have one more question to ask:

▶ Problem 5.4 (Res(k)-variant). Is there a constant $k > 0$ such that $\log S_{T,\mathrm{Res}(k)} \approx \mathrm{Res}(k)\mathrm{Space}$ or at least $\log S_{T,\mathrm{Res}(k)} \preceq \mathrm{CSpace}$?

Let us also mention that as k increases, both hierarchies, $\log S_{T,\mathrm{Res}(k)}$ (and, hence, also $\mathrm{Res}(k)\mathrm{Space}^*$) and $\mathrm{Res}(k)\mathrm{Space}$ are proper ([15] and [7] respectively). This excludes the dual version of Remark 3.5: while the formula space of DNF resolution refutations can be reduced to constant, this is not true for the widths of individual formulas in the memory.

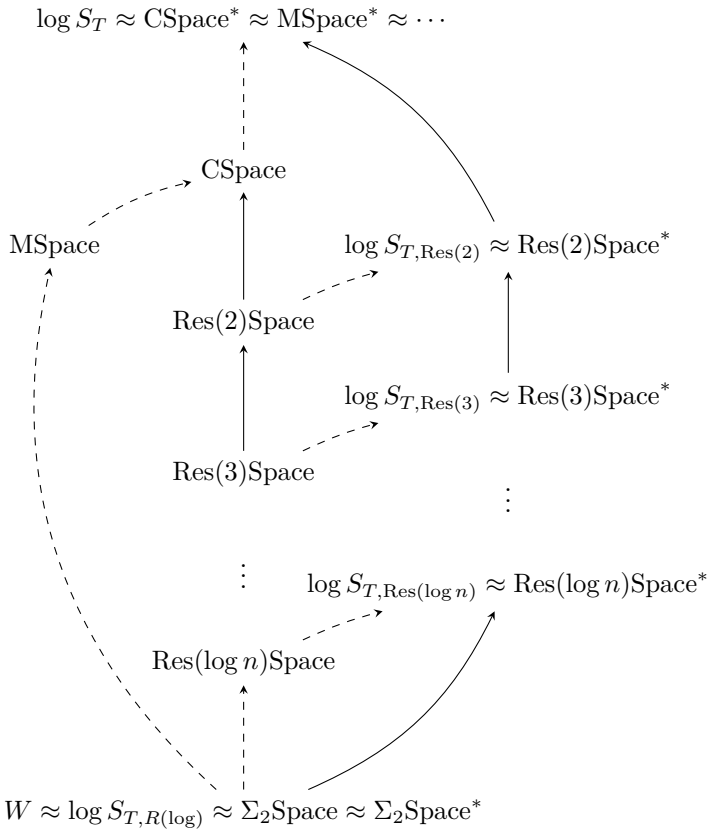


Figure 2 Σ_2 space and tree-like size for subsystems of DNF resolution.

The relation between VSpace and CSpace is also unknown in one direction (the opposite one is taken care of by [4]). Let us re-iterate the problem posed e.g. in [37]:

▶ Problem 5.5. Is it true that CSpace $\preceq$ VSpace?

Just as with the questions of similar nature discussed above, a negative answer would also imply a separation between VSpace and VSpace*, hence we can expect it to be very difficult.

References

1. Michael Alekhnovich, Eli Ben-Sasson, Alexander Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal of Computing*, 31:1184–1211, 2002.
2. Albert Atserias and Victor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74:323–334, 2008.
3. Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus: extended abstract. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 813–822, 2013.
4. Eli Ben-Sasson. Size-space tradeoffs for resolution. *SIAM Journal of Computing*, 38, 2009.
5. Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24:585–603, 2004.
6. Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 709–718, 2008.

7 Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proceedings of the 2nd Symposium on Innovations in Computer Science*, pages 401–416, 2011.

8 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48:149–169, 2001.

9 Ilario Bonacina. Total space in resolution is at least width squared. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming*, volume 55, pages 56:1–56:13, 2016.

10 Maria Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Computational Complexity*, 10:261–276, 2002.

11 Samuel Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 7, pages 233–350. IOS Press, 2nd edition, 2021.

12 Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 174–183, 1996.

13 Stephen Cook and Robert Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

14 Stephen Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *Journal of Computer and System Sciences*, 13:25–37, 1976.

15 Juan Luis Esteban, Nicola Galesi, and Jochen Messner. On the complexity of resolution with bounded conjunctions. *Theoretical Computer Science*, 321:347–370, 2004.

16 Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171:84–97, 2001.

17 Juan Luis Esteban and Jacobo Torán. A combinatorial characterization of treelike resolution space. *Information Processing Letters*, 87:295–300, 2003.

18 Yuval Filmus, Massimo Lauria, Mladen Miksa, Jakob Nordström, and Marc Vinyals. From small space to small width in resolution. *ACM Transactions of Computational Logic*, 16:28:1–28:15, 2015.

19 Nicola Galesi, Leszek Kołodziejczyk, and Neil Thapen. Polynomial calculus space and resolution width. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science*, pages 1325–1337, 2019.

20 Nicola Galesi and Neil Thapen. Resolution and pebbling games. In *Proceedings of the 8th Theory and Applications of Satisfiability Testing Conference*, volume 3569, pages 76–90, 2005.

21 Mika Göös and Toniann Pitassi. Communication lower bounds via critical block sensitivity. *SIAM Journal of Computing*, 47:1778–1806, 2018.

22 Trinh Huynh and Jakob Nordström. On the virtue of succinct proofs: amplifying communication complexity hardness to time-space trade-offs in proof complexity. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing Conference*, pages 233–248, 2012.

23 Russell Impagliazzo, Pavel Pudlák, and Jirí Sgall. Lower bounds for the polynomial calculus and the gröbner basis algorithm. *Computational Complexity*, 8:127–144, 1999.

24 Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *Journal of Symbolic Logic*, 59:73–86, 1994.

25 Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170:123–140, 2001.

26 Jan Krajíček. *Proof Complexity*. Cambridge University Press, 2019.

27 Massimo Lauria. A note about k-DNF resolution. *Information Processing Letters*, 137:33–39, 2018.

28 Thomas Lengauer and Robert Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *Journal of the ACM*, 29:1087–1130, 1982.

29 Bruno Loff and Sagnik Mukhopadhyay. Lifting theorems for equality. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science*, volume 126, pages 50:1–50:19, 2019.

30 Jakob Nordström. Pebble games, proof complexity, and time-space trade-offs. *Logical Methods in Computer Science*, 9, 2013.

31 Jakob Nordström and Johan Håstad. Towards an optimal separation of space and length in resolution. *Theory of Computing*, 9:471–557, 2013.

32 Theodoros Papamakarios. Space characterizations of complexity measures and size-space trade-offs in propositional proof systems. *Electronic Colloqium on Computational Complexity, Report No. 176*, 2021.

33 Theodoros Papamakarios and Alexander Razborov. Space characterizations of complexity measures and size-space trade-offs in propositional proof systems. *Electronic Colloqium on Computational Complexity, Report No. 74*, 2021.

34 Wolfgang Paul, Robert Tarjan, and James Celoni. Space bounds for a game on graphs. *Mathematical Systems Theory*, 10:239–251, 1977.

35 Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 128–136, 2000.

36 Alexander Razborov. A new kind of tradeoffs in propositional proof complexity. *Journal of the ACM*, 63:16:1–16:14, 2016.

37 Alexander Razborov. On space and depth in resolution. *Computational Complexity*, 27:511–559, 2018.

38 Alasdair Urquhart. The depth of resolution proofs. *Studia Logica*, 99:349–364, 2011.

Learning Algorithms Versus Automatability of Frege Systems

Ján Pich
University of Oxford, UK

Rahul Santhanam
University of Oxford, UK

Abstract

We connect learning algorithms and algorithms automating proof search in propositional proof systems: for every sufficiently strong, well-behaved propositional proof system P, we prove that the following statements are equivalent,

- **Provable learning.** P proves efficiently that p-size circuits are learnable by subexponential-size circuits over the uniform distribution with membership queries.
- **Provable automatability.** P proves efficiently that P is automatable by non-uniform circuits on propositional formulas expressing p-size circuit lower bounds.

Here, P is sufficiently strong and well-behaved if I.-III. holds: I. P p-simulates Jeřábek's system WF (which strengthens the Extended Frege system EF by a surjective weak pigeonhole principle); II. P satisfies some basic properties of standard proof systems which p-simulate WF; III. P proves efficiently for some Boolean function h that h is hard on average for circuits of subexponential size. For example, if III. holds for $P = \mathsf{WF}$, then Items 1 and 2 are equivalent for $P = \mathsf{WF}$. The notion of automatability in Item 2 is slightly modified so that the automating algorithm outputs a proof of a given formula (expressing a p-size circuit lower bound) in p-time in the length of the shortest proof of a closely related but different formula (expressing an average-case subexponential-size circuit lower bound).

If there is a function $h \in \mathsf{NE} \cap \mathsf{coNE}$ which is hard on average for circuits of size $2^{n/4}$, for each sufficiently big n, then there is an explicit propositional proof system P satisfying properties I.-III., i.e. the equivalence of Items 1 and 2 holds for P.

2012 ACM Subject Classification Theory of computation

Keywords and phrases learning algorithms, automatability, proof complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.101

Category Track A: Algorithms, Complexity and Games

Related Version *Previous Version*: https://arxiv.org/abs/2111.10626

Funding *Ján Pich*: This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodovska-Curie grant agreement No 890220. Ján Pich received support from the Royal Society University Research Fellowship URF\R1\211106. *Rahul Santhanam*: Rahul Santhanam is partially funded by the EPSRC New Horizons grant EP/V048201/1: "Structure versus Randomness in Algorithms and Computation".

Acknowledgements We would like to thank Moritz Müller, Jan Krajíček, Iddo Tzameret and anonymous reviewers for comments on a draft of the paper.

1 Introduction

Learning algorithms and automatability algorithms searching for proofs in propositional proof systems are central concepts in complexity theory, but a priori they appear rather unrelated.

Learning algorithms. In the PAC model of learning introduced by Valiant [34], a circuit class $\mathcal{C}$ is learnable by a randomized algorithm L over the uniform distribution, up to error ϵ, with confidence δ and membership queries, if for every Boolean function f computable by a circuit from $\mathcal{C}$, when given oracle access to f, L outputs with probability $\geq \delta$ over the uniform distribution a circuit computing f on $\geq (1-\epsilon)$ inputs. An important task of learning theory is to find out if standard circuit classes such as $\mathsf{P/poly}$ are learnable by efficient circuits. A way to approach the question is to connect the existence of efficient learning algorithms to other standard conjectures in complexity theory. For example, we can try to prove that efficient learning of $\mathsf{P/poly}$ is equivalent to $\mathsf{P} = \mathsf{NP}$ or to the non-existence of strong pseudorandom generators. In both cases one implication is known: $\mathsf{P} = \mathsf{NP}$ implies efficient learning of $\mathsf{P/poly}$ (with small error and high confidence) which in turn breaks pseudorandom generators. However, while some progress on the opposite implications has been made, they remain open, cf. [2, 33].

Automatability. The notion of automatability was introduced in the work of Bonet, Pitassi and Raz [6]. A propositional proof system P is automatable if there is an algorithm A such that for every tautology ϕ, A finds a P-proof of ϕ in p-time in the size of the shortest P-proof of ϕ. That is, even if P does not prove all tautologies efficiently, it can still be automatable. Establishing (non-)automatability results for concrete proof systems is one of the main tasks of proof complexity. This led to many attempts to link the notion of automatability to other standard complexity-theoretic conjectures. For example, recently Atserias and Müller [3] proved that automating Resolution is NP-hard and their work has been extended to other weak proof systems, e.g. [12, 13, 14]. For stronger systems, it is known that automating Extended Frege system EF, Frege or even constant-depth Frege would break specific cryptographic assumptions such as the security of RSA or Diffie-Hellman scheme, cf. [23, 6, 5]. It remains, however, open to obtain non-automatability of strong systems like Frege under a generic assumption such as the existence of strong pseudorandom generators, let alone to prove the equivalence between such notions.

In the present paper we derive a conditional equivalence between learning algorithms for p-size circuits and automatability of proof systems on tautologies encoding circuit lower bounds.

1.1 Our result

An ideal connection between learning and automatability would say that for standard proof systems P,

"P is automatable if and only if $\mathsf{P/poly}$ is learnable efficiently".

We establish this modulo some provability conditions and a change of parameters. Additionally, we need to consider automatability only w.r.t. formulas encoding circuit lower bounds. More precisely, denote by $\mathsf{tt}(f,s)$ a propositional formula which expresses that boolean function f represented by its truth-table is not computable by a boolean circuit of size s represented by free variables, see Section 3. So $\mathsf{tt}(f,s)$ is a tautology if and only if f is hard for circuits of size s. Note that f is represented by 2^n bits, if n is the number of inputs of f, so the size of $\mathsf{tt}(f,s)$ is $2^{O(n)}$. Similarly, let $\mathsf{tt}(f,s,t)$ be a formula expressing that circuits of size s fail to compute f on $\geq t$-fraction of inputs. In our main result (Theorem 1)

we use a slightly modified notion of automatability where the automating algorithm for a proof system P is non-uniform and outputs a P-proof of a given formula $\mathsf{tt}(f, n^{O(1)})$ in p-time in the size of the shortest P-proof of $\mathsf{tt}(f, 2^{n^{o(1)}}, 1/2 - 1/2^{n^{o(1)}})$, see Section 3.[1]

▶ **Theorem 1** (Informal, cf. Theorem 18)**.** *Let P be any propositional proof system which APC_1-provably p-simulates WF and satisfies some basic properties, e.g. $P = \mathsf{WF}$. Moreover, assume that P proves efficiently $\mathsf{tt}(h, 2^{n/4}, 1/2 - 1/2^{n/4})$ for some boolean function h. Then, the following statements are equivalent:*

1. **Provable learning.** *P proves efficiently that p-size circuits are learnable by $2^{n^{o(1)}}$-size circuits, over the uniform distribution, up to error $1/2 - 1/2^{n^{o(1)}}$, with membership queries and confidence $1/2^{n^{o(1)}}$.*
2. **Provable automatability.** *P proves efficiently that P is automatable by non-uniform circuits on formulas $\mathsf{tt}(f, n^{O(1)})$.*

WF is an elegant strengthening of EF introduced by Jeřábek [15], which corresponds to the theory of approximate counting APC_1, a theory formalizing probabilistic p-time reasoning, see Section 2.2. Concrete proof systems which APC_1-provably p-simulate WF and satisfy the basic properties from Theorem 1 include WF itself or even much stronger systems such as set theory ZFC (if we interpret ZFC as a suitable system for proving tautologies, see Section 5). We emphasize that the conditional equivalence from Theorem 1 holds for any sufficiently strong proof system satisfying some basic properties. The error and confidence of learning algorithms can be amplified "for free", see Section 2.1, but we did not make the attempts to prove that the amplification is efficiently provable already in WF.

Perhaps the most unusual aspect of Theorem 1 is its usage of metamathematics: we do not prove the equivalence between automatability and learning but between *provable* automatability and *provable* learning. We believe that the usage of metamathematics is not a substantial deviation. It would be very surprising if, e.g., an efficient learning algorithm existed but not provably (in some natural proof system).

Plausibility of the assumption. The main assumption in Theorem 1 is the provability of a circuit lower bound $\mathsf{tt}(h, 2^{n/4}, 1/2 - 1/2^{n/4})$. This assumption has an interesting status. Razborov's conjecture about hardness of Nisan-Wigderson generators implies a conditional hardness of formulas $\mathsf{tt}(h, n^{O(1)})$ for Frege (for every h), cf. [31], and it is possible to consider extensions of the conjecture to all standard proof systems, even set theory ZFC. On the other hand, all major circuit lower bounds for weak circuit classes and explicit boolean functions are known to be effciently provable in EF [2], cf. [29, 25]. If we believe that explicit circuit lower bounds such as $\mathsf{tt}(h, 2^{n/4}, 1/2 - 1/2^{n/4})$, for some $h \in \mathsf{EXP}$, are true, it is also perfectly plausible that they are efficiently provable in a standard proof system such as ZFC [3] or EF. Notably, if EF proves efficiently $\mathsf{tt}(h, 2^{n/4})$ for some boolean function h, then EF simulates WF, cf. [22, Lemma 19.5.4]. If there is a p-time algorithm which given a string of length 2^n (specifying the size of $\mathsf{tt}(h, 2^{n/4})$) generates an EF-proof of $\mathsf{tt}(h, 2^{n/4})$, then EF is p-equivalent to WF. To see that, combine Lemma 12 with the fact (proved in [15]) that APC_1 proves the reflection principle for WF.

---

[1] We believe that the gap between $\mathsf{tt}(f, n^{O(1)})$ and $\mathsf{tt}(f, 2^{n^{o(1)}}, 1/2 - 1/2^{n^{o(1)}})$ can be almost closed, if one uses learning of subexponential-size circuits instead of p-size circuits in Item 1 of Theorem 1 and tt-formulas expressing subexponential-size circuit lower bounds in Item 2.

[2] This has not been verified for lower bounds obtained via the algorithmic method of Williams [35].

[3] Efficient provability of $\mathsf{tt}(h, 2^{n/4}, 1/2 - 1/2^{n/4})$ in ZFC, for some $h \in \mathsf{EXP}$, would follow from the standard provability of this lower bound in ZFC.

As a corollary of Theorem 1 we show that, under a standard hardness assumption, there is an explicit proof system P for which the equivalence holds. This, follows, essentially, by "hard-wiring" tautologies $\mathsf{tt}(h, 2^{n/4}, 1/2 - 1/2^{n/4})$ to WF.

▶ **Corollary 2** (cf. Corollary 22). *Assume there is a* $\mathsf{NE} \cap \mathsf{coNE}$*-function* $h_n : \{0,1\}^n \mapsto \{0,1\}$ *such that for each sufficiently big* n*,* h_n *is not* $(1/2 + 1/2^{n/4})$*-approximable by* $2^{n/4}$*-size circuits.*[4] *Then there is a proof system* P *(which can be described explicitly given the definition of* h_n*) such that Items 1 and 2 from Theorem 1 are equivalent.*

The proof of Theorem 1 reveals also a proof complexity collapse which we discuss in the arXiv version of the paper.

1.2 Outline of the proof

Our starting point for the derivation of Theorem 1 is a relation between natural proofs and automatability which goes back to a work of Razborov and Krajíček. Razborov [30, 28] proved that certain theories of bounded arithmetic cannot prove explicit circuit lower bounds assuming strong pseudorandom generators exist. Krajíček [19, 21] developed the concept of feasible interpolation (a weaker version of automatability, cf. [22]) and reformulated Razborov's unprovability result in this language, see [22, Section 17.9] for more historical remarks.

▶ **Theorem 3** (Razborov-Krajíček [30, 28, 19] - informal version). *Let* P *be a proof system which simulates* EF *and satisfies some basic properties. If* P *is automatable and* P *proves efficiently* $\mathsf{tt}(h, n^{O(1)})$ *for some function* h*, then there are* $\mathsf{P/poly}$*-natural proofs useful against* $\mathsf{P/poly}$*.*

The second crucial ingredient we will use is a result of Carmosino, Impagliazzo, Kabanets and Kolokolova, who showed that natural proofs can be turned into learning algorithms [8]. This allows us to conclude the following.

▶ **Theorem 4** (Informal). *Let* P *be a proof system simulating* EF *and satisfying some basic properties. If* P *proves efficiently* $\mathsf{tt}(h, n^{O(1)})$ *for some function* h*, then automatability of* P *implies the existence of subexponential-size circuits learning p-size circuits over the uniform distribution, with membership queries.*

Theorem 4 directly implies that if strong pseudorandom generators exist and EF proves efficiently $\mathsf{tt}(h, n^{O(1)})$ for some h, then EF is automatable if and only if there are subexponential-size circuits learning p-size circuits over the uniform distribution, with membership queries. The disadvantage of this observation is that, unlike in Theorem 1, its assumptions are known to imply that both sides of the desired equivalence are false.

We note that the proof of Theorem 4 can be used to show also that optimal and automatable proof systems imply learning algorithms. In fact, it is possible to prove, unconditionally, that there is some propositional proof system P such that automatability of P is equivalent to the existence of subexponential-size circuits infnitely often learning $\mathsf{P/poly}$ over the uniform distribution. The proof is, however, non-constructive so (unlike in Corollary 2) we do not know which system P satisfies the equivalence. We discuss these results in more detail in the arXiv version of the paper.

[4] A circuit C with n inputs γ-approximates function $f : \{0,1\}^n \mapsto \{0,1\}$ if $\Pr_{x \in \{0,1\}^n}[C(x) = f(x)] \geq \gamma$.

The entrance of metamathematics. Unfortunately, it is unclear how to derive the opposite implication in Theorem 4. We do not know how to automate, say, EF assuming just the existence of efficient learning algorithms. In order to get the reverse, we need to assume that an efficient learning algorithm is provably correct in a proof system P, which p-simulates WF. For simplicity, let $P = \mathsf{WF}$. If we assume that WF proves efficiently for some small circuits that they can learn p-size circuits, we can show that there are small circuits such that WF proves efficiently that these circuits automate WF on formulas $\mathsf{tt}(f, n^{O(1)})$. In more detail, we first formalize in APC_1 the implication that WF-provable learning yields automatability of WF on $\mathsf{tt}(f, n^{O(1)})$ - if a learning circuit A does not find a small circuit for a given function f, the automating circuit uses WF-proof of the correctness of A to produce a short WF-proof of $\mathsf{tt}(f, n^{O(1)})$. Then, we translate the APC_1-proof to WF and conclude that WF proves that WF-provable learning implies automatability of WF. This allows us to show that if we have WF-provable learning, then WF is WF-provably automatable on $\mathsf{tt}(f, n^{O(1)})$.

It is important that assuming WF-provable learning, we are able to derive WF-provable automatability of WF, and not just automatability of WF. This makes it possible to obtain the opposite direction and establish the desired equivalence: If we know that WF proves that WF is automatable, we can formalize the proof of Theorem 4 in WF and conclude the existence of WF-provable learning algorithms.

Benefits of bounded arithmetic. The proof of Theorem 1 relies heavily on formalizations. Among other things we need to formalize the result of Carmosino, Impagliazzo, Kabanets and Kolokolova in APC_1[5], and use an elaborated way of expressing complex statements about metacomplexity by propositional formulas: existential quantifiers often need to be witnessed before translating them to propositional setting. The framework of bounded arithmetic allows us to deal with these complications in an elegant way: we often reason in bounded arithmetic, possibly using statements of higher quantifier complexity, and only subsequently translate the outcomes to propositional logic, if the resulting (proved) statement has coNP form. Notably, already propositional formulas expressing probabilities in the definition of learning algorithms require more advanced tools - the probabilities are encoded using suitable Nisan-Wigderson generators which come out of the notion of approximate counting in APC_1, cf. Section 3.2.

1.3 Related results

Learning algorithms and automatability have been linked already in the work of Alekhnovich, Braverman, Feldman, Klivans and Pitassi [1], who showed an informal connection between learning of weak circuit classes and automatability of some weak systems such as tree-like Resolution. As already mentioned, Atserias and Müller [3] proved that automating Resolution is NP-hard and their work has been extended to other weak proof systems, see e.g. [12, 13, 14]. A direct consequence of these results is that efficient algorithms automating the respective proof systems can be used to learn efficiently classes like P/poly. A major difference between these results and ours is that for our results to apply, the proof system needs to be sufficiently strong, while for the other results, the proof system needs to be weak (in the sense that lower bounds for the system are already known).

[5] We will actually formalize "CIKK" just conditionally, in order to avoid the formalization of Bertrand's postulate.

1.4 Open problems

Unconditional equivalence between learning and automatability. Is it possible to avoid the assumption on the provability of a circuit lower bound in Theorem 1 and establish an unconditional equivalence between learning and automatability?

Complexity theory from the perspective of metamathematics. Our results demonstrate that in the context of metamathematics it is possible to establish some complexity-theoretic connections which we are not able to establish otherwise. We exploit the metamathematical nature of the notion of automatability: efficient P-provability of the correctness of an algorithm implies efficient P-provability of automatability of P. Is it possible to take advantage of metamathematics in other contexts and resolve other important open problems in this setting? For example, could we get a version of the desired equivalence between the existence of efficient learning algorithms and the non-existence of cryptographic pseudorandom generators, cf. [26, 33, 27]? The question of basing cryptography on a worst-case assumption such as $\mathsf{P} \neq \mathsf{NP}$ could be addressed in this setting by showing that if a sufficiently strong proof system P proves efficiently that there is no strong pseudorandom generator[6], then P is p-bounded.

Circuit lower bound tautologies. How essential are circuit lower bound tautologies in our results? Consider fundamental questions of proof complexity (p-boundness, optimality, automatability) w.r.t. formulas $\mathsf{tt}(f, s)$. Do they coincide with the original ones? Are formulas $\mathsf{tt}(f, s)$ the hardest ones, do they admit optimal proof systems, or can we turn automatability on formulas $\mathsf{tt}(f, s)$ into automatability on all formulas?

2 Preliminaries

2.1 Natural proofs and learning algorithms

$[n]$ denotes $\{1, \ldots, n\}$. $\mathsf{Circuit}[s]$ denotes fan-in two Boolean circuits of size at most s. The size of a circuit is the number of gates.

▶ **Definition 5** (Natural property [32]). *Let $m = 2^n$ and $s, d : \mathbb{N} \mapsto \mathbb{N}$. A sequence of circuits $\{C_{2^n}\}_{n=1}^{\infty}$ is a $\mathsf{Circuit}[s(m)]$-natural property useful against $\mathsf{Circuit}[d(n)]$ if*

1. Constructivity. *C_m has m inputs and size $s(m)$,*
2. Largeness. *$\Pr_x[C_m(x) = 1] \geq 1/m^{O(1)}$,*
3. Usefulness. *For each sufficiently big m, $C_m(x) = 1$ implies that x is a truth-table of a function on n inputs which is not computable by circuits of size $d(n)$.*

▶ **Definition 6** (PAC learning). *A circuit class $\mathcal{C}$ is learnable over the uniform distribution by a circuit class $\mathcal{D}$ up to error ϵ with confidence δ, if there are randomized oracle circuits L^f from $\mathcal{D}$ such that for every Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$ computable by a circuit from $\mathcal{C}$, when given oracle access to f, input 1^n and the internal randomness $w \in \{0,1\}^*$, L^f outputs the description of a circuit satisfying*

$$\Pr_w[L^f(1^n, w) \text{ } (1-\epsilon)\text{-approximates } f] \geq \delta.$$

[6] The formalization of this statement would assume the existence of a p-size circuit which for any p-size circuit defining a potential pseudorandom generator outputs its distinguisher.

L^f uses non-adaptive membership queries if the set of queries which L^f makes to the oracle does not depend on the answers to previous queries. L^f uses random examples if the set of queries which L^f makes to the oracle is chosen uniformly at random.

In this paper, PAC learning always refers to learning over the uniform distribution. While, a priori, learning over the uniform distribution might not reflect real-world scenarios very well (and on the opposite end, learning over all distributions is perhaps overly restrictive), as far as we can tell it is possible that PAC learning of p-size circuits over the uniform distribution implies PAC learning of p-size circuits over all p-samplable distributions. Binnendyk, Carmosino, Kolokolova, Ramyaa and Sabin [4] proved the implication, if the learning algorithm in the conclusion is allowed to depend on the p-samplable distribution.

Boosting confidence and reducing error. The confidence of the learner and its error can be improved generically, see the arXiv version of the paper. We can thus often ignore the optimisation of these parameters.

Natural proofs vs learning algorithms. Natural proofs are actually equivalent to efficient learning algorithms with suitable parameters. In this paper we need just one implication.

▶ **Theorem 7** (Carmosino-Impagliazzo-Kabanets-Kolokolova [8]). *Let R be a* P/poly*-natural property useful against* Circuit$[n^k]$ *for $k \geq 1$. Then, for each $\gamma \in (0,1)$,* Circuit$[n^{k\gamma/a}]$ *is learnable by* Circuit$[2^{O(n^\gamma)}]$ *over the uniform distribution with non-adaptive membership queries, confidence 1, up to error $1/n^{k\gamma/a}$, where a is an absolute constant.*

2.2 Bounded arithmetic and propositional logic

Theories of bounded arithmetic capture various levels of feasible reasoning and present a uniform counterpart to propositional proof systems.

The first theory of bounded arithmetic formalizing p-time reasoning was introduced by Cook [10] as an equational theory PV. We work with its first-order conservative extension PV_1 from [24]. The language of PV_1, denoted PV as well, consists of symbols for all p-time algorithms given by Cobham's characterization of p-time functions, cf. [9]. A PV-formula is a first-order formula in the language PV. Σ^b_0 ($=\Pi^b_0$) denotes PV-formulas with only sharply bounded quantifiers $\exists x, x \leq |t|$, $\forall x, x \leq |t|$, where $|t|$ is "the length of the binary representation of t". Inductively, Σ^b_{i+1} resp. Π^b_{i+1} is the closure of Π^b_i resp. Σ^b_i under positive Boolean combinations, sharply bounded quantifiers, and bounded quantifiers $\exists x, x \leq t$ resp. $\forall x, x \leq t$. Predicates definable by Σ^b_i resp. Π^b_i formulas are in the Σ^p_i resp. Π^p_i level of the polynomial hierarchy, and vice versa. PV_1 is known to prove Σ^b_0(PV)-induction:

$$A(0) \wedge \forall x\ (A(x) \to A(x+1)) \to \forall x A(x),$$

for Σ^b_0-formulas A, cf. Krajíček [18].

Buss [7] introduced the theory S^1_2 extending PV_1 with the Σ^b_1-length induction:

$$A(0) \wedge \forall x < |a|, (A(x) \to A(x+1)) \to A(|a|),$$

for $A \in \Sigma^b_1$. S^1_2 proves the sharply bounded collection scheme $BB(\Sigma^b_1)$:

$$\forall i < |a|\ \exists x < a, A(i,x) \to \exists w\ \forall i < |a|, A(i, [w]_i),$$

for $A \in \Sigma^b_1$ ($[w]_i$ is the ith element of the sequence coded by w), which is unprovable in PV_1 under a cryptographic assumption, cf. [11]. On the other hand, S^1_2 is $\forall\Sigma^b_1$-conservative over PV_1. This is a consequence of Buss's witnessing theorem stating that $\mathsf{S}^1_2 \vdash \exists y, A(x,y)$ for $A \in \Sigma^b_1$ implies $\mathsf{PV}_1 \vdash A(x, f(x))$ for some PV-function f.

Following a work by Krajíček [20], Jeřábek [15, 16, 17] systematically developed a theory APC_1 capturing probabilistic p-time reasoning by means of approximate counting.[7] The theory APC_1 is defined as $\mathsf{PV}_1 + dWPHP(\mathsf{PV})$ where $dWPHP(\mathsf{PV})$ stands for the dual (surjective) pigeonhole principle for PV-functions, i.e. for the set of all formulas

$$x > 0 \to \exists v < x(|y|+1)\forall u < x|y|,\ f(u) \neq v,$$

where f is a PV-function which might involve other parameters not explicitly shown. We devote Section 2.3 to a more detailed description of the machinery of approximate counting in APC_1.

Any Π^b_1-formula ϕ provable in PV_1 can be expressed as a sequence of tautologies $||\phi||_n$ with proofs in the Extended Frege system EF which are constructible in p-time (given a string of the length n), cf. [10]. Similarly, Π^b_1-formulas provable in APC_1 translate to tautologies with p-time constructible proofs in WF, an extension of EF introduced by Jeřábek [15]. We describe the translation and system WF in more detail below.

As it is often easier to present a proof in a theory of bounded arithmetic than in the corresponding propositional system, bounded arithmetic functions, so to speak, as a uniform language for propositional logic.

We refer to Krajíček [22] for basic notions in proof complexity.

▶ **Definition 8** (WF (WPHP Frege), cf. Jeřábek [15])**.** *Let L be a finite and complete language for propositional logic, i.e. L consists of finitely many boolean connectives of constant arity such that each boolean function of every arity can be expressed by an L-formula, and let $\mathcal{R}$ be a finite, sound and implicationally complete set of Frege rules (in the langauge L). A* WF*-proof of a (L-)circuit A is a sequence of circuits $A_0, \dots, A_k$ such that $A_k = A$, and each A_i is derived from some $A_{j_1}, \dots, A_{j_\ell}$, $j_1, \dots, j_\ell < i$ by a Frege rule from $\mathcal{R}$, or it is similar to some A_j, $j < i$, or it is the dWPHP axiom,*

$$\bigvee_{\ell=1}^{m} (r_\ell \neq C_{i,\ell}(D_{i,1}, \dots, D_{i,n})),$$

where $n < m$ and r_ℓ are pairwise distinct variables which do not occur in circuits A, $C_{i,\ell'}$, or A_j for $j < i$, but may occur in circuits $D_{i,1}, \dots, D_{i,n}$.

The similarity rule in Definition 8 is verified by a specific p-time algorithm which checks that circuits A_i and A_j can be "unfolded" to the same (possible huge) formula, cf. [15, Lemma 2.2.]. Intuitively, the NLOG ($\subseteq$ P) algorithm recognizes if two circuits are not similar by guessing a partial path through them, going from the output to the inputs, where on at least one instruction the circuits disagree. As defined WF depends on the choice of Frege rules and language L, but for each choice the resulting systems are p-equivalent, so we can identify them. The dWPHP axiom refers to "dual weak pigeonhole principle" postulating the existence of an element $r_1, \dots, r_m$ outside the range of a p-size map $C_{i,1}, \dots, C_{i,m} : \{0,1\}^n \mapsto \{0,1\}^m$.

[7] Krajíček [20] introduced a theory BT defined as $\mathsf{S}^1_2 + dWPHP(\mathsf{PV})$ and proposed it as a theory for probabilistic p-time reasoning.

The dWPHP axiom comes with a specification of circuits $C_{i,1},\dots,C_{i,m},D_{i,1},\dots,D_{i,n}$ so that we can recognize the axiom efficiently. The role of circuits $D_{i,1},\dots,D_{i,n}$ in the dWPHP axiom is to allow WF to postulate not only that $r_1,\dots,r_m$ is not the output of $C_{i,1},\dots,C_{i,m}$ on a specific input $x_1,\dots x_n$ but to postulate that $r_1,\dots,r_m$ is not the output of $C_{i,1},\dots,C_{i,m}$ on other inputs (which could depend on $r_1,\dots,r_m$) either.

The translation of a Π^b_1 formula ϕ into a sequence of propositional formulas $||\phi||_{\overline{n}}$ works as follows. For each PV-function $f(x_1,\dots,x_k)$ and numbers $n_1,\dots,n_k$ we have a p-size circuit C_f computing the restriction $f:2^{n_1}\times\cdots\times 2^{n_k}\mapsto 2^{b(n_1,\dots,n_k)}$, where b is a suitable "bounding" polynomial for f. The formula $||f||_{\overline{n}}(p,q,r)$ expresses that C_f outputs r on input p, with q being the auxiliary variables corresponding to the nodes of C_f. The formula $||\phi(x)||_{\overline{n}}(p,q)$ is defined as $||\phi'(x)||_{\overline{n}}(p,q)$, where ϕ' is the negation normal form of ϕ, i.e. negations in ϕ' are only in front of atomic formulas. The formula $||\phi'(x)||_{\overline{n}}(p,q)$ is defined inductively in a straightforward way so that $||\dots||$ commutes with $\vee,\wedge$. The atoms p correspond to variables x, atoms q correspond to the universally quantified variables of ϕ and to the outputs and auxiliary variables of circuits C_f for functions f appearing in ϕ. Sharply bounded quantifiers are replaced by polynomially big conjuctions resp. disjunctions. For the atomic formulas we have,

$$||f(x)=g(x)||_{\overline{n}} :=||f(x)||_{\overline{n}}(p,q,r)\wedge||g(x)||_{\overline{n}}(p,q',r')\to\bigwedge_i r_i=r'_i,$$

$$||\neg f(x)=g(x)||_{\overline{n}} :=||f(x)||_{\overline{n}}(p,q,r)\wedge||g(x)||_{\overline{n}}(p,q',r')\to\neg\bigwedge_i r_i=r'_i,$$

$$||f(x)\le g(x)||_{\overline{n}} :=||f(x)||_{\overline{n}}(p,q,r)\wedge||g(x)||_{\overline{n}}(p,q',r')\to\bigwedge_i(r_i\wedge\bigwedge_{j>i}(r_j=r'_j)\to r'_i),$$

$$||\neg f(x)\le g(x)||_{\overline{n}} :=||f(x)||_{\overline{n}}(p,q,r)\wedge||g(x)||_{\overline{n}}(p,q',r')\to\neg\bigwedge_i(r_i\wedge\bigwedge_{j>i}(r_j=r'_j)\to r'_i).$$

2.3 Approximate counting

In order to prove our results we will need to use Jeřábek's theory of approximate counting. This section recalls the properties of APC_1 we will need.

By a definable set we mean a collection of numbers satisfying some formula, possibly with parameters. When a number a is used in a context which asks for a set it is assumed to represent the integer interval $[0,a)$, e.g. $X\subseteq a$ means that all elements of set X are less than a. If $X\subseteq a$, $Y\subseteq b$, then $X\times Y:=\{bx+y\mid x\in X,y\in Y\}\subseteq ab$ and $X\dot\cup Y:=X\cup\{y+a\mid y\in Y\}\subseteq a+b$. Rational numbers are assumed to be represented by pairs of integers in the natural way. We use the notation $x\in Log\leftrightarrow\exists y,\ x=|y|$ and $x\in LogLog\leftrightarrow\exists y,\ x=||y||$.

Let $C:2^n\to 2^m$ be a circuit and $X\subseteq 2^n, Y\subseteq 2^m$ definable sets. We write $C:X\twoheadrightarrow Y$ if $\forall y\in Y\exists x\in X,\ C(x)=y$. Jeřábek [17] gives the following definitions in APC_1 (but they can be considered in weaker theories as well).

▶ **Definition 9.** *Let $X,Y\subseteq 2^n$ be definable sets, and $\epsilon\le 1$. The size of X is approximately less than the size of Y with error ϵ, written as $X\preceq_\epsilon Y$, if there exists a circuit C, and $v\neq 0$ such that*

$$C:v\times(Y\dot\cup\epsilon 2^n)\twoheadrightarrow v\times X.$$

$X\approx_\epsilon Y$ stands for $X\preceq_\epsilon Y$ and $Y\preceq_\epsilon X$.

Since a number s is identified with the interval $[0, s)$, $X \preceq_\epsilon s$ means that the size of X is at most s with error ϵ.

The definition of $X \preceq_\epsilon Y$ is an unbounded $\exists\Pi^b_2$-formula even if X, Y are defined by circuits so it cannot be used freely in bounded induction. Jeřábek [17] solved this problem by working in HARD^A, a conservative extension of APC_1, defined as a relativized theory $\mathsf{PV}_1(\alpha) + dWPHP(\mathsf{PV}(\alpha))$ extended with axioms postulating that $\alpha(x)$ is a truth-table of a function on $||x||$ variables hard on average for circuits of size $2^{||x||/4}$, see Section 3.2. In HARD^A there is a $\mathsf{PV}_1(\alpha)$ function $Size$ approximating the size of any set $X \subseteq 2^n$ defined by a circuit C so that $X \approx_\epsilon Size(C, 2^n, 2^{\epsilon^{-1}})$ for $\epsilon^{-1} \in Log$, cf. [17, Lemma 2.14]. If $X \cap t \subseteq 2^{|t|}$ is defined by a circuit C and $\epsilon^{-1} \in Log$, we can define

$$\Pr_{x<t}[x \in X]_\epsilon := \frac{1}{t} Size(C, 2^{|t|}, 2^{\epsilon^{-1}}).$$

The presented definitions of approximate counting are well-behaved:

▶ **Proposition 10** (Jeřábek [17]). *(in PV_1) Let $X, X', Y, Y', Z \subseteq 2^n$ and $W, W' \subseteq 2^m$ be definable sets, and $\epsilon, \delta < 1$. Then*

- **i)** $X \subseteq Y \Rightarrow X \preceq_0 Y$,
- **ii)** $X \preceq_\epsilon Y \wedge Y \preceq_\delta Z \Rightarrow X \preceq_{\epsilon+\delta} Z$,
- **iii)** $X \preceq_\epsilon X' \wedge W \preceq_\delta W' \Rightarrow X \times W \preceq_{\epsilon+\delta+\epsilon\delta} X' \times W'$.
- **iv)** $X \preceq_\epsilon X' \wedge Y \preceq_\delta Y'$ *and* X', Y' *are separable by a circuit, then* $X \cup Y \preceq_{\epsilon+\delta} X' \cup Y'$.

▶ **Proposition 11** (Jeřábek [17]). *(in APC_1)*

1. *Let $X, Y \subseteq 2^n$ be definable by circuits, $s, t, u \leq 2^n$, $\epsilon, \delta, \theta, \gamma < 1, \gamma^{-1} \in Log$. Then*
   - **(i)** $X \preceq_\gamma Y$ *or* $Y \preceq_\gamma X$,
   - **(ii)** $s \preceq_\epsilon X \preceq_\delta t \Rightarrow s < t + (\epsilon + \delta + \gamma)2^n$,
   - **(iii)** $X \preceq_\epsilon Y \Rightarrow 2^n \backslash Y \preceq_{\epsilon+\gamma} 2^n \backslash X$,
   - **(iv)** $X \approx_\epsilon s \wedge Y \approx_\delta t \wedge X \cap Y \approx_\theta u \Rightarrow X \cup Y \approx_{\epsilon+\delta+\theta+\gamma} s + t - u$.
2. *(Disjoint union) Let $X_i \subseteq 2^n$, $i < m$ be defined by a sequence of circuits and $\epsilon, \delta \leq 1$, $\delta^{-1} \in Log$. If $X_i \preceq_\epsilon s_i$ for every $i < m$, then $\bigcup_{i<m}(X_i \times \{i\}) \preceq_{\epsilon+\delta} \sum_{i<m} s_i$.*
3. *(Averaging) Let $X \subseteq 2^n \times 2^m$ and $Y \subseteq 2^m$ be definable by circuits, $Y \preceq_\epsilon t$ and $X_y \preceq_\delta s$ for every $y \in Y$, where $X_y := \{x | \langle x, y\rangle \in X\}$. Then for any $\gamma^{-1} \in Log$,*

   $$X \cap (2^n \times Y) \preceq_{\epsilon+\delta+\epsilon\delta+\gamma} st.$$

When proving Σ^b_1 statements in APC_1 we can afford to work in $\mathsf{S}^1_2 + dWPHP(\mathsf{PV}) + BB(\Sigma^b_2)$ and, in fact, assuming the existence of a single hard function in PV_1 gives us the full power of APC_1. Here, $BB(\Sigma^b_2)$ is defined as $BB(\Sigma^b_1)$ but with $A \in \Sigma^b_2$.

▶ **Lemma 12** ([25]). *Suppose $\mathsf{S}^1_2 + dWPHP(\mathsf{PV}) + BB(\Sigma^b_2) \vdash \exists y A(x, y)$ for $A \in \Sigma^b_1$. Then, for every $\epsilon < 1$, there is k and PV-functions g, h such that PV_1 proves*

$$|f| \geq |x|^k \wedge \exists y, |y| = ||f||, C_h(y) \neq f(y) \to A(x, g(x, f))$$

where $f(y)$ is the yth bit of f, $f(y) = 0$ for $y > |f|$, and C_h is a circuit of size $\leq 2^{\epsilon||f||}$ generated by h on f, x. Moreover, $\mathsf{APC}_1 \vdash \exists y A(x, y)$.

3 Formalizing complexity-theoretic statements

3.1 Circuit lower bounds

An "almost everywhere" formulation of a circuit lower bound for circuits of size s and a function f says that for every sufficiently big n, for each circuit C with n inputs and size s, there exists an input y on which the circuit C fails to compute $f(y)$.

If $f : \{0,1\}^n \to \{0,1\}$ is an NP function and $s = n^k$ for a constant k, this can be written down as a $\forall\Sigma^b_2$ formula $\mathsf{LB}(f, n^k)$,

$$\forall n,\ n > n_0\ \forall \text{ circuit } C \text{ of size } \leq n^k\ \exists y,\ |y| = n,\ C(y) \neq f(y),$$

where n_0 is a constant and $C(y) \neq f(y)$ is a Σ^b_2 formula stating that a circuit C on input y outputs the opposite value of $f(y)$.

If we want to express $s(n)$-size lower bounds for $s(n)$ as big as $2^{O(n)}$, we add an extra assumption on n stating that $\exists x,\ n = ||x||$. That is, the resulting formula $\mathsf{LB}_{\mathsf{tt}}(f, s(n))$ has form "$\forall x, n; n = ||x|| \to \dots$". Treating x, n as free variables, $\mathsf{LB}_{\mathsf{tt}}(f, s(n))$ is Π^b_1 if f is, for instance, SAT because $n = ||x||$ implies that the quantifiers bounded by $2^{O(n)}$ are sharply bounded. Moreover, allowing $f \in \mathsf{NE}$ lifts the complexity of $\mathsf{LB}_{\mathsf{tt}}(f, s(n))$ just to $\forall\Sigma^b_1$. The function $s(n)$ in $\mathsf{LB}_{\mathsf{tt}}(f, s(n))$ is assumed to be a PV-function with input x (satisfying $||x|| = n$).

In terms of the *Log*-notation, $\mathsf{LB}(f, n^k)$ implicitly assumes $n \in Log$ while $\mathsf{LB}_{\mathsf{tt}}(f, n^k)$ assumes $n \in LogLog$. By chosing the scale of n we are determining how big objects are going to be "feasible" for theories reasoning about the statement. In the case $n \in LogLog$, the truth-table of f (and everything polynomial in it) is feasible. Assuming just $n \in Log$ means that only the objects of polynomial-size in the size of the circuit are feasible. Likewise, the theory reasoning about the circuit lower bound is less powerful when working with $\mathsf{LB}(f, n^k)$ than with $\mathsf{LB}_{\mathsf{tt}}(f, n^k)$. (The scaling in $\mathsf{LB}_{\mathsf{tt}}(f, s)$ corresponds to the choice of parameters in natural proofs and in the formalizations by Razborov [29].)

We can analogously define formulas $\mathsf{LB}_{\mathsf{tt}}(f, s(n), t(n))$ expressing an average-case lower bound for f, where f is a free variable (with $f(y)$ being the yth bit of f and $f(y) = 0$ for $y > |f|$). More precisely, $\mathsf{LB}_{\mathsf{tt}}(f, s(n), t(n))$ generalizes $\mathsf{LB}_{\mathsf{tt}}(f, s(n))$ by saying that each circuit of size $s(n)$ fails to compute f on at least $t(n)$ inputs, for PV-functions $s(n), t(n)$. Since $n \in LogLog$, $\mathsf{LB}_{\mathsf{tt}}(f, s(n), t(n))$ is Π^b_1.

Propositional version. An $s(n)$-size circuit lower bound for a function $f : \{0,1\}^n \to \{0,1\}$ can be expressed by a $2^{O(n)}$-size propositional formula $\mathsf{tt}(f, s)$,

$$\bigvee_{y \in \{0,1\}^n} f(y) \neq C(y)$$

where the formula $f(y) \neq C(y)$ says that an $s(n)$-size circuit C represented by $poly(s)$ variables does not output $f(y)$ on input y. The values $f(y)$ are fixed bits. That is, the whole truth-table of f is hard-wired in $\mathsf{tt}(f, s)$.

The details of the encoding of the formula $\mathsf{tt}(f, s)$ are not important for us as long as the encoding is natural because systems like EF considered in this paper can reason efficiently about them. We will assume that $\mathsf{tt}(f, s)$ is the formula resulting from the translation of Π^b_1 formula $\mathsf{LB}_{\mathsf{tt}}(h, s)$, where $n_0 = 0$, n, x are substituted after the translation by fixed constants so that $x = 2^{2^n}$, and h is a free variable (with $h(y)$ being the yth bit of h and $h(y) = 0$ for $y > |h|$) which is substituted after the translation by constants defining f.

Analogously, we can express average-case lower bounds by propositional formulas $\mathsf{tt}(f, s(n), t(n))$ obtained by translating $\mathsf{LB}_{\mathsf{tt}}(h, s(n), t(n)2^n)$, with $n_0 = 0$, fixed $x = 2^{2^n}$ and h substituted after the translation by f.

3.2 Learning algorithms

A circuit class $\mathcal{C}$ is defined by a PV-formula if there is a PV-formula defining the predicate $C \in \mathcal{C}$. Definition 6 can be formulated in the language of HARD^A: A circuit class $\mathcal{C}$ (defined by a PV-formula) is learnable over the uniform disribution by a circuit class $\mathcal{D}$ (defined by a PV-formula) up to error ϵ with confidence δ, if there are randomized oracle circuits L^f from $\mathcal{D}$ such that for every Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$ (represented by its truth-table) computable by a circuit from $\mathcal{C}$, for each $\gamma^{-1} \in Log$, when given oracle access to f, input 1^n and the internal randomness $w \in \{0,1\}^*$, L^f outputs the description of a circuit satisfying

$$\Pr_w[L^f(1^n, w) \ (1-\epsilon)\text{-approximates } f]_\gamma \geq \delta.$$

The inner probability of approximability of f by $L^f(1^n, w)$ is counted exactly. This is possible because f is represented by its truth-table, which implies that $2^n \in Log$.[8]

Propositional version. In order, to translate the definition of learning algorithms to propositional formulas we need to look more closely at the definition of HARD^A.

PV_1 can be relativized to $\mathsf{PV}_1(\alpha)$. The new function symbol α is then allowed in the inductive clauses for introduction of new function symbols. This means that the language of $\mathsf{PV}_1(\alpha)$, denoted also $\mathsf{PV}(\alpha)$, contains symbols for all p-time oracle algorithms.

▶ **Proposition 13** (Jeřábek [15]). *For every constant $\epsilon < 1/3$ there exists a constant n_0 such that APC_1 proves: for every $n \in LogLog$ such that $n > n_0$, there exist a function $f : 2^n \to 2$ such that no circuit of size $2^{\epsilon n}$ computes f on $\geq (1/2 + 1/2^{\epsilon n})2^n$ inputs.*

▶ **Definition 14** (Jeřábek [15]). *The theory HARD^A is an extension of the theory $\mathsf{PV}_1(\alpha) + dWPHP(\mathsf{PV}(\alpha))$ by the axioms*

1. *$\alpha(x)$ is a truth-table of a Boolean function in $||x||$ variables,*
2. *$\mathsf{LB}_{\mathsf{tt}}(\alpha(x), 2^{||x||/4}, 2^{||x||}(1/2 - 1/2^{||x||/4}))$, for constant n_0 from Proposition 13,*
3. *$||x|| = ||y|| \to \alpha(x) = \alpha(y)$.*

By inspecting the proof of Lemma 2.14 in [17], we can observe that on each input $C, 2^n, 2^{\epsilon^{-1}}$ the $\mathsf{PV}_1(\alpha)$-function $Size$ calls α just once (to get the truth-table of a hard function which is then used as the base function of the Nisan-Wgiderson generator). In fact, $Size$ calls α on input x which depends only on $|C|$, the number of inputs of C and w.l.o.g. also just on $|\epsilon^{-1}|$ (since decreasing ϵ leads only to a better approximation). In combination with the fact that the approximation $Size(C, 2^n, 2^{\epsilon^{-1}}) \approx_\epsilon X$, for $X \subseteq 2^n$ defined by C, is not affected by a particular choice of the hard boolean function generated by α, we get that APC_1 proves

$$\mathsf{LB}_{\mathsf{tt}}(y, 2^{||y||/4}, 2^{||y||}(1/2 - 1/2^{||y||/4})) \wedge ||y|| = S(C, 2^n, 2^{\epsilon^{-1}}) \to \ Sz(C, 2^n, 2^{\epsilon^{-1}}, y) \approx_\epsilon X,$$

where Sz is defined as $Size$ with the only difference that the call to $\alpha(x)$ on $C, 2^n, 2^{\epsilon^{-1}}$ is replaced by y and $S(C, 2^n, 2^{\epsilon^{-1}}) = ||x||$ for a PV-function S.

[8] It could be interesting to develop systematically a standard theory of learning algorithms in APC_1 and WF, but it is not our goal here. Note, for example, that when we are learning small circuits it is not clear how to boost the confidence to 1 in APC_1, because we don't have counting with exponential precision.

This allows us to express $\Pr_{x<t}[x \in X]_\epsilon = a$, where $\epsilon^{-1} \in Log$ and $X \cap t \subseteq 2^{|t|}$ is defined by a circuit C, without a $\mathsf{PV}_1(\alpha)$ function, by formula

$$\forall y\,(\mathsf{LB}_{\mathsf{tt}}(y, 2^{||y||/4}, 2^{||y||}(1/2{-}1/2^{||y||/4}))\wedge||y|| = S(C, 2^{|t|}, 2^{\epsilon^{-1}}) \to Sz(C, 2^{|t|}, 2^{\epsilon^{-1}}, y)/t = a).$$

We denote the resulting formula by $\Pr^y_{x<t}[x \in X]_\epsilon = a$. We will use the notation $\Pr^y_{x<t}[x \in X]_\epsilon$ in equations with the intended meaning that the equation holds for the value $Sz(\cdot,\cdot,\cdot,\cdot)/t$ under corresponding assumptions. For example, $t \cdot \Pr^y_{x<t}[x \in X]_\epsilon \preceq_\delta a$ stands for "$\forall y, \exists v, \exists$ circuit $\hat{C}$ (defining a surjection) which witnesses that $\mathsf{LB}_{\mathsf{tt}}(y, 2^{||y||/4}, 2^{||y||}(1/2 - 1/2^{||y||/4})) \wedge ||y|| = S(C, 2^{|t|}, 2^{\epsilon^{-1}})$ implies $Sz(C, 2^{|t|}, 2^{\epsilon^{-1}}, y) \preceq_\delta a$".

The definition of learning can be now expressed without a $\mathsf{PV}_1(\alpha)$ function: If circuit class $\mathcal{C}$ is defined by a PV-function, the statement that a given oracle algorithm L (given by a PV-function with oracle queries) learns a circuit class $\mathcal{C}$ over the uniform distribution up to error ϵ with confidence δ can be expressed as before with the only difference that we replace $\Pr_w[L^f(1^n, w)\ (1-\epsilon)\text{-approximates } f]_\gamma \geq \delta$ by

$$\Pr^y_w[L^f(1^n, w)\ (1-\epsilon)\text{-approximates } f]_\gamma \geq \delta.$$

Since the resulting formula A defining learning is not Π^b_1 (because of the assumption $\mathsf{LB}_{\mathsf{tt}}$) we cannot translate it to propositional logic. We will sidestep the issue by translating only the formula B obtained from A by deleting subformula $\mathsf{LB}_{\mathsf{tt}}$ (but leaving $||y|| = S(\cdot,\cdot,\cdot)$ intact) and replacing the variables y by fixed bits representing a hard boolean function. In more detail, Π^b_1 formula B can be translated into a sequence of propositional formulas $\mathsf{lear}^y_\gamma(L, \mathcal{C}, \epsilon, \delta)$ expressing that "if $C \in \mathcal{C}$ is a circuit computing f, then L querying f generates a circuit D such that $\Pr[D(x) = f(x)] \geq 1 - \epsilon$ with probability $\geq \delta$, which is counted approximately with precision γ". Note that C, f are represented by free variables and that there are also free variables for error γ from approximate counting and for boolean functions y. As in the case of tt-formulas, we fix $|f| = 2^n$, so n is not a free variable. Importantly, $\mathsf{lear}^y_\gamma(L, \mathcal{C}, \epsilon, \delta)$ does not postulate that y is a truth-table of a hard boolean function. Nevertheless, for any fixed (possibly non-uniform) bits representing a sequence of boolean functions $h = \{h_m\}_{m>n_0}$ such that h_m is not $(1/2 + 1/2^{m/4})$-approximable by any circuit of size $2^{m/4}$, we can obtain formulas $\mathsf{lear}^h_\gamma(L, \mathcal{C}, \epsilon, \delta)$ by substituting bits h for y.

Using a single function h in $\mathsf{lear}^h_\gamma(L, \mathcal{C}, \epsilon, \delta)$ does not ruin the fact that (the translation of function) Sz approximates the respective probability with accuracy γ because Sz queries a boolean function y which depends just on the number of atoms representing γ^{-1} and on the size of the circuit D defining the predicate we count together with the number of inputs of D. The size of D and the number of its inputs are w.l.o.g. determined by the number of inputs of f.

If we are working with formulas $\mathsf{lear}^h_\gamma(L, \mathcal{C}, \epsilon, \delta)$, where h is a sequence of bits representing a hard boolean function, in a proof system which cannot prove efficiently that h is hard, our proof system might not be able to show that the definition is well-behaved - it might not be able to derive some standard properties of the function Sz used inside the formula. Nevertheless, in our theorems this will never be the case: our proof systems will always know that h is hard.

In formulas $\mathsf{lear}^y_\gamma(L, \mathcal{C}, \epsilon, \delta)$ we can allow L to be a sequence of nonuniform circuits, with a different advice string for each input length. One way to see that is to use additional input to L in Π^b_1 formula B, then translate the formula to propositional logic and substitute the right bits of advice for the additional input. Again, the precise encoding of the formula $\mathsf{lear}^y_\gamma(L, \mathcal{C}, \epsilon, \delta)$ does not matter very much to us but in order to simplify proofs we will

assume that $\mathsf{lear}^y_\gamma(L, \mathsf{Circuit}[n^k], \epsilon, \delta)$ has the from $\neg\mathsf{tt}(f, n^k) \to R$, where n, k are fixed, f is represented by free variables and R is the remaining part of the formula expressing that L generates a suitable circuit with high probability.

3.3 Automatability

Let Φ be a class of propositional formulas. We say that a proof system P is automatable w.r.t. Φ up to proofs of size s, where $s : \Phi \mapsto \mathbb{N}$ is a function, if there is a PV-function A such that for each $\phi \in \Phi$ and each t-size P-proof of ϕ with $t \leq s$, $A(\phi, 1^t)$ is a P-proof of ϕ.

In our main theorem we will need a slightly modified notion of automatability where the automating algorithm outputs a proof of a given tautology ϕ which is not much longer than a proof of an associated tautology ψ. (Formula ψ will be closely related to ϕ: while ϕ will express a worst-case lower bound, ψ will express an average-case lower bound for the same function.)

Let Φ be a class of pairs of propositional formulas. We say that a proof system P is automatable w.r.t. Φ up to proofs of size s if there is a PV-function A such that for each pair $\langle \psi, \phi \rangle \in \Phi$ and each t-size P-proof of ψ with $t \leq s$, $A(\phi, 1^t)$ is a P-proof of ϕ.

Propositional version. If Φ is defined by a PV-function and $s \in Log$ is a PV-function, the statement that an algorithm A (given by a PV-function) automates system P w.r.t. Φ up to proofs of size s is Π^b_1. Therefore, it can be translated into a sequence of propositional formulas $\mathsf{aut}_P(A, \Phi, s)$. Again, in formulas $\mathsf{aut}_P(A, \Phi, s)$ we can allow A to be a sequence of nonuniform circuits and s to be arbitrary possibly nonuniform parameter.

4 Learning algorithms from natural proofs in APC_1

The formalization of the transformation of natural proofs into learning algorithms follows from a straightforward inspection of the original proof. The proof can be found in the arXiv version of the paper.

▶ **Theorem 15.** *There is a* PV*-function L such that* APC_1 *proves: For $k \geq 1$, $d \geq 2$, $2^{n^d}, n^{dk}, \delta^{-1} \in Log$, $\delta < 1/N^3$ and a prime $n^d \leq p \leq 2n^d$, let R_N be a circuit with $N = 2^n$ inputs such that for sufficiently big N,*

1. *$R_N(x) = 1$ implies that x is a truth-table of a boolean function with n inputs hard for* $\mathsf{Circuit}[n^{10dk}]$,
2. *$\{x \mid R_N(x) = 1\} \succeq_\delta 2^N/N$.*

Then, circuits with n^d inputs and size n^{dk} are learnable by circuit $L(R_N, p)$ over the uniform distribution with membership queries, confidence $1/N^4$, up to error $1/2 - 1/N^3$. Here, the confidence is counted approximately with error δ using PV*-function Sz and the corresponding assumptions* $\mathsf{LB}_{\mathsf{tt}}$ *expressing hardness of a boolean function y, i.e. using formulas* $\Pr^y[\cdot]_\delta$.

5 Main theorem

Our main theorem holds for any "decent" proof system p-simulating WF, which is well-behaved in the sense that it APC_1-provably satisfies some basic properties.

▶ **Definition 16** (APC_1-decent proof system)**.** *A propositional proof system P is* APC_1*-decent if the language L of P is finite and complete, i.e. L consists of connectives of constant arity such that each boolean function of every arity can be expressed by an L-formula, P proves efficiently its own reflection principle, i.e. formulas stating that if π is a P-proof of ϕ then ϕ holds, cf. [22], and there is a* PV*-function F such that* APC_1 *proves:*

1. *P p-simulates* WF*, i.e. F maps each* WF*-proof of ϕ to a P-proof of ϕ.*
2. *P admits substitution property: F maps each triple $\langle \phi, \rho, \pi \rangle$ to a P-proof of $\phi|_\rho$, where π is a P-proof of ϕ and $\phi|_\rho$ is formula ϕ after applying substitution ρ which replaces atoms of ϕ by formulas.*
3. *F maps each pair $\langle \pi, \pi' \rangle$, where π is a P-proof of ϕ and π' is a P-proof of $\phi \to \psi$, to a P-proof of ψ.*

In Definition 16, WF refers to some fixed system from the set of all WF systems. It follows from the proof of Lemma 17 that if APC_1 proves that P p-simulates a WF-system Q, then for every WF-system R, APC_1 proves that P p-simulates R, so the particular choice of the WF-system does not matter. When we use connectives $\wedge, \vee, \neg, \to$ in an APC_1-decent system P, we assume that these are expressed in the language of P.

▶ **Lemma 17.** *Each* WF *system is APC_1-decent. Moreover, for each APC_1-decent proof system P the following holds.*

1. *For every Frege rule which derives ϕ from $\phi_1, \ldots, \phi_k$, there is a* PV*-function F such that APC_1 proves that F maps each $(k+1)$-tuple $\langle \pi_1, \ldots, \pi_k, \rho \rangle$ to a P-proof of $\phi|_\rho$, where π_i is a P-proof of $\phi_i|_\rho$ for a substitution ρ replacing each atom of $\phi, \phi_1, \ldots, \phi_k$ by a formula.*
2. *There is a* PV*-function F such that APC_1 proves that F maps each pair $\langle \phi, b \rangle$, for assignment b satisfying formula ϕ, to a P-proof of $\phi(b)$.*
3. *Let π be a P-proof of $E \to \phi$, where E defines a computation of a circuit which is allowed to use atoms from ϕ as inputs but other atoms of E do not appear in ϕ, i.e. E is the conjunction of extension axioms of* EF *built on atoms from ϕ. Then, there is a $poly(|\pi|)$-size P-proof of ϕ.*

Proof. WF is known to prove efficiently its own reflection principle, cf. [15]. In order to show that it is APC_1-decent, it thus suffices to prove that it satisfies Items 1-3 from Definition 16.

Item 2 is established already in PV_1 by Σ^b_1-induction on the length of the proof π (which can be used because of $\forall\Sigma^b_1$-conservativity of S^1_2 over PV_1): F replaces each circuit C from π by $C|_\rho$ and preserves all WF-derivation rules.

Item 1 holds trivially if the given WF-system P is the WF-system P' from Definition 16. Otherwise, we use implicational completeness of P and the completeness of the language of P to simulate all $O(1)$ Frege rules of P' by $O(1)$ steps in P. (This does not require that the implicational completeness of P is provable in APC_1 because we need to simulate only $O(1)$ Frege rules of finite size). Similarly, by Σ^b_1-induction and the completeness of the language of P, we simulate each circuit in the language of P' by a circuit in the language of P and show that this simulation preserves the similarity rule. Then, given an s-size P'-proof of ϕ, we obtain a $poly(s)$-size P-proof of ϕ using the simulation of Frege rules of P', the similarity rule and dWPHP axiom, together with substituting the right circuits in Frege rules. This is done again in PV_1 by Σ^b_1-induction on the length of the P'-proof.

Item 3 follows by simulating modus ponens as in the proof of Item 1.

For the "moreover" part, see the arXiv version of the paper. ◀

APC_1-decent proof systems can be much stronger than WF. For example, consider ZFC as a propositional proof system: a ZFC-proof of propositional formula ϕ is a ZFC-proof of the statement encoding that ϕ is a tautology. We can add the reflection of ZFC to WF, i.e. we will allow WF to derive (substitutional instances of) formulas stating that "If π is a ZFC-proof of ϕ, then ϕ holds." The new system is as strong as ZFC w.r.t. tautologies and it is easy to see that it is APC_1-decent. (The reflection of the system can be proved in APC_1 extended with an axiom postulating the reflection for ZFC.)

▶ **Theorem 18** (Learning versus automatability). *Let P be an APC_1-decent proof system and assume there is a sequence of boolean functions $h = \{h_n\}_{n>n_1}$, for a constant n_1, such that P proves efficiently $\mathsf{tt}(h_n, 2^{n/4}, 1/2 - 1/2^{n/4})$. Then, for each constant K and constant $\gamma < 1$, the following statements are equivalent.*

1. **Provable learning.** *For each $k \geq 1$ and $\ell \geq K + 1$, there are 2^{Kn^γ}-size circuits A such that for each sufficiently big n, P proves efficiently*

   $$\mathsf{lear}^h_{1/2^{\ell n^\gamma}}(A, \mathsf{Circuit}[n^k], 1/2 - 1/2^{Kn^\gamma}, 1/2^{Kn^\gamma}).$$

2. **Provable automatability.** *For each $k \geq 1$, for each function $s(n) \geq 2^n$, there is a constant K' and $s^{K'}$-size circuits B such that P proves efficiently*

   $$\mathsf{aut}_P(B, \Phi, s),$$

   where Φ is the set of pairs $\langle \mathsf{tt}(f, 2^{Kn^\gamma}, 1/2 - 1/2^{Kn^\gamma}), \mathsf{tt}(f, n^k)\rangle$ for all boolean functions f with n inputs.

Proof. (**1.** $\to$ **2.**) We first prove the following statement in APC_1.

▷ Claim 19 (in APC_1). Assume that π is a P-proof of $\mathsf{lear}^y_{1/2^{\ell n^\gamma}}(A, \mathsf{Circuit}[n^k], 1/2 - 1/2^{Kn^\gamma}, \delta)$ for a circuit A and a boolean function y represented by fixed bits in formula $\mathsf{lear}^y_{1/2^{\ell n^\gamma}}(\cdot, \cdot, \cdot, \cdot)$. Further, assume that the probability that A on queries to f outputs a circuit D such that $\Pr[D(x) = f(x)] \geq 1/2 + 1/2^{Kn^\gamma}$ is $< \delta$, where the outermost probability is counted approximately with error $1/2^{\ell n^\gamma}$ using PV-function Sz and the corresponding assumptions $\mathsf{LB}_{\mathsf{tt}}$ expressing hardness of y, i.e. using formulas $\Pr^y[\cdot]_{1/2^{\ell n^\gamma}}$ for the same y as above - we treat y as a free variable here. Then there is a $poly(|\pi|)$-size P-proof of $\mathsf{tt}(f, n^k)$ or y does not satisfy the assumptions of $\Pr^y[\cdot]_{1/2^{\ell n^\gamma}}$.

To see that the claim holds, we reason in APC_1 as follows. Assume π is a P-proof of $\mathsf{lear}^y_{1/2^{\ell n^\gamma}}(A, \mathsf{Circuit}[n^k], 1/2 - 1/2^{Kn^\gamma}, \delta)$ but A on queries to f outputs a circuit $(1/2 + 1/2^{Kn^\gamma})$-approximating f with probability $< \delta$. Then, either y does not satisfy the assumptions of $\Pr^y[\cdot]_{1/2^{\ell n^\gamma}}$ or there is a trivial $2^{O(n)}$-size P-proof of $\neg\mathsf{tt}(f, n^k) \to \neg R(b)$, for predicate R from the definition of $\mathsf{lear}^y_{1/2^{\ell n^\gamma}}(A, \mathsf{Circuit}[n^k], 1/2 - 1/2^{Kn^\gamma}, \delta)$ and a complete assignment b. The P-proof is obtained by evaluating function Sz which counts the confidence of A - note that functions f, y and algorithm A are represented inside P by fixed bits so the P-proof just evaluates a $2^{O(n)}$-size circuit on some input, which is possible by Lemma 17, Item 2. (We use here also the fact that APC_1 knows that the probability statement expressed by function Sz translates to $\neg R$ in the negation normal form.) The formula $\neg\mathsf{tt}(f, n^k) \to \neg R(b)$ is obtained from $\neg R(b)$ by an instantiation of a single Frege rule, which is available by Lemma 17, Item 1. Applying again Lemma 17, Item 1, from a P-proof of $\mathsf{lear}^y_{1/2^{\ell n^\gamma}}(A, \mathsf{Circuit}[n^k], 1/2 - 1/2^{Kn^\gamma}, \delta)$ and a P-proof of $\neg\mathsf{tt}(f, n^k) \to \neg R(b)$, we construct a $poly(|\pi|)$-size P-proof of $\mathsf{tt}(f, n^k)$. This proves the claim.

Next, observe that APC_1 proves that "If for a sufficiently big n and $\ell \geq K + 1$ the probability that a circuit A on queries to f outputs a circuit $(1/2 + 1/2^{Kn^\gamma})$-approximating f is $\geq 1/2^{Kn^\gamma}$, where the probability is counted approximately with error $1/2^{\ell n^\gamma}$ using PV-function Sz and the corresponding assumptions $\mathsf{LB}_{\mathsf{tt}}$, then there is a circuit of size $|A|$ $(1/2 + 1/2^{Kn^\gamma})$-approximating f or y does not satisfy the assumptions of $\Pr^y[\cdot]_{1/2^{\ell n^\gamma}}$." This is because, if such a circuit did not exist, a trivial surjection would witness that 2^m times the probability that A outputs a circuit $(1/2 + 1/2^{Kn^\gamma})$-approximating f, counted approximately with error $1/2^{\ell n^\gamma}$ using function Sz, is $\preceq_{1/2^{\ell n^\gamma}} 0$. Here, 2^m is the domain of the surjection. By Proposition 11 1.*ii*), this would imply $2^m/2^{Kn^\gamma} < 2^{m+1}/2^{\ell n^\gamma}$, which is a contradiction for $\ell \geq K + 1$ and sufficiently big n.

Therefore, Claim 19 implies that APC_1 proves that "For sufficiently big n and $\ell \geq K+1$, if π is a P-proof of $\mathsf{lear}^{y}_{1/2^{\ell n^\gamma}}(A, \mathsf{Circuit}[n^k], 1/2 - 1/2^{Kn^\gamma}, 1/2^{Kn^\gamma})$ for circuits A of size 2^{Kn^γ}, then there is a P-proof of $\mathsf{tt}(f, n^k)$ or there is a 2^{Kn^γ}-size circuit $(1/2 + 1/2^{Kn^\gamma})$-approximating f or there is a $2^{||y||/4}$-size circuit $(1/2+1/2^{||y||/4})$-approximating y or $||y|| \leq n_0$ or $||y|| \neq S(\cdot, 2^m, 2^{2^{\ell n^\gamma}})$;" for n_0 from Definition 14. Since this is a Σ^b_1-statement, by Lemma 12, PV_1 proves the same statement with the existential quantifiers witnessed by PV-functions assuming they are given a boolean function h' which is hard for circuits of size $2^{||h'||/4}$, for sufficiently big $|h'|$.

The last statement provable in PV_1 is Π^b_1 so we can translate it to EF. This gives us $poly(|\pi|, 2^n)$-size circuits B_0 such that for sufficiently big n, EF proves efficiently

"If $\ell \geq K+1$,
h' is not computable by a particular circuit of size $2^{||h'||/4}$, $|h'|$ is sufficiently big,
y is not $(1/2 + 1/2^{||y||/4})$-approximable by a particular circuit of size $2^{||y||/4}$, $||y|| > n_0$,
$||y|| = S(\cdot, 2^m, 2^{2^{\ell n^\gamma}})$
and π is a P-proof of $\mathsf{lear}^{y}_{1/2^{\ell n^\gamma}}(A, \mathsf{Circuit}[n^k], 1/2 - 1/2^{Kn^\gamma}, 1/2^{Kn^\gamma})$ for 2^{Kn^γ}-size A,
then B_0 (given π, h' and formula $\mathsf{tt}(f, n^k)$) outputs a P-proof of $\mathsf{tt}(f, n^k)$
or B_0 outputs a 2^{Kn^γ}-size circuit $(1/2 + 1/2^{Kn^\gamma})$-approximating f.".[9]

If we now assume that P proves efficiently $\mathsf{tt}(h_n, 2^{n/4}, 1/2 - 1/2^{n/4})$ and that Item 1 holds, then by Definition 16, Items 1-3, for each k, there are p-size circuits B_1 such that for each sufficiently big n, P proves efficiently "B_1 (given just formula $\mathsf{tt}(f, n^k)$) outputs a P-proof of $\mathsf{tt}(f, n^k)$ or B_1 outputs a 2^{Kn^γ}-size circuit $(1/2 + 1/2^{Kn^\gamma})$-approximating f." (We use here also the fact that PV_1 knows that $S(\cdot, 2^m, 2^{2^{\ell n^\gamma}})$ depends just on n.) Consequently, since P proves efficiently its own reflection, for each sufficiently big n, P proves efficiently that "if π is a P-proof of $\mathsf{tt}(f, 2^{Kn^\gamma}, 1/2 - 1/2^{Kn^\gamma})$ then B_1 outputs a P-proof of $\mathsf{tt}(f, n^k)$".[10] Finally, we make the P-proofs work for all n by increasing the size of B_1 by a constant. This finishes the proof of case $(1. \to 2.)$.

$(\mathbf{2.} \to \mathbf{1.})$ The opposite implication can be obtained from Lemma 20 and 21 which formalize Theorem 4.

▶ **Lemma 20.** *For each $d \geq 2$, each $k \geq 10d$ and each sufficiently big c, there is a PV-function L such that for each PV-function B the theory APC_1 proves: Assume the reflection principle for P holds, π is a P-proof of*

$$\mathsf{tt}(h_n \oplus g, 2^{Kn^\gamma}, 1/2 - 1/2^{Kn^\gamma}) \vee \mathsf{tt}(g, 2^{Kn^\gamma}), \tag{1}$$

where g is represented by free variables, and that B automates P on Φ up to size $|\pi|^c$. Then, for prime $n^d \leq p \leq 2n^d$, where $2^{n^d} \in Log$, for $\delta^{-1} \in Log$ such that $\delta < 1/N^3 = 2^{3n}$, $L(B, \pi, p)$ is a $poly(2^n, |\pi|)$-size circuit learning circuits with $m = n^d$ inputs and size $m^{k/10d}$,

[9] Formally, the statement "If a particular assignment a satisfies formula ϕ, then formula ψ holds" means that "If a is the output of a computation of a specific circuit W (where W is allowed to use as inputs atoms from ψ, but other atoms of W do not appear in ψ), and a satisfies ϕ, then ψ". By Lemma 17, Item 3, if we assume that the statement is efficiently provable in P and that P proves efficiently ϕ, then P proves efficiently ψ. Note also that for $A, B \in \Sigma^b_0$, the translation $||A \to B||$ is $\neg||\neg A|| \to ||B||$, which might not be the same formula as $||A|| \to ||B||$. Nevertheless, EF proves efficiently that $E \to (||A|| \leftrightarrow \neg||\neg A||)$, where E postulates that auxiliary variables of $||A||$ encode the computation of a suitable circuit. Therefore, in systems like EF or P, if we have a proof of $||A||$ and $||A \to B||$, we can remove the assumption E after proving $E \to ||B||$, assuming "non-input" variables of E do not occur in $||B||$, and ignore the difference between $||A||$ and $\neg||\neg A||$.

[10] It is assumed that the encoding of the statement coincides with the encoding of aut_P.

with confidence $1/N^4$*, up to error* $1/2 - 1/N^3$*, where the confidence is counted approximately with error* δ *using* PV*-function* Sz *and the corresponding assumptions* $\mathsf{LB}_{\mathsf{tt}}$ *expressing hardness of a boolean function* y*, i.e. using formulas* $\Pr^y[\cdot]_\delta$*.*

▶ **Lemma 21** ("XOR trick")**.** PV_1 *proves that for all boolean functions* g, h'' *with* n *inputs, for sufficiently big* n*,* $\mathsf{LB}_{\mathsf{tt}}{}'(h'', 3 \cdot 2^{Kn^\gamma}, 2^n(1/2 - 1/2^{Kn^\gamma}))$ *implies* $\mathsf{LB}_{\mathsf{tt}}{}'(h'' \oplus g, 2^{Kn^\gamma}, 2^n(1/2 - 1/2^{Kn^\gamma})) \vee \mathsf{LB}_{\mathsf{tt}}{}'(g, 2^{Kn^\gamma})$*, where* $\mathsf{LB}_{\mathsf{tt}}{}'$ *is obtained from* $\mathsf{LB}_{\mathsf{tt}}$ *by setting* $n_0 = 0$ *and skipping the universal quantifier on* n*, i.e. all formulas* $\mathsf{LB}_{\mathsf{tt}}{}'$ *refer to the same* n*.*

The proof of Lemma 21 is almost immediate: By Σ^b_1-induction, a 2^{Kn^γ}-size circuit C_1 computing g and a 2^{Kn^γ}-size circuit C_2 $(1/2+1/2^{Kn^\gamma})$-approximating $h'' \oplus g$ can be combined into a circuit $C_1 \oplus C_2$ of size $3 \cdot 2^{Kn^\gamma}$ which $(1/2 + 1/2^{Kn^\gamma})$-approximates h''.

The implication (2. $\to$ 1.) can be derived from Lemma 20 and 21 as follows. Since the APC_1-provable statement from Lemma 20 is Σ^b_1, similarly as above, we can witness it and translate to EF at the expense of introducing an additional assumption about the hardness of a boolean function h'. That is, for each p-size circuit B there are $poly(|\pi|, 2^{n^d})$-size circuits A and $poly(|\pi|, 2^{n^d})$-size EF-proofs of

"If the reflection principle for P is satisfied by a particular assignment,
π is a P-proof of (1),
h' is not computable by a particular circuit of size $2^{||h'||/4}$, $|h'|$ is sufficiently big,
y is not $(1/2 + 1/2^{||y||/4})$-approximable by a particular circuit of size $2^{||y||/4}$, $||y|| > n_0$,
$||y|| = S(\cdot, \cdot, 2^{|\delta^{-1}|})$
and $n^d \le p \le 2n^d$ is a prime,
then, for $\delta < 1/N^3$, $\mathsf{lear}^y_\delta(L(B, \pi, p), \mathsf{Circuit}(m^{k/10d}), 1/2 - 1/N^3, 1/N^4)$
or $A(B, \pi, h')$ outputs a falsifying assignment of $\mathsf{aut}_P(B, \Phi, |\pi|^c)$.".

Analogously, PV_1-proof from Lemma 21 yields p-size EF-proofs of the implication "$\mathsf{tt}(h_n, 3 \cdot 2^{Kn^\gamma}, 1/2 - 1/2^{Kn^\gamma})$ is falsified by a particular assignment or (1) holds". By the assumption of the theorem, there are p-size P-proofs of $\mathsf{tt}(h_n, 3 \cdot 2^{Kn^\gamma}, 1/2 - 1/2^{Kn^\gamma})$ for sufficiently big n. Hence, by Definition 16, Items 1-3, there are p-size P-proofs of (1) for sufficiently big n. As P proves efficiently also its own reflection, this yields $poly(2^{n^d})$-size P-proofs of

"If h' is not computable by a particular circuit of size $2^{||h'||/4}$, $|h'|$ is sufficiently big,
y is not $(1/2 + 1/2^{||y||/4})$-approximable by a particular circuit of size $2^{||y||/4}$, $||y|| > n_0$,
$||y|| = S(\cdot, \cdot, 2^{|\delta^{-1}|})$
and $n^d \le p \le 2n^d$ is a prime,
then, for $\delta < 1/N^3$, $\mathsf{lear}^y_\delta(L(B, \pi, p), \mathsf{Circuit}(m^{k/10d}), 1/2 - 1/N^3, 1/N^4)$
or $A(B, \pi, h')$ outputs a falsifying assignment of $\mathsf{aut}_P(B, \Phi, |\pi|^c)$.".

By Bertrand's postulate there is a prime $n^d \le p \le 2n^d$, so EF proves that p is a prime by a trivial $2^{O(n^d)}$-size proof which verifies all possible divisors. Therefore, choosing $d > 1/\gamma$, Item 2 and p-size P-proofs of $\mathsf{tt}(h_n, 2^{n/4}, 1/2 - 1/2^{n/4})$ imply Item 1.

It remains to prove Lemma 20.

Suppose π is a P-proof of (1). Assuming that B automates P on Φ, we want to obtain a $\mathsf{P/poly}$-natural property useful against $\mathsf{Circuit}[n^k]$. To do so, observe (first, without formalizing it in APC_1) that for each g, B can be used to find a proof of $\mathsf{tt}(h_n \oplus g, n^k)$ or to recognize that $\mathsf{tt}(g, 2^{Kn^\gamma})$ holds - if $\mathsf{tt}(g, 2^{Kn^\gamma})$ was falsifiable, there would exist a $poly(|\pi|)$-size P-proof of $\mathsf{tt}(h_n \oplus g, 2^{Kn^\gamma}, 1/2 - 1/2^{Kn^\gamma})$ obtained by substituting the falsifying assignment to the proof of (1) and thus B would find a short proof of $\mathsf{tt}(h_n \oplus g, n^k)$, for

sufficiently big c. Since for random g, both $h_n \oplus g$ and g are random functions, we know that with probability $\geq 1/2$ B finds a proof of $\mathsf{tt}(h_n \oplus g, n^k)$ or with probability $\geq 1/2$ it recognizes that $\mathsf{tt}(g, 2^{Kn^\gamma})$ holds. In both cases, B yields a $\mathsf{P/poly}$-natural property useful against $\mathsf{Circuit}[n^k]$.

Let us formalize reasoning from the previous paragraph in $\mathsf{APC_1}$. Let $N = 2^n$ and B' be the algorithm which uses B to search for P-proofs of $\mathsf{tt}(h_n \oplus g, n^k)$ or to recognize that $\mathsf{tt}(g, 2^{Kn^\gamma})$ holds. B' uses π to know how long it needs to run B. Assume for the sake of contradiction that

$$G_0 := \{g \oplus h_n \mid B'(g) \text{ outputs a } P\text{-proof of } \mathsf{tt}(h_n \oplus g, n^k)\} \preceq_0 2^N/3$$

$$G_1 := \{g \mid B'(g) \text{ recognizes that } \mathsf{tt}(g, 2^{Kn^\gamma}) \text{ holds}\} \preceq_0 2^N/3.$$

It is easy to construct a surjection S witnessing that $2^N \preceq_0 G_0 \cup G_1$: S maps $g \in G_1$ to g and $g \in G_0$ to $g \oplus h_n$. Following the argument above we conclude that S is a surjection: for each g, either $g \in G_1$ (and $S(g) = g$) or $g \oplus h_n \in G_0$ (and $S(g \oplus h_n) = g$). Here, we use the assumption that $\mathsf{APC_1}$ knows that P admits the substitution property and simulates Frege rules. Thus, by Proposition 10 *iv*), $2^N \preceq_0 2 \cdot 2^N/3$, which yields a contradiction by Proposition 11 1.*ii*). Consequently, by Proposition 11 1.*i*), $G_0 \succeq_\delta 2^N/3$ or $G_1 \succeq_\delta 2^N/3$ for $\delta^{-1} \in \mathit{Log}$. Since $g \in G_0$ and $g \in G_1$ are decidable by p-size circuits and we assume the reflection principle for P (which implies that G_0 is useful), this means that either G_0 or G_1 defines a $\mathsf{P/poly}$-natural property useful against $\mathsf{Circuit}[n^k]$.

Finally, by the $\mathsf{APC_1}$-formalization of [8], Theorem 15, we obtain $\mathit{poly}(2^n, |\pi|)$-size circuit $L(B, \pi, p)$ learning circuits with $m = n^d$ inputs and size $n^{k/10}$, over the uniform distribution, with membership queries, confidence $1/N^4$, up to error $1/2 - 1/N^3$. ◀

▶ **Corollary 22.** *Assume there is a* $\mathsf{NE} \cap \mathsf{coNE}$*-function* $h_n : \{0,1\}^n \mapsto \{0,1\}$ *such that for each sufficiently big* n*,* h_n *is not* $(1/2 + 1/2^{n/4})$*-approximable by* $2^{n/4}$*-size circuits. Then there is a proof system* P *(which can be described explicitly*[11] *given the definition of* h_n*) such that for each constant* K *and* $\gamma < 1$*, Items 1 and 2 from Theorem 18 are equivalent. Moreover, the equivalence holds for each* $\mathsf{APC_1}$*-decent system simulating* P*.*

Proof. See the arXiv version of the paper. ◀

References

1 M. Alekhnovich, M. Braverman, V. Feldman, A. R. Klivans, and T. Pitassi. Learnability and automatizability. *IEEE Symposium on Foundations of Computer Science*, 2004.

2 B. Applebaum, B. Barak, and D. Xiao. On basing lower bounds for learning on worst-case assumptions. *IEEE Symposium on Foundations of Computer Science*, 2008.

3 A. Atserias and M. Müller. Automating resolution is NP-hard. *IEEE Symposium on Foundations of Computer Science*, 2019.

4 E. Binnendyk, M. Carmosino, A. Kolokolova, R. Ramyaa, and M. Sabin. Learning with distributional inverters. *Algorithmic Learning Theory*, 2022.

5 M.L. Bonet, C. Domingo, R. Gavaldá, A. Maciel, and T. Pitassi. Non-automatizability of bounded-depth Frege proofs. *Computational Complexity*, 13(1–2):47–68, 2004.

6 M.L. Bonet, T. Pitassi, and R. Raz. On interpolation and automatization for Frege proof systems. *SIAM Journal of Computing*, 29(6):1939–1967, 2000.

7 S. Buss. *Bounded arithmetic*. Bibliopolis, 1986.

[11] More formally, there is a p-time algorithm R such that given predicates H_0, H_1 defining h_n (see the proof of Corollary 22), R outputs a p-time algorithm defining system P.

8 M. Carmosino, R. Impagliazzo, V. Kabanets, and A. Kolokolova. Learning algorithms from natural proofs. *IEEE Symposium on Computational Complexity*, 2016.

9 A. Cobham. The intrinsic computational difficulty of functions. *Proceedings of the 2nd International Congress of Logic, Methodology and Philosophy of Science*, 1965.

10 S.A. Cook. Feasibly constructive proofs and the propositional calculus. *ACM Symposium on Theory of Computing*, 1975.

11 S.A. Cook and N. Thapen. The strength of replacement in weak arithmetic. *ACM Transactions on Computational Logic*, 7(4):749–764, 2006.

12 S. de Rezende, M. Göös, J. Nördstrom, T. Pitassi, R. Robere, and D. Sokolov. Automating algebraic proof systems is NP-hard. *IEEE Symposium on Computational Complexity*, 2020.

13 M. Garlík. Failure of feasible disjunction property for k-DNF resolution and NP-hardness of automating it. *Electronic Colloquium on Computational Complexity*, 2020.

14 M. Göös, S. Koroth, I. Mertz, and T. Pitassi. Automating cutting planes is NP-hard. *ACM Symposium on Theory of Computing*, 2020.

15 E. Jeřábek. Dual weak pigeonhole principle, Boolean complexity and derandomization. *Annals of Pure and Applied Logic*, 129:1–37, 2004.

16 E. Jeřábek. Weak pigeonhole principle and randomized computation. *Ph.D. thesis, Charles University in Prague*, 2005.

17 E. Jeřábek. Approximate counting in bounded arithmetic. *Journal of Symbolic Logic*, 72:959–993, 2007.

18 J. Krajíček. *Bounded arithmetic, propositional logic, and complexity theory.* Cambridge University Press, 1995.

19 J. Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 66(2):457–486, 1997.

20 J. Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1–3):123–140, 2001.

21 J. Krajíček. *Forcing with random variables and proof complexity.* Cambridge University Press, 2011.

22 J. Krajíček. *Proof complexity.* Cambridge University Press, 2019.

23 J. Krajíček and P. Pudlák. Some consequences of cryptographical conjectures for S_2^1 and EF. *Information and Computation*, 140(1):82–94, 1998.

24 J. Krajíček, P. Pudlák, and G. Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52:143–153, 1991.

25 M. Müller and J. Pich. Feasibly constructive proofs of succinct weak circuit lower bounds. *Annals of Pure and Applied Logic*, 2019.

26 I.C. Oliveira and R. Santhanam. Conspiracies between learning algorithms, circuit lower bounds, and pseudorandomness. *IEEE Symposium on Computational Complexity*, 2017.

27 J. Pich. Learning algorithms from circuit lower bounds. *preprint*, 2020.

28 A.A. Razborov. On provably disjoint NP pairs. *BRICS*, 1994.

29 A.A. Razborov. Bounded arithmetic and lower bounds in boolean complexity. *Feasible Mathematics II*, pages 344–386, 1995.

30 A.A. Razborov. Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic. *Izvestiya of the Russian Academy of Science*, 59:201–224, 1995.

31 A.A. Razborov. Pseudorandom generators hard for k-DNF Resolution and Polynomial Calculus. *Annals of Mathematics*, 181(2):415–472, 2015.

32 A.A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

33 R. Santhanam. Pseudorandomness and the Minimum Circuit Size Problem. *Innovations in Theoretical Computer Science*, 2020.

34 L. Valiant. A theory of the learnable. *Communications of the ACM*, 27, 1984.

35 R. Williams. Non-uniform ACC circuit lower bounds. *IEEE Symposium on Computational Complexity*, 2011.

Algorithms and Data Structures for First-Order Logic with Connectivity Under Vertex Failures

Michał Pilipczuk
University of Warsaw, Poland

Nicole Schirrmacher
Universität Bremen, Germany

Sebastian Siebertz
Universität Bremen, Germany

Szymon Toruńczyk
University of Warsaw, Poland

Alexandre Vigny
Universität Bremen, Germany

Abstract

We introduce a new data structure for answering connectivity queries in undirected graphs subject to batched vertex failures. Precisely, given any graph G and integer parameter k, we can in fixed-parameter time construct a data structure that can later be used to answer queries of the form: "are vertices s and t connected via a path that avoids vertices $u_1, \ldots, u_k$?" in time $2^{\mathcal{O}(k)}$. In the terminology of the literature on data structures, this gives the first deterministic data structure for connectivity under vertex failures where for every fixed number of failures, all operations can be performed in constant time.

With the aim to understand the power and the limitations of our new techniques, we prove an algorithmic meta theorem for the recently introduced *separator logic*, which extends first-order logic with atoms for connectivity under vertex failures. We prove that the model-checking problem for separator logic is fixed-parameter tractable on every class of graphs that exclude a fixed topological minor. We also show a weak converse. This implies that from the point of view of parameterized complexity, under standard complexity theoretical assumptions, the frontier of tractability of separator logic is almost exactly delimited by classes excluding a fixed topological minor.

The backbone of our proof relies on a decomposition theorem of Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [SICOMP '19], which provides a tree decomposition of a given graph into bags that are unbreakable. Crucially, unbreakability allows to reduce separator logic to plain first-order logic within each bag individually. Guided by this observation, we design our model-checking algorithm using dynamic programming over the tree decomposition, where the transition at each bag amounts to running a suitable model-checking subprocedure for plain first-order logic. This approach is robust enough to provide also an extension to efficient enumeration of answers to a query expressed in separator logic.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Finite Model Theory; Mathematics of computing → Graph algorithms

Keywords and phrases Combinatorics and graph theory, Computational applications of logic, Data structures, Fixed-parameter algorithms and complexity, Graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.102

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2111.03725

Funding This paper is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 948057) and part of the French-German Collaboration ANR/DFG Project UTMA supported by the German Research Foundation (DFG) through grant agreement No 446200270.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 102; pp. 102:1–102:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We thank Ismaël Jecker, Pierre Ohlmann and Wojciech Przybyszewski for useful discussions. In particular, Wojciech Przybyszewski showed how to improve the dependency on k in the running time from double exponential to single exponential in Theorem 1.1 (see Theorem 2.4).

1 Introduction

1.1 Connectivity under vertex failures

In many applications, we do not need to answer a query once but rather need to repeatedly answer queries over dynamically changing data. A prime example are databases, where a database is repeatedly modified and queried by the users. In most cases, the modifications to the database can be expected to be small compared to its size, which suggests that in a preprocessing step, we can set up a data structure that is efficiently updated after each modification and that allows for efficient querying. In 1997, Frigioni and Italiano introduced the *dynamic subgraph model* [22], which allows to conveniently model such dynamic situations in the particular case of connectivity in network infrastructures that are subject to node or link failures. This model, which has been studied intensively in the data structures community under the name of *connectivity oracles under vertex failures*, is as follows. We are given a graph G and an integer k. We imagine that G is a network in which at every moment, at most k vertices (or edges) are inactive (are subject to *failures*). The goal is to construct a data structure that supports the following two operations. First, one can *update* G by resetting the set of failed vertices to a given set of size at most k. Second, one can *query* G by asking, for a given pair of vertices s and t, whether s and t can be connected by a path that avoids the failed vertices.

Following the introduction of the problem by Frigioni and Italiano [22] and the more general problem of batched vertex or edge failures by Pătraşcu and Thorup [37][1], there has been a long line of work on data structures for connectivity oracles under vertex and edge failures. We refer to a remarkably comprehensive literature overview in the work of Duan and Pettie [17], which also provides the currently best bounds for the problem as far as combinatorial and deterministic data structures are concerned. Their data structure can be initialized in time $\mathcal{O}(|G|\|G\| \cdot \log|G|)$, takes $\mathcal{O}(k\|G\| \log|G|)$ space, and supports updates in $\mathcal{O}(k^3 \log^3|G|)$ time and queries in $\mathcal{O}(k)$ time. Here $|G|$ is the vertex count of G and $\|G\|$ is the joint number of edges and vertices in G. These bounds can be slightly improved at the cost of allowing randomization, however, the polylogarithmic dependency on the size of the graph in the update time persists. As noted in [17], from known results it is possible to derive data structures with constant time complexity of operations for $k \leqslant 3$ (or $k \leqslant 4$ for edge failures), but, citing their words, "scaling these solutions up, even to an arbitrarily large constant k, becomes prohibitively complex, even in the simpler case of edge failures."

Recently, van den Brand and Saranurak [42] proposed a very different approach to the problem, using which they obtained a randomized data structure that can be initialized in time $\mathcal{O}(|G|^\omega)$, takes $\mathcal{O}(|G|^2 \log|G|)$ space, and supports updates in $\mathcal{O}(k^\omega)$ time and queries in $\mathcal{O}(k^2)$ time (where ω is the exponent for the boolean matrix multiplication).

In particular, the update and the query time are constant for constant k; to the best of our knowledge, this is the only data structure that has this property known so far. The approach is algebraic and, simplifying it substantially, boils down to storing a matrix of

[1] Strictly speaking, in [37] Pătraşcu and Thorup considered only edge failures. To the best of our knowledge, (batched) vertex failures on general graphs were first investigated by Duan and Pettie [16].

counts of walks between pairs of vertices and extracting answers to queries using algebraic operations on those counts. To manipulate the counts efficiently, one needs to work on their short hashes, for instance in the form of elements from a fixed finite field. This introduces randomization and avoiding it within this methodology seems difficult. We remark that the data structure of van den Brand and Saranurak [42] works even for the more general problem of reachability in directed graphs under vertex failures, and can handle arc insertions.

As the first main contribution of this paper, we prove the following theorem. Note that querying whether two vertices are connected by a path avoiding failed vertices is equivalent to an update followed by a connectivity query in the terminology of previous works [42, 16, 17, 37].

▶ **Theorem 1.1.** *Given a graph G and an integer k, one can in time $2^{2^{\mathcal{O}(k)}} \cdot |G|^2\|G\|$ construct a data structure that may answer the following queries in time $2^{\mathcal{O}(k)}$: given $s,t \in V(G)$ and k vertices $u_1,\ldots,u_k$ of G, are s and t connected in G by a path that avoids the vertices $u_1,\ldots,u_k$? The space usage of the data structure is $2^{2^{\mathcal{O}(k)}} \cdot \|G\|$.*

Note that in Theorem 1.1, for every fixed k, all operations are supported in constant time (which is exponential in k). Also, the data structure is entirely deterministic and purely combinatorial. To the best of our knowledge, this is the first deterministic (non-randomized) data structure with constant time queries. Note also that for a fixed k, the space usage of our data structure is linear in the size of the graph, as opposed to the quadratic dependency in the result of van den Brand and Saranurak [42].

We remark that in the first version of this paper [36], the query time in Theorem 1.1 is stated as $2^{2^{\mathcal{O}(k)}}$ instead of $2^{\mathcal{O}(k)}$. The improved query time is due to an improvement in one of the ingredients of our proof (see Theorem 2.4 below), provided by Wojciech Przybyszewski.

The doubly-exponential dependencies on k may seem high, however we also derive two variants of our data structure that achieve the following tradeoffs:

- The space usage and the query time can be reduced to $2^{\mathcal{O}(k^2)} \cdot \|G\|$ and $k^{\mathcal{O}(1)}$, respectively, while the construction time becomes $2^{\mathcal{O}(k^2)} \cdot |G|^{\mathcal{O}(1)}$.
- The space usage and the query time can be replaced with $k^{\mathcal{O}(1)} \cdot |G|^2$ and $k^{\mathcal{O}(1)}$, respectively, while the construction time becomes $2^{\mathcal{O}(k \log k)} \cdot |G|^{\mathcal{O}(1)}$.

Note the first statement offers both more efficient queries (the query time is even polynomial in k), and smaller space usage. The slight drawback is that the polynomial factor in the construction time becomes unspecified, but it is still of the form $|G|^{\mathcal{O}(1)}$. In the second statement, the space usage becomes polynomial in k at the expense of making it quadratic in $|G|$. The tradeoffs are obtained by a non-trivial replacement of particular components in the proof of Theorem 1.1; see the discussion in Section 6 of the full version. We consider the statement provided in Theorem 1.1 to be the cleanest formulation, hence we put a primary focus on it.

As for the proof of Theorem 1.1, our approach is completely new compared to the previous approaches [42, 16, 17, 37]. Our key combinatorial observation is that connectivity queries over a constant number k of vertex failures can be evaluated in constant time provided the underlying graph G is sufficiently well-connected. The requirement on the well-connectedness of G depends on the number k. Obviously, there is no guarantee that the input graph G satisfies this property. We therefore use a decomposition theorem of Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [15] that (roughly) states the following (see Theorem 2.1). For every graph G and fixed number k, there is a tree decomposition where every intersection of adjacent bags has bounded size and every bag is well-connected for parameter k. Moreover, such a tree decomposition can be computed in time $2^{\mathcal{O}(k^2)}|G|^2\|G\|$. The data structure of

Theorem 1.1 is constructed around the tree decomposition $\mathcal{T}$ provided by the theorem of Cygan et al. [15]. Intuitively speaking, whether s and t are connected via a path that avoids the vertices $u_1, \ldots, u_k$ can be decided using dynamic programming over $\mathcal{T}$. We employ known techniques developed for answering queries on trees in constant time, for instance a deterministic variant of Simon's factorization [12], to be able to compute the outcome of this dynamic programming efficiently for any query given on input.

We provide a more detailed overview of the proof of Theorem 1.1 in Section 2. All proofs are provided in the full version [36] (numbers referring to `https://arxiv.org/abs/2111.03725v1`).

1.2 Algorithmic meta theorems

Once an algorithmic technique has been applied to solve a specific problem, it is very desirable to understand the power and the limitations of that technique, that is, to understand which other problems can be solved with that technique and which problems cannot. An approach to this very general goal is to prove an *algorithmic meta theorem*: a theorem that establishes tractability for a whole class of problems, possibly on certain restricted input instances. As logic allows to conveniently define classes of algorithmic problems, namely, the class of all problems that can be expressed in the logic under consideration, algorithmic meta theorems are often formulated as model-checking problems for a logic. In the *model-checking problem* for a logic $\mathcal{L}$ on a class of graphs $\mathscr{C}$ we are given a graph $G \in \mathscr{C}$ and a sentence $\varphi \in \mathcal{L}$, and the task is to decide whether φ holds in G. The archetypal result of this form is a result of Courcelle stating that every formula of monadic-second order logic (MSO_2) can be evaluated in linear time on all graphs whose treewidth is bounded by some fixed constant [13]. This result has been extended to various other logics $\mathcal{L}$ and graph classes $\mathscr{C}$.

This approach to algorithmic meta theorems can be made precise in the language of parameterized complexity, as follows. Say that $\mathcal{L}$ is *tractable* on $\mathscr{C}$ if the model-checking problem for $\mathscr{C}$ and $\mathcal{L}$ is *fixed-parameter tractable (fpt)*: it can be solved in time $f(\varphi) \cdot \|G\|^c$, for some computable function f and a constant c, both depending only on $\mathscr{C}$. Results about tractability of a logic $\mathcal{L}$ on a class $\mathscr{C}$ imply the tractability of a vast array of problems – namely all problems that can be expressed in the logic $\mathcal{L}$ – on a given class of graphs. For instance, if model-checking *first-order logic* (FO) is fpt on a class of graphs $\mathscr{C}$, then in particular the dominating set problem is fpt on $\mathscr{C}$, since the existence of a dominating set of size k can be expressed by a first-order sentence with $k+1$ quantifiers that range over the vertices of the graph. Similarly, the independent set problem is then also fpt on $\mathscr{C}$. On the other hand, if the more powerful logic MSO_2 is tractable on a class of graphs $\mathscr{C}$, then the 3-colorability problem, or the hamiltonicity problem are polynomial-time solvable on $\mathscr{C}$, since both those problems can be expressed using an MSO_2 sentence, whose quantifiers that range over *sets* of vertices and edges of the graph.

There has been a long line of work on fixed-parameter tractability of model-checking FO, as it is arguably the most fundamental logic. This culminated in the work of Grohe, Kreutzer and Siebertz [24], who proved that this problem is fixed-parameter tractable on every class that is *nowhere dense*. Those classes include for example the class of planar graphs, or every class with bounded maximum degree, or of bounded genus. Without going into details, nowhere denseness is a general notion of uniform sparseness in graphs, and it is broader than most well-studied concepts of sparseness considered in structural graph theory, such as excluding a fixed (topological) minor. As observed by Dvořák et al. [19], the result of Grohe et al. is tight in the following sense: whenever $\mathscr{C}$ is not nowhere dense and is closed under taking subgraphs, then FO model-checking on $\mathscr{C}$ is as hard as on general graphs, that

is, AW[⋆]-hard. This means that as far as subgraph-closed classes are concerned, sparsity – described formally through the notion of nowhere denseness – exactly delimits the area of tractability of FO and the limits of the locality method for FO model-checking.

The aforementioned classic theorem of Courcelle [13] states that the model-checking problem for MSO_2 is fixed-parameter tractable on a class $\mathscr{C}$ if $\mathscr{C}$ has bounded treewidth. The proof translates the given sentence into a suitable tree automaton working on a tree decomposition of the input graph; this approach brings a wide range of automata-based tools to the study of MSO_2 on graphs. Again, as shown by Kreutzer and Tazari [31] and Ganian et al. [23], under certain complexity assumptions this is (almost) the most one could hope for: on subgraph-closed classes whose treewidth is poly-logarithmically unbounded, model-checking of MSO_2 becomes intractable from the parameterized perspective. So for MSO_2, bounded treewidth (almost) delimits the frontier of tractability, at least for subgraph-closed classes, and automata-based techniques exactly explain what can be done algorithmically.

With the aim to provide a meta theorem that captures the essence of the techniques developed to answer connectivity under vertex failures, we search for a logic that can express these queries. It is easy to see that MSO_2 is strong enough, but in fact, MSO_2 is much stronger and as explained above, it is intractable beyond graphs of bounded treewidth. On the other hand, FO can only express local properties and in particular, it cannot even express the very simple query of whether two vertices are in the same connected component. This naturally leads to an extension of FO that may use connectivity under vertex failures as atomic formulas. Such a logic was very recently introduced independently by Bojańczyk [6] and by Schirrmacher et al. [39] under the name *separator logic*, and denoted $\mathsf{FO}+\mathsf{conn}$. This logic extends FO by predicates $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ (one for each $k \geqslant 0$) with the following semantics: if $s,t,u_1,\ldots,u_k$ are vertices of a graph G, then $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ holds if and only if there is a path that connects s with t and does not pass through any of the vertices $u_1,\ldots,u_k$. Thus, separator logic involves very basic connectivity queries that have a global character, unlike plain FO, which is local. As a consequence, separator logic can express many interesting algorithmic properties, such as acyclicity, k-connectivity, the feedback vertex set problem parameterized by solution size and many recently studied elimination distance problems [9]. On the other hand, while $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ predicates are expressible in MSO_2, it is no surprise that the expressive power of $\mathsf{FO}+\mathsf{conn}$ is strictly weaker than that of MSO_2. For example, MSO_2 can express bipartiteness, while $\mathsf{FO}+\mathsf{conn}$ cannot [39, Theorem 3.11]. Separator logic is also incomparable with the recently introduced compound logic [20]. While compound logic can express the atomic connectivity predicates under vertex failures, it is not closed under negation. On the other hand, compound logic can express the exclusion of minors, a property not expressible in separator logic.

As the second main result of the paper, we prove an algorithmic meta theorem for separator logic. We show that the frontier of tractability of $\mathsf{FO}+\mathsf{conn}$ is almost exactly delimited by classes of graphs that exclude a fixed topological minor. More precisely, we prove the following complementary results.

▶ **Theorem 1.2.** *Let $\mathscr{C}$ be a class of graphs that exclude a fixed graph as a topological minor. Then given $G \in \mathscr{C}$ and an $\mathsf{FO}+\mathsf{conn}$ sentence φ, one can decide whether φ holds in G in time $f(\varphi)\cdot\|G\|^3$, where f is a computable function depending on $\mathscr{C}$.*

Our result implies e.g. the recent result about elimination distance to bounded degree [3], but it is much more general as it unifies in particular all elimination distance problems to classes definable in separator logic on classes that exclude a topological minor. Note that it does not subsume the result of [27] as e.g. the elimination distance to bipartite graphs and vertex planarization cannot be expressed by separator logic.

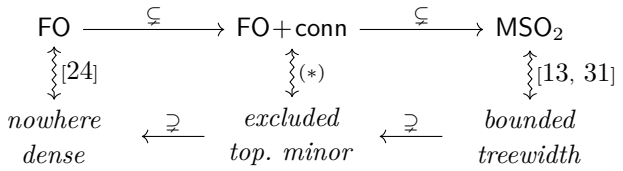


Figure 1 Correspondences between logics and properties of graph classes. An arrow $\mathcal{L} \rightsquigarrow \mathcal{P}$ between a logic $\mathcal{L}$ and a property $\mathcal{P}$ of graph classes denotes a correspondence: the logic $\mathcal{L}$ is tractable on a graph class $\mathscr{C}$ if and, under additional assumptions, only if, the class $\mathscr{C}$ has the property $\mathcal{P}$. Our result is the central correspondence $(*)$.

For the lower bound, we will need a technical assumption. We say that a class $\mathscr{C}$ admits *efficient encoding of topological minors* if for every graph H there exists $G \in \mathscr{C}$ such that H is a topological minor of G, and, given H, such G together with a suitable topological minor model can be computed in time polynomial in $|H|$. Note that in particular such G can be only polynomially larger than H.

▶ **Theorem 1.3.** *Let $\mathscr{C}$ be a subgraph-closed class of graphs that admits efficient encoding of topological minors. Then the model-checking of* $\mathsf{FO}{+}\mathsf{conn}$ *on $\mathscr{C}$ is* $\mathsf{AW}[\star]$*-hard, even for the fragment that uses only* conn_k *predicates with $k \leqslant 2$.*

On the one hand, these results show that separator logic is a natural logic whose expressive power lies strictly between FO and MSO_2, and which corresponds, in the sense described above, to a natural property of classes of graphs that lies between bounded treewidth and nowhere denseness, see Figure 1. On the other hand, the proof of Theorem 1.2 provides a general meta-explanation of the technique of dynamic programming over tree decompositions with unbreakable parts. In particular, the tractability result provided by Theorem 1.2 generalizes several recent algorithmic results for concrete problems expressible in separator logic [3, 10, 32]. However, there are some concrete problems where the general methodology non-trivially applies and which seem not to be captured by our meta-theorem [1, 27].

The proof of Theorem 1.3 is easy (see Section 3), so we focus on sketching the proof of Theorem 1.2, which is the main technical contribution of the second part of the paper. Let us fix an $\mathsf{FO}{+}\mathsf{conn}$ sentence φ, and let $k+2$ be the maximum arity of connectivity predicates appearing in φ. Again, the key observation is that conn_k predicates can be rewritten to plain FO provided the graph is well-connected, and the reason for this is the same combinatorial observations that underlies the proof of Theorem 1.1. Coming back to our dynamic programming on the tree decomposition, we choose to present it using a new framework based on automata, as this brings us closer to the classic understanding of logic on tree-decomposable graphs. Intuitively, the automaton processes a given tree in a bottom-up manner, but we assume that on the children of every node there is an additional structure of a graph. We say that such trees are *augmented* with graphs. When the automaton processes a node x, it chooses a transition based on an FO query executed on the graph on the children of x, where each child is labeled with the state computed for it before in the run. Then testing whether a formula $\varphi \in \mathsf{FO}{+}\mathsf{conn}$ holds on a graph G is reduced to deciding whether an automaton constructed from φ accepts such an *augmented tree*, constructed from the tree decomposition provided by the theorem of Cygan et al. [15]. The run of the automaton can be computed efficiently when the graphs augmenting the tree admit efficient FO model-checking, which is the case when the input graph excludes a fixed topological minor.

The benefit of this approach is that other questions related to the logic FO+conn can be reduced in the same way to questions about tree automata over augmented trees. We showcase this by proving that given an FO+conn query $\varphi(\bar{x})$ with free variables and a graph G from a fixed topological-minor-free class $\mathscr{C}$, the answers to $\varphi(\bar{x})$ on G can be enumerated with constant delay after fpt preprocessing (see Theorem 8.2 of the full version for a precise statement). Similarly, the formula $\varphi(\bar{x})$ can be queried in constant time after fpt preprocessing (see Theorem 8.1 of the full version). In general, we believe that the framework of automata over augmented trees may be of independent interest, as it seems to be a convenient model for understanding dynamic programming procedures over tree-decomposable structures.

Similar fpt query-answering and enumeration algorithms for plain FO have been obtained for classes with bounded expansion [30, 41], low degree [18], and nowhere dense classes [40] and for MSO_2 on trees and graphs of bounded treewidth [5, 29, 4]. Our general approach based on augmented trees allows us to generalize those results to trees that are augmented with graphs coming from a nowhere dense class, and to a certain logic combining the power of MSO on trees and FO on graphs.

Let us comment on the novelty of using the decomposition theorem of Cygan et al. [15]. The result of Cygan et al. [15] has been used several times for various graph problems [14, 15, 35, 33, 38]. Our application is the most similar to (and in fact, draws inspiration from) the work of Lokshtanov et al. [34], who proved the following statement: for every CMSO_2 sentence φ and $k \in \mathbb{N}$, model-checking φ on general graphs can be reduced to model-checking φ on (q,k)-unbreakable graphs, where q is a constant depending on φ and k. As FO+conn is subsumed by CMSO_2, this result can be almost applied to establish Theorem 1.2. The caveat is that the unbreakable graph output by the reduction needs to admit efficient FO model-checking so that the considered FO+conn sentence can be decided in fpt time after rewriting it to plain FO. In essence, we show that it is possible to guarantee this provided the input graph excludes a fixed topological minor. We remark that the work of Lokshtanov et al. [34] does not use the decomposition theorem of Cygan et al. [15] directly, but relies on its conceptual predecessor, the *recursive understanding* technique [11, 28].

Organization. In the extended abstract we give a detailed overview over the proofs of our main results: Theorem 1.1, Theorem 1.2, and Theorem 1.3. We provide all formal definitions and proofs in the full version [36]. The theorem numbers for references in the full version are provided in parenthesis.

2 Connectivity under vertex failures: overview of Theorem 1.1

In this section we provide an overview of the proofs of Theorem 1.1. This proof exposes all main ideas in a purely algorithmic setting. Then we discuss how the same methodology can be used for Theorem 1.2. In this introduction we assume familiarity with basic terminology of tree decompositions.

Unbreakability. As mentioned in Section 1, the central idea of this work is to tackle problems involving connectivity predicates using a decomposition into well-connected – or, more formally, *unbreakable* – parts. We first need to recall a few definitions.

A *separation* in a graph G is a pair (A,B) of vertex subsets such that $A \cup B = V(G)$ and there are no edges in G between $A - B$ and $B - A$. The *order* of the separation is the cardinality of the *separator* $A \cap B$. A vertex subset X is (q,k)-*unbreakable* in a graph G if for every separation (A,B) in G of order at most k in G, either $|A \cap X| \leqslant q$ or $|B \cap X| \leqslant q$.

So intuitively speaking, a separation of order k cannot break X in a balanced way: one of the sides must contain at most q vertices of X. For example, cliques and $k+1$-connected graphs are (k,k)-unbreakable, and square grids are $(\mathcal{O}(k^2),k)$-unbreakable.

The notion of unbreakability has been implicitly introduced in the context of parameterized algorithms by Kawarabayashi and Thorup in their work on the k-Way Cut problem [28]. This work has brought about the method of *recursive understanding*, which was then made explicit by Chitnis, Cygan, Hajiaghayi, Pilipczuk, and Pilipczuk in the technique of *randomized contractions* [11]. Intuitively, recursive understanding is a Divide&Conquer scheme using which one can reduce problems on general graphs to problems on suitably unbreakable graphs, provided certain technical conditions are satisfied. The technique was also applied in the context of model-checking: Lokshtanov, Ramanujan, Saurabh, and Zehavi [34] proved that the problem of deciding a property expressed in CMSO_2 in fpt time on general graphs can be reduced to the same problem on suitably unbreakable graphs. This result was used to provide algorithms for computing elimination distance to certain graph classes [2, 21, 26], which is very much related to separator logic (see [39, Example 3.5]).

In this work, we will not use recursive understanding or randomized contractions *per se*, but their conceptual successor: the decomposition into unbreakable parts. Precisely, the following theorem was proved by Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [15].

▶ **Theorem 2.1** ([15]). *There is a function $q(k) \in 2^{\mathcal{O}(k)}$ such that given a graph G and $k \in \mathbb{N}$, one can in time $2^{\mathcal{O}(k^2)} \cdot |G|^2\|G\|$ construct a (rooted) tree decomposition $\mathcal{T} = (T, \mathsf{bag})$ of G satisfying the following.*

- *All adhesions in $\mathcal{T}$ are of size at most $q(k)$; and*
- *every bag of $\mathcal{T}$, say at node x, is $(q(k),k)$-unbreakable in the subgraph of G induced by the union of bags at all descendants[2] of x.*

We remark that a different variant of Theorem 2.1 was later proved by Cygan, Komosa, Lokshtanov, Pilipczuk, Pilipczuk, and Wahlström [14]. This variant provides much stronger unbreakability guarantees – $q(k) = k$ – at the cost of relaxing the unbreakability to hold only in the whole graph G. See Section 3 of the full version for a discussion. The variant of [14] can be also used in our context and this leads to improving some quantitative bounds, however for simplicity, we focus on Theorem 2.1 in this overview.

A typical usage of Theorem 2.1 is to apply bottom-up dynamic programming on the obtained tree decomposition $\mathcal{T} = (T, \mathsf{bag})$. The subproblem for a node x of T corresponds to finding partial solutions in the subgraph of G induced by the union of bags at descendants of x, for every possible behavior of such a partial solution on the adhesion connecting x with its parent. That this adhesion has size at most $q(k)$ gives an upper bound on the number of different behaviors. To solve such a subproblem one needs to aggregate the solutions computed for the children of x by solving an auxiliary problem on the subgraph induced by the bag of x. In various problems of interest, the unbreakability of the bag becomes helpful in solving the auxiliary problem. This general methodology has been successfully used to give multiple fpt algorithms for cut problems [15, 14, 38], most prominently for Minimum Bisection [15]. More recent uses include a parameterized approximation scheme for the Min k-Cut problem [35] and a fixed-parameter algorithm for Graph Isomorphism parameterized by the size of the excluded minor [33]. On a high level, we apply the same methodology here.

[2] We follow the convention that every node is its own ancestor and descendant.

The case of totally unbreakable graphs. Let us come back to the problem of evaluating connectivity queries: we are given a graph G and after some preprocessing of G, we would like to be able to quickly evaluate for given vertices $s, t, u_1, \ldots, u_k$ whether $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. Consider first the following special case: the whole G is (q, k)-unbreakable for some parameter q (or more formally, $V(G)$ is (q, k)-unbreakable in G). The key observation is that then, whether $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds can be easily decided in time polynomial in q and k as follows.

Run a breadth-first search from s in $G - \{u_1, \ldots, u_k\}$, but terminate the search once $q+1$ different vertices have been reached. If the search reached t, then for sure $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. If the search finished before reaching $q+1$ different vertices and it did not reach t, then for sure $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ does not hold. In the remaining case, perform a symmetric procedure starting from t. The key observation is that if both breadth-first searches – from s and from t – got terminated after reaching $q+1$ different vertices, then the (q, k)-unbreakability of G implies that $\{u_1, \ldots, u_k\}$ do not separate s from t. So then $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. It is straightforward to implement this procedure in time polynomial in q and k given the standard encoding of G through adjacency lists.

In the general case, we cannot expect the given graph G to be (q, k)-unbreakable for any parameter q bounded in terms of k. However, we can use Theorem 2.1 to compute a tree decomposition $\mathcal{T}$ of G into parts that are (q, k)-unbreakable for $q \in 2^{\mathcal{O}(k)}$. The idea is that then, a query $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ can be evaluated by bottom-up dynamic programming on $\mathcal{T}$. By suitably precomputing enough auxiliary information about $\mathcal{T}$, this evaluation can be performed in time depending only on q and k.

Evaluating a query on the decomposition. Consider then the following setting: we are given the decomposition $\mathcal{T} = (T, \mathsf{bag})$ provided by Theorem 2.1, and for given $s, t, u_1, \ldots, u_k$ we would like to decide whether $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. So far let us not optimize the complexity: the goal is only to design a general algorithmic mechanism which will be later implemented efficiently using appropriate data structures.

For every node x of T, let $\mathsf{adh}(x)$ be the adhesion between x and its parent (or $\emptyset$ if x is the root), let $\mathsf{cone}(x)$ be the union of the bags at the descendants of x, and let $\mathsf{comp}(x) = \mathsf{cone}(x) - \mathsf{adh}(x)$. Call a node x *affected* if $\mathsf{comp}(x)$ contains a vertex of $\{s, t, u_1, \ldots, u_k\}$, and *unaffected* otherwise. Further, let $D(x) = \mathsf{adh}(x) \cup (\mathsf{comp}(x) \cap \{s, t\})$; note that $|D(x)| \leqslant q + 2$. Our goal is to compute the following *(connectivity) profile* for every node x of T:

$$\mathsf{profile}(x) = \left\{ \{a, b\} \in \binom{D(x)}{2} \;\middle|\; a \text{ and } b \text{ are connected in } G[\mathsf{cone}(x) - \{u_1, \ldots, u_k\}] \right\}.$$

We remark that the tree decomposition $\mathcal{T}$ can be chosen so that it satisfies the following basic connectivity property: for every node x and vertices $a, b \in \mathsf{adh}(x)$, there is a path connecting a and b whose all internal vertices belong to $\mathsf{comp}(x)$. Thus, for every unaffected node x, we have that $\mathsf{profile}(x) = \binom{D(x) - \{u_1, \ldots, u_k\}}{2}$.

It is easy to see that the profile of x is uniquely determined by the profiles of the children of x and the subgraph induced by the bag of x. The following lemma shows that this can be done efficiently.

▶ **Lemma 2.2** (informal statement, Lemma 4.4 in the full version). *Let x be a node of T. Suppose that for each affected child z of x we are given* $\mathsf{profile}(z)$*. Then, subject to suitable preprocessing of G, one can compute* $\mathsf{profile}(x)$ *in time polynomial in q.*

Note that every node has at most $k+2$ affected children, hence the input to the algorithm of Lemma 2.2 is of size polynomial in q.

Let us sketch the proof of Lemma 2.2. In essence, we apply the same strategy as the one from the case when the whole graph G is unbreakable. We define the *bag graph of* x, denoted $\mathsf{bgraph}(x)$, as the graph[3] obtained from $G[\mathsf{bag}(x)]$ by turning the adhesion $\mathsf{adh}(z)$ into a clique, for every child z of x. Note that $\mathsf{bgraph}(x)$ does not depend on the query, and thus can be precomputed upon initialization. As $\mathsf{bag}(x)$ is (q,k)-unbreakable in $G[\mathsf{cone}(x)]$, to test whether two vertices $a, b \in \mathsf{bag}(x)$ are connected in $G[\mathsf{cone}(x) - \{u_1, \ldots, u_k\}]$ it suffices to apply breadth-first searches from a and b in $\mathsf{bgraph}(x)$ that are terminated after reaching $q+1$ different vertices, except that these searches are forbidden to use vertices from $\{u_1, \ldots, u_k\} \cap \mathsf{bag}(x)$ as well as those edges originating from contracting the adhesion of affected children into cliques that are not in respective profiles. Provided $\mathsf{bgraph}(x)$ is precomputed and a list of affected children together with their profiles is given on input, it is easy to implement this algorithm so that it runs in time polynomial in q. Thus we can decide for every pair $\{a,b\} \in \binom{\mathsf{adh}(x)}{2}$ whether it should be included in $\mathsf{profile}(x)$. A similar reasoning can be applied to pairs in $\binom{D(x)}{2}$ that include s or t.

By applying Lemma 2.2 bottom-up, we can compute all the profiles in time $q^{\mathcal{O}(1)} \cdot |T| \leqslant q^{\mathcal{O}(1)} \cdot |G|$. Then we can read whether $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ holds by checking whether $\{s,t\}$ belongs to the profile of the root of T. Hence, the query $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ can be evaluated in time $q^{\mathcal{O}(1)} \cdot |G|$.

Data structure. Our final goal is to enrich the decomposition $\mathcal{T}$ with some additional information so that the mechanism presented above can be executed in time depending only on q.

For a vertex v of G, let $\mathsf{top}(v)$ be the unique top-most node of T whose bag contains v. Let $\widehat{S} := \{s,t,u_1,\ldots,u_k\}$ and let $X = \{\mathsf{top}(v) \colon v \in \widehat{S}\}$.

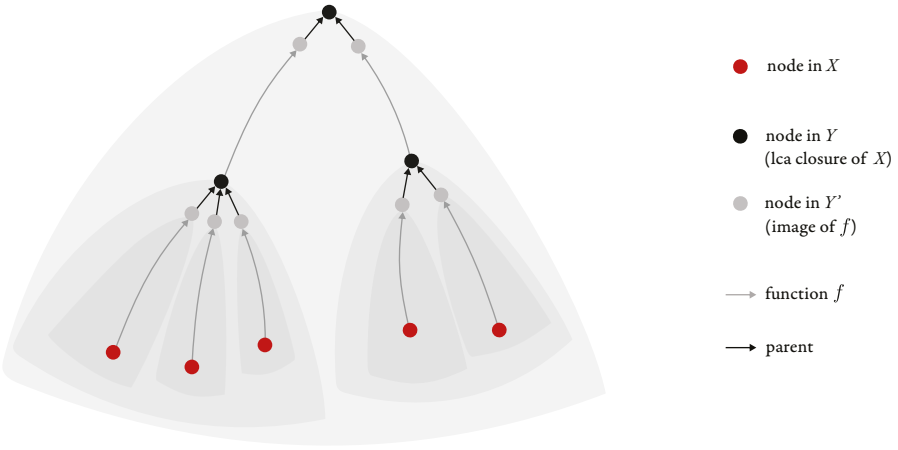


Figure 2 The sets X, Y and Y'. Only the nodes from $Y \cup Y'$ are marked in the figure, wheras the shaded areas indicate subtrees consisting of nodes which are not marked in the figure. Every affected node lies on a path from some $y \in Y$ to $f(y) \in Y'$.

[3] The bag graph $\mathsf{bgraph}(x)$ is actually defined slightly differently for technical reasons, as the graph obtained from $G[\mathsf{cone}(x)]$ by contracting, for each $y \in \mathsf{children}(x)$, the set $\mathsf{comp}(y)$ into a single vertex, identified with y. See the full version for the details.

Note that $|X| \leqslant |\widehat{S}| \leqslant k+2$ and that a node of T is affected if and only if it is an ancestor of a node from X. Let now Y be the *lowest common ancestor closure* of X: the set comprising X and all lowest common ancestors of pairs of vertices of X. It is well-known that then $|Y| \leqslant 2|X| - 1 \leqslant 2k+3$. We also add the root of T to Y in case it is not already present; thus $|Y| \leqslant 2k+4$. Observe that if for every vertex v we store a pointer to $\mathsf{top}(v)$, and we set up the data structure for lowest common ancestor queries of Harel and Tarjan [25] on T, then for given $s, t, u_1, \ldots, u_k$ the set Y can be computed in time polynomial in k. For every node y in Y let $f(y)$ denote the closest ancestor of y whose parent belongs to Y, or y if no such ancestor exists (see Figure 2). Note that every affected node lies on a path joining y with $f(y)$, for some $y \in Y$. Let $Y' = \{f(y) \mid y \in Y\}$. Then Y' is the set of all nodes that are affected children of a node in Y. Note that every element in Y' has exactly one preimage under f, that is, there is an function $f^{-1}\colon Y' \to Y$ such that $f(f^{-1}(y')) = y'$ for all $y' \in Y'$. In particular, $|Y'| \leqslant |Y|$. Using a slight modification of the data structure of Harel and Tarjan [25], we can compute Y' as well as the function $f^{-1}\colon Y' \to Y$ in time polynomial in k.

The idea is that when evaluating query $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$, we can compute the profiles only for the nodes of $Y \cup Y'$, instead of all affected nodes of T. Note that the root of T was explicitly added to Y, so we will still be able to read the answer to $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ from the profile of the root.

We process the nodes of $Y \cup Y'$ in a bottom-up manner. Consider then any element $x \in Y \cup Y'$; our task is to compute its profile based on the profiles of its strict descendants belonging to $Y \cup Y'$. We distinguish two cases. In the first case, $x \in Y$, so we may compute $\mathsf{profile}(x)$ using the algorithm of Lemma 2.2. The second case is of an element $x \in Y'$; then $x = f(y)$ where $y = f^{-1}(x)$. Observe that the path from y to x consists of affected nodes, whose siblings are unaffected nodes. We now show how $\mathsf{profile}(x)$ can be computed from $\mathsf{profile}(y)$.

For two nodes c, d of T, where c is an ancestor of d, let $\mathsf{torso}(c, d)$ be the set of all pairs $\{a, b\} \in \binom{\mathsf{adh}(c) \cup \mathsf{adh}(d)}{2}$ such that in G there is a path from a to b whose all internal vertices belong to $\mathsf{comp}(c) - \mathsf{cone}(d)$. Then to compute $\mathsf{profile}(x)$ from $\mathsf{profile}(y)$, construct an auxiliary graph with vertices $(\mathsf{adh}(x) \cup \mathsf{adh}(y)$ and edges $\mathsf{profile}(y) \cup \mathsf{torso}(x, y)$. Now observe that $\mathsf{profile}(x)$ is the reachability relation in this graph restricted to $\mathsf{adh}(x) - \{u_1, \ldots, u_k\}$. The key point implying the correctness of this observation is that there are no vertices from X in $\mathsf{comp}(x) - \mathsf{comp}(y)$.

Observe now that the values of $\mathsf{torso}(c, d)$ for (c, d) ranging over ancestor/descendants pairs in T are independent of the query. Therefore, upon initialization we can compute a data structure that can be queried for those valued efficiently. The statement below describes two possible implementations.

▶ **Lemma 2.3** (Lemmas 4.2 and 6.2 of the full version, combined and improved). *Given $\mathcal{T}$, one can set up data structures that can answer $\mathsf{torso}(c, d)$ queries with the following specifications:*

1. *initialization time $q^{\mathcal{O}(1)} \cdot |G|^2\|G\|$, memory usage $q^{\mathcal{O}(1)} \cdot |G|^2$, query time $q^{\mathcal{O}(1)}$; or*
2. *initialization time $2^{\mathcal{O}(q^2)} \cdot |G|$, memory usage $2^{\mathcal{O}(q^2)} \cdot |G|$, query time $q^{\mathcal{O}(1)}$.*

The first point of Lemma 2.3 is actually straightforward: just compute and store all the $\mathcal{O}(|T|^2) = \mathcal{O}(|G|^2)$ answers to the torso queries, each in time $q^{\mathcal{O}(1)} \cdot \|G\|$ using $q^{\mathcal{O}(1)}$ applications of breadth-first search. The second point, which trades exponential dependency on q for obtaining linear memory usage and initialization time, is more interesting. Before we discuss it, let us observe that we may now complete the proof of Theorem 1.1. Indeed, using the data structure from the second point of Lemma 2.3, we can compute $\mathsf{profile}(x)$

from $\mathsf{profile}(y)$ for each $x \in Y'$ and $y = f^{-1}(x)$. By performing this procedure for all the at most $2(2k+4)$ nodes of $Y \cup Y'$ in a bottom-up manner, we eventually compute the profile of the root of T, from which the answer to the query can be read. The running time is dominated by $q^{\mathcal{O}(1)}$ calls to the data structure of the second point of Lemma 2.2, each taking $2^{\mathcal{O}(q^2)} = 2^{2^{\mathcal{O}(k)}}$ time.

Finally, let us discuss the proof of the second point of Lemma 2.3. We reduce this problem to the problem of evaluating *product queries* in a semigroup-labeled tree, defined as follows. Suppose T is a rooted tree whose edges are labeled with elements of a finite semigroup S. The task is to set up a data structure that for given nodes x, y, where x is an ancestor of y, outputs the product of the elements of S on the path in T from x to y. The problem of evaluating torso queries can be reduced to this abstract setting by considering a suitable semigroup of *bi-interface graphs*, which represent connectivity between corresponding adhesions; see [7, 8] for the origins of this concept. This semigroup has size $2^{\mathcal{O}(q^2)}$. So it then suffices to use the following result.

▶ **Theorem 2.4** (Przybyszewski). *There is a data structure for the problem of evaluating product queries in a tree T labeled with elements of a finite semigroup S that achieves query time $\log(|S|)^{\mathcal{O}(1)}$, memory usage $|S|^{\mathcal{O}(1)} \cdot |T|$, and initialization time $|S|^{\mathcal{O}(1)} \cdot |T|$.*

Note that this formulation is an improvement over Theorem 5.1 in the first version of our paper [36], as the query time $|S|^{\mathcal{O}(1)}$ is replaced by $\log(|S|)^{\mathcal{O}(1)}$. This improvement is due to Wojciech Przybyszewski.

This finishes the proof of Theorem 1.1. Note that in the argumentation there are two modules that can be replaced with other solutions:

- The decomposition of Theorem 2.1 can be replaced with the more recent variant from [14]. This allows us to have $q = k$, implying that all the $2^{2^{\mathcal{O}(k)}}$ factors are replaced with $2^{\mathcal{O}(k^2)}$. The cost is increasing the polynomial factor of the initialization time to an unspecified term $|G|^{\mathcal{O}(1)}$ and several technical complications in the analysis, which nevertheless can be overcome.
- Instead of using the data structure of the second point of Lemma 2.3, we can use the first point. This allows us to have polynomial dependency on q in the query time and space usage at the cost of quadratic dependence on $|G|$ in the memory usage.

In particular, if only the first replacement is performed, then we obtain a data structure with initialization time $2^{\mathcal{O}(k^2)} \cdot |G|^{\mathcal{O}(1)}$, memory usage $2^{\mathcal{O}(k^2)} \cdot \|G\|$, and query time $k^{\mathcal{O}(1)}$, and if both are performed, then we obtain a data structure with initialization time $2^{\mathcal{O}(k \log k)} \cdot |G|^{\mathcal{O}(1)}$, memory usage $k^{\mathcal{O}(1)} \cdot |G|^2$, and query time $k^{\mathcal{O}(1)}$.

3 Model-checking FO+conn: Theorems 1.2 and 1.3

We now turn to the proofs of Theorem 1.2 and Theorem 1.3. In those theorems, we fix a class $\mathscr{C}$ and consider the model-checking problem for the logic $\mathsf{FO}{+}\mathsf{conn}$, that is, the problem of deciding whether a given $\mathsf{FO}{+}\mathsf{conn}$ sentence holds in a given graph $G \in \mathscr{C}$.

Lower bound. First let us explain why we require $\mathscr{C}$ to exclude a fixed topological minor in Theorem 1.2. Clearly, the model-checking problem for $\mathsf{FO}{+}\mathsf{conn}$ generalizes the model-checking problem for FO, so $\mathscr{C}$ should be in particular a class for which it is known that FO model-checking is FPT. This is the case not only for classes that exclude a topological minor but also for all nowhere dense classes [24]. So let us see why it is not enough to merely assume that $\mathscr{C}$ is nowhere dense.

For a graph G and $k \in \mathbb{N}$, the *k-subdivision* $G^{(k)}$ of G is obtained from G by replacing each edge by a path of length $k+1$. It follows from the definition of nowhere denseness that the class $\mathscr{C}$ of all graphs of the form $G^{(n)}$, where G is an arbitrary graph on n vertices, is nowhere dense. However, a formula of $\mathsf{FO}+\mathsf{conn}$ can easily recover G from $G^{(n)}$, at least when G has no vertices of degree 2. Indeed, the original vertices of G are precisely the vertices of $G^{(n)}$ that have degree at least 3, which can be expressed by an FO formula $\varphi_V(x)$. Furthermore, two such vertices u, v are adjacent in G if and only if there is some vertex z of degree 2 in $G^{(n)}$ such that for every vertex w of $G^{(n)}$, $\mathsf{conn}_2(z, w, u, v)$ holds only if w has degree 2 in $G^{(n)}$. This condition can be written using an $\mathsf{FO}+\mathsf{conn}$ formula $\varphi_E(u, v)$. Using this observation, any FO sentence φ can be replaced by an $\mathsf{FO}+\mathsf{conn}$ sentence φ', so that for every graph G as above, φ holds in G if and only if φ' holds in $G^{(n)}$. Essentially, φ' is obtained from φ by replacing every atomic formula $E(x, y)$ (denoting adjacency) by $\varphi_E(x, y)$, and guarding all quantifiers by $\varphi_V(x)$. Therefore, if we could efficiently model-check φ' on the nowhere dense class $\mathscr{C}$, we could as well model-check φ on the class of all graphs (of size at least 3 and minimum degree at least 3), as follows: given an arbitrary graph G, first construct the graph $G^{(n)} \in \mathscr{C}$ (in time polynomial in $n = |V(G)|$) and then test φ' on $G^{(n)}$. Hence, if $\mathsf{FO}+\mathsf{conn}$ model-checking was fpt on $\mathscr{C}$, then FO model-checking would be fpt on the class of all graphs, implying $\mathsf{FPT} = \mathsf{AW}[\star]$. So nowhere dense graph classes are too general for the statement of Theorem 1.2 to hold. Intuitively, the notion of nowhere denseness is not preserved under contracting long paths, which can be simulated in the logic $\mathsf{FO}+\mathsf{conn}$.

The same simple argument proves Theorem 1.3. Indeed, the argument works not only for the graph $G^{(n)}$ obtained as the n-subdivision of G but also for any subdivision G' (obtained by replacing each edge independently by any number of vertices) of G, as long as G' can be computed from G in polynomial time. The condition that $\mathscr{C}$ is subgraph-closed and admits efficient encoding of topological minors guarantees precisely that for any given graph G we can construct, in polynomial time, a graph G' which is a subdivision of G and belongs to $\mathscr{C}$. This yields Theorem 1.3 (see Section 9 of the full version for details).

Upper bound. We now give some details concerning the proof of Theorem 1.2, which is the second main contribution of the paper.

The starting observation is that for a (q, k)-unbreakable graph G, the query $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$ can be expressed in plain FO. Indeed, this query fails if and only if there is a set A of at most q vertices which contains exactly one of the vertices u and v, and such that all neighbors of vertices of A outside of A are contained in $\{x_1, \ldots, x_k\}$. The existence of such a set of q vertices can be expressed using q existential quantifiers followed by a universal quantifier. So $\mathsf{FO}+\mathsf{conn}$ with connectivity queries of arity at most $k+2$ is equally expressive as plain FO on (q, k)-unbreakable graphs, as long as q is a constant.

Now, if the graph G is not (q, k)-unbreakable, we work with the tree decomposition into (q, k)-unbreakable parts given by Theorem 2.1, where $q = q(k)$ is a constant depending on k. The next observation is that the constant-time querying algorithm used in the proof of Theorem 1.1 and explained above, can be seen as a formula on a suitably defined logical structure. Indeed, if we look into the data structure answering the queries $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$, we notice that each such query triggers a constant number of operations asking about the least common ancestor of two nodes in the tree decomposition, or asking about the membership of a vertex to a bag. Additionally, the algorithm may ask, for two vertices u, v belonging to a bag x, whether u and v are adjacent in the original graph, or whether they are adjacent in the bag graph $\mathsf{bgraph}(x)$. Finally, the algorithm also performs torso queries. We may therefore construct a logical structure, corresponding to the data structure, such that this

basic functionality is expressible by a first-order formula in this structure. Then the query $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$ becomes expressible using a plain FO formula in the logical structure. It will then remain to show that we can model check FO formulas in fpt on the resulting logical structure.

It is convenient to abstract away the details of our logical structure, and represent it in the following form: It is a tree T (possibly vertex-labeled), together with additional edges (possibly labeled) between some nodes which are siblings in the tree. We call such a structure an *augmented tree*. In such a tree, the set of children of any node v carries a structure of a graph. In the particular case of our construction, those graphs will precisely correspond to bag graphs. Recall that a bag graph is obtained from the subgraph induced by a bag by turning all adhesions towards children into cliques. It is not difficult to show (see Lemma 3.4 of the full version) that if the original graph G is H-topological-minor free, then the resulting bag graphs are H'-topological-minor-free for some H' depending on H and on k.[4] Hence, we obtain a tree augmented with H'-topological-minor-free graphs. And the original query $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$ can be now represented as an FO formula in the augmented tree, where the FO formula may use the ancestor relation $\leqslant$ on the tree nodes, as well as the adjacency relation between the siblings. Showing this basically amounts to revisiting the proof of Lemma 2.2 and arguing that the atomic algorithmic operations can now be simulated by FO formulas in the augmented tree.

Let us note here that the remainder of the argument would go through even if we assumed that the resulting tree is augmented with graphs from a fixed nowhere dense class $\mathscr{C}'$ which is not necessarily topological-minor-free. However, if the original graph G merely belongs to a nowhere dense class $\mathscr{C}$, then the bag graphs, and hence the graphs in the augmented tree, no longer need to belong to any fixed nowhere dense class. In fact, they may contain arbitrarily large cliques.

Continuing with the proof, now it remains to show that we can solve FO model-checking on trees augmented with H'-topological-minor-free graphs in fixed-parameter time, where the FO formulas may use the ancestor relation in the tree as well as the edge relation among the siblings. For this, we lift the usual, automata-based techniques for model-checking on trees, to the case of augmented trees. We define automata that process augmented trees in a bottom-up fashion, labeling the nodes of the tree with states from a finite set of states, starting from the leaves and proceeding towards the root. At any node x, to determine the state of the automaton at x, we consider the graph induced on the children of x in the augmented tree, vertex-labeled by the states computed earlier. Then the state at x is determined by evaluating a fixed collection of first-order sentences on this labeled graph. In this way, all nodes of the tree are labeled by states, and the automaton accepts the augmented tree depending on the state at the root.

By construction, it can be decided in fpt time whether or not a given such automaton accepts a given tree that is augmented with graphs that come from, say, a nowhere dense class. Additionally, using standard techniques, we show that for every first-order sentence φ on augmented trees there is such an automaton that determines whether φ holds in a given augmented tree. Therefore, FO model-checking is fpt on trees augmented with graphs from a nowhere dense class; this in particular applies to classes with excluded topological minors.

[4] Here we refer to the bgraph as defined in Footnote 3.

To summarize, to model-check a sentence φ of $\mathsf{FO}+\mathsf{conn}$ on a H-topological-minor-free graph G, we perform the following reasoning:

1. Using Theorem 2.1, compute a tree decomposition $\mathcal{T}$ of G with (q,k)-unbreakable bags, where $k+2$ is the maximal arity of the connectivity predicates in φ, and $q = q(k)$.
2. Turn $\mathcal{T}$ into a tree $\mathcal{T}'$ augmented with H'-topological-minor free graphs, where H' depends on H and k.
3. In the augmented tree $\mathcal{T}'$, the connectivity predicate $\mathsf{conn}_k(u,v,x_1,\ldots,x_k)$ can be expressed using a plain FO formula. This follows by analyzing how connectivity predicates are evaluated in constant time, and observing that the atomic operations can be performed using FO formulas in the augmented trees.
4. Therefore, also φ can be expressed as a plain FO formula φ', which is evaluated in $\mathcal{T}'$.
5. Convert φ' into an automaton $\mathcal{A}$ processing augmented trees, using standard automata-based techniques.
6. Run $\mathcal{A}$ on $\mathcal{T}'$ in a bottom-up fashion. This involves performing FO queries on the graphs induced on the children of any given node, and uses one of the known algorithms for FO model-checking on H'-topological-minor-free graphs.

In Section 7 of the full version we describe the basic toolbox of automata on augmented trees, explaining in detail the last two steps above, abstracting away from the specific application, and providing further results on augmented trees, e.g. concerning query enumeration. In fact, with the same methods we can show that not only FO model-checking is fpt on trees augmented with graphs from a nowhere dense class, but also a more general logic $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A) \cup \Sigma)$ combining the power of FO on the augmenting graphs with MSO on trees can be solved in fpt on augmented trees. We then show in Section 8 (of the full version) how to express the predicates conn_k on augmented trees using this more general logic $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A) \cup \Sigma)$.

4 Discussion

In this work we have added computational problems related to conn queries in graphs and the logic $\mathsf{FO}+\mathsf{conn}$ to the growing list of applications of the decomposition theorem of Cygan et al. [15]. Several questions can be asked about possible improvements of the obtained complexity bounds, especially regarding the data structure of Theorem 1.1.

First, in both the model-checking algorithm of Theorem 1.2 and the construction algorithm of the data structure of Theorem 1.1, the main bottleneck for the polynomial factor is the running time of the algorithm to compute the tree decomposition of Cygan et al. (see Theorem 2.1). This running time is cubic in the size of the graph, while in both applications presented in this work, the remainder of the construction takes fpt time with at most quadratic dependence on the graph size. Therefore, it seems imperative to revisit Theorem 2.1 with the purpose of improving the time complexity of the construction algorithm.

Second, in Section 6 of the full version we showed how to improve the time complexity of queries to polynomial in k by using the weakly unbreakable tree decomposition provided by [14]. The drawback is that the space usage of the data structure becomes $k^{\mathcal{O}(1)} \cdot |G|^2$. However, the quadratic dependence on $|G|$ is caused only by the brute-force implementation of the data structure for torso queries provided by Lemma 6.2 of the full version. So now we have two implementations of this data structure:

- Theorem 2.4 and Lemma 4.2 (of the full version) offers query time $q^{\mathcal{O}(1)}$ and space usage $2^{\mathcal{O}(q^2)} \cdot |T|$.
- Lemma 6.2 (of the full version) offers query time $q^{\mathcal{O}(1)}$ and space usage $q^{\mathcal{O}(1)} \cdot |T|^2$.

Is it possible to design a data structure that would offer query time $q^{\mathcal{O}(1)}$ while keeping the space usage at $q^{\mathcal{O}(1)} \cdot |T|$?

References

1 Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, eliminating and decomposing to hereditary classes are all fpt-equivalent. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1976–2004. SIAM, 2022.

2 Akanksha Agrawal, Lawqueen Kanesh, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. An FPT algorithm for elimination distance to bounded degree graphs. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021*, volume 187 of *LIPIcs*, pages 5:1–5:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

3 Akanksha Agrawal, Lawqueen Kanesh, Fahad Panolan, MS Ramanujan, and Saket Saurabh. An fpt algorithm for elimination distance to bounded degree graphs. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

4 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019*, pages 89–103. ACM, 2019.

5 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proceedings of the 15th International Conference on Computer Science Logic, CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006. doi:10.1007/11874683_11.

6 Mikołaj Bojańczyk. Separator logic and star-free expressions for graphs. *CoRR*, abs/2107.13953, 2021. arXiv:2107.13953.

7 Mikołaj Bojańczyk and Michał Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016*, pages 407–416. ACM, 2016. doi:10.1145/2933575.2934508.

8 Glencora Borradaile, Seth Pettie, and Christian Wulff-Nilsen. Connectivity oracles for planar graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 316–327. Springer, 2012.

9 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016.

10 Jannis Bulian and Anuj Dawar. Fixed-parameter tractable distances to sparse graph classes. *Algorithmica*, 79(1):139–158, 2017.

11 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

12 Thomas Colcombet. A combinatorial theorem for trees. In *Proceedings of the 34th International Colloquium Automata, Languages and Programming, ICALP 2007*, volume 4596 of *Lecture Notes in Computer Science*, pages 901–912. Springer, 2007. doi:10.1007/978-3-540-73420-8_77.

13 Bruno Courcelle. The Monadic Second-Order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

14 Marek Cygan, Paweł Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Transactions on Algorithms*, 17(1):6:1–6:30, 2021. doi:10.1145/3426738.

15 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum Bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48(2):417–450, 2019. doi:10.1137/140988553.

16 Ran Duan and Seth Pettie. Connectivity oracles for failure prone graphs. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 465–474. ACM, 2010. doi:10.1145/1806689.1806754.

17 Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. *SIAM Journal on Computing*, 49(6):1363–1396, 2020. doi:10.1137/17M1146610.

18 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 121–131, 2014.

19 Zdeněk Dvořák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

20 Fedor V Fomin, Petr A Golovach, Ignasi Sau, Giannos Stamoulis, and Dimitrios M Thilikos. A compound logic for modification problems: Big kingdoms fall from within. *arXiv preprint*, 2021. arXiv:2111.02755.

21 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Parameterized complexity of elimination distance to first-order logic properties. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*, pages 1–13. IEEE, 2021.

22 Daniele Frigioni and Giuseppe F. Italiano. Dynamically switching vertices in planar graphs. In *Proceedings of the 5th Annual European Symposium on Algorithms, ESA 1997*, volume 1284 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 1997.

23 Robert Ganian, Petr Hliněný, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of mso1 model-checking. *Journal of Computer and System Sciences*, 80(1):180–194, 2014.

24 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

25 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

26 Bart M. P. Jansen and Jari J. H. de Kroon. FPT algorithms to compute the elimination distance to bipartite graphs and more. In *47th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2021*, volume 12911 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 2021. doi:10.1007/978-3-030-86838-3_6.

27 Bart MP Jansen, Jari JH de Kroon, and Michał Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 1757–1769, 2021.

28 Ken-ichi Kawarabayashi and Mikkel Thorup. The Minimum k-way Cut of bounded size is fixed-parameter tractable. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 160–169. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.53.

29 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Transactions on Computational Logic*, 14(4):25:1–25:12, 2013. doi:10.1145/2528928.

30 Wojciech Kazana and Luc Segoufin. First-order queries on classes of structures with bounded expansion. *Log. Methods Comput. Sci.*, 16(1), 2020.

31 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010*, pages 189–198. IEEE Computer Society, 2010.

32 Alexander Lindermayr, Sebastian Siebertz, and Alexandre Vigny. Elimination distance to bounded degree on planar graphs. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020*, volume 170 of *LIPIcs*, pages 65:1–65:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

33 Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Fixed-parameter tractability of Graph Isomorphism in graphs with an excluded minor, 2021. Manuscript.

34 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.135.

35 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for Min k-Cut. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 798–809. IEEE, 2020. doi:10.1109/FOCS46700.2020.00079.

36 Michał Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Toruńczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. *arXiv preprint v1*, 2021. arXiv:2111.03725.

37 Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007*, pages 263–271. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.54.

38 Saket Saurabh and Meirav Zehavi. Parameterized complexity of multi-node hubs. In *Proceedings of the 13th International Symposium on Parameterized and Exact Computation, IPEC 2018*, volume 115 of *LIPIcs*, pages 8:1–8:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.IPEC.2018.8.

39 Nicole Schirrmacher, Sebastian Siebertz, and Alexandre Vigny. First-order logic with connectivity operators. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

40 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, pages 151–163. ACM, 2018. doi:10.1145/3196959.3196971.

41 Szymon Toruńczyk. Aggregate queries on sparse databases. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020*, pages 427–443. ACM, 2020.

42 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 424–435. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00034.

A Perfect Sampler for Hypergraph Independent Sets

Guoliang Qiu
Shanghai Jiao Tong University, China

Yanheng Wang[1]
ETH Zürich, Switzerland

Chihao Zhang
Shanghai Jiao Tong University, China

Abstract

The problem of uniformly sampling hypergraph independent sets is revisited. We design an efficient *perfect* sampler for the problem under a similar condition of the asymmetric Lovász local lemma. When specialized to d-regular k-uniform hypergraphs on n vertices, our sampler terminates in expected $O(n \log n)$ time provided $d \leq c \cdot 2^{\frac{k}{2}}$ where $c > 0$ is a constant, matching the rapid mixing condition for Glauber dynamics in Hermon, Sly and Zhang [10]. The analysis of our algorithm is simple and clean.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains

Keywords and phrases Coupling from the past, Markov chains, Hypergraph independent sets

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.103

Category Track A: Algorithms, Complexity and Games

Funding *Chihao Zhang*: The author is supported by National Natural Science Foundation of China Grant 61902241.

1 Introduction

The problem of uniformly sampling hypergraph independent sets, or equivalently the solutions of monotone CNF formulas, has been well-studied in recent years. Consider a hypergraph $\Phi = (V, \mathcal{C})$ on $|V| = n$ vertices. A set $S \subseteq V$ is an independent set if $C \cap S \neq C$ for all $C \in \mathcal{C}$. Assuming the hypergraph is d-regular and k-uniform, [10] showed that the natural Glauber dynamics mixes in $O(n \log n)$ time when $d \leq c \cdot 2^{\frac{k}{2}}$ for some constant $c > 0$. The sampler implies a fully polynomial-time randomized approximation scheme (FPRAS) for counting hypergraph independent sets [13]. On the other hand, it was shown in [1] that there is no FPRAS for the problem when $d \geq 5 \cdot 2^{\frac{k}{2}}$, unless **NP** = **RP**. Therefore, the result of [10] is tight up to a multiplicative constant.

The proof in [10] analyzes the continuous-time Glauber dynamics under the framework of *information percolation* developed in [14] for studying the cutoff phenomenon of the Ising model. In this framework, one can view the coupling history of a Markov chain as time-space slabs, and the failure of the coupling at time t as a discrepancy path percolating from t back to the beginning. The analysis of this structure can result in (almost) optimal bounds in many interesting settings.

The percolation analysis in [10] is technically complicated due to the *continuous* nature of the chain which leads to involved dependencies in both time and space. Recently, discrete analogs of the time-space slabs have been introduced in sampling solutions of general CNF

[1] Part of the work was done while the author was an undergraduate student at Shanghai Jiao Tong University.

formulas in the Lovász local lemma regime [11, 8]. Notably in [8], an elegant discrete time-space structure, tailored to *systematic scan*, was introduced to support the analysis of *coupling from the past* (CFTP) paradigm [17]. In contrast to simulating Glauber dynamics, the method of CFTP allows to produce *perfect samples* (i.e. without approximation error) from stationary distribution.

In this article, we refine the discrete time-space structure in [8], which we call the *witness graph*, and apply it to the problem of sampling hypergraph independent sets. This leads to an efficient CFTP sampler that (1) outputs perfect samples; (2) matches the bound in [10] and operates also in the asymmetric case under Lovász local lemma-like condition; and (3) has significantly simpler analysis. More specifically, we study a natural grand coupling of the systematic scan for sampling hypergraph independent sets. To apply the method of CFTP, one needs to detect the coalescence of the grand coupling efficiently at each stage of the algorithm. The monotone property of hypergraph independent sets allows us to reduce the detection to arguing about percolation on the witness graph. This observation provides sufficient flexibility for us to carefully bound related probabilities. We first show that, under a condition similar to the *asymmetric* Lovász local lemma, a perfect sampler exists.

▶ **Theorem 1.** *Let $\mathcal{G}$ collects all hypergraphs $\Phi = (V, \mathcal{C})$ such that*

$$\forall C \in \mathcal{C} : 2|C| \cdot 2^{-|C|} \leq (1-\varepsilon) \cdot x(C) \cdot \prod_{C' \in \Gamma_{\Phi^2}(C)} (1 - x(C')),$$

for some constant $\varepsilon \in (0,1)$ and function $x : \mathcal{C} \to (0,1)$. Here $\Gamma_{\Phi^2}(C)$ denotes the set of hyperedges within distance 2 to C in Φ (excluding C itself).

There exists an algorithm that inputs a hypergraph $\Phi \in \mathcal{G}$ and outputs an independent set of Φ uniformly at random, in expected time

$$O\left(-\frac{1}{\log(1-\varepsilon)} \cdot \log\left(\sum_{C \in \mathcal{C}} \frac{x(C)}{1-x(C)}\right) \cdot \sum_{C \in \mathcal{C}} \sum_{v \in C} d_v |C|\right),$$

where d_v is number of hyperedges in Φ that contains v.

In the proof of Theorem 1, we view a discrepancy path of the coupling percolating from time t back to the beginning as an object similar to the *witness tree* in [16] for certifying the non-termination of a randomized algorithm. This is an interesting analogy between an algorithm that *samples* the solutions of CNF formulas and an algorithm that *finds* them. We map the discrepancy paths in the witness graph to connected trees generated by *multi-type Galton-Watson process* in the hypergraph. As a result, a certain spatial mixing property implies the rapid mixing of the chain.

For k-uniform d-regular hypergraphs, the result in Theorem 1 translates to the condition $d \leq \frac{c \cdot 2^{k/2}}{k^{1.5}}$ for some constant $c > 0$ by choosing $x(C) = \frac{1}{d^2 k^2}$. We give a refined analysis for this symmetric case to remove the denominator:

▶ **Theorem 2.** *Let $\mathcal{G}$ collects all k-uniform d-regular hypergraphs $\Phi = (V, \mathcal{C})$ such that*

$$d \leq \left(\frac{1}{4}\sqrt{\frac{9-\varepsilon}{2}} - \frac{1}{2}\right) \cdot 2^{k/2}.$$

for some constant $\varepsilon \in (0,1)$. There exists an algorithm that inputs a hypergraph $\Phi \in \mathcal{G}$ and outputs a hypergraph independent set of Φ uniformly at random, in expected time $O\left(-\frac{1}{\log(1-\varepsilon)} \cdot k^2 d^2 n \cdot (\log n + \log d)\right)$ where $n := |V|$.

Compared to the result in [10], our sampler has the advantage of being perfect. Our analysis inductively enumerates discrepancy paths, taking into account the structure of the overlapping between hyperedges. Thanks to the clean structure of the witness graph, our proof is much simpler than the one in [10].

Relation to sampling solutions of general CNF formulas

The problem of sampling hypergraph independent sets is a special case of sampling the solutions of general CNF formulas or CSP instances which draw a lot of recent attention [15, 3, 11, 8, 12, 7, 6, 5, 4, 2, 9]. For general k-CNF formulas where each variable is of degree d (corresponding to the k-uniform d-regular hypergraphs here), the best condition needed for an efficient sampler is $d < \frac{1}{\mathrm{poly}(k)} \cdot 2^{\frac{k}{40/7}}$ [11, 8] which is much worse than the condition $d < c \cdot 2^{\frac{k}{2}}$ for hypergraph independent sets. A main reason is that for general CNF formulas, the state space of the local Markov chain is no longer connected, and therefore one needs to project the chain onto a sub-instance induced by some special set of variables. The loss of the projection step, however, is not well-understood. On the other hand, this is not an issue for hypergraph independent sets and therefore (almost) optimal bounds can be obtained.

We remark that the technique developed here for non-uniform graphs can also be applied to general CNF formulas to analyze the projection chain.

2 Preliminaries

2.1 Hypergraph independent sets

Recall in the introduction we mentioned that a set $S \subseteq V$ is an independent set of a hypergraph $\Phi = (V, \mathcal{C})$ if $S \cap C \neq C$ for every $C \in \mathcal{C}$. Sometimes we represent it as a (binary) coloring $\sigma \in \{0,1\}^V$, which is the indicator for S. That is, we say σ is an hypergraph independent set if $\forall C \in \mathcal{C}, \exists v \in C : \sigma(v) = 0$. Denoting by Ω_Φ the collection of all independent sets of Φ, our goal is to efficiently produce a sample from the uniform measure μ on Ω_Φ.

Let us fix some notations used throughout our discussion:

- We assume $V := [n]$ and $m := |\mathcal{C}|$. We denote the degree of a vertex $v \in V$ as $d_v := |\{C \in \mathcal{C} : v \in C\}|$ and the maximum degree as $d := \max_{v \in V} d_v$. A hypergraph is d-regular if every vertex is of degree d and is k-uniform if every hyperedge is of size k.
- For any $C \in \mathcal{C}$, we use $\Gamma^+_{\Phi^2}(C)$ to denote the set of hyperedges C' such that either $C \cap C' \neq \varnothing$ or there exists C^* such that $C \cap C^* \neq \varnothing$ and $C^* \cap C' \neq \varnothing$. Furthermore, let $\Gamma_{\Phi^2}(C) := \Gamma^+_{\Phi^2}(C) \setminus \{C\}$.
- For any coloring $\sigma \in \{0,1\}^V$, we use $\sigma^{v \leftarrow r}$ to denote the coloring obtained after recoloring $v \in V$ by $r \in \{0,1\}$ in σ.

2.2 Systematic scan and coupling from the past

Fix a hypergraph $\Phi = (V, \mathcal{C})$ and the uniform distribution μ over Ω_Φ. We use the so-called *systematic scan* Markov chain to sample independent sets from μ.

Let us define a transition map $f : \Omega_\Phi \times V \times \{0,1\} \to \Omega_\Phi$ by

$$f(\sigma; v, r) := \begin{cases} \sigma^{v \leftarrow r} & \text{if } \sigma^{v \leftarrow r} \in \Omega_\Phi; \\ \sigma & \text{otherwise.} \end{cases}$$

It takes the given coloring $\sigma \in \Omega_\Phi$ and tries to recolor vertex v with the proposed r. Next, we fix a deterministic *scan sequence* $v_1, v_2, \ldots$ where $v_i := (i \bmod n) + 1$ and write

$$F(\sigma; r_1, \ldots, r_t) := f(f(\cdots f(\sigma; v_1, r_1) \cdots); v_t, r_t).$$

Basically, it begins with the given coloring $\sigma \in \Omega_\Phi$ and runs f for t steps to update vertex colors. The vertices are updated periodically as specified by the scan sequence; the proposed colors for every steps are provided by the arguments $r_1, \ldots, r_t$. For a collection of initial states $S \subseteq \Omega_\Phi$, we use $F(S; r_1, \ldots, r_t)$ to denote the set $\{F(\sigma; r_1, \ldots, r_t) : \sigma \in S\}$.

▶ **Definition 3.** *The* systematic scan *is a Markov chain* (X_t) *defined by*

$$X_t := F(\sigma; R_1, \ldots, R_t) \quad \forall t$$

for some $\sigma \in \Omega_\Phi$ *and some independent* $\texttt{Bernoulli}(1/2)$ *variables* $R_1, R_2, \ldots$.

The systematic scan is not a time-homogeneous Markov chain. However, we can (and will) bundle every n steps into an atomic round so that the bundled version – denoting its transition matrix P_Φ – is homogeneous. It is easy to check that P_Φ is irreducible and aperiodic with stationary distribution μ.

Given a Markov chain with stationary μ, in the usual Markov chain Monte Carlo method, one obtains a sampler for μ as follows: Starting from some initial X_0, simulate the chain for t steps with sufficiently large t and output X_t. The fundamental theorem of Markov chains says that if the chain is finite, irreducible and aperiodic, then the distribution of X_t converges to μ when t approaches infinity. However, since we always terminate the simulation after some fixed finite steps, the sampler obtained is always an "approximate" sampler instead of a "perfect" one.

The work of [17] proposed an ingenious method called *coupling from the past* (CFTP) to simulate the given chain in a reverse way with a random stopping time. A perfect sampler for μ can be obtained in this way. Roughly speaking, it essentially simulates an infinite long chain using only finitely many steps. To achieve this, it relies upon a routine to detect whether the (finite) simulation coalesces with the virtual infinite chain. Once the coalescence happens, one can output the result of the simulation – which is also the result of the infinite chain and thus follows the stationary distribution perfectly.

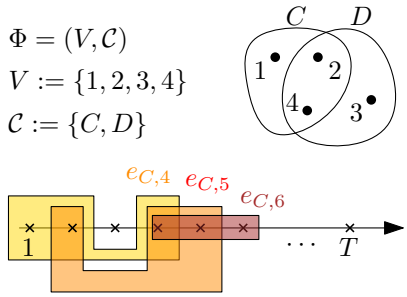
We use CFTP to simulate our systematic scan and obtain a desired perfect sampler for hypergraph independent sets. The key ingredient to apply CFTP is how to detect the coalescence. We describe and analyze our algorithm in Section 3.

3 Perfect Sampler via Information Percolation

In this section, we describe our perfect sampler for hypergraph independent sets. With the help of a data structure introduced in [8], we apply the argument of information percolation to analyze the algorithm. We utilize the monotonicity of the hypergraph independent sets and establish a sufficient condition for the algorithm to terminate.

3.1 The witness graph

In this section, we introduce the notion of witness graph $H_T = (V_T, E_T)$ for the systematic scan up to some time $T \in \mathbb{N}$. A similar structure was used in [8] for sampling general CNF formulas.



$e_{C,1} = \{1\}$	
$e_{C,2} = \{1, 2\}$	$e_{D,2} = \{2\}$
	$e_{D,3} = \{2, 3\}$
$e_{C,4} = \{1, 2, 4\}$	$e_{D,4} = \{2, 3, 4\}$
$e_{C,5} = \{5, 2, 4\}$	
$e_{C,6} = \{5, 6, 4\}$	$e_{D,6} = \{6, 3, 4\}$
	$e_{D,7} = \{6, 7, 4\}$
$e_{C,8} = \{5, 6, 8\}$	$e_{D,8} = \{6, 7, 8\}$
...	...

Figure 1 An illustration of the witness graph. In this example, the hypergraph Φ has four vertices and two hyperedges. We list the vertices of the witness graph in the table on the right. The lower left picture visualizes three vertices in H_T; all of them have label C.

Given a vertex $v \in V$, we denote its last update time up to moment t as

$$\texttt{UpdTime}(v, t) = \max\{t^* \le t : v_{t^*} = v\}.$$

Clearly $\texttt{UpdTime}(v, t) \in (t - n, t]$ is a deterministic number.

For $C \in \mathcal{C}$ and $t \in [T]$, let $e_{C,t} := \{\texttt{UpdTime}(v, t) : v \in C\}$ be timestamps when the elements in C got their latest updates up to time t. Conversely, we say $e_{C,t}$ has *label* C and denote it by $C(e_{C,t}) := C$.

The vertex set of witness graph is given by

$$V_T := \{e_{C,t} : t \in [T], v_t \in C \in \mathcal{C}\}.$$

We put a directed edge $e_{C,t} \to e_{C',t'}$ into E_T if and only if $t' \in (e_{C,t} \cap e_{C',t'}) \setminus \{t\}$. Note that H_T is acyclic, since $\max e > \max e'$ for any directed edge $e \to e'$. We remark that the witness graph is a deterministic object which by itself does not incorporate any randomness.

The following lemma measures the number of vertices labelled by a certain $C \in \mathcal{C}$ that are 2-distant from a given vertex in the witness graph. It is useful throughout the enumeration in later discussions.

▶ **Lemma 4.** *For any $e_{C,t} \in V_T$ and $C' \in \mathcal{C}$, we have $|e_{C',t'} \in V_T : \texttt{dist}(e_{C,t}, e_{C',t'}) = 2| \le 2|C'|$.*

Proof. If $C' \notin \Gamma^+_{\Phi^2}(C)$ then $|e_{C',t'} \in V_T : \texttt{dist}(e_{C,t}, e_{C',t'}) = 2| = 0$. Otherwise, $t' \in [t - 2n + 2, t)$, and there are at most $2|C'|$ timestamps $e_{C',t'}$ satisfying $t' \in [t - 2n + 2, t)$. ◀

3.2 Coalescence and percolation

For any $L \in \mathbb{N}$ we fix $T = T(L) := n(L + 1)$. On top of the witness graph H_T we define a probability space as follows. We tie an independent $\texttt{Bernoulli}(1/2)$ variable B_t to each time point $t \in [T]$. We say a vertex $e \in V_T$ is *open* if $B_t = 1$ for all $t \in e$, and call a set of vertices $P \subseteq V$ *open* if all $e \in P$ are open. The event $\mathcal{B}_L$ is defined as

> "H_T contains an induced open path $P = (e_1, \ldots, e_L) \subseteq V_T$ of length L where $e_1 \cap (T - n, T] \neq \varnothing$".

For the coming few pages, the notation $\mathcal{B}_L(R_1, \ldots, R_T)$ indicates that we are using (external) random variables $R_1, \ldots, R_T$ as *concrete realizations* of our abstract variables $B_1, \ldots, B_T$. Needless to say, $R_1, \ldots, R_T$ themselves should be independent $\texttt{Bernoulli}(1/2)$ for such notation to make sense.

Starting from Section 4, however, we will switch back to the abstract setting and sweep the concrete realization under the rug.

Recall F is the transition map for our systematic scan.

▶ **Lemma 5.** *Assume $L \in \mathbb{N}$, $T := n(L+1)$ and $R_1, \ldots, R_T$ are independent* `Bernoulli`$(1/2)$ *variables. If $|F(\Omega_\Phi; R_1, \ldots, R_T)| > 1$ then $\mathcal{B}_L = \mathcal{B}_L(R_1, \ldots, R_T)$ happens.*

To prove Lemma 5 we specify a *grand coupling*, namely a family of Markov chains that share the same random sequence $R_1, \ldots, R_T$ provided by the lemma.

For every $\sigma \in \Omega_\Phi$, define a copy of systematic scan $(X_{\sigma,t})_{0 \le t \le T}$ by

$$X_{\sigma,t} := F(\sigma, R_1, \ldots, R_t) \quad \forall 0 \le t \le T.$$

In addition, we define an auxiliary Markov chain $(Y_t)_{0 \le t \le T}$ by

$$Y_0 := \{1\}^V, \qquad Y_t := Y_{t-1}^{v_t \leftarrow R_t} \quad \forall 1 \le t \le T.$$

The chain $(Y_t)_{0 \le t \le T}$ dominates the execution of $(X_{\sigma,t})_{0 \le t \le T}$ by monotonicity:

▶ **Proposition 6.** *For all $\sigma \in \Omega_\Phi$ and $0 \le t \le T$, we have $X_{\sigma,t} \le Y_t$.*

Proof. Initially, $X_{\sigma,0} = \sigma \le \{1\}^V = Y_0$ for all $\sigma \in \Omega_\Phi$. At any time $t \ge 1$, all the chains update the same vertex v_t and (i) if $R_t = 0$ then $X_{\sigma,t}(v_t) = 0$; (ii) if $R_t = 1$ then $Y_t(v_t) = 1$. So the ordering $X_{\sigma,t} \le Y_t$ is preserved throughout. ◀

▶ **Proposition 7.** *Let $0 \le t \le T$ be a time point. If there exist $\sigma, \tau : X_{\sigma,t}(v_t) \ne X_{\tau,t}(v_t)$, then there is a hyperedge $C \in \mathcal{C}$ containing v_t such that $C \setminus \{v_t\}$ was fully colored by "1" in exactly one of $X_{\sigma,t}$ and $X_{\tau,t}$. Furthermore, $Y_t(C) = \{1\}^C$.*

Proof. Since $X_{\sigma,t}(v_t) \ne X_{\tau,t}(v_t)$, we know that $R_t = 1$ and thus $Y_t(v_t) = 1$. According to the definition of transition map f, one of the two chains failed to recolor v_t by "1" because there exists a hyperedge C such that $C \setminus \{v_t\}$ was fully colored "1" in that chain just before time t. Hence $Y_t(C \setminus \{v_t\}) = \{1\}^{C \setminus \{v_t\}}$ by Proposition 6. Putting together we have $Y_t(C) = \{1\}^C$. ◀

We are now ready to prove Lemma 5.

Proof of Lemma 5. Assume $|F(\Omega_\Phi; R_1, \ldots, R_T)| > 1$; that is, there exist $\sigma, \tau \in \Omega_\Phi$ and $v \in V$ such that $X_{\sigma,T}(v) \ne X_{\tau,T}(v)$. We inductively construct a path in H_T as follows:

1. Set $i \leftarrow 1$. Let $t_1 := \texttt{UpdTime}(v, T) \in (T-n, T]$.
2. While $t_i \ge n$ do the following. Regarding Proposition 7 and the fact $X_{\sigma,t_i}(v_{t_i}) \ne X_{\tau,t_i}(v_{t_i})$, there is a hyperedge $C_i \in \mathcal{C}$ containing v_{t_i} such that $C_i \setminus \{v_{t_i}\}$ was fully colored by "1" in exactly one of X_{σ,t_i} and X_{τ,t_i}. So we may find an earliest time $t_{i+1} \in e_i := e_{C_i,t_i}$ such that $X_{\sigma,t_{i+1}}(v_{t_{i+1}}) \ne X_{\tau,t_{i+1}}(v_{t_{i+1}})$ and $t_{i+1} < t_i$. Moreover, $Y_{t_i}(C_i) = \{1\}^{C_i}$ implies that e_i is open. Let $i \leftarrow i+1$ and repeat.

Note that $t_{i+1} \in e_i$ by definition. On the other hand $t_{i+1} \in e_{i+1} = e_{C_{i+1},t_{i+1}}$ since $v_{t_{i+1}} \in C_{i+1}$. Combining with the condition $t_{i+1} < t_i$, we see $t_{i+1} \in (e_i \cap e_{i+1}) \setminus \{t_i\}$ and consequently $e_i \to e_{i+1}$ is indeed an edge in H_T. Therefore the above procedure returns an open (not necessarily induced) path $P^\circ = (e_1, \ldots, e_r) \subseteq V_T$ where r is the number of rounds. It starts at vertex e_1 that intersects $(T-n, T]$, and ends at vertex e_r that intersects $[1, n)$.

Let $P \subseteq P^\circ$ be a *shortest path* from e_1 to e_r in $H_T[P^\circ]$. It must be an induced path in H_T since H_T is acyclic. It is open since $P \subseteq P^\circ$. Finally, every vertex $e \in P$ "spans" a time interval of at most n, so the length of P is at least $T/n - 1 = L$. ◀

3.3 The perfect sampler

We are ready to introduce our perfect sampler for hypergraph independent sets. Let $L \in \mathbb{N}$ be a parameter to be fixed later (roughly in the order of $O(\log m)$) and $T = T(L) := n(L+1)$ as usual. The algorithm follows the standard framework of CFTP.

Algorithm 1 The CFTP sampler of hypergraph independent sets.

Input: A hypergraph $\Phi = (V, \mathcal{C})$.
Output: An independent set of Φ sampled uniformly from μ.

$j \leftarrow 0$;
repeat
  $j \leftarrow j + 1$;
  generate independent $\texttt{Bernoulli}(1/2)$ variables $R_{-jT+1}, \ldots, R_{-(j-1)T}$;
until *not* $\texttt{Detect}(\Phi, R_{-jT+1}, \ldots, R_{-(j-1)T})$;
return $F\left(\{0\}^V; R_{-jT+1}, \ldots, R_0\right)$.

Algorithm 2 The Detect subroutine.

Input: A hypergraph $\Phi = (V, \mathcal{C})$ and a sequence $R_1, \ldots, R_T$.
Output: Whether $\mathcal{B}_L(R_1, \ldots, R_T)$ happens.

foreach $C \in \mathcal{C}$ **do**
  foreach $t \in [T] : v_t \in C$ **do**
    foreach $v \in C \setminus \{v_t\}$ **do**
      foreach $C' \in \mathcal{C} : v \in C' \neq C$ **do**
        $t' \leftarrow \texttt{UpdTime}(v, t)$;
        connect $e_{C,t} \to e_{C',t'}$ if both of them are open with regard to $R_1, \ldots, R_T$;
breadth-first search from all $e_{C,t} : C \in \mathcal{C}, t \in (T-n, T]$ until the current vertex intersects $[1, n)$;
return *if the path has length* $\geq L$.

▶ **Theorem 8.** *Suppose there exist constant $\varepsilon \in (0,1)$ and β such that $\mathbf{Pr}[\mathcal{B}_L] \leq \beta \cdot (1-\varepsilon)^L$ for all L. With the concrete choice $L := \left\lceil \frac{\log(2\beta)}{-\log(1-\varepsilon)} \right\rceil$, Algorithm 1 terminates with probability 1 and has expected running time $O\left(\frac{\log(2\beta)}{-\log(1-\varepsilon)} \cdot \sum_{C \in \mathcal{C}} \sum_{v \in C} d_v |C|\right)$. Moreover, its output distribution ν is exactly μ upon termination.*

Proof. The `Detect` subroutine (Algorithm 2) essentially constructs the witness graph H_T and decides if $\mathcal{B}_L(R_1, \ldots, R_T)$ happens. Since the inputs fed by Algorithm 1 are always independent $\texttt{Bernoulli}(1/2)$ variables as required, we have

$$p := \mathbf{Pr}\left[\texttt{Detect}(\Phi, R_{-jT+1}, \ldots, R_{(-j-1)T})\right] = \mathbf{Pr}[\mathcal{B}_L] \leq \beta \cdot (1-\varepsilon)^L = \frac{1}{2}.$$

where the inequality follows from the assumption of the theorem.

Note that Algorithm 1 feeds disjoint (hence independent) sequences into Algorithm 2 in different rounds, so the corresponding return values are independent. Therefore, the total number of rounds J follows geometric distribution $\texttt{Geom}(1-p)$. In particular $\mathbf{Pr}[J < \infty] = 1$ and $\mathbf{E}[J] = \frac{1}{1-p} \leq 2$. So the algorithm terminates with probability 1.

Next we analyze the running time of a single call to Algorithm 2. Each iteration of the nested loop can be implemented in constant time: We index each vertex $e_{C,t}$ by the pair (C,t) to allow random access; the test of openness can be computed and stored beforehand to facilitate fast lookup. So the nested loop takes

$$O\left(\sum_{C\in\mathcal{C}}|C|\cdot L\cdot\sum_{v\in C}d_v\right)$$

time to finish. The standard breadth-first search consumes time proportional to the number of edges in the witness graph, which is bounded by the same quantity.

Finally, recall that Algorithm 1 calls Algorithm 2 J times. But $\mathbf{E}[J]\le 2$, so the expected running time of our sampler is $O\left(L\cdot\sum_{C\in\mathcal{C}}\sum_{v\in C}d_v|C|\right)$.

We could now turn to show $\nu=\mu$ as advertised. Upon termination, the percolation event $\mathcal{B}_L(R_{-JT+1},\dots,R_{-(J-1)T})$ does not happen and hence $\left|F(\Omega_\Phi;R_{-JT+1},\dots,R_{-(J-1)T})\right|=1$ by Lemma 5. But this in particular means

$$|F(\Omega_\Phi;R_{-JT+1},\dots,R_0)|=1. \tag{1}$$

For each integer $s\ge 0$, we define a (virtual) copy $(Z_t^s)_{-sn\le t\le 0}$ of systematic scan which starts from $\{0\}^V$ at time $-sn$:

$$Z_t^s:=F\left(\{0\}^V;R_{-sn+1},\dots,R_t\right).$$

Let ν^s denote the distribution of Z_0^s. Then $\lim_{s\to\infty}\nu^s=\mu$ by convergence theorem. On the other hand,

$$\begin{aligned}\|\nu^s-\nu\|&\le\mathbf{Pr}\left[F\left(\{0\}^V;R_{-sn+1},\dots,R_0\right)\ne F\left(\{0\}^V;R_{-JT+1},\dots,R_0\right)\right] && \text{(coupling)}\\ &\le\mathbf{Pr}[sn<JT] && \text{(by (1))}\\ &\to 0. && \text{(as } s\to\infty)\end{aligned}$$

Hence $\nu=\lim_{s\to\infty}\nu^s=\mu$. ◀

3.4 Proofs of Theorem 1 and Theorem 2

Armed with Theorem 8, we only need to bound $\mathbf{Pr}[\mathcal{B}_L]$ under various conditions. We bound the quantity in Section 4. As a result, Theorem 1 follows from Lemma 9 and Theorem 2 follows from Lemma 13.

4 Percolation on Witness Graphs

Let $\Phi=(V,\mathcal{C})$ be a hypergraph with $|V|=n$ and let H_T be its witness graph for any $T=(L+1)n$, $L\in\mathbb{N}$. We will analyze the probability $\mathbf{Pr}[\mathcal{B}_L]$ in the abstract probability space, where each time point $t\in[T]$ is associated with an independent $\texttt{Bernoulli}(1/2)$ variable.

4.1 General hypergraphs

▶ **Lemma 9.** *If there exists a constant $\varepsilon\in(0,1)$ and a function $x:\mathcal{C}\to(0,1)$ satisfying*

$$\forall C\in\mathcal{C}:2|C|\cdot 2^{-|C|}\le(1-\varepsilon)\cdot x(C)\cdot\prod_{C'\in\Gamma_{\Phi^2}(C)}(1-x(C')),$$

then $\mathbf{Pr}[\mathcal{B}_L]\le\sum_{C\in\mathcal{C}}\frac{x(C)}{1-x(C)}\cdot(1-\varepsilon)^{\lfloor\frac{L+1}{2}\rfloor}$ for all $L\in\mathbb{N}$.

For every $C \in \mathcal{C}$, we use $\mathcal{P}_{C,L}$ to denote the collection of induced paths $(e_1, e_2, \ldots, e_L)$ in H_T where $e_1 \cap (T-n, T] \neq \varnothing$ and the label of e_1 is C (namely $C(e_1) = C$). Let

$$\mathcal{P}^2_{C,\lfloor \frac{L+1}{2} \rfloor} := \left\{ (e_1, e_3, e_5 \ldots, e_{2\lfloor \frac{L-1}{2} \rfloor + 1}) \,:\, (e_1, e_2, e_3 \ldots, e_L) \in \mathcal{P}_C \right\}.$$

Then by the union bound, we have

$$\begin{aligned} \mathbf{Pr}\left[\mathcal{B}_L\right] &\leq \sum_{C \in \mathcal{C}} \sum_{P \in \mathcal{P}_{C,L}} \mathbf{Pr}\left[P \text{ is open}\right] \leq \sum_{C \in \mathcal{C}} \sum_{P \in \mathcal{P}^2_{C,\lfloor \frac{L+1}{2} \rfloor}} \mathbf{Pr}\left[P \text{ is open}\right] \\ &= \sum_{C \in \mathcal{C}} \sum_{(e_1, \ldots, e_{\lfloor \frac{L+1}{2} \rfloor}) \in \mathcal{P}^2_{C,\lfloor \frac{L+1}{2} \rfloor}} \prod_{i=1}^{\lfloor \frac{L+1}{2} \rfloor} 2^{-|C(e_i)|}, \end{aligned} \tag{2}$$

where the last equality follows from the fact that $e_i \cap e_{i+1} = \varnothing$ holds for every pair of consecutive vertices of a path in $\mathcal{P}^2_{C,\lfloor \frac{L+1}{2} \rfloor}$.

▶ **Lemma 10.** *If there exists a function $x : \mathcal{C} \to (0,1)$ satisfying*

$$\forall C \in \mathcal{C} : 2|C| \cdot 2^{-|C|} \leq (1-\varepsilon) \cdot x(C) \cdot \prod_{C' \in \Gamma^+_{\Phi^2}(C)} (1 - x(C')),$$

then

$$\sum_{C \in \mathcal{C}} \sum_{(e_1, \ldots, e_{\lfloor \frac{L+1}{2} \rfloor}) \in \mathcal{P}^2_{C,\lfloor \frac{L+1}{2} \rfloor}} \prod_{i=1}^{\lfloor \frac{L+1}{2} \rfloor} 2^{-|C(e_i)|} \leq \sum_{C \in \mathcal{C}} \frac{x(C)}{1 - x(C)} \cdot (1-\varepsilon)^{\lfloor \frac{L+1}{2} \rfloor}.$$

Lemma 9 is clearly a consequence of Lemma 10. We prove the latter by analyzing a *multi-type Galton-Watson branching process*.

4.1.1 Multi-type Galton-Watson branching process

Recall $\Phi = (V, \mathcal{C})$ is a fixed hypergraph and we assume that there exists a function $x : \mathcal{C} \to (0,1)$ assigning each $C \in \mathcal{C}$ a number in $(0,1)$. A $\mathcal{C}$-labelled tree is a tuple $\tau = (V_\tau, E_\tau, \mathcal{L})$ where $\mathcal{L} : V_\tau \to \mathcal{C}$ labels each vertex in V_τ with a hyperedge in $\mathcal{C}$.

Let $C \in \mathcal{C}$ be a hyperedge. Consider the following process which generates a random $\mathcal{C}$-labelled tree with the root labelled with C:

- First produce a root vertex u with label C. Initialize the active set A as $\{u\}$.
- Repeat the following until A is empty:
  - Pick some $u \in A$.
  - For every $C' \in \Gamma^+_{\Phi^2}(\mathcal{L}(u))$: create a new child for u labelled with C' *with probability* $x(C')$ independently; Add the new child to A.
  - Remove u from A.

We let $\mathcal{T}_C$ be the set of all labelled trees that can be generated by the above process and let $\mu_{\mathcal{T}_C}$ be the distribution over $\mathcal{T}_C$ induced by the process.

▶ **Lemma 11.** *For every labelled tree $\tau = (V_\tau, E_\tau, \mathcal{L}) \in \mathcal{T}_C$, it holds that*

$$\mu_{\mathcal{T}_C}(\tau) = \frac{1 - x(C)}{x(C)} \cdot \prod_{v \in V_\tau} x(\mathcal{L}(v)) \cdot \prod_{C' \in \Gamma^+_{\Phi^2}(\mathcal{L}(v))} (1 - x(C')).$$

Proof. For a vertex $v \in V_\tau$ we use $W_v \subseteq \Gamma^+_{\Phi^2}(\mathcal{L}(v))$ to denote the set of labels that do not occur as a label of some child nodes of v. Then clearly, the probability that the Galton-Watson branching process produces exactly the tree τ is given by

$$\mu_{\mathcal{T}_C}(\tau) = \frac{1}{x(C)} \prod_{v \in V_\tau} x(\mathcal{L}(v)) \prod_{C' \in W_v} (1 - x(C')).$$

In order to get rid of the W_v, we can rewrite the expression as

$$\begin{aligned}\mu_{\mathcal{T}_C}(\tau) &= \frac{1 - x(C)}{x(C)} \prod_{v \in V_\tau} \frac{x(\mathcal{L}(v))}{1 - x(\mathcal{L}(v))} \prod_{C' \in \Gamma^+_{\Phi^2}(\mathcal{L}(v))} (1 - x(C')) \\ &= \frac{1 - x(C)}{x(C)} \prod_{v \in V_\tau} x(\mathcal{L}(v)) \prod_{C' \in \Gamma_{\Phi^2}(\mathcal{L}(v))} (1 - x(C')).\end{aligned}$$

◀

For every $\ell \geq 1$, we also use $\mathcal{T}_{C,\ell}$ to denote the set of labelled trees in $\mathcal{T}_C$ containing exactly ℓ vertices. In the following, we slightly abuse notation and say that a labelled tree $\tau = (V_\tau, E_\tau, \mathcal{L}) \in \mathcal{T}_C$ (or $\mathcal{T}_{C,\ell}$) if there exists some $\tau' \in \mathcal{T}_C$ (or $\mathcal{T}_{C,\ell}$) such that τ and τ' are isomorphic.

4.1.2 Proof of Lemma 10

Let $\ell = \lfloor \frac{L+1}{2} \rfloor$. It follows from the assumption that for every $C \in \mathcal{C}$,

$$\begin{aligned}&\sum_{P=(e_1,e_2,\ldots,e_\ell) \in \mathcal{P}^2_{C,\ell}} \prod_{i=1}^{\ell} 2^{-|C(e_i)|} \\ &\leq (1-\varepsilon)^\ell \cdot \sum_{P=(e_1,e_2,\ldots,e_\ell) \in \mathcal{P}^2_{C,\ell}} \prod_{i=1}^{\ell} \frac{1}{2|C(e_i)|} \cdot x(C(e_i)) \prod_{C' \in \Gamma_{\Phi^2}(C(e_i))} (1 - x(C')). \qquad (3)\end{aligned}$$

We now define a mapping $\Psi : \mathcal{P}^2_{C,\ell} \to \mathcal{T}_{C,\ell}$ that maps each $P = (e_1, e_2, \ldots, e_\ell)$ to a labelled directed path $\tau = (V_\tau, E_\tau, \mathcal{L})$ where

- $V_\tau = \{u_1, u_2, \ldots, u_\ell\}$;
- $E_\tau = \{(u_i, u_{i+1}) : i \in [\ell - 1]\}$; and
- for every $i \in [\ell]$, $\mathcal{L}(u_i) = C(e_i)$.

The mapping Ψ is not necessarily an injection, but we can bound its multiplicity.

▶ **Lemma 12.** *For every labelled tree $\tau = (V_\tau, E_\tau, \mathcal{L}) \in \mathcal{T}_{C,\ell}$ with $V_\tau = \{u_1, \ldots, u_\ell\}$ and $E_\tau = \{(u_i, u_{i+1}) : i \in [\ell - 1]\}$, it holds that*

$$\left|\Psi^{-1}(\tau)\right| \leq 2^{\ell-1} \cdot \prod_{i=1}^{\ell} |\mathcal{L}(u_i)|.$$

Proof. We prove it by induction:

1. When $\ell = 1$, the lemma holds because there are at most $|C|$ timestamps e with label C such that $e \cap (T - n + 1, T] \neq \varnothing$;
2. We assume the statement holds for $\ell \geq 1$. Let $V_\tau = \{u_1, u_2, \ldots, u_\ell, u_{\ell+1}\}$ and $V_{\tau'} = \{u_1, u_2, \ldots, u_\ell\}$. According to the definition of Ψ, we know that for all $P \in \Psi^{-1}(\tau)$, it was extended from some $P' \in \Psi^{-1}(\tau')$. Therefore, it suffices to analyze the possible extension from P' to $P \in \Psi^{-1}(\tau)$. Assuming $P' = (e_1, e_2, \ldots, e_\ell)$ and $P = \{e_1, e_2, \ldots, e_\ell, e_{\ell+1}\}$, if P

was extended from P', then $\texttt{dist}(e_\ell, e_{\ell+1}) = 2$ and $C(e_{\ell+1}) = \mathcal{L}(u_{\ell+1})$. Therefore, there are at most $2|\mathcal{L}(u_{\ell+1})|$ timestamps $e_{\ell+1}$ satisfying $P \in \Psi^{-1}(\tau)$ according to Lemma 4. Combining with the induction hypothesis, the statement holds for $\ell + 1$. ◀

Clearly for those $\tau \in \mathcal{T}_{C,\ell}$ which are not directed paths, its pre-image $\Psi^{-1}(\tau) = \varnothing$. Therefore, it follows from Lemma 11 and Lemma 12 that

$$\begin{aligned}
&\sum_{P=(e_1,e_2,\ldots,e_\ell)\in\mathcal{P}^2_{C,\ell}} \prod_{i=1}^{\ell} \frac{1}{2|C(e_i)|} \cdot x(C(e_i)) \prod_{C'\in\Gamma_{\Phi^2}(C(e_i))} (1 - x(C')) \\
&\leq \sum_{\substack{\tau=(V_\tau,E_\tau,\mathcal{L})\in\mathcal{T}_{C,\ell}:\\ V_\tau=\{u_1,\ldots,u_\ell\}}} \prod_{i=1}^{\ell} x(\mathcal{L}(u_i)) \prod_{C'\in\Gamma_{\Phi^2}(\mathcal{L}(u_i))} (1 - x(C')) \\
&= \frac{x(C)}{1 - x(C)} \sum_{\tau\in\mathcal{T}_{C,\ell}} \mu_{\mathcal{T}_C}(\tau) \\
&< \frac{x(C)}{1 - x(C)}.
\end{aligned}$$

4.2 Refined analysis for uniform hypergraphs

If the instance $\Phi = (V, \mathcal{C})$ is a d-regular k-uniform hypergraph, we can choose $x(C) = \frac{1}{d^2k^2}$ in Lemma 9 and the condition becomes $d \leq \frac{c\cdot 2^{k/2}}{k^{1.5}}$ for some constant $c > 0$. In this section, we present a refined analysis for this case which removes the denominator $k^{1.5}$.

▶ **Lemma 13.** *For all $\varepsilon \in (0, 1)$ and $k \geq 2$, if*

$$d \leq \left(\frac{1}{4}\sqrt{\frac{9-\varepsilon}{2}} - \frac{1}{2}\right) \cdot 2^{k/2}$$

then $\mathbf{Pr}[\mathcal{B}_L] \leq \frac{k}{2^k} \cdot m \cdot (1-\varepsilon)^{\lfloor \frac{L}{2}-1 \rfloor}$ for all $L \in \mathbb{N}$.

For every induced path $(u_1, \ldots, u_\ell)$ in H_T, and $L > 0$, we use $\mathcal{P}_{(u_1,\ldots,u_\ell),L}$ to denote the collection of induced paths in H_T of length L with prefix $(u_1, \ldots, u_\ell)$[2].

▶ **Lemma 14.** *For all $\varepsilon \in (0, 1)$ and $k \geq 2$, if*

$$d \leq \left(\frac{1}{4}\sqrt{\frac{9-\varepsilon}{2}} - \frac{1}{2}\right) \cdot 2^{k/2}$$

then for every $u \in V_{H_T}$ and every $L \in \mathbb{N}$,

$$\mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(u),L} \;\middle|\; u \text{ is open}\right] \leq (1-\varepsilon)^{\lfloor \frac{L}{2}-1 \rfloor}.$$

We first show that Lemma 13 follows from Lemma 14.

Proof of Lemma 13. By the union bound, we have

$$\begin{aligned}
\mathbf{Pr}[\mathcal{B}_L] &\leq \sum_{C\in\mathcal{C}} \sum_{e_{C,t}:t\in(T-n,T]} \mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(e_{C,t}),L}\right] \\
&= \sum_{C\in\mathcal{C}} \sum_{e_{C,t}:t\in(T-n,T]} \mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(e_{C,t}),L} \;\middle|\; e_{C,t} \text{ is open}\right] \cdot \mathbf{Pr}\left[e_{C,t} \text{ is open}\right] \\
&\leq km(1-\varepsilon)^{\lfloor \frac{L}{2}-1 \rfloor} \cdot 2^{-k}.
\end{aligned}$$

◀

[2] Unlike the notation $\mathcal{P}_{C,L}$ defined in Section 4.1, we do not require $u_1 \cap \{T - n + 1, \ldots, T\} \neq \varnothing$ here.

The remaining part of the section is devoted to a proof of Lemma 14. We apply induction on L. The base case is that $L = 1$ and $L = 2$, in which the lemma trivially holds. For larger L and every path $P = (u_1, u_2, u_3 \dots, u_L) \in \mathcal{P}_{(u_1),L}$, we discuss how the vertices u_1, u_2 and u_3 overlap.

Recall for every $u = e_{C,t} \in V_T$, $C(u) = C$ is its label. Similar to [10], we classify the tuple (u_1, u_2, u_3) into three categories.

- We say u_2 is *good* if $|C(u_2) \cap C(u_1)| \leq \alpha \cdot k$ where $\alpha \in [0, 1]$ is a parameter to be set;
- If u_2 is not good, we say (u_2, u_3) is of *type I* if $C(u_3) \cap C(u_1) \neq \varnothing$;
- If u_2 is not good, we say (u_2, u_3) is of *type II* if $C(u_3) \cap C(u_1) = \varnothing$.

Then we can write

$$\begin{aligned}
&\mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(u_1),L} \;\middle|\; u_1 \text{ is open}\right] \qquad (4)\\
\leq\ &\mathbf{Pr}\left[\exists \text{ open } P = (u_1, u_2, u_3, \dots) \in \mathcal{P}_{(u_1),L} \text{ where } u_2 \text{ is good} \;\middle|\; u_1 \text{ is open}\right]\\
&+ \mathbf{Pr}\left[\exists \text{ open } P = (u_1, u_2, u_3, \dots) \in \mathcal{P}_{(u_1),L} \text{ where } (u_2, u_3) \text{ is of type I} \;\middle|\; u_1 \text{ is open}\right]\\
&+ \mathbf{Pr}\left[\exists \text{ open } P = (u_1, u_2, u_3, \dots) \in \mathcal{P}_{(u_1),L} \text{ where } (u_2, u_3) \text{ is of type II} \;\middle|\; u_1 \text{ is open}\right]
\end{aligned}$$

In the following, we bound the probabilities in the three cases respectively.

u_2 is good

In this case, we have $|C(u_2) \cap C(u_1)| \leq \alpha \cdot k$. Therefore, conditioned on u_1 being open, at least $(1 - \alpha) \cdot k$ variables in u_2 are free (i.e. independent of those variables related to u_1). The number of choices of $C(u_2)$ is at most $k \cdot d$ and the number of choices of u_2 with fixed $C = C(u_2)$ is at most k. Combining this with the induction hypothesis, we have

$$\begin{aligned}
&\mathbf{Pr}\left[\exists \text{ open } P = (u_1, u_2, u_3, \dots) \in \mathcal{P}_{(u_1),L} \text{ where } u_2 \text{ is good} \;\middle|\; u_1 \text{ is open}\right]\\
=\ &\mathbf{Pr}\left[\bigcup_{\text{good } u_2^*} \left[\exists \text{ open } P \in \mathcal{P}_{(u_1,u_2^*),L}\right] \;\middle|\; u_1 \text{ is open}\right]\\
\leq\ &\sum_{\text{good } u_2^*} \mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(u_1,u_2^*),L} \;\middle|\; u_1 \text{ is open}\right]\\
=\ &\sum_{\text{good } u_2^*} \mathbf{Pr}\left[u_2^* \text{ is open} \mid u_1 \text{ is open}\right] \cdot \mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(u_1,u_2^*),L} \;\middle|\; (u_1, u_2^*) \text{ is open}\right]\\
=\ &\sum_{\text{good } u_2^*} \mathbf{Pr}\left[u_2^* \text{ is open} \mid u_1 \text{ is open}\right] \cdot \mathbf{Pr}\left[\exists \text{ open } P' \in \mathcal{P}_{(u_2^*),L-1} \;\middle|\; u_2^* \text{ is open}\right]\\
\leq\ &k^2 d \cdot 2^{-(1-\alpha)k} \cdot (1 - \varepsilon)^{\lfloor \frac{L-3}{2} \rfloor}. \qquad (5)
\end{aligned}$$

(u_2, u_3) is of type I

In this case, we know $C(u_3)$ is adjacent to $C(u_1)$ in Φ and therefore the number of the choices of $C(u_3)$ is at most kd. The choice of u_3 with fixed $C = C(u_3)$ is at most $2k$ by Lemma 4. On the other hand, since P is an induced path, $u_1 \cap u_3 = \varnothing$ and $\mathbf{Pr}\left[u_3 \text{ is open} \mid u_1 \text{ is open}\right] = 2^{-k}$. Combining these facts with the induction hypothesis, we have

$$
\begin{aligned}
&\mathbf{Pr}\left[\exists \text{ open } P=(u_1,u_2,u_3,\dots)\in\mathcal{P}_{(u_1),L} \text{ where } (u_2,u_3) \text{ is of type I} \;\middle|\; u_1 \text{ is open}\right]\\
\le\ &\mathbf{Pr}\left[\bigcup_{u_3^*:\exists u_2,\ (u_2,u_3^*) \text{ type I}} \left[\exists \text{ open } P'\in\mathcal{P}_{(u_3^*),L-2}\right] \;\middle|\; u_1 \text{ is open}\right]\\
\le\ &\sum_{u_3^*:\exists u_2,\ (u_2,u_3^*) \text{ type I}} \mathbf{Pr}\left[\exists \text{ open } P'\in\mathcal{P}_{(u_3^*),L-2} \;\middle|\; u_1 \text{ is open}\right]\\
=\ &\sum_{u_3^*:\exists u_2,\ (u_2,u_3^*) \text{ type I}} \mathbf{Pr}\left[u_3^* \text{ is open} \;\middle|\; u_1 \text{ is open}\right]\cdot\mathbf{Pr}\left[\exists \text{ open } P'\in\mathcal{P}_{(u_3^*),L-2} \;\middle|\; (u_1,u_3^*) \text{ is open}\right]\\
=\ &\sum_{u_3^*:\exists u_2,\ (u_2,u_3^*) \text{ type I}} \mathbf{Pr}\left[u_3^* \text{ is open} \;\middle|\; u_1 \text{ is open}\right]\cdot\mathbf{Pr}\left[\exists \text{ open } P'\in\mathcal{P}_{(u_3^*),L-2} \;\middle|\; u_3^* \text{ is open}\right]\\
\le\ &2k^2d\cdot 2^{-k}\cdot(1-\varepsilon)^{\lfloor\frac{L-4}{2}\rfloor}. \qquad (6)
\end{aligned}
$$

(u_2, u_3) is of type II

This case is more complicated. We will enumerate all type II pairs (u_2, u_3) and bound the sum of probabilities that some path in $\mathcal{P}_{(u_2,u_3),L-1}$ is open conditioned on u_1 being open. To this end, we first enumerate all pairs $(C_2, C_3) \in \mathcal{C}^2$ and then consider all pairs (u_2, u_3) with $C(u_2) = C_2$ and $C(u_3) = C_3$.

For a fixed pair (C_2, C_3), we aim to bound the probability

$$
\sum_{\substack{(u_2,u_3) \text{ is of type II,}\\ C(u_2)=C_2, C(u_3)=C_3}} \mathbf{Pr}\left[\exists \text{ open } P\in\mathcal{P}_{(u_1,u_2,u_3),L} \;\middle|\; u_1 \text{ is open}\right]. \qquad (7)
$$

To ease the notation, we let $C_1 := C(u_1)$. In order for the above sum to be nonzero, we must have $|C_2 \cap C_1| > \alpha \cdot k$ and $C_3 \cap C_1 = \varnothing$. Let $a := |C_2 \cap C_1|$ and $b := |C_3 \cap C_2|$. Recall the notations in Section 3.1. For those (u_2, u_3) with $C(u_2) = C_2$ and $C(u_3) = C_3$, we can write them as $u_2 = e_{C_2,t_2}$ and $u_3 = e_{C_3,t_3}$ respectively. In the following, all pairs (u_2, u_3) discussed are of type II.

Note that $e_{C_2,t_2} =: u_2 \neq u_2' := e_{C_2,t_2'}$ if and only if $t_2 \neq t_2'$. Similarly $e_{C_2,t_2} =: u_3' \neq u_3' := e_{C_2,t_2'}$ if and only if $t_3 \neq t_3'$. Moreover, if $u_2 \neq u_2'$, then $|u_2 \cap u_1| \neq |u_2' \cap u_1|$ and similarly if $u_3 \neq u_3'$, then $|u_3 \cap u_2| \neq |u_3' \cap u_2|$. As a result, we can enumerate type II pairs (u_2, u_3) by enumerating the sizes of $u_2 \cap u_1$ and $u_3 \cap u_2$ respectively. On the other hand, if we know that $|u_2 \cap u_1| = i$ and $|u_3 \cap u_2| = j$, then,

$$
\begin{aligned}
&\mathbf{Pr}\left[\exists \text{ open } P\in\mathcal{P}_{(u_1,u_2,u_3),L} \;\middle|\; u_1 \text{ is open}\right]\\
=\ &\mathbf{Pr}\left[u_2 \text{ is open} \mid u_1 \text{ is open}\right]\cdot\mathbf{Pr}\left[u_3 \text{ is open} \mid (u_1,u_2) \text{ is open}\right]\\
&\cdot\mathbf{Pr}\left[\exists \text{ open } P\in\mathcal{P}_{(u_1,u_2,u_3),L} \;\middle|\; (u_1,u_2,u_3) \text{ is open}\right]\\
\le\ &\mathbf{Pr}\left[u_2 \text{ is open} \mid u_1 \text{ is open}\right]\cdot\mathbf{Pr}\left[u_3 \text{ is open} \mid u_2 \text{ is open}\right]\\
&\cdot\mathbf{Pr}\left[\exists \text{ open } P'\in\mathcal{P}_{(u_3),L-2} \;\middle|\; u_3 \text{ is open}\right]\\
=\ &2^{-(k-i)}\cdot 2^{-(k-j)}\cdot\mathbf{Pr}\left[\exists \text{ open } P'\in\mathcal{P}_{(u_3),L-2} \;\middle|\; u_3 \text{ is open}\right].
\end{aligned}
$$

These facts together with the induction hypothesis give

$$\begin{aligned}
&\sum_{\substack{(u_2,u_3) \text{ is of type II:}\\ C(u_2)=C_2, C(u_3)=C_3}} \mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(u_1,u_2,u_3),L} \;\middle|\; u_1 \text{ is open}\right] \\
&\le \sum_{i=1}^{a}\sum_{j=1}^{b} 2^{-(k-i)} \cdot 2^{-(k-j)} \cdot (1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor} \\
&\le 2^{a+b-2k} \cdot 4(1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor}.
\end{aligned}$$

It remains to enumerate (C_2, C_3) pairs. Since we know that $|C_2 \cap C_1| > \alpha \cdot k$, the number of choices of C_2 is at most $\frac{d \cdot k}{\alpha \cdot k} = \frac{d}{\alpha}$[3]. For every fixed C_2, we let $\mathcal{C} = \left\{C_3^{(1)}, C_3^{(2)}, \ldots, C_3^{(m)}\right\}$ be the collection of all possible C_3. Denote $a := |C_2 \cap C_1|$ as before and for every $i \in [m]$ let $b_i := \left|C_3^{(i)} \cap C_2\right|$. The probability of interest can therefore be written as

$$\begin{aligned}
&\mathbf{Pr}\left[\exists \text{ open } (u_1,u_2,u_3,\ldots) \in \mathcal{P}_{(u_1),L} \text{ where } (u_2,u_3) \text{ is of type II and } C(u_2)=C_2 \;\middle|\; u_1 \text{ is open}\right] \\
&\le \sum_{\substack{(u_1,u_2,u_3,\ldots)\in\mathcal{P}_{(u_1),L}:\\ (u_2,u_3) \text{ is of type II,}\\ C(u_2)=C_2}} \mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(u_1,u_2,u_3),L} \;\middle|\; u_1 \text{ is open}\right] \\
&\le \left(\sum_{i=1}^{m} 2^{a+b_i-2k}\right) \cdot 4(1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor}.
\end{aligned}$$

To bound the term $\sum_{i=1}^{m} 2^{a+b_i-2k}$, let us list some properties of the numbers involved:

- $a \le k$ and $1 \le m \le (k-a)d$;
- $1 \le b_i \le k-a$ for all $i \in [m]$;
- $\sum_{i=1}^{m} b_i \le (k-a)d$. (This can be seen by an argument similar to the last footnote and the fact that $C_3 \cap C_1 = \varnothing$.)

Roughly the numbers are acting against each other: If a and m are large then the b_i's are small. So it is possible to control $\sum_{i=1}^{m} 2^{a+b_i-2k}$ reasonably:

▶ **Lemma 15.** *Suppose $a \le k$ and $1 \le m \le (k-a)d$. Then for any integers $b_1, \ldots, b_m \in [k-a]$ such that $\sum_{i=1}^{m} b_i \le (k-a)d$, we have the inequality $\sum_{i=1}^{m} 2^{a+b_i-2k} \le d \cdot 2^{-k}$.*

We arrange the proof in Appendix A. Returning to previous discussion, we derive

$$\begin{aligned}
&\mathbf{Pr}\left[\exists \text{ open } P = (u_1,u_2,u_3,\ldots) \in \mathcal{P}_{(u_1),L} \text{ where } (u_2,u_3) \text{ is of type II} \;\middle|\; u_1 \text{ is open}\right] \\
&\le \frac{4d^2}{\alpha} \cdot 2^{-k} \cdot (1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor}. \qquad (8)
\end{aligned}$$

Putting all together

Plugging Equations (5), (6) and (8) into Equation (4), we have

$$\begin{aligned}
&\mathbf{Pr}\left[\exists \text{ open } P \in \mathcal{P}_{(u_1),L} \;\middle|\; u_1 \text{ is open}\right] \\
&\le \left(k^2 d \cdot 2^{-(1-\alpha)k} + 2k^2 d \cdot 2^{-k} + \frac{4d^2}{\alpha} \cdot 2^{-k}\right) \cdot (1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor} \\
&< \left(2^{\alpha k+2} k^2 d + \frac{4}{\alpha} d^2\right) 2^{-k} \cdot (1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor}
\end{aligned}$$

[3] To see this, consider the following way to enumerate all C_2 incident to C_1 in Φ: First pick a vertex in C_1 and then pick one of its incident hyperedges. This way we enumerated (with repetitions) $k \cdot d$ hyperedges in total, and every hyperedge $C_2 : |C_2 \cap C_1| > \alpha \cdot k$ is enumerated at least $\alpha \cdot k$ times.

where we used a very crude inequality $2^{\alpha k}+2 < 2^{\alpha k+2}$. Let us take $\alpha = \alpha(k) := \frac{1}{2} + \frac{3-2\log k}{k} \in [0,1]$ for any $k \geq 2$. Note that $2^{\alpha k+2}k^2 = 2^{5+k/2}$ by definition. On the other hand, from basic calculus one may find that $\alpha(k) : k \in \mathbb{N}$ attains minimum value $1/8$ at $k = 8$, thus $4/\alpha \leq 32 = 2^5$. Hence the above bound continues as:

$$\begin{aligned}\cdots &\leq \left(2^{k/2}d + d^2\right) 2^{5-k} \cdot (1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor} \\ &\leq (1-\varepsilon) \cdot (1-\varepsilon)^{\lfloor \frac{L-4}{2} \rfloor} \\ &\leq (1-\varepsilon)^{\lfloor \frac{L-2}{2} \rfloor}\end{aligned}$$

where the second line is due to our assumption $d \leq \left(\frac{1}{4}\sqrt{\frac{9-\varepsilon}{2}} - \frac{1}{2}\right) \cdot 2^{k/2} =: d^*$. To see this, observe that the function $g(d) := 2^{k/2}d + d^2$ is monotonically increasing when $d \geq 0$, and that $g(d^*) = (1-\varepsilon) \cdot 2^{k-5}$.

▶ Remark 16. When $\varepsilon := 0.1$, say, the condition becomes (approximately) $d \leq 0.027 \cdot 2^{k/2}$. Taking smaller ε will allow larger applicable range of d, but it comes at the price of slowing down the sampler (by a multiplicative log-factor; see Theorem 8).

If k is sufficiently large we can further improve our bound to $d \lesssim \frac{\sqrt{33-\varepsilon}-1}{16} \cdot 2^{k/2}$.

References

1. Ivona Bezáková, Andreas Galanis, Leslie Ann Goldberg, Heng Guo, and Daniel Stefankovic. Approximation via correlation decay when strong spatial mixing fails. *SIAM Journal on Computing*, 48(2):279–349, 2019.
2. Weiming Feng, Heng Guo, and Jiaheng Wang. Improved bounds for randomly colouring simple hypergraphs. *arXiv preprint*, 2022. `arXiv:2202.05554`.
3. Weiming Feng, Heng Guo, Yitong Yin, and Chihao Zhang. Fast sampling and counting k-sat solutions in the local lemma regime. *Journal of the ACM*, 68(6):40:1–40:42, 2021.
4. Weiming Feng, Kun He, and Yitong Yin. Sampling constraint satisfaction solutions in the local lemma regime. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1565–1578, 2021.
5. Andreas Galanis, Leslie Ann Goldberg, Heng Guo, and Kuan Yang. Counting solutions to random cnf formulas. *SIAM Journal on Computing*, 50(6):1701–1738, 2021.
6. Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovász local lemma. *Journal of the ACM*, 66(3):1–31, 2019.
7. Heng Guo, Chao Liao, Pinyan Lu, and Chihao Zhang. Counting hypergraph colorings in the local lemma regime. *SIAM Journal on Computing*, 48(4):1397–1424, 2019.
8. Kun He, Xiaoming Sun, and Kewen Wu. Perfect sampling for (atomic) Lovász local lemma. *arXiv preprint*, 2021. `arXiv:2107.03932`.
9. Kun He, Chunyang Wang, and Yitong Yin. Sampling Lovász local lemma for general constraint satisfaction solutions in near-linear time. *arXiv preprint*, 2022. `arXiv:2204.01520`.
10. Jonathan Hermon, Allan Sly, and Yumeng Zhang. Rapid mixing of hypergraph independent sets. *Random Structures & Algorithms*, 54(4):730–767, 2019.
11. Vishesh Jain, Huy Tuan Pham, and Thuy-Duong Vuong. On the sampling Lovász local lemma for atomic constraint satisfaction problems. *arXiv preprint*, 2021. `arXiv:2102.08342`.
12. Vishesh Jain, Huy Tuan Pham, and Thuy Duong Vuong. Towards the sampling Lovász local lemma. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science*, 2021.
13. Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.
14. Eyal Lubetzky and Allan Sly. Information percolation and cutoff for the stochastic ising model. *Journal of the American Mathematical Society*, 29(3):729–774, 2016.

15 Ankur Moitra. Approximate counting, the Lovász local lemma, and inference in graphical models. *Journal of the ACM*, 66(2):10:1–10:25, 2019.

16 Robin A Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. *Journal of the ACM*, 57(2):1–15, 2010.

17 James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Structures & Algorithms*, 9(1-2):223–252, 1996.

A Proof of Lemma 15

For each $j \in [k-a]$ we introduce a counter $x_j := |\{i : b_i = j\}|$ that counts the number of b_i's taking a specific value j. Then the constraint of the lemma translates to

$$\sum_{j=1}^{k-a} j \cdot x_j \leq (k-a)d,$$

and we want to show

$$\sum_{j=1}^{k-a} 2^{a+j-2k} \cdot x_j \leq d \cdot 2^{-k}.$$

To this end, we consider a (relaxed) linear program with variables $x_1, \ldots, x_{k-a}$:

$$\begin{aligned} \max \quad & \sum_{j=1}^{k-a} 2^{a+j-2k} \cdot x_j \\ \text{s.t} \quad & \sum_{j=1}^{k-a} j \cdot x_j \leq (k-a)d \\ & x_j \geq 0 \quad (\forall j \in [k-a]). \end{aligned}$$

By the *strong duality theorem of linear programming*, its maximum value equals the minimum value of the dual program

$$\begin{aligned} \min \quad & (k-a)d \cdot y \\ \text{s.t} \quad & j \cdot y \geq 2^{a+j-2k} \quad (\forall j \in [k-a]) \\ & y \geq 0. \end{aligned}$$

Clearly the minimum value is obtained when y is as small as possible. It is clear that the sequence $h_j := \frac{2^j}{j}$ is monotonically increasing for integers $j \geq 1$, so the minimum possible y is given by the $(k-a)$-th constraint, namely

$$y^* := \frac{2^{a+(k-a)-2k}}{k-a} = \frac{2^{-k}}{k-a}.$$

Therefore, the minimum value of the dual – also the maximum value of the primal – is exactly $d \cdot 2^{-k}$, as desired.

Threshold Rates of Code Ensembles: Linear Is Best

Nicolas Resch
Cryptology Group, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Chen Yuan
School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China

Abstract

In this work, we prove new results concerning the combinatorial properties of random linear codes. By applying the thresholds framework from Mosheiff et al. (FOCS 2020) we derive fine-grained results concerning the list-decodability and -recoverability of random linear codes.

Firstly, we prove a lower bound on the list-size required for random linear codes over $\mathbb{F}_q$ ε-close to capacity to list-recover with error radius ρ and input lists of size ℓ. We show that the list-size L must be at least $\frac{\log_q \binom{q}{\ell} - R}{\varepsilon}$, where R is the rate of the random linear code. This is analogous to a lower bound for list-decoding that was recently obtained by Guruswami et al. (IEEE TIT 2021B). As a comparison, we also pin down the list size of random codes which is $\frac{\log_q \binom{q}{\ell}}{\varepsilon}$. This result almost closes the $O(\frac{q \log L}{L})$ gap left by Guruswami et al. (IEEE TIT 2021A). This leaves open the possibility (that we consider likely) that random linear codes perform better than the random codes for list-recoverability, which is in contrast to a recent gap shown for the case of list-recovery from erasures (Guruswami et al., IEEE TIT 2021B).

Next, we consider list-decoding with constant list-sizes. Specifically, we obtain new lower bounds on the rate required for:

- List-of-3 decodability of random linear codes over $\mathbb{F}_2$;
- List-of-2 decodability of random linear codes over $\mathbb{F}_q$ (for any q).

This expands upon Guruswami et al. (IEEE TIT 2021A) which only studied list-of-2 decodability of random linear codes over $\mathbb{F}_2$. Further, in both cases we are able to show that the rate is larger than that which is possible for uniformly random codes.

A conclusion that we draw from our work is that, for many combinatorial properties of interest, random linear codes actually perform *better* than uniformly random codes, in contrast to the apparently standard intuition that uniformly random codes are best.

2012 ACM Subject Classification Mathematics of computing → Coding theory

Keywords and phrases Random Linear Codes, List-Decoding, List-Recovery, Threshold Rates

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.104

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2205.01513`

Funding *Nicolas Resch*: Research supported in part by ERC H2020 grant No.74079 (ALGSTRONGCRYPTO).
Chen Yuan: Research supported in part by the National Natural Science Foundation of China under Grant 12101403, the National Natural Science Foundation of China under Grant 12031011 and National Key Research and Development Project 2021YFE0109900.

1 Introduction

Coding theory is concerned with developing efficient means to makes data robust to noise. The mathematical objects used for this purpose are *(error-correcting) codes*, which are just subsets $\mathcal{C} \subseteq \Sigma^n$, where Σ is a finite alphabet of size q. It is often convenient to set $\Sigma = \mathbb{F}_q$,

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 104; pp. 104:1–104:19


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where $\mathbb{F}_q$ is the finite field of order q,[1] in which case we can insist that $\mathcal{C}$ be a subspace of $\mathbb{F}_q^n$. We call such a code *linear* and denote it $\mathcal{C} \leq \mathbb{F}_q^n$. As we are mostly concerned with linear codes in the sequel we will always set $\Sigma = \mathbb{F}_q$.[2]

In order for a code to be useful for information transmission in noisy environments, we require $\mathcal{C}$ to satisfy noise-resilience properties, which amounts to insisting that the codewords are "difficult to confuse." A basic way to do this is to define a distance metric on $\mathbb{F}_q^n$ and then insist that the codewords are not too clustered. The standard choice is the (relative) *Hamming distance* which is defined as $d(\vec{x}, \vec{y}) = \frac{1}{n}|\{i \in [n] : x_i \neq y_i\}|$ for $\vec{x}, \vec{y} \in \mathbb{F}_q^n$: in words, it is the fraction of coordinates on which the vectors $\vec{x}$ and $\vec{y}$ differ. The *minimum distance* of a code is then the minimum distance between two distinct codewords, i.e., $\delta := \min\{d(\vec{c}, \vec{d}) : \vec{c}, \vec{d} \in \mathcal{C}, \vec{c} \neq \vec{d}\}$.

Beyond the minimum distance, there are other proxies for a code's noise-resilience that are widely studied. First and foremost, a popular relaxed notion of noise-resilience is provided by *list-decodability*, which informally asks that the code not be "too" clustered around any one point. More precisely, a code is said to be (ρ, L)-list-decodable if there are never L or more codewords that are all within distance ρ of some vector $z \in \mathbb{F}_q^n$, i.e.,

$$\forall \vec{z} \in \mathbb{F}_q^n,\ |\{\vec{x} \in \mathcal{C} : d(\vec{x}, \vec{z}) \leq \rho\}| < L\ .$$

The integer L is called the *list-size*. This notion, originally introduced by Elias and Wozencraft [6, 35], finds uses within coding theory and beyond in, e.g., complexity theory [27, 2, 33], cryptography [25], and learning theory [8].

We will also investigate another relaxation of list-decoding: *list-recovery*. Here, we are given a collection of input lists $S_1, \ldots, S_n \subseteq \mathbb{F}_q$ of bounded size, and the requirement is that there are not too many codewords that agree too much with these input lists. More precisely, for an integer $\ell \leq q$ we require that

$$\forall \vec{S} = (S_1, \ldots, S_n) \in \binom{\mathbb{F}_q}{\ell}^n,\ |\{\vec{x} \in \mathcal{C} : d(\vec{x}, \vec{S}) \leq \rho\}| < L\ .$$

In the above, we are denoting by $\binom{\mathbb{F}_q}{\ell}$ the family of all ℓ-element subsets of $\mathbb{F}_q$, and we are extending the Hamming distance notation $d(\cdot, \cdot)$ via

$$d(\vec{x}, \vec{S}) = \frac{1}{n}|\{i \in [n] : x_i \notin S_i\}|\ .$$

Note that $(\rho, 1, L)$-list-recovery is equivalent to list-decoding, demonstrating that list-recoverability is indeed a generalization of list-decodability. While list-recovery was initially introduced as a stepping stone towards list-decoding [11, 12, 13, 14] it has since found many new uses in theoretical computer science more broadly [20, 24, 29, 7, 22, 23].

In order for a code to provide for efficient information transmission, we would like for the code's *rate* to be as large as possible, which is a measure of the amount of information transmitted per symbol of a codeword. More precisely, the code's rate R is defined as $\frac{\log_q |\mathcal{C}|}{n}$; when the code is linear, this is simply $\frac{\dim(\mathcal{C})}{n}$.

At its core, coding theory is concerned with determining the achievable tradeoffs between a code's rate and its noise-resilience for various noise models. In this work, we focus upon the list-decodability and list-recoverability of codes. An important question we ask is how

[1] In this case, we will of course insist that q be a prime power.

[2] For nonlinear codes this does potentially lose some generality, as the alphabet size in that case could be any integer. We do remark that our results concerning arbitrary codes hold for all integer q, but emphasizing this point is not relevant to our purposes.

large the list-size L must be for these tasks. This is useful in practice, as the main constraint on the run time of most list-decoding/recovery algorithms is due to the need to process the list. Further, in applications of list-recoverable codes to constructions of expanders [20] the quality of the expansion is directly governed by the list-size.

Random Ensembles of Codes. As a stepping-stone towards a thorough understanding of the achievable tradeoffs (which is believed to be a very challenging problem), we take cues from much of the literature and study the behaviour of "typical" codes. That is, we sample codes of a prescribed rate according to natural distributions and investigate their list-decodability/-recoverability. In particular, we consider random *linear* codes, which are uniformaly sampled subspaces of $\mathbb{F}_q^n$ of the prescribed dimension. We also study uniformly random subsets of $\mathbb{F}_q^n$ of the prescribed size, which we call *random codes*.

In our work, we endeavour to provide a more fine-grained understanding of the combinatorial properties of these code ensembles. In this way, we help to uncover the landscape of achievable parameters for various code properties of interest. Beyond its theoretical interest, many code constructions [14, 19, 22, 23] use (small) linear codes as a component, and better list-decodability/recoverability of these inner codes improves these constructions.

In our results, we highlight a (perhaps surprising) fact: for list-decoding/recovery, random linear codes seem to perform *better* than uniformly random codes. On the one hand, even for the basic property of minimum distance it has already been observed that random linear codes (which achieve the Gilbert-Varshamov bound) outperform uniformly random codes. On the other hand, for problems such as list-decoding and list-recovery much of the literature appears to be focused on showing that random linear codes are "not too much worse" than uniformly random codes. We hope our work encourages the coding theory community to change perspective and endeavour to prove that random linear codes are in fact better.

1.1 Our Results

List-Recoverability of Random Linear Codes. As a first result, we provide a new lower bound on the list-size of random linear codes for list-recoverability. For context, we recall the list-recoverability capacity theorem, which gives us some coarse-grained information regarding achievable tradeoffs. For an integer $1 \leq \ell < q$, error-radius $\rho \in (1 - \ell/q)$ and $\varepsilon > 0$ it states the following:

- If $R \leq 1 - h_{q,\ell}(\rho) - \varepsilon$, there exist (ρ, ℓ, L)-list-recoverable codes with $L = O(\ell/\varepsilon)$.
- If $R \geq 1 - h_{q,\ell}(\rho) + \varepsilon$, there *do not* exist (ρ, ℓ, L)-list-recoverable codes with $L = o(q^{\varepsilon n})$.

In the above, the function $h_{q,\ell}(\cdot)$ is the (q, ℓ)-entropy function; its precise definition is not important at the moment so we defer it to Section 2. Informally, when studying codes of rate ε below the capacity for a small $\varepsilon > 0$ we refer to them as *capacity-approaching* and call ε as the *gap-to-capacity*.

This already tells us that the capacity for (ρ, ℓ, L)-list-recovery is $1 - h_{q,\ell}(\rho)$ if we insist that L be subexponential in n. However, we can ask for more fine-grained information: in particular, exactly how large must the list-size L be as a function of ε and the other parameters?

For random linear codes, we prove the following lower bound.

▶ **Theorem 1** (List-Recoverability Lower Bound for Random Linear Codes). *Let $1 \leq \ell \leq q$ be integers with q a prime power and fix $\rho \in (0, 1 - \ell/q)$. Fix $\delta > 0$. For sufficiently small $\varepsilon > 0$, a random linear code in $\mathbb{F}_q^n$ of rate $1 - h_{q,\ell}(\rho) - \varepsilon$ is whp not $\left(\rho, \ell, \lfloor \frac{\log_q \binom{q}{\ell} - (1 - h_{q,\ell}(\rho))}{\varepsilon} - \delta \rfloor\right)$-list-recoverable.*

Table 1 This table summarizes much of the work on the list-recoverability of random linear codes (RLC) and random codes (RC). The lower bound of [15] only applies when $q = p^{\Omega(1/\varepsilon)}$ for a prime p, and in [32] $\eta > 0$ is viewed as a small constant. [15] also offers a similar lower bound for the case of list-recovery from erasures.

Source	Model	Radius	Rate	List-size bound
Folklore	RC	$\rho > 0$	$1 - h_{q,\ell}(\rho) - \varepsilon$	$\leq O(\ell/\varepsilon)$
[37]	RLC	$\rho > 0$	$1 - h_{q,\ell}(\rho) - \varepsilon$	$\leq q^{O(\ell/\varepsilon)}$
[32]	RLC	$\rho = 1 - \frac{\ell}{q} - \eta$	$0.99(1 - h_{q/\ell}(\alpha) - \log_q(\ell))$	$\leq q^{O(\ln^2(\ell/\eta))}$
[15]	RLC	$\rho = 0$	$1 - \log_q(\ell) - \varepsilon$	$\geq \ell^{\Omega(1/\varepsilon)}$
Theorem 1	RLC	$\rho > 0$	$1 - h_{q,\ell}(\rho) - \varepsilon$	$> \frac{\log_q \binom{q}{\ell} - (1 - h_{q,\ell}(\rho))}{\varepsilon}$
Theorem 2	RC	$\rho > 0$	$1 - h_{q,\ell}(\rho) - \varepsilon$	$\approx \frac{\log_q \binom{q}{\ell}}{\varepsilon}$

For context, we consider the case of uniformly random codes. In this case, we obtain a tight result.

▶ **Theorem 2** (List-Recoverability for Random Codes). *Let $1 \leq \ell \leq q$ be integers with q a prime power and fix $\rho \in (0, 1 - \ell/q)$. Fix $\delta > 0$. For sufficiently small $\varepsilon > 0$, a random code in $\mathbb{F}_q^n$ of rate $1 - h_{q,\ell}(\rho) - \varepsilon$ is whp* not *$\left(\rho, \ell, \lfloor \frac{\log_q \binom{q}{\ell}}{\varepsilon} - \delta \rfloor\right)$-list-recoverable.*

On the other hand, for any $\varepsilon > 0$ and n sufficiently large, a random code in $\mathbb{F}_q^n$ of rate $1 - h_{q,\ell}(\rho) - \varepsilon$ is whp $\left(\rho, \ell, \lceil \frac{\log_q \binom{q}{\ell}}{\varepsilon} \rceil + 1\right)$-list-recoverable.

In this way, we pin down the list-recoverability for random codes to one of two or three possible values: $\lfloor \frac{\log_q \binom{q}{\ell}}{\varepsilon} + 0.99 \rfloor$, $\lceil \frac{\log_q \binom{q}{\ell}}{\varepsilon} \rceil$ (if it's different) or $\lceil \frac{\log_q \binom{q}{\ell}}{\varepsilon} \rceil + 1$.

Comparing Theorems 1 and 2 we see that our lower bound on random linear codes is less than the precise bound we have on random codes. One could potentially draw the conclusion that Theorem 1 should be improved. However, we believe that it is in fact tight. For the case of list-decoding binary codes it has already been shown that random linear performs better than uniformly random, and the bounds we obtain are the natural generalizations of the (tight) results for that case. We therefore conjecture that Theorem 1 is indeed tight. This stands in stark contrast to *erasure* list-recovery:[3] for this model, it is known that random linear codes can require lists of size $\ell^{\Omega(1/\varepsilon)}$ [15] (at least, if the field has large characteristic), whereas the lists for random codes can be just $O(\ell/\varepsilon)$. A summary of the state-of-the-art for list-recovery of RLCs and RCs is provided in Table 1.

▶ Remark 3. It might appear that our conjecture that random linear codes outperform random codes for list-recovery is contradicted by the result of [15]. However, we emphasize that the capacity for erasure list-recovery is larger, so if a code is ε-close to capacity for list-recovery from erasures for small $\varepsilon > 0$ it is above capacity for list-recovery from errors, the model we study. Hence, this lower bound does not contradict our conjecture. One can also consider the model where ρ approaches the limit $1 - \ell/q$ as is done in [32]; in this case we still suspect that random linear codes outperform uniformly random codes, but this is just speculation and further investigation is required.

[3] Here, the requirement is that for all subsets $S_1, \ldots, S_n \subseteq \mathbb{F}_q$ where at least $(1 - \rho)n$ of the S_i's satisfy $|S_i| \leq \ell$ (and the others may be all of $\mathbb{F}_q$), the number of codewords in $S_1 \times \cdots \times S_n$ is less than L.

List-decoding with small lists. Next, we turn our attention to the challenge of list-decoding when the output list-size L is a (small) constant. Thus, we are no longer in the regime where we can expect to approach the list-decoding capacity, and we are interested to know by how much we are required to back off if, say, $L = 3, 4$.

First, we consider the case where $L = 4$ for the binary field, which we also refer to as *list-of-*3 decoding. Here and throughout, we also use the following notation (which is slightly abusive): for $q \geq 2$ and nonnegative reals $x_1, \ldots, x_t$ with $x_1 + \cdots + x_t \leq 1$, $H_q(x_1, \ldots, x_t) = \sum_{i=1}^{t} x_i \log_q \frac{1}{x_i} + (1 - x_1 - \cdots - x_t) \log_q \frac{1}{1-x_1-\cdots-x_t}$.

We first prove the following possibility result for random linear codes. In the following,

$$\mathcal{B}_\rho = \left\{(x_1, x_2) \in \mathbb{R}^2 : x_1 + 2x_2 \leq 4\rho, x_1 + x_2 \leq 1, x_1, x_2 \geq 0\right\} .$$

▶ **Theorem 4** (List-of-3 decoding Random Linear Binary Codes). *Let $\rho \in (0, 5/16)$[4] and suppose*

$$R < 1 - \max_{(x_1,x_2)\in\mathcal{B}_\rho} \frac{H_2(x_1, x_2) + 2x_1 + x_2 \log_2 3}{3} .$$

Then a random linear code over $\mathbb{F}_q$ of rate R is whp $(\rho, 4)$-list-decodable.

For context, we also study the list-of-3 decodability of random codes over the binary alphabet. In this case, we can prove the following:

▶ **Theorem 5** (List-of-3 decoding Random Binary Codes). *Let $\rho \in (0, 5/16)$ and suppose*

$$R > 1 - \max_{(x_1,x_2)\in\mathcal{B}_\rho} \frac{1 + H_2(x_1, x_2) + 2x_1 + x_2 \log_2 3}{4} .$$

Then a random code over $\{0, 1\}$ of rate R is whp not *$(\rho, 4)$-list-decodable.*

On the other hand, if

$$R < 1 - \max_{(x_1,x_2)\in\mathcal{B}_\rho} \frac{1 + H_2(x_1, x_2) + 2x_1 + x_2 \log_2 3}{4} ,$$

then a random code over $\{0, 1\}$ is whp $(\rho, 4)$-list-decodable.

As $\frac{1+F}{4} \geq \frac{F}{3}$ whenever $F \leq 3$, we see that the bound in Theorem 4 is greater than the bound from Theorem 5. Using terminology that we later make precise, we see that the *threshold rate* for list-of-3 decoding binary random linear codes strictly exceeds that of binary random codes.

Next, we study list-of-2 decoding over alphabets of size $q > 2$. And again, our theorems demonstrate that random linear codes strictly outperform random codes. Define

$$\mathcal{D}_\rho := \{(x_1, x_2) \in \mathbb{R}^2 : x_1 + x_2 \leq 3\rho, x_1 + x_2 \leq 1, x_1, x_2 \geq 0\}.$$

▶ **Theorem 6** (List-of-2 decoding Random Linear q-ary Codes). *Let $\rho \in (0, 1/3)$ and suppose*

$$R < 1 - \max_{(x_1,x_2)\in\mathcal{D}_\rho} \frac{H_q(x_1, x_2) + x_1 \log_q 3(q-1) + x_2 \log_q (q-1)(q-2)}{2} .$$

Then a random linear code over $\mathbb{F}_q$ of rate R is whp $(\rho, 3)$-list-decodable.

[4] If $\rho \geq 5/16$ it is known that there are no $(\rho, 4)$-list-decodable codes with postive rate [1].

▶ **Theorem 7** (List-of-2 decoding Random q-ary Codes). *Let $\mathbb{F}_q$ be an alphabet of size q. Let $\rho \in (0, 1/3)$ and suppose*

$$R > 1 - \max_{(x_1,x_2)\in\mathcal{D}_\rho} \frac{1 + H_q(x_1, x_2) + x_1 \log_q 3(q-1) + x_2 \log_q (q-1)(q-2)}{3} .$$

Then a random code over $\mathbb{F}_q$ of rate R is whp not $(\rho, 3)$*-list-decodable.*

On the other hand, if

$$R < 1 - \max_{(x_1,x_2)\in\mathcal{D}_\rho} \frac{1 + H_q(x_1, x_2) + x_1 \log_q 3(q-1) + x_2 \log_q (q-1)(q-2)}{3} ,$$

then a random code over $\mathbb{F}_q$ is whp $(\rho, 3)$-list-decodable.

Again, we can see that the bound from Theorem 6 is greater than the bound from Theorem 7. We therefore conjecture that this phenomenon of random linear codes outperforming random codes extends to more values of L. To provide more evidence for this conjecture, we extend an argument for binary random linear codes of [10, 26] to larger values of L, and by comparing it to a computation of the threshold rate for random binary codes, show that for many parameter regimes of interest we do indeed have random linear codes outperforming random codes.

1.2 Techniques

In order to obtain our results, we rely on a recently developed toolkit for proving threshold rates for combinatorial properties of random (linear) codes. This toolkit was developed by Mosheiff et al. [28] on the way to proving that LDPC codes achieve list-decoding capacity; recent works [15, 16] have found further uses for the techniques in investing combinatorial properties of random linear codes. An analogous threshold toolkit for *random codes* was provided in [17].

Broadly speaking, the techniques of [28, 17] apply when considering a property of codes defined by forbidding a family of "bad" subsets, each of which have constant cardinality (independent of n). For example, the property of (ρ, L)-list-decodability is defined by forbidding all L-element subsets $B = \{x_1, \ldots, x_L\}$ of a Hamming ball $B(z, \rho) = \{x \in \mathbb{F}_q^n : d(x, z) \leq \rho\}$ from appearing in the code. In [28], it is proved that for any such local property there is a *threshold* rate R^* such that:

- If $R < R^*$, a random linear code satisfies the property with high probability;
- If $R > R^*$, a random linear code fails to satisfy the property with high probability.

The theorem furthermore characterizes the threshold rate R^* as the solution to a certain optimization problem. In this work, we endeavour to compute new bounds on the threshold rate R^* for various properties of interest.

In the remainder, we provide intuition for the characterization of the threshold rate from [28]. First, we identify subsets $B \subseteq \mathbb{F}_q^n$ of size L with the matrix in $\mathbb{F}_q^{n\times L}$ whose columns are given by B (the choice of ordering is immaterial), and we say that a matrix M is contained in a code $\mathcal{C}$ if $\mathcal{C}$ contains all of M's columns. For a collection of matrices $\mathcal{M} \subseteq \mathbb{F}_q^{n\times L}$, we would like to compute the threshold rate R^* for "$\mathcal{M}$-freeness," i.e., the code property of not containing a matrix in $\mathcal{M}$.

As we are interested in list-decoding/recovery, we define a set of matrices $\mathcal{M}$ such that if $\mathcal{C}$ contains a matrix from $\mathcal{M}$ then $\mathcal{C}$ is not list-decodable/recoverable. We say that the collection $\mathcal{M}$ is "bad" for list-decoding/recovery. As intuition, for list-decoding we can just take the set of matrices where each column lies in some ball $B(z, \rho)$. Next, we would like

to show that $\mathcal{M}$ is "abundant" in the sense that it is very likely that $\mathcal{C}$ contains a matrix $M \in \mathcal{M}$. In other words, if X_M denotes the indicator random variable for the event $M \subseteq \mathcal{C}$, then we should expect $X_{\mathcal{M}} := \sum_{M \in \mathcal{M}} X_M \geq 1$.

It is relatively easy to compute $\mathbb{E}[X_{\mathcal{M}}]$ and see when it exceeds 1; however, to conclude that $X_{\mathcal{M}}$ is likely to be large one needs a concentration bound. Such a bound is often provided by estimating the variance of $X_{\mathcal{M}}$. Broadly construed, [28] applies the second moment method to demonstrate that there is really only one reason that $X_{\mathcal{M}}$ would fail to be concentrated: it is because for some compressing matrix $A \in \mathbb{F}_q^{L \times L'}$ with $L' \leq L$ the set $\{MA : M \in \mathcal{M}\}$ is too small.

List-Recovery. First, we endeavour to prove a lower bound on the list-size for list-recovery. This means that we need to say that if the list-size is too small then the random linear code quite likely contains a matrix from a set $\mathcal{M}$ of bad matrices for list-recovery. In light of the above, to conclude our argument we need to show that for any compressing matrix A, the set $\{MA : M \in \mathcal{M}\}$ remains large.

To do this, we use information-theoretic techniques: we identify each of our bad matrices $M \in \mathcal{M}$ with an appropriate *type*, which is a distribution $\tau \sim \mathbb{F}_q^L$ defined as the empirical distribution of M's rows. A lower bound on $\{MA : M \in \mathcal{M}\}$ is then implied by a lower bound on the entropy of the random variable $A\vec{\boldsymbol{u}}$ for $\vec{\boldsymbol{u}} \sim \tau$. We are also free to choose the type τ which is "bad" for a certain property, in the sense that if a code contains a matrix of type τ then it fails to satisfy the property.

For the case of (ρ, ℓ, L)-list-recovery, the following type is bad: one samples uniformly $\boldsymbol{S} \in \binom{\mathbb{F}_q}{\ell}$ and then outputs $\vec{\boldsymbol{u}} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_L) \in \mathbb{F}_q^L$, where each $\boldsymbol{u}_i$ is independently uniform over $\boldsymbol{S}$ with probability $1 - \rho$ and uniform over $\mathbb{F}_q \setminus \boldsymbol{S}$ otherwise. It thus follows that a lower bound on $\{AM : M \in \mathcal{M}\}$ is implied by a lower bound on the entropy of the random variable $A\vec{\boldsymbol{u}}$ for $\vec{\boldsymbol{u}} \sim \tau$.

Obtaining this lower bound requires a rather lengthy argument; we overview the main ideas now. We begin by partitioning the coordinates of $A\vec{\boldsymbol{u}}$ into subsets $J_1, \ldots, J_k \subseteq [L']$, where each J_i depends on at least 2 "fresh" coordinates from $\vec{\boldsymbol{u}}$, along with (perhaps) a set of leftover coordinates J_{k+1}. We then provide two arguments depending on the maximum size of a part. If, say, $|J_1|$ is large, then we can show that $(A\vec{\boldsymbol{u}})_{J_1}$ already experiences a large entropy increase. This is shown by demonstrating that these coordinates alone already allow us to nontrivially guess the subset $\boldsymbol{S}$. Otherwise, we argue that all the parts provide a nontrivial increase in the entropy, and since there must be a large number of parts in this case, by summing over all the parts we provide an adequate lower bound.

This result generalizes the list-decoding lower bound that was provided in [15, Theorem IV.1]. The argument in that paper exploited the fact that a sample from the bad type for list-decoding has a simpler structure: it looks like $\vec{\boldsymbol{v}} + \boldsymbol{\alpha}\vec{1}$, where $\vec{\boldsymbol{v}}$ is a q-ary Bernoulli random variable and $\boldsymbol{\alpha} \in \mathbb{F}_q$ is uniformly random. In our case, we do not have this nice linear structure,[5] making the analysis more intricate.

List-Decoding with Small Lists. For our results concerning list-decoding with small lists, we again use the thresholds framework. In this case, we need to consider *any* type that is bad for $(\rho, 3)$ or $(\rho, 4)$-list-decoding. For these small values of L, we are able to identify the linear map A which leads to the maximum relative entropy $\frac{H_q(A\tau)}{\dim(A\tau)}$: in each case, it is given by the map sending $(x_1, \ldots, x_L) \mapsto (x_1 - x_L, \ldots, x_{L-1} - x_L)$.

[5] One might be tempted to look at $\vec{\boldsymbol{v}} + \vec{\boldsymbol{w}}$ where $\vec{\boldsymbol{v}}$ is q-ary Bernoulli and $\vec{\boldsymbol{w}}$ is uniform over $\boldsymbol{S}$, but note that for $\ell - 1$ choices for $\boldsymbol{v}_i \in \mathbb{F}_q^*$ the sum $\boldsymbol{v}_i + \boldsymbol{w}_i$ still lies in $\boldsymbol{S}$.

To provide the proof, we break up the vector spaces based on the number of distinct coordinates of the entries, and observe that a type which is bad for list-decodability can only put so much probability mass on each of these parts. To conclude, we rely on the concavity of the entropy function as well as some combinatorial reasoning concerning the subspaces of $\mathbb{F}_2^4$ and $\mathbb{F}_q^3$. Even for these small values of L we need to be quite careful to avoid a massive explosion in the number of cases to consider, as we must look at all compressing linear maps A.

Random Codes. For the case of random codes, we can compute the threshold rates for all the properties of interest in a relatively straightforward way, as the characterization from [17] does not require us to consider any sort of compressing mapping on the types. Quite notably, in all cases we see that random linear codes appear to perform better than random codes. This is perhaps in contrast to commonly held beliefs: in this sense, a main goal of our work is to disseminate this counterintuitive phenomenon.

1.3 Related Work

In Section 1.2 we outlined the works [28, 15, 17] which developed and studied the thresholds toolkit that we apply. In this section, we provide more context for the study of random linear codes and their list-decodability/-recoverability. In what follows, q always denotes the alphabet size and ε the "gap-to-capacity" for a capacity-approaching code.

List Size Lower Bounds for Random (Linear) Codes. As we provide lower bounds for list-recovery of random linear codes, we briefly survey the known lower bounds for list-decoding. First, Guruswami and Narayanan [21] showed that capacity-approaching random (linear) codes require lists of size $\Omega_{\rho,q}(1/\varepsilon)$: by inspecting the proof one can note that the implied constant tends to 0 as $\rho \to 1 - 1/q$, or if $q \to \infty$. While on the surface their approach appears very different to ours, their use of a second-moment method is akin to the proofs underlying the thresholds framework from [28], so the approaches are in fact somewhat similar. Later, Li and Wootters [26] gave a $\sim 1/\varepsilon$ list-size lower bound for capacity-approaching random codes. Again, the argument relies on the second-moment method.

In [15], a lower bound for the list-decodability of capacity-approaching random linear codes is given, showing that lists of size $\sim \frac{h_q(\rho)}{\varepsilon}$ are required: our list-recovery list-size lower bound is a generalization of this result. Lastly, in [17] the threshold rate for $(\rho, 2)$-list-decodability is computed, providing a lower bound and an upper bound: this segues us nicely into a discussion of the work on computing upper bounds on list-sizes.

List Size Upper Bounds for Random Linear Codes. There has been a long line of work [37, 10, 9, 5, 34, 31, 32, 26, 17] studying the list-decodability of capacity-approaching random linear codes, and we now highlight some relevant results. First, Zyablov and Pinkser [37] demonstrated that capacity-approaching RLCs are indeed (ρ, L)-list-decodable, albeit with $L = q^{\Omega(1/\varepsilon)}$. Subsequent work has endeavoured to prove list-decodability with $L = O(1/\varepsilon)$. The existence of such linear codes over $\mathbb{F}_2$ was first demonstrated by [10]; later, [26] showed that this holds with high probability for randomly sampled linear codes, and subsequently [15] showed this is true for *average-radius*[6] list-decoding.

[6] In this model, it is required that the code does not contain L points whose average distance from a centre is less than ρ. Thus, it is a stricter requirement than standard list-decoding.

As for larger alphabets, [9] showed that lists of size $O_{\rho,q}(1/\varepsilon)$ do indeed suffice for random linear codes. We further remark that their argument uses a certain Ramsey-theoretic concept called a 2-increasing sequence to choose the order in which to reveal coordinates, which is vaguely reminiscent of the "fresh" coordinates that we have defined by the J_i's in our list-recovery lower bound argument. A drawback of this work is that the implied constant in the $O_{\rho,q}(\cdot)$ notation degrades as $\rho \to 1-1/q$ or if q grows too large. In light of this restriction, a line of works [5, 34, 31] has studied the "high noise regime," where $\rho = 1-1/q-\eta$ and one endeavours to show that lists of size $O(1/\eta^2)$ suffice for codes of rate $\Omega(\eta^2)$. These results are still not quite optimal in the sense that the implied constants (even for the rate) lag behind the parameters achievable by random codes. Lastly, for list-recoverability with input list-size ℓ it appears that the best upper bound on the list-size is due to [32], where it is shown that lists of size $(q\ell)^{O(\log(\ell)/\varepsilon)}$ suffice.

Lower Bounds for List Sizes of Arbitrary Codes. While we exclusively study random (linear) codes, we view these as a proxy for determining the actual achievable tradeoffs. As lists of size $\Theta(1/\varepsilon)$ are required for random codes, it is natural to wonder if all capacity-approaching (ρ, L)-list-decodable codes require lists of size $\Omega(1/\varepsilon)$. Blinovsky [4, 3] has shown a lower bound of $\Omega_\rho(\log(1/\varepsilon))$. In the high noise regime, viz., $\rho = 1-1/q-\eta$, Guruswami and Vadhan [21] provided a $\Omega_q(1/\eta^2)$ lower bound on the list size. Lastly, for *average-radius* list-decoding Guruswami and Narayanan [18] proved a $\Omega_\rho(1/\sqrt{\varepsilon})$ lower bound.

1.4 Open Problems

In this work, we have progressed our understanding of combinatorial properties of random (linear) codes. A main conclusion of our work is that for list-decoding/recovery, random linear codes perform better.[7]

There are many open problems which remain to be studied and we list some below.

- Provide the corresponding upper bounds on the threshold rate for $(\rho, 4)$-list-decoding binary random linear codes, and the threshold rate for $(\rho, 3)$-list-decoding q-ary random linear codes.
- Provide the corresponding lower bound on the threshold rate for (ρ, ℓ, L)-list-recovery in the capacity-approaching regime. In fact, for $q > 2$, the threshold rate for (ρ, L)-list-decoding is still open. This is quite likely a very challenging problem; the only tight argument we have is due to [10, 26] (see also [15]) which only applies to list-decoding over the binary field, and this argument appears too "rigid" to apply in more generality.
- Get a better understanding for *worst-case* codes. In particular, to the best of our knowledge the *Plotkin points* for (ρ, L)-list-decoding for $q > 2$ are not known. That is, compute the minimum value ρ^* such that for all $\rho > \rho^*$, there are no q-ary (ρ, L)-list-decodable code families with positive rate. (Recent work [36] expresses the Plotkin point as a solution to a certain optimization problem, but we do not see how to extract a simple expression from this.)

1.5 Organization

In the subequent section, we introduce the necessary notations and definitions that we will use in this work, along with the tools from [28, 17] that we apply. In Section 3, we provide our lower bound on the list-size for the list-recoverability of random linear codes which

[7] For list-recovery, we admittedly only provide some evidence in this direction.

approach capacity. In Section 4, we lower bound the threshold rate for list-of-2 decoding (for general q) and list-of-3 decoding (in the binary case). We also compare random linear codes to random codes over the binary alphabet for more values of L. For space reasons, most of the technical proofs are deferred to the full version.

2 Preliminaries

Miscellaneous Notations. For an integer $n \geq 1$, we denote $[n] := \{1, 2, \ldots, n\}$. For a set X we denote by $\binom{X}{\ell}$ the family of all subsets of X with ℓ elements, and similarly $\binom{X}{\leq \ell}$ denotes the family of all subsets of X with $\leq \ell$ elements. Throughout, $\mathbb{F}_q$ denotes the finite field with q elements, for q a prime power.

For clarity, vectors are typically denoted with an arrow overtop. Given a vector $\vec{x} \in \mathbb{F}_q^n$ and a subset $I \subseteq [n]$ we denote by $\vec{x}_I$ the length $|I|$ vector $(x_i : i \in I) \in \mathbb{F}_q^{|I|}$. We reserve $\vec{1}$ for the all-1's vector; if we wish to emphasize its length we subscript it, i.e., $\vec{1}_D$ is the all-1's vector of length D. Random variables are typically written in boldface, e.g., $\boldsymbol{x}, \boldsymbol{y}$, etc. In particular, random vectors are denoted, e.g., $\vec{\boldsymbol{u}}$.

Coding Theory Terminology. A *code* $\mathcal{C}$ is a subset of $\mathbb{F}_q^n$ for $\mathbb{F}_q$ the finite field of order q, a prime power. Elements $\vec{c} \in \mathcal{C}$ are called *codewords*, the integer n is the *block-length*, and the integer q is the *alphabet size*; such a code is also called *q-ary*. When $q = 2$ the code is deemed *binary*. We are typically interested in *linear* codes, which are $\mathcal{C} \leq \mathbb{F}_q^n$, i.e., they are subspaces. The *rate* of a code $\mathcal{C}$ is $R = R(\mathcal{C}) := \frac{\log_q |\mathcal{C}|}{n}$ and its minimum distance is $\delta = \delta(\mathcal{C}) := \min\{d(\vec{c}, \vec{d}) : \vec{c} \neq \vec{d}, \vec{c}, \vec{d} \in \mathcal{C}\}$, where $d(\vec{x}, \vec{y}) = \frac{1}{n}|\{i \in [n] : x_i \neq y_i\}|$ is the (relative) Hamming distance from $\vec{x}$ to $\vec{y}$. We also slightly extend this notation as follows: for a vector $\vec{x} \in \mathbb{F}_q^n$ and a tuple of subsets $\vec{S} = (S_1, \ldots, S_n)$, $S_i \subseteq \mathbb{F}_q$, we define $d(\vec{x}, \vec{S}) := \frac{1}{n}|\{i \in [n] : x_i \notin S_i\}|$, i.e., the fraction of coordinates i for which $\vec{x}$ "disagrees" with the corresponding subset of $\vec{S}$.

A *random linear code* of rate R is a uniformly random subspace of $\mathbb{F}_q^n$ of dimension Rn.[8] As this concept will arise regularly in this work, we occasionally use the abbreviation *RLC*. A *random code* of rate R is a random subset of $\mathbb{F}_q^n$ obtained by including each element independently with probability $q^{(R-1)n}$.[9] For this concept, we use the abbreviation *RC*.

2.1 List-decodability and List-recoverability

In this work, we study combinatorial properties of linear codes. Of primary interest to us are list-decodability and list-recoverability, which we now define.

▶ **Definition 8** (List-decodability). *Let $\rho \in (0, 1 - 1/q)$ and $L \geq 1$. A code $\mathcal{C} \subseteq \mathbb{F}_q^n$ is called* (ρ, L)-list-decodable *if for all $\vec{z} \in \mathbb{F}_q^n$,*

$$|\{\vec{c} \in \mathcal{C} : d(\vec{c}, \vec{z}) \leq \rho\}| < L \ .$$

[8] In fact, there are different ways to sample linear codes. For concreteness, we typically implicitly use the model where a random parity check matrix $\boldsymbol{H} \in \mathbb{F}_q^{(1-R)n \times n}$ is sampled and we output $\boldsymbol{\mathcal{C}} = \ker(\boldsymbol{H})$. Of course, there is a small chance $\boldsymbol{\mathcal{C}}$ has rate larger than R, but as this probability is exponentially small in n it is immaterial to our conclusions. We also briefly use the model where a random $\boldsymbol{G} \in \mathbb{F}_q^{Rn \times n}$ is sampled and we output $\boldsymbol{\mathcal{C}} = \mathrm{im}(\boldsymbol{G})$.

[9] By Chernoff bounds, such a code as rate $R \pm o(1)$ with high probability.

We also use the terminology "list-of-L-decoding" for $(\rho, L+1)$-list-decoding, e.g., list-of-2-decoding corresponds to $(\rho, 3)$-list-decoding.

The list-decoding *capacity* is the value $R^*(\rho)$ such that for any $R < R^*(\rho)$ there exists $L > 1$ such that infinite families of (ρ, L)-list-decodable codes of rate at least R exist, but for any $R > R^*(\rho)$ such an infinite family does not exist. It is known that

$$R^*(\rho) = 1 - h_q(\rho) \ ,$$

where

$$h_q(\rho) = \rho \log_q \frac{q-1}{\rho} + \log_q \frac{1}{1-\rho}$$

is the q-ary entropy function.

▶ **Definition 9** (List-recoverability). *Let $\rho \in (0, 1-1/q)$, $1 \le \ell \le q$ and $L \ge 1$. A code $\mathcal{C} \subseteq \mathbb{F}_q^n$ is called* (ρ, ℓ, L)-list-recoverable *if for all tuples of subsets $\vec{S} = (S_1, \ldots, S_n) \in \binom{\mathbb{F}_q}{\le \ell}^n$,*

$$|\{\vec{c} \in \mathcal{C} : d(\vec{c}, \vec{S}) \le \rho\}| < L \ .$$

In analogy to the list-decoding capacity, the *list-recovery capacity* is the value $R^*(\rho, \ell)$ such that for any $R < R^*(\rho, \ell)$ there exists $L > 1$ such that infinite families of (ρ, ℓ, L)-list-recoverable codes of rate at least R exist, but for any $R > R^*(\rho, \ell)$ such an infinite family does not exist. It is known that

$$R^*(\rho, \ell) = 1 - h_{q,\ell}(\rho) \ ,$$

where

$$h_{q,\ell}(\rho) = \rho \log_q \frac{q-\ell}{\rho} + (1-\rho) \log_q \frac{\ell}{1-\rho}$$

is the (q, ℓ)-entropy function.

2.2 Information-Theoretic Concepts

For a random variable $\boldsymbol{x}$ over a domain $\mathcal{X}$ we denote its entropy by

$$H(\boldsymbol{x}) = \sum_{x \in \mathcal{X}} \Pr[\boldsymbol{x} = x] \log \frac{1}{\Pr[\boldsymbol{x} = x]} \ ,$$

where we use the convention $0 \log \frac{1}{0} = 0$. If τ is a distribution then we define $H(\tau)$ to be the entropy of a random variable distributed according to τ.

Given another random variable $\boldsymbol{y}$ supported on a set $\mathcal{Y}$, the *conditional entropy* of $\boldsymbol{x}$ given $\boldsymbol{y}$ is

$$H(\boldsymbol{x}|\boldsymbol{y}) = \mathop{\mathbb{E}}_{y \sim \boldsymbol{y}} [H(\boldsymbol{x}|\boldsymbol{y} = y)] = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \Pr[\boldsymbol{x} = x, \boldsymbol{y} = y] \log \frac{\Pr[\boldsymbol{x} = x]}{\Pr[\boldsymbol{x} = x, \boldsymbol{y} = y]} \ .$$

Intuitively, this is the expected amount of entropy remaining in $\boldsymbol{x}$ after revealing $\boldsymbol{y}$. Conditional entropy satisfies the *chain rule* $H(\boldsymbol{x}, \boldsymbol{y}) = H(\boldsymbol{x}|\boldsymbol{y}) + H(\boldsymbol{y})$, which can be extended by induction to larger collections of random variables.

We also use the notion of *mutual information*, which is a measure of the amount of information one random variable gives about another and is defined as follows:

$$I(\boldsymbol{x}; \boldsymbol{y}) = H(\boldsymbol{x}) - H(\boldsymbol{x}|\boldsymbol{y}) = H(\boldsymbol{y}) - H(\boldsymbol{y}|\boldsymbol{x}) = H(\boldsymbol{x}, \boldsymbol{y}) - H(\boldsymbol{x}) - H(\boldsymbol{y}) \ .$$

(The equalities are justified by the chain rule.) We also consider the *conditional mutual information*, defined as follows:

$$I(\boldsymbol{x};\boldsymbol{y}|\boldsymbol{z}) = H(\boldsymbol{x}|\boldsymbol{z}) - H(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{y}) = H(\boldsymbol{y}|\boldsymbol{z}) - H(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z}) = H(\boldsymbol{x},\boldsymbol{y}|\boldsymbol{z}) - H(\boldsymbol{x}|\boldsymbol{z}) - H(\boldsymbol{y}|\boldsymbol{z}) \ ,$$

where $\boldsymbol{z}$ is another random variable.

Conditional entropy, mutual information and conditional mutual information all satisfy the *data-processing inequality*: for any function f supported on $\mathcal{Y}$ (the domain of Y), we have

$$H(\boldsymbol{x}|f(\boldsymbol{y})) \geq H(\boldsymbol{x}|\boldsymbol{y}) \ , I(\boldsymbol{x};\boldsymbol{y}) \geq I(\boldsymbol{x};f(\boldsymbol{y})) \ , I(\boldsymbol{x};\boldsymbol{y}|\boldsymbol{z}) \geq I(\boldsymbol{x};f(\boldsymbol{y})|\boldsymbol{z}) \ .$$

We will also use *Fano's inequality*.

▶ **Theorem 10** (Fano's Inequality.). *Let $\boldsymbol{x}$ be a random variable supported on $\mathcal{X}$, $\boldsymbol{y}$ a random variable supported on $\mathcal{Y}$ and $f : \mathcal{Y} \to \mathcal{X}$. Define $p_{\mathrm{err}} := \Pr[f(\boldsymbol{y}) \neq \boldsymbol{x}]$. Then,*

$$H(\boldsymbol{x}|\boldsymbol{y}) \leq h(p_{\mathrm{err}}) + p_{\mathrm{err}} \cdot \log(|\mathcal{X}| - 1) \ .$$

When we wish to change the base of the logarithm with which the entropy or mutual information, the desired base is subscripted. That is,

$$H_q(\boldsymbol{x}) := \frac{H(\boldsymbol{x})}{\log q} \ , \qquad I_q(\boldsymbol{x};\boldsymbol{y}) := \frac{I(\boldsymbol{x};\boldsymbol{y})}{\log q} \ ,$$

and similarly for the conditional versions of these quantities. Finally, as a slight abuse of notation, we also write

$$H_q(x_1,\ldots,x_t) = \sum_{i=1}^{t} x_i \log_q \frac{1}{x_i} + (1 - x_1 - \cdots - x_t) \log_q \frac{1}{1 - x_1 - \cdots - x_t}$$

if $x_1,\ldots,x_t$ are positive numbers satisfying $\sum_{i=1}^{t} x_i \leq 1$. (We caution that for $q > 2$, $H_q(x) \neq h_q(x)$.)

2.3 Thresholds

We now introduce the specialized notations and tools that we will need in order to apply the machinery of [28]. First, for a distribution $\tau \sim \mathbb{F}_q^b$ and a linear map $A : \mathbb{F}_q^b \to \mathbb{F}_q^c$, we let $A\tau$ denote the distribution of the random vector $A\vec{\boldsymbol{u}}$ for $\vec{\boldsymbol{u}} \sim \tau$. In more detail, $A\tau$ has the following probability mass function:

$$\Pr_{\vec{\boldsymbol{v}} \sim A\tau}[\vec{\boldsymbol{v}} = \vec{y}] = \sum_{\vec{x} \in A^{-1}(\vec{y})} \Pr_{\vec{\boldsymbol{u}} \sim \tau}[\vec{\boldsymbol{u}} = \vec{x}] \ .$$

While we are generally concerned with understanding the probability that certain "bad sets" lie in our code, it is in fact more convenient to work with matrices. For a matrix $M \in \mathbb{F}_q^{n \times b}$ and a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ we say that $\mathcal{C}$ contains M if the columns of M are contained in $\mathcal{C}$.

Every matrix is assigned a *type*, and the type of a matrix is determined by the matrix's empirical row distribution as follows:

▶ **Definition 11** ($\tau_M, \dim(\tau), \mathcal{M}_{n,\tau}$). *For a matrix $M \in \mathbb{F}_q^{n \times b}$, we define its* type τ_M *to be the distribution given by the empirical distribution of M's rows. That is, for all $\vec{v} \in \mathbb{F}_q^b$ we have*

$$\tau_M(\vec{v}) := \frac{|\{i \in [n] : i\text{th row of } M \text{ equals } \vec{v}\}|}{n} \ .$$

For a distribution τ on $\mathbb{F}_q^b$, $\dim(\tau)$ denotes the dimension of the span of τ's support, i.e.,

$$\dim(\tau) := \dim(\mathrm{span}(\mathrm{supp}(\tau))).$$

*We denote by $\mathcal{M}_{n,\tau}$ the set of all matrices in $\mathbb{F}_q^{b\times n}$ with empirical row distribution τ. We call a type τ b-*local *if $\tau \sim \mathbb{F}_q^b$; note that a b-local type has $\dim(\tau) \le b$.*

▶ Remark 12. Technically, for a distribution $\tau \sim \mathbb{F}_q^b$ it could be the case that $\mathcal{M}_{n,\tau}$ is empty just because, for some $\vec{v} \in \mathbb{F}_q^b$, $\tau(\vec{v}) \notin \{0, 1/n, 2/n, \dots, (n-1)/n, 1\}$. For such τ, we can define $\mathcal{M}_{n,\tau}$ to consist of those matrices which contain either $\lfloor n \cdot \tau(\vec{v})\rfloor$ or $\lceil n \cdot \tau(\vec{v})\rceil$ copies of $\vec{v}$. As we are always dealing with the setting where n is assumed to be sufficiently large compared to all other parameters, this does not affect the analysis. Hence, we may safely ignore this technicality, which we do for the clarity of exposition.

Our target is an understanding of the threshold rate for a combinatorial property of random linear codes. The combinatorial properties that we will study are those that are defined by excluding a set of types, as follows.

▶ **Definition 13** (τ-freeness, local properties)**.** *Given a code $\mathcal{C}$ and a type τ, we say that $\mathcal{C}$ is* τ-free *if $\mathcal{C}$ does not contain any matrix $M \in \mathcal{M}_{n,\tau}$, i.e., no matrix M of type τ.*

For a set $\mathcal{T}$ of types, where each $\tau \sim \mathbb{F}_q^b$ for some $b \in \mathbb{N}$, we say that $\mathcal{C}$ is $\mathcal{T}$-free *if it is τ-free for all $\tau \in \mathcal{T}$. We refer to $\mathcal{T}$-freeness as a* b-local property *of codes.*

For a more in-depth discussion of the definition, we refer the reader to, [28, Section 2] or [30, Chapter 3]. To provide some intuition, we demonstrate how (ρ, ℓ, L)-list-recoverability may be described as an L-local property. We define $\mathcal{T}$ to be the set of all types $\tau \sim \mathbb{F}_q^L$ such that for some (correlated) distribution $\nu \sim \binom{\mathbb{F}_q}{\ell}$,

$$\forall i \in [L], \quad \Pr_{(\vec{\boldsymbol{u}},\boldsymbol{S})\sim(\tau,\nu)}[\boldsymbol{u}_i \notin \boldsymbol{S}] \le \rho \tag{1}$$

and furthemore we require

$$\forall 1 \le i < j \le L, \quad \Pr_{\vec{\boldsymbol{u}}\sim\tau}[\boldsymbol{u}_i \neq \boldsymbol{u}_j] > 0 \ .$$

(This second condition amounts to requiring that any matrix of type τ has distinct columns.) We refer to the collection of all these types as $\mathcal{T}_{\rho,\ell,L}$.

We now characterize (up to $o(1)$ terms) the threshold rate of a property.

▶ **Theorem 14** ([30], Theorem 3.3.9: Thresholds for Random Linear Codes)**.** *Fix $b \in \mathbb{N}$ and let $\mathcal{T}$ be a set of b-local types. Then the threshold rate for $\mathcal{T}$-freeness is*

$$1 - \max_{\tau\in\mathcal{T}} \min_{A} \left\{ \frac{H_q(A\tau)}{\dim(A\tau)} \right\} \pm o_{n\to\infty}(1) \ , \tag{2}$$

where the minimum is taken over all surjective linear maps $A : \mathbb{F}_q^b \to \mathbb{F}_q^c$ with $c \le b$.

Let us specialize to the case of τ-freeness for a single type τ. Suppose that $R > 1 - \min_A \left\{ \frac{H_q(A\tau)}{\dim(A\tau)} \right\}$. Theorem 14 tells us that it is unlikely that a RLC of rate R is τ-free. Stated differently, we can expect that there is at least one matrix of type τ contained in such an RLC. In fact, while we do not prove this, it is in fact likely that there will be *many* such matrices. For this reason, we use the following terminology for types τ satisfying $R > 1 - \min_A \left\{ \frac{H_q(A\tau)}{\dim(A\tau)} \right\}$: we call them *abundant*.

In proving an upper bound R_{upper} on the threshold rate for a property of interest (e.g., (ρ, ℓ, L)-list-recovery), we will follow the following steps. First, we define an appropriate set type τ and prove that a code satisfies the property of interest only if it is τ-free. Informally, we refer to this as a proof that τ is *bad* for the property of interest. Next, we show that for RLCs of rate R_{upper}, the type τ is abundant. This is the more challenging part of the theorem, as the minimization over the set of all linear maps A is quite challenging to control. Nonetheless, we are able to carry out this program for (ρ, ℓ, L)-list-recovery, as advertised.

In proving a lower bound on R_{low} on the threshold rate for a property of interest (e.g., $(\rho, 3)$-list-decoding), we need to consider *any* type that is bad for list-decoding, and then show that it is *implicitly rare*: that is, for some matrix A, there are relatively few matrices of type $A\tau$, and hence it is likely no matrix of that type lies in the RLC. That is, we must upper bound the ratio of the entropy of $A\tau$ with the dimension of $A\tau$. Here, we have the freedom to choose A, but the argument must apply to all types τ. This is especially tricky when given a type τ whose support is contained in a strict subspace, as then the bound on the entropy must be commensurately smaller. It is for this reason that we only consider small values of L, as one suffers from a combinatorial explosion in the number of possible support spaces for the types.

Thresholds for Random Codes. For thresholds of random codes, the characterization theorem is simpler in the sense that we do not have to minimize over compressive mappings, at least if the property satisfies certain technical conditions. Fortunately, the characterization applies to list-recoverability, and hence also list-decodability.

▶ **Theorem 15** ([17], Theorem 2: Thresholds for Random Codes)**.** *Let $b \in \mathbb{N}$ and let $\mathcal{T}$ be a set of b-local types. Let T be a convex approximation for $\mathcal{T}$. Then the threshold rate for $\mathcal{T}$-freeness is*

$$1 - \frac{\max_{\tau \in T} H_q(\tau)}{b} \ .$$

▶ **Proposition 16** ([17], Lemma 1)**.** *$\mathcal{T}_{\rho,\ell,L}$ is a convex approximation for the property of (ρ, ℓ, L)-list-recoverability.*

3 Lower Bound on List-Size for List-Recovery

Througout this section, the following notations are fixed:

- $q \in \mathbb{N}$ is a (fixed) prime power;[10]
- $\ell \in \mathbb{N}$ satisfies $1 \leq \ell < q$;
- $\rho \in \mathbb{R}$ satisfies $0 < \rho < 1 - \frac{\ell}{q}$; and
- $\delta > 0$ is a small constant.

All these parameters are constants, independent of the growing parameter n. Our main result in this section is the following theorem.

▶ **Theorem 17.** *There exists $\varepsilon_{q,\ell,\rho,\delta} > 0$ such that for all $0 < \varepsilon < \varepsilon_{q,\ell,\rho,\delta}$ and n sufficiently large, a random linear code in $\mathbb{F}_q^n$ of rate $1 - h_{q,\ell}(\rho) - \varepsilon$ is not $\left(\rho, \ell, \lfloor \frac{\log_q \binom{q}{\ell} - (1 - h_{q,\ell}(\rho))}{\varepsilon} - \delta \rfloor\right)$-list-recoverable with probability $1 - o(1)$.*

[10] When we discuss random codes, q may be any positive integer.

The proof of this theorem follows the same outline as has been used in, e.g., [15]. Namely, we begin by defining a L-local type which we show is *bad* for (ρ, ℓ, L)-list-recovery. Later, we prove that the type is indeed *abundant*, which is the more challenging part of the theorem.

The bad L-local type is defined as follows.

▶ **Definition 18** (The bad type for (ρ, ℓ, L)-list-recoverability)**.** *Fix $L \in \mathbb{N}$. Define the distribution $\tau \sim \mathbb{F}_q^L$ via the following procedure for sampling a random vector $\vec{\boldsymbol{u}} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_L)$:*

- *First, $\boldsymbol{S} \sim \binom{\mathbb{F}_q}{\ell}$ is sampled uniformly at random;*
- *Second, for $i = 1, \ldots, L$, we sample $\boldsymbol{u}_i \sim \mathbb{F}_q$ as*

$$\Pr[\boldsymbol{u}_i = x | \boldsymbol{S} = S] = \begin{cases} \frac{1-\rho}{\ell} & \text{if } x \in S \\ \frac{\rho}{q-\ell} & \text{if } x \notin S \end{cases} ,$$

and conditioned on $\boldsymbol{S} = S$, the coordinates $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_L$ are independent*.*

Note that such a type does indeed lie in the set $\mathcal{T}_{\rho,\ell,L}$. Indeed, if $\nu \sim \boldsymbol{S}$ we clearly have

$$\forall i \in [L], \quad \Pr_{(\vec{\boldsymbol{u}}, \boldsymbol{S}) \sim (\tau, \nu)} [\boldsymbol{u}_i \notin \boldsymbol{S}] = \rho$$

and we also readily have $\Pr_{\vec{\boldsymbol{u}} \sim \tau}[\boldsymbol{u}_i \neq \boldsymbol{u}_j] > 0$. From [17], we conclude that τ is bad for (ρ, ℓ, L)-list-recovery.

We now claim that the type τ is indeed abundant, i.e., that it has sufficiently large (relative) entropy. This is the more technical part of the proof, and its proof is deferred to the full version.

▶ **Lemma 19.** *There exists an integer $L_{\rho,q,\ell,\delta}$ such that for all integers $L \geq L_{\rho,q,\ell,\delta}$, the following holds. Let $\vec{\boldsymbol{u}} \sim \tau$, and let $A \in \mathbb{F}_q^{L' \times L}$ with $L' \leq L$ and $\operatorname{rank}(A) = L'$. Then*

$$H_q(A\vec{\boldsymbol{u}}) \geq L' \cdot h_{q,\ell}(\rho) + \log_q \binom{q}{\ell} - 1 + h_{q,\ell}(\rho) - \delta \geq L' \cdot \left(h_{q,\ell}(\rho) + \frac{\log_q \binom{q}{\ell} - 1 + h_{q,\ell}(\rho) - \delta}{L} \right) .$$

Assuming Lemma 19, we now show that this does indeed yield our target Theorem 17.

Proof of Theorem 17. Let $L_{\rho,q,\ell,\delta/2}$ be the promised constant from Lemma 19, and choose $\varepsilon_{q,\ell,\rho,\delta} := \frac{\log_q \binom{q}{\ell} - 1 + h_{q,\ell}(\rho)}{L_{\rho,q,\ell,\delta/2}+1}$. Let $\varepsilon < \varepsilon_{q,\ell,\rho,\delta}$. Let $L = \left\lfloor \frac{\log_q \binom{q}{\ell} - 1 + h_{q,\ell}(\rho)}{\varepsilon} - \delta \right\rfloor$, and define τ as in Definition 18 with this choice of L.

By Lemma 19, as $L \geq L_{\rho,q,\ell,\delta/2}$ we have that for all surjective linear maps $A : \mathbb{F}_q^L \to \mathbb{F}_q^{L'}$

$$\frac{H_q(A\tau)}{L'} \geq h_{q,\ell}(\rho) + \frac{\log_q \binom{q}{\ell} - 1 + h_{q,\ell}(\rho) - \delta/2}{L} .$$

We note further that as τ has full support the same is true for $A\tau$, i.e., $\dim(A\tau) = L'$. Thus, by Theorem 14 we have that the threshold rate for τ-freeness is at most

$$1 - h_{q,\ell}(\rho) - \frac{\log_q \binom{q}{\ell} - 1 + h_{q,\ell}(\rho) - \delta/2}{L} - o_{n\to\infty} < 1 - h_{q,\ell}(\rho) - \varepsilon ,$$

where the last inequality holds for large enough n. In other words, a random linear code of rate $1 - h_{q,\ell}(\rho) - \varepsilon$ contains a matrix $M \in \mathcal{M}_{n,\tau}$ with probability $1 - o(1)$. As we know that a code $\mathcal{C}$ which contains a matrix of type τ is not (ρ, ℓ, L)-list-recoverable, our theorem is proved. ◀

3.1 List-recoverability lower bound for random codes

For context, we provide nearly matching upper and lower bounds for list-recovery for *uniformly* random codes. There is a similar result for list-recovery provided in [17], but it is not optimized for the case of capacity-approaching codes.

▶ **Theorem 20.** *There exists $\varepsilon_{q,\ell,\rho,\delta}$ such that for all $0 < \varepsilon < \varepsilon_{q,\ell,\rho,\delta}$ and n sufficiently large, a random code in $\mathbb{F}_q^n$ of rate $1 - h_{q,\ell}(\rho) - \varepsilon$ is not $\left(\rho, \ell, \lfloor \frac{\log_q \binom{q}{\ell}}{\varepsilon} - \delta \rfloor\right)$-list-recoverable.*

On the other hand, for any $\varepsilon > 0$ and n sufficiently large, a random code in $\mathbb{F}_q^n$ of rate $1 - h_{q,\ell}(\rho) - \varepsilon$ is $\left(\rho, \ell, \lceil \frac{\log_q \binom{q}{\ell}}{\varepsilon} + 1 \rceil\right)$-list-recoverable.

In this way, we can essentially pin-down the list size of a rate $1 - h_{q,\ell}(\rho) - \varepsilon$ random code to one of three possible values. This is similar to the result on the list-decodability of binary random linear codes from [15]. Again, the proof is deferred to the full version.

4 List-Decoding with Small Lists

In this section, we investigate the list-decodability of random codes and random linear codes with constant list size. Specifically, for list-of-3 decoding over the binary field, we can show that the threshold rate for list-decoding of random linear codes is strictly better than that for list-decoding uniformly random codes. Further, for larger field sizes we are able to show that the threshold rate for list-of-2 decoding over $\mathbb{F}_q$ is strictly better for random linear codes than for uniformly random codes. This extends the result of [17] which only applies to list-of-2 decoding for binary codes.

For our lower bound on the threshold rates for RLCs, we follow the following procedure. First, we consider any type that is bad for, e.g., $(\rho, 3)$-list-decoding, i.e., a type from $\mathcal{T}_{\rho,1,3}$. For any such type τ, we upper bound $\frac{H_q(A\tau)}{\dim(A\tau)}$ for the linear map A sending $(x_1, x_2, x_3) \mapsto (x_1 - x_3, x_2 - x_3)$. This is straightforward when the $\dim(A\tau)$ is full (requiring essentially only the concavity of the entropy function); when it is smaller, more careful reasoning is required. For space reasons, all the proofs of this section are deferred to the full version.

As a final contribution, we recall that in [15] it is shown that over the binary field the threshold rate for random linear codes is strictly better than random codes in the capacity-approaching regime. We observe that their techniques can be extended to show that such a trend holds for any constant list size L (assuming the decoding radius ρ is not too large). To do this, we first prove a lower bound on the threshold rate of binary random linear codes by applying the argument in [26] and an upper bound on the threshold rate of binary random codes following the argument in [15]. Although our proof resorts to known techniques, such results were not stated before and greatly strengthen our belief that random linear codes perform better than random codes. In light of the available evidence, a reasonable conjecture would be that the for all alphabet sizes, the threshold rate of random linear codes is strictly better than that of random codes.

4.1 List-of-3 Decoding for Binary Alphabet

In this section, we study the threshold rate for list-of-3 decoding binary codes. We recall that the Plotkin point for list-of-3 decoding binary codes, i.e., the maximum value of ρ for which $(\rho, 4)$-list-decoding with positive rate is possible, is $5/16$ [1]. Our main theorem is the following:

▶ **Theorem 21.** *Let $\rho \in (0, 5/16)$. The threshold rate for $(\rho, 4)$-list-decoding a random linear code over $\mathbb{F}_2$ is at least*

$$1 - \max\left\{ \frac{H_2(x_1, x_2) + 2x_1 + x_2 \log_2 3}{3} : x_1 + 2x_2 \leq 4\rho, x_1 + x_2 \leq 1, x_1, x_2 \geq 0 \right\}.$$

Next, for context, we consider the threshold rate for $(\rho, 4)$-list-decoding uniformly random codes.

▶ **Theorem 22.** *Let $\rho \in (0, 5/16)$. The threshold rate for $(\rho, 4)$-list decoding random code over $\{0, 1\}$ is*

$$1 - \max\left\{ \frac{1 + H_2(x_1, x_2) + 2x_1 + x_2 \log_2 3}{4} : x_1 + 2x_2 \leq 4\rho, x_1 + x_2 \leq 1, x_1, x_2 \geq 0 \right\}.$$

As $\frac{1+F}{4} \geq \frac{F}{3}$ for all $F \leq 3$, the lower bound on the threshold rate provided by Theorem 21 is greater than the exact value from Theorem 22. This demonstrates that random linear codes do indeed perform better.

4.2 List-of-2 Decoding for Arbitrary Alphabets

We now study list-of-2 decoding over $\mathbb{F}_q$ for $q \geq 3$. Here, the Plotkin point is to the best of our knowledge unknown, and we just prove our result for $\rho < 1/3$.

▶ **Theorem 23.** *Let $\rho \in (0, 1/3)$. The threshold rate for $(\rho, 3)$-list decoding random linear code over $\mathbb{F}_q$ with $q \geq 3$ is at least*

$$1 - \max\left\{ \frac{H_q(x_1, x_2) + x_1 \log_q 3(q-1) + x_2 \log_q (q-1)(q-2)}{2} : x_1 + 2x_2 \leq 3\rho, x_1 + x_2 \leq 1, x_1, x_2 \geq 0 \right\}.$$

For context, we again consider random codes.

▶ **Theorem 24.** *Let $\rho \in (0, 1/3)$. The threshold rate for $(\rho, 3)$-list decoding random code over $\mathbb{F}_q$ is*

$$1 - \max\left\{ \frac{1 + H_q(x_1, x_2) + x_1 \log_q 3(q-1) + x_2 \log_q (q-1)(q-2)}{3} : x_1 + 2x_2 \leq 3\rho, x_1 + x_2 \leq 1, x_1, x_2 \geq 0 \right\}.$$

Again, by noting $\frac{1+F}{3} \geq \frac{F}{2}$ for all $F \leq 2$, we conclude that random linear codes do indeed perform better: the lower bound on the threshold rate furnished by Theorem 23 is strictly greater than the exact threshold rate of Theorem 24.

4.3 List Decoding for Binary Alphabets with Larger Lists

In this subsection, we observe that the list-decodability of random linear codes is better than random codes over the binary field for any list size L.

We begin by stating our possibility result for random linear codes. The proof is an adaptation of the argument from [10, 26] which we omit due to the space limit.

▶ **Theorem 25.** *For any fixed list size L and $\delta > 0$, a random linear code over the binary field of rate $1 - h_2(\rho) - \frac{h_2(\rho)}{L-1-2\delta} - \delta$ is (ρ, L)-list decodable with probability $1 - 2^{-\Omega_{\delta,L}(n)}$.*

Next, we provide an upper bound on the list size of a random code. The proof, which appears in the full version, uses the threshold framework.

▶ **Theorem 26.** *Let L be a fixed constant list size and δ be any positive constant. With high probability, a random code with rate $\frac{L-1}{L}(1-h_2(\rho)) - \frac{h_2(2\rho-2\rho^2)-h_2(\rho)}{L} + \delta$ is not (ρ, L)-list decodable.*

From these two theorems, we note the following. If we let δ tend to 0, the upper bound provided by Theorem 26 is smaller than that provided by Theorem 25 as $(3+\frac{1}{L-1})h_2(\rho) - h_2(2\rho-2\rho^2) < 1$, assuming ρ is not too large.

References

1 Noga Alon, Boris Bukh, and Yury Polyanskiy. List-decodable zero-rate codes. *IEEE Transactions on Information Theory*, 65(3):1657–1667, 2018.

2 Laszlo Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. Bpp has weak subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1990.

3 Vladimir M Blinovsky. Code bounds for multiple packings over a nonbinary finite alphabet. *Problems of Information Transmission*, 41(1):23–32, 2005.

4 Volodia M. Blinovsky. Bounds for codes in the case of list decoding of finite volume. *Problems of Information Transmission*, 22(1):7–19, 1986.

5 Mahdi Cheraghchi, Venkatesan Guruswami, and Ameya Velingker. Restricted isometry of fourier matrices and list decodability of random linear codes. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 432–442, 2013. doi:10.1137/1.9781611973105.31.

6 Peter Elias. List decoding for noisy channels. *Wescon Convention Record, Part 2*, pages 94–104, 1957.

7 Anna C Gilbert, Hung Q Ngo, Ely Porat, Atri Rudra, and Martin J Strauss. l2/l2-foreach sparse recovery with low risk. In *International Colloquium on Automata, Languages, and Programming*, pages 461–472. Springer, 2013.

8 Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32. ACM, 1989.

9 Venkatesan Guruswami, Johan Håstad, and Swastik Kopparty. On the list-decodability of random linear codes. *IEEE Trans. Information Theory*, 57(2):718–725, 2011. doi:10.1109/TIT.2010.2095170.

10 Venkatesan Guruswami, Johan Håstad, Madhu Sudan, and David Zuckerman. Combinatorial bounds for list decoding. *IEEE Trans. Information Theory*, 48(5):1021–1034, 2002. doi:10.1109/18.995539.

11 Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 658–667, 2001. doi:10.1109/SFCS.2001.959942.

12 Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 812–821, 2002.

13 Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 126–135, 2003.

14 Venkatesan Guruswami and Piotr Indyk. Efficiently decodable codes meeting gilbert-varshamov bound for low rates. In *SODA*, volume 4, pages 756–757. Citeseer, 2004.

15 Venkatesan Guruswami, Ray Li, Jonathan Mosheiff, Nicolas Resch, Shashwat Silas, and Mary Wootters. Bounds for list-decoding and list-recovery of random linear codes. *IEEE Transactions on Information Theory*, 2021.

16 Venkatesan Guruswami and Jonathan Mosheiff. Punctured large distance codes, and many reed-solomon codes, achieve list-decoding capacity. *arXiv preprint*, 2021. arXiv:2109.11725.

17 Venkatesan Guruswami, Jonathan Mosheiff, Nicolas Resch, Shashwat Silas, and Mary Wootters. Threshold rates for properties of random codes. *IEEE Transactions on Information Theory*, 2021.

18 Venkatesan Guruswami and Srivatsan Narayanan. Combinatorial limitations of average-radius list-decoding. *IEEE Transactions on Information Theory*, 60(10):5827–5842, 2014.

19 Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

20 Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *Journal of the ACM (JACM)*, 56(4):1–34, 2009.

21 Venkatesan Guruswami and Salil Vadhan. A lower bound on list size for list decoding. *IEEE Transactions on Information Theory*, 56(11):5681–5688, 2010.

22 Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes & applications. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 204–215. IEEE, 2017.

23 Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. *Information and Computation*, 261:202–218, 2018.

24 Piotr Indyk, Hung Q Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1126–1142. SIAM, 2010.

25 Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

26 Ray Li and Mary Wootters. Improved list-decodability of random linear binary codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

27 Richard J Lipton. Efficient checking of computations. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 207–215. Springer, 1990.

28 Jonathan Mosheiff, Nicolas Resch, Noga Ron-Zewi, Shashwat Silas, and Mary Wootters. Ldpc codes achieve list decoding capacity. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 458–469. IEEE, 2020.

29 Hung Q Ngo, Ely Porat, and Atri Rudra. Efficiently decodable error-correcting list disjunct matrices and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 557–568. Springer, 2011.

30 Nicolas Resch. List-decodable codes: (randomized) constructions and applications. *School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., CMU-CS-20-113*, 2020.

31 Atri Rudra and Mary Wootters. Every list-decodable code for high noise has abundant near-optimal rate puncturings. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 764–773. ACM, 2014.

32 Atri Rudra and Mary Wootters. Average-radius list-recovery of random linear codes. In *Proceedings of the 2018 ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2018.

33 Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

34 Mary Wootters. On the list decodability of random linear codes with large error rates. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 853–860, 2013. doi:10.1145/2488608.2488716.

35 Jack Wozencraft. List decoding. *Quarter Progress Report*, 48:90–95, 1958.

36 Yihan Zhang, Amitalok J Budkuley, and Sidharth Jaggi. Generalized list decoding. In *2020 Information Theory and Applications Workshop (ITA)*, pages 51–1. IEEE, 2020.

37 Victor Vasilievich Zyablov and Mark Semenovich Pinsker. List concatenated decoding. *Problemy Peredachi Informatsii*, 17(4):29–33, 1981.

Explicit and Efficient Construction of Nearly Optimal Rate Codes for the Binary Deletion Channel and the Poisson Repeat Channel

Ittai Rubinstein
Blavatnik School of Computer Science, Tel-Aviv University, Israel
QEDMA Quantum Computing, Tel-Aviv, Israel

Abstract

Two of the most common models for channels with synchronisation errors are the Binary Deletion Channel with parameter p (BDC_p) – a channel where every bit of the codeword is deleted i.i.d with probability p, and the Poisson Repeat Channel with parameter λ (PRC_λ) – a channel where every bit of the codeword is repeated Poisson(λ) times.

Previous constructions based on synchronisation strings yielded codes with rates far lower than the capacities of these channels [6, 9], and the only efficient construction to achieve capacity on the BDC at the time of writing this paper is based on the far more advanced methods of polar codes [23].

In this work, we present a new method for concatenating synchronisation codes and use it to construct simple and efficient encoding and decoding algorithms for both channels with nearly optimal rates.

2012 ACM Subject Classification Mathematics of computing → Coding theory

Keywords and phrases Error Correcting Codes, Algorithmic Coding Theory, Binary Deletion Channel

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.105

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2111.00261?context=math.IT

Funding *Ittai Rubinstein*: The Deutsch institute fund.

Acknowledgements We thank Roni Con, Aviad Rubinstein and Muli Safra for their helpful comments on previous drafts.

1 Introduction

The theory of error-correcting–codes deals with methods for encoding messages to be sent over noisy media in such a manner that they can be correctly decoded afterwards. Initially introduced by Shannon [22], this field has proven to be instrumental in understanding the theory of computation, and has had a wide variety of applications in other fields, such as communications, and computational biology [24].

The most commonly considered models are "Synchronous Models" - models where the message may be altered or erased, but every letter that was received can be traced back to its original position in the transmitted message. This category includes models such as the Binary Symmetry Channel (BSC) where some of the bits in the transmitted message are flipped (i.e. changed from 1 to 0 or vice versa), and the Binary Erasure Channels (BEC) where some of the bits of the transmitted message are replaced with a question mark (but are not removed, thus preserving the alignment between the transmitted message and the received message). These models can be adversarial (such as [14]), where the code must correct any error the channel may produce, or average-case (such as [22]), where the effect of the channel is random and decoding only needs to succeed w.h.p.


Synchronous models are very well researched, and have a variety of efficient encoding and decoding algorithms [24]. The main method used for such channels are linear error-correcting–codes and they rely heavily on the fact that the letters of the received messages can be mapped back into letters of the transmitted messages.

However, in many real world applications, deletions and insertions can cause the received codewords to be misaligned and the decoder must also deal with synchronisation errors [18, 17]. Perhaps the most intuitive asynchronous channel is the Binary Deletion Channel. This is a channel where every transmitted bit is deleted i.i.d with probability p (where $1 > p > 0$ is a parameter of the channel).

Here, unlike with the binary erasure channel, deleted bits are not replaced with a question mark, but are completely removed from the sequence, shifting the rest of the received codeword. For instance, when transmitting the codeword $w = (1, 0, 0, 0, 1, 1)$, the BEC may result in the received codeword $r_{\text{BEC}} = (1, 0, ?, ?, 1, ?)$ while a similar pattern of deletions would result in the received codeword $r_{\text{BDC}} = (1, 0, 1)$.

This channel represents a simple model for many real-life systems in which there is a loss of information due to synchronisation errors. Moreover, the tools developed for this channel have been instrumental in a variety of other fields [18, 17].

A slightly more complex model which we will also consider, is the Poisson Repeat Channel with parameter $\lambda > 0$ (PRC_λ). This is a channel where every transmitted bit is received Poisson(λ) times. While originally used to help prove lower bounds on the capacity of binary deletion channels, Poisson repeat channels are interesting in their own right. Indeed, Poisson repeats can model every-day examples like a sticky key in a keyboard, as well as deeper technological issues such as errors in single photon generation (a crucial step in light-based quantum computing) [2].

However, the PRC also presents a slightly greater challenge. This is because bits can now be received more than once, resulting in several new types of synchronisation errors. Continuing with our previous example where the codeword $w = (1, 0, 0, 0, 1, 1)$ was transmitted, the received codeword in the BDC model will always start with at most a single 1 bit, unless all three 0s were deleted. In the Poisson repeat channel, this is not the case and the received codeword $r_{\text{PRC}} = (1, 1, 0, 1)$ is a possible outcome.

Several previous results have shown an interesting connection between these two noise models. For instance, Mitzenmacher and Drinea's lower bound for the capacity of the BDC channel [19] is based on their previous lower bound for the capacity of the PRC channel, and Con and Shpilka's constructive codes for the BDC channel [6] are also applicable to the PRC channel.

In this paper we will focus our attention on these two channels, but we believe our tools and approaches can be applicable to other asynchronous channels as well.

1.1 Previous Work

Asynchronous channels present us with a varied field of research, and we will not be able to cover all of its results here. The excellent surveys by Mitzenmacher, Cheraghchi et al and Mercier et al [18, 17, 5] give a more detailed background.

Determining the capacity of the BDC_p channel, remains an open problem, and so far it has been answered only for some extremal cases. When $p \to 0$, the capacity of this channel is $1 - h(p)$ [15] (where $h(\cdot)$ is the binary entropy function), when $p \to 1$, the capacity is $\mu(1-p)$ where $\frac{1}{9} < \mu \leq 0.4143 - o(1)$ [19, 8] (where $o(1)$ is w.r.t the block size n), and [25] give lower bounds for some of the intermediate values of p.

However, the lower bound on the capacity by Mitzenmacher and Drinea [19] is not based on an efficient construction. Recently, Con and Shpilka constructed a family of error-correcting-codes with rate $r \geq \frac{1-p}{16}$ for the BDC_p channel and $r \geq \frac{\lambda}{17}$ for the PRC_λ (when $\lambda < \frac{1}{2}$) [6], improving upon the results of Guruswami and Li who presented the first explicit codes with rate $\Theta(1-p)$ in [9]. This was improved upon by, Tal et al and by Pfister and Tal who showed that polar codes can be used to construct efficiently decodable codes with optimal rates for the BDC channel [23, 20].

Haeupler and Shahrasbi [13] construct a family of efficient codes for InsDel channels with a sufficiently large alphabet, and Haeupler, Rubinstein and Shahrasbi improve the decoder in [12], reaching quasi-linear complexity.

In the adversarial model, Guruswami and Wang [10] showed that there are codes with rate $1 - \widetilde{O}\left(\sqrt{\delta}\right)$ that can correct up to δn errors. This rate was improved by Cheng et al. [3] and further by Haeupler [11]. More recently it was shown by Con, Shpilka and Tamo [7] that, surprisingly, linear error correcting codes are also effective for adversarial InsDel channels.

1.2 Main Contribution

In this work we will construct a family of codes, with efficient encoding and decoding algorithms, whose rates are arbitrarily close to the capacities of the BDC and PRC channels. Unlike the previous results of Tal et al. and of Pfister and Tal [23, 20], this construction does not require the more advanced machinery of polar codes. Furthermore, the construction presented has a decoding error probability of $\exp\left(-\Theta\left(n^{\frac{1}{6}}\right)\right)$ (where n is the block length) with a quasi-linear complexity decoder, while Pfister and Tal's code requires $n^{\frac{3}{2}+\varepsilon}$ time for the same error probability.

Both our code and Pfister and Tal's construction assume that we are given some inner code which achieves a high rate on the channel but which does not necessarily have efficient encoding and decoding algorithms, and both methods produce a new version of this code with efficient encoding and decoding. However, Pfister and Tal's construction requires this code to be generated by a hidden-Markov distribution, while the construction presented here can be used with any inner code. Li et al. proved that there exists a finite hidden-Markov distribution code that achieves capacity for this channel [16] and this model can clearly be found in $O(1)$ time using an enumeration technique similar to the one described in Section 4.1, but such a code has not been found yet.

▶ **Theorem 1** (Main Result (informal)). *Any (possibly inefficient) family of codes for either the BDC or the PRC channel can be converted into a family of codes for the same channel with an arbitrarily close rate, that has encoding and decoding algorithms with a quasi-linear complexity.*

It should be noted that while this complexity is asymptotically very good, it hides within it a very large constant factor, and while we do not have an exact bound on it, we expect it to grow as the rate of the code approaches the capacity of the channel.

Nonetheless, this allows us to construct to construct a family of efficient codes which achieve rates of $\frac{1-p}{9}$ for the BDC_p channel, thus completing the line of works started by Guruswami and Li [9] and continued by Con and Shpilka [6].

Our construction is based on a new technique for tracking inner codewords in a concatenation of synchronisation codes, which we hope may be useful in other cases as well:

Most previous constructions use buffers of 0s as the delimiters between inner codewords. Then, by bounding the probability that the channel would transform any substring of the inner codeword into a sufficiently long sequence of 0s, they can ensure that long sequences of

0s in the received codeword mostly correspond to delimiters. In other words, when separating the received codeword into inner codewords, one searches the entire string for patterns that may have come from a delimiter.

In our construction, we will use delimiters in a very different manner. Instead of searching the entire codeword for the delimiters, we will use our knowledge of the length of the inner codeword and the average expansion of the channel to produce a prior estimate for the distance between consecutive delimiters. Using this prior estimate, we are able to find the delimiters one after another, without looking at the entire codeword.

This new method allows us to drastically reduce the probability that even a single inner delimiter will be missed, while reducing the overhead of the delimiters to a negligible fraction of the codeword. The ability to decode under the assumption that all delimiters will be found simplifies the outer code in our construction, and the fact that we will not search for a delimiter within an inner codeword allows us to use a general inner code, resulting in a nearly optimal rate.

1.3 An Overview of Con and Shpilka's Construction

Since our approach will be similar to that of [6], we will begin with a short overview of their construction, which is based on a concatenation of codes.

Initially, the message is divided into segments of length $\sigma = O(1)$. These are thought of as members of an alphabet Σ of size $|\Sigma| = 2^\sigma$ and can be encoded using [13]. Each letter in the encoded message is then converted back to a string of σ binary symbols and is encoded using an inner code. Since the inner code is only applied to strings of length $\sigma = O(1)$, it can be inefficient without affecting the asymptotic complexity of the encoding / decoding algorithms. The encoded strings are appended and separated by delimiters - in this case buffers of 0s.

Con and Shpilka's construction has a quasi-linear encoding algorithm and a quadratic decoding algorithm, where the computational bottleneck of the decoding algorithm comes from decoding the outer code. The improved decoding algorithm for Haupler and Shahrasbi's code [12] can be used with Con and Shpilka's code to reduce the complexity of their decoding algorithm to a quasi-linear time as well.

1.4 Sketch of the Proof

Our construction will be based on a similar strategy, but with a few key differences. Firstly, we note that most of the overhead of this code is caused by the fact that the inner code is designed to preserve a certain structure.

By removing this structure we are able to significantly increase the rate of our code. However, this comes at a cost - separating unstructured codewords from the delimiters is made far more difficult. We overcome this using a slightly more complex construction of delimiters and a careful analysis.

In addition, the delimiters themselves account for another constant fraction of the overhead of Con and Shpilka's code. By using a recursive concatenation, we are able to reduce the cost of these delimiters to a negligible fraction of the overhead.

At each step of this recursion, we will assume that there exists a BDC/PRC code with message length k and block length n, and we will construct a code with message length k^2 and block length $n' = (1 + o(1))\, nk$.

We will do this by separating the k^2 bit message into k strings of k bits each. We will think of each of these k-bit strings as a member of an alphabet of size $|\Sigma| = 2^k$ and use a ReedSolomon $[k + 2t, k, t]_{2^k}$ code to give it some redundancy (i.e. a Reed-Solomon code over a field of size 2^k, message length k and block length $k + 2t$ with distance $t = o(k)$).

If we were constructing a code for a discrete memoryless channel (DMC) such as the BEC, this step might not be very surprising, because without synchronisation errors we could map the received codeword back into the letters of the Reed-Solomon codeword. However, in our case this might seem somewhat counter-intuitive, since Reed Solomon codes offer no protection against the synchronisation errors we are trying to correct. This step works for asynchronous channels as well because our delimiters are designed to fully preserve the synchronisation *between* inner codewords and the Reed Solomon code will only need to compensate for a small number of local decoding failures of inner codewords.

We will encode each of the $k + 2t$ letters of the Reed Solomon codeword using our inner code. This will output a list of $k + 2t$ strings of length n bits each. Finally, we will append these strings after inserting a delimiter between each two.

The delimiters will be made up of two parts: a positioning string which will help us find the delimiter and two partitioning strings which will help us separate between the delimiters and the inner codewords themselves. The reason that we need partitioning strings, is that we make no assumptions about the structure of the inner code. Therefore, any sequence of bits that originated from the delimiter could have originated from the inner codeword.

For instance, suppose we had used buffers of 0s as our delimiters. Since we make no assumptions about the structure of the inner code, we have no way of knowing whether or not the inner codeword begins with a sequence of 0s, so we can't tell where the delimiter ends and the inner codeword begins.

This makes separating the two a very difficult task and will be at the heart of our construction. Our separation between inner codewords and delimiters will not be completely accurate, but we will be able to bound the effect this has on the decoding failure probability by using the fact that the inner code is designed to deal with (some) deletions.

Both parts of the delimiter will be based on an idea we call "valleys". Similar to the markers defined by Cheraghchi et al [4], we will define valleys to be a long sequence of 0s followed by a long sequence of 1s (when looking at the cumulative sum of the string minus $\frac{1}{2}$, these translate to local minima - see Figure 1). Unless one of these two sequences is completely deleted by the channel, a valley in the transmitted message will result in a valley in the received message.

We will use this observation to align indices within the received message to their source in the transmitted message. We will start with an estimate of where the center of some valley from the transmitted message should be in the received message, and then we will go downhill to the nearest local minima (see Algorithm 1). By bounding both the probability that one of these sequences was removed and the probability that our initial estimate was outside the bounds of the received valley, we can correlate the center of the received valley to the center of the transmitted valley with high probability.

Each positioning string will be a long valley, and each partitioning string will be a short valley. We set the positioning string to be long, because we need to be able to find its center, given only a very rough estimate. On the other hand, setting the partitioning strings this long would reduce the accuracy of its separation from the inner codewords.

The decoding algorithm will be similar to the encoding algorithm, but in a reversed order. First, we will align the received message by locating the centers of the positioning strings. Then we will use the partitioning strings to separate the delimiters from the inner codewords, and apply the inner code decoding to obtain the inner codewords. Finally, we use the Reed Solomon decoding to correct up to t errors that may have occurred.

1.5 Organization

In Section 2 we will define basic notations, show some well-known inequalities that we will use in our analysis and present the basic building block of our construction. Section 3 contains the construction of our recursive step and in Section 4 we will prove the basis of the recursion and show how we can connect it to the recursive step. In Section 5 we will extend our results to the Poisson repeat channel. Finally, in Section 6 we will discuss the implications and limitations of these results, as well as potential avenues for future research. We leave the slightly more technical proof that of our bound for the decoding failure probabilities to the full version (see Section 6 of the full version [21]).

2 Preliminaries

2.1 Average-case Codes

▶ **Definition 2.** *Let Σ be a finite set and let $k, n \in \mathbb{N}$ be positive integers.*

We will say that C is a random channel *acting on the* alphabet Σ *and* block-length n *if it maps any member of Σ^n to a distribution on some set $\mathcal{Y}$.*

Furthermore, we will say that encoding and decoding algorithms $E : \Sigma^k \to \Sigma^n, D : \mathcal{Y} \to \Sigma^k$ *for this channel with* message length k *have* rate $\rho = \frac{k}{n}$ *and a* decoding falure probability (DFP) *of*

$$\delta = \max_{m \in \{0,1\}^k} \{\Pr[D(C(E(m))) \neq m]\}.$$

In other words, the DFP of a code is the probability that a message will be decoded incorrectly if the message was chosen adversarially, but the effects of the channel were random.

▶ **Definition 3.** *Let C be a random channel. We will say that $\mathcal{F} = \{(E_i, D_i)\}_{i \in \mathbb{N}}$ is a family of codes for C if:*

- *The message lengths k_i of E_i, D_i are unbounded ($k_i \xrightarrow[i \to \infty]{} \infty$).*
- *The DFPs δ_i of E_i, D_i in C are vanishing ($\delta_i \xrightarrow[i \to \infty]{} 0$).*

Throughout the decoding process we will often attempt to align the received message with the transmitted codeword.

▶ **Definition 4.** *When the channel acts independently on each letter of the input (i.e. when $C(b_1, \ldots, b_n) = C(b_1) \ldots C(b_n)$), we will say that the ith coordinate of a message transmitted over some asynchronous channel and the jth coordinate of the received message are* aligned, *if the first $i-1$ letters of the transmitted message were mapped to a string of length at most j by the channel and the first i letters of message were mapped to at least j letters by the channel.*

2.2 Probability Inequalities

Throughout this paper we will bound the probability that several parts of our construction will fail. This will require several bounds on the tails of Poisson and binomial distributions, which we will present in this section. Perhaps the most important tool at our disposal will be the Chernoff bound.

▶ **Theorem 5** (Chernoff Bound for Binomial and Poisson Distributions). *For all* $\frac{1}{2} > \varepsilon, p > 0$, $n \in \mathbb{N}$ *and* $\lambda > 0$*:*

$$\Pr\left[|Bin(p,n) - pn| > \varepsilon pn\right] \leq 2\exp\left(-\frac{1}{4}\varepsilon^2 pn\right)$$

$$\Pr\left[|Poisson(\lambda) - \lambda| > \varepsilon\lambda\right] \leq 2\exp\left(-\frac{1}{4}\varepsilon^2\lambda\right).$$

This theorem is based on the Chernoff bound (see Section 2.2 of [1]) and an analysis of entropy functions. We leave its slightly technical proof to the full version (see Appendix 8.1 of [21]).

Another special case of the Chernoff bound (see Appendix 8.2 of [21]) is

▶ **Theorem 6** (Binomial Tail). *For all* $\alpha > e^2$, $\frac{1}{\alpha+1} > p > 0$ *and* $n \in \mathbb{N}$*:*

$$\Pr\left[Bin(p,n) > (\alpha+1)pn\right] \leq \exp\left(-\frac{1}{2}\log(\alpha)\,\alpha pn\right).$$

2.3 Aligning Valleys

Our main tool in aligning the messages will be an idea we call valleys.

▶ **Definition 7.** *We will define a* valley *with* faces *of length* x *and* y *to be a string of* x *0s followed by* y *1s.*

In other words $Valley(x,y) = 0^x 1^y$.

We will define the center *of a valley to be the index where it transitions from* 0*s to* 1*s (i.e. the index of the last* 0 *bit).*

Suppose the codeword contains some valley $V = \text{Valley}(x,y)$. This is encoded by the channel into a valley whose face lengths are i.i.d $\text{Bin}(1-p,x)$ and $\text{Bin}(1-p,y)$ (or $\text{Pois}(\lambda,x)$ and $\text{Pois}(\lambda,y)$ for the PRC channel). Assuming neither side of the valley was completely deleted, the center of the received valley and the center of the transmitted valley should be aligned.

Our definition of valleys is similar to the markers considered in [4], but we will view our valleys very differently. Cheraghchi et al construct their inner code so that is is unlikely to contain a long sequence of 0s. This allows them to find the delimiters by searching for the next sequence of 0s and then use the marker to zero in on the center of the delimiter.

In our decoding algorithm, we will produce an approximate estimate for where the center of a certain valley should be, and by finding its actual center we will be able to align that index to a position within the transmitted message. We will do so using Algorithm 1.

Intuitively, this algorithm works by using the observation that if the valley was not distorted by the channel too much and our initial estimate for its center was not off by too much, then we known whether or not our guess was to the right or to the left (since the left face of the valley has 0s and the right face has 1s). We can then correct our position. If we are on the left face, go right until we reach the center and vice versa.

In order to clarify this approach we will consider a simple example. Suppose we are building a code for the BDC_p channel with parameter $p = \frac{1}{2}$ and we decide to use a valley of length 32 on either side to align some index.

That is, our transmitted message would be as follows (where the bold digit signifies the center of the valley).

$$m = [\ldots]00000000000000000000000000000000\mathbf{0}11111111111111111111111111111111[\ldots]$$

Algorithm 1 Align Valley.

Input : a received codeword $w \in \{0,1\}^*$, estimated center of valley j
Output : the center of the valley j'
$j' \leftarrow j$;
if $w[j] = 0$ **then**
  while $w[j'] \neq 1$ **do**
    $i \leftarrow i + 1$;
  end
  return $j' - 1$;
else
  while $w[j'] \neq 0$ **do**
    $i \leftarrow i - 1$;
  end
  return j' ;
end

Suppose the channel made the following deletions:

$$d = [\ldots] \cdots 0 \cdot \cdot \cdot \cdot \cdot \cdot 000 \cdot 0 \cdot \cdot 00 \cdot \cdot \cdot \cdot 0 \cdot 0 \cdot 0 \cdot \cdot 0 \cdot \cdot 1 \cdot 1 \cdot 11 \cdot 11 \cdot 1 \cdots 111 \cdot 1111111 \cdot \cdot \cdot \cdot \cdot 1[\ldots]$$

Furthermore, assume that we have some prior estimate that the center of the received valley should be 5 bits from its actual position.

Then the received codeword would be as follows, where the underlined digit signifies our prior estimate for the center of the valley.

$$w = [\ldots]00000000000111111\underline{1}111111111111[\ldots]$$

Algorithm 1 would start from this initial estimate and advance to the left, returning the correct center of the valley.

$$w = [\ldots]00000000000\mathbf{0}11111111111111111[\ldots]$$

In Figure 1 we show a geometric representation of this algorithm.

3 Recursive Step

In this section we will define the recursive step in our construction, explain the rationale behind it and begin to prove its correctness. Intuitively, the main theorem we will prove here is that any error correcting code for the BDC with message length k can be transformed into a code with message length k^2 and that the DFP, rate and complexity of the new code "scale well". More formally we will show that:

▶ **Theorem 8** (Recursive Step). *For some constants $c_1, c_2, c_3, k_0, \delta_0 > 0$ and for all $k > k_0$, $d > 0$, $2^{k-1} - k > t > 0$, $\delta_0 > \delta > 0$, $1 > p > 0$ and any error correcting code C for the BDC_p channel with message length k, block length n and DFP δ, there exists an error correcting code C' for the BDC_p with message length k^2, block length $n' \leq (k + 2t)(n + \frac{d}{1-p})$ and DFP:*

$$\delta' \leq \Pr\left[Bin\left(\delta^{c_1}, k + 2t\right) > t\right] + c_3\left(k + 2t\right) \exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right).$$

Furthermore, there exist an encoder and decoder for C' with time complexity $\widetilde{O}\left(n'\right)$ using up to $k + 2t$ calls to the encoder and decoder of C.

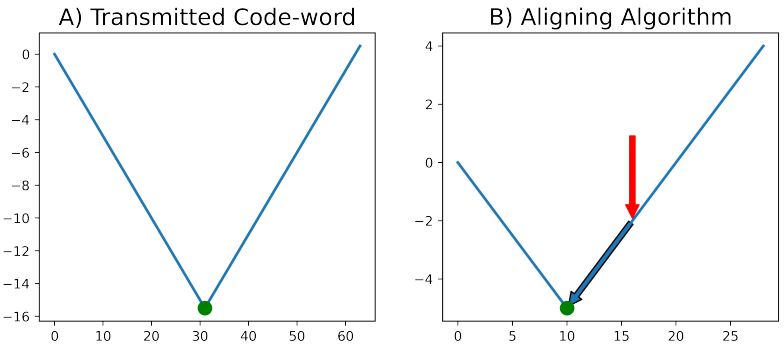


Figure 1 A visual representation of the algorithm for aligning valley centers on a specific example of a valley with parameters $(32, 32)$, being sent over a BDC_p channel with parameter $p = \frac{1}{2}$. The plots go up by $\frac{1}{2}$ whenever the relevant string has a 1 and down by $\frac{1}{2}$ whenever it has a 0. **A**) The transmitted message, with the green dot representing the center of the valley. **B**) First, the channel deletes some of the bits resulting in a skewed valley. We are given some prior estimate for the center of the valley (red arrow). The algorithm goes down the valley (blue arrow), until it terminates at the center (green dot).

We will use this construction in two scenarios: to improve the base of the recursion and for the steps of the recursion. In Table 1, we present the parameters of Theorem 8 and the asymptotic values for both use cases.

Table 1 The parameters of Theorem 8, and their asymptotic values in the two use cases.

Parameter	Description	Recursion Base | Step
k	Message Length	
n	Block Length	$\Theta\left(\frac{k}{1-p}\right)$
d	Delimiter Length	$\Theta\left(k^{\frac{2}{3}}\right)$
t	Reed Solomon Redundancy	$o(k) \mid \Theta\left(k^{\frac{2}{3}}\right)$
δ	Inner Code DFP	$o(1) \mid \exp\left(-\frac{c_2}{2}k^{\frac{1}{6}}\right)$
c_1, c_2, c_3	Constants	$\frac{1}{34}$, $\frac{1}{256}$, 6

The first set of values is used for improving of the base of the recursion. In this case, our only bound on δ will be that it is an arbitrarily small constant, but its relation to k will not be exactly known. Our goal in this step of the construction will be to reduce the error probability at the cost of an arbitrarily small but non-negligible cost to the rate of the code, and we will accomplish this by setting t to be of the order of $\Theta\left(\delta^{\frac{c_1}{2}}k\right)$.

The second scenario is that of a step in our recursion. We will construct our recursion in such a manner that the DFP of the inner code will be bounded by $\delta < \exp\left(-\frac{c_2}{2}k^{\frac{1}{6}}\right)$. By setting $t = d = \Theta\left(k^{\frac{2}{3}}\right)$ we will be able ensure that on the one hand, the DFP of the final code will be $\delta' \ll \exp\left(-\frac{c_2}{2}\left(k^2\right)^{\frac{1}{6}}\right)$ while on the other, the rate of the code will be reduced by only a factor of $1 - O\left(\frac{t+d}{k}\right) = 1 - O\left(k^{\frac{1}{3}}\right) = 1 - o(1)$.

The construction of the code $C' : \{0,1\}^{k^2} \to \{0,1\}^{n'}$ is as described in the introduction. First, the input string is split into k parts of length k each. These are considered as elements in an alphabet of size 2^k and a Reed Solomon encoding with parameters $[k+2t, k, t]_{2^k}$ is applied to them. Each of these is encoded using the encoder of C, a delimiter is appended to each codeword and the concatenation of all of these strings is outputted.

Similarly, the decoding algorithm works by locating the delimiters, separating them from the inner codewords and then decoding each inner codeword using the decoder of C. The decoded inner codewords are once again viewed as letters in an alphabet of size 2^k and the Reed Solomon decoding is applied.

We will define a delimiter with parameters α, β to be a valley of length β surrounded by two valleys of length α. In other words

$$\text{Delimiter}\,(\alpha,\beta) = \text{Valley}\,(\alpha)\,\text{Valley}\,(\beta)\,\text{Valley}\,(\alpha) = 0^\alpha 1^\alpha 0^\beta 1^\beta 0^\alpha 1^\alpha$$

The exact values of α and β are presented in the full version (see Section 6 of [21]), but they will be of the order of $\alpha = \Theta\left(\frac{\log\left(\frac{1}{\delta}\right)}{1-p}\right)$ and $\beta = \frac{d}{2(1-p)} - 2\alpha = \Theta\left(\frac{k^{\frac{2}{3}}}{1-p}\right)$.

In order to complete the construction, we still need to provide methods of locating the delimiters in the received codeword and separating them from the inner codewords. These steps will be explained in the following subsections.

3.1 Positioning Strings

We will use the positioning strings to locate the delimiters one at a time.

Let us denote by L_i the location of the center of the ith positioning string in the received codeword. We can estimate the location of the center of the first positioning string as being around $\mathbb{E}[L_1] = (1-p)(n+2\alpha+\beta)$ bits from the beginning of the received codeword. However, this is only a rough estimate and in reality $L_1 \sim \text{Bin}\,(n+2\alpha+\beta, 1-p)$. We want to find the exact center.

This is where the valleys come into play. With high probability, $|L_1 - \mathbb{E}[L_1]|$ will not be much larger than $\sigma_{L_1} = \sqrt{p(1-p)(n+2\alpha+\beta)}$ (where σ_{L_1} is the standard deviation of L_1), and at least $\frac{1-p}{2}\beta$ of the bits on either side of the valley will survive the channel. Therefore, so long as $\sqrt{p(1-p)(n+2\alpha+\beta)} \ll \frac{1-p}{2}\beta$, we can expect Algorithm 1 to find the value of L_1 w.h.p.

Once we have found the ith delimiter, we go on to search for the $i+1$-th. At each step we use the aligned center of the previous positioning string L_i to obtain an estimate for $L_{i+1} \sim L_i + \text{Bin}\,(n+4\alpha+2\beta, 1-p)$, and use its valley to correct our estimate. In the full version we show that the probability that even a single positioning string will not be correctly found is at most $(k+2t)\exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right)$ (see Section 6 of [21]).

3.2 Partitioning Strings

Once we have located the centers of all of the delimiters, we still need to separate them from the inner codewords. This step is surprisingly difficult, because, unlike Con and Shpilka [6] who designed their inner code to preserve a certain structure that would help them differentiate it from the delimiters, we reduce our overhead precisely by making no assumptions about the structure of the inner code.

For instance, if we were to use buffers of 0s as our delimiters and the inner codewords surrounding the delimiter happened to start / end with sequences of 0s, then we would have had a hard time telling which of the 0s belonged to the delimiter and which belonged to the inner codeword. The same applies for any choice of the delimiters.

Our solution to this problem will be based on two ideas. First, we will attempt to approximate the correct separation as accurately as possible. However, this will not yield a perfect separation and we will need to mitigate the effects of this inaccuracy.

The first step will be accomplished by surrounding the positioning string with two valleys. When decoding we can find the centers of these valleys by going from the positioning string until the ends of its valley and then proceeding to the bottom of the next/previous valleys.

Once we have found the center of a partitioning string, we can estimate the length of the faces of its valley. Each of those will be i.i.d distributed according to $F \sim \text{Bin}\,(1-p,\alpha)$. By guessing $f = (1-p)\alpha$ we will get a good approximation of F (to within an error of $\Theta\left(\sqrt{p(1-p)\alpha}\right)$).

If we were to use this estimate as the separation between the delimiter and the inner codeword we would have a two sided error probability, either due to overshooting (attributing some bits of the inner codeword to the delimiter) or due to undershooting.

In the former case, this would cause us to delete some additional bits from the inner codeword. Since the inner code is designed to deal with deletions, it stands to reason that it might also be able to decode received messages if they were also subject to a small amount of additional deletions. This claim is not straightforward, since the inner code deals with random deletions and we will be subjecting it to a very structured set of deletions. However, with a careful analysis we can bound the effect these deletions can have on the DFP of the inner code.

However, in the latter case we would end up erroneously inserting part of the delimiter to one end of the inner codeword. Since the inner code is for a deletion channel, it might not be able to correctly decode the inner codeword, even after a small number of insertions. Therefore, we have no way of bounding the effect this could have on its DFP.

That is why we want to reduce the probability of undershooting significantly more than the probability of overshooting. To do this, instead of taking the estimate $f = \mathbb{E}F$, we will use the estimate $f = \mathbb{E}\left[F\right] + \eta\sigma_F$ for some parameter η (where σ_F is the standard deviation of F).

If F was at most η standard deviations from its expectancy, then we would have $f - F \in [0, 2\eta\sigma_F]$. The lower bound means that no bits from the delimiter would ever trickle into the inner codeword, and the upper bound limits the number of bits from the inner codeword we will delete by accidentally attributing them to the delimiter.

In the full version of the paper we bound both the probability that $f - F \notin [0, 2\eta\sigma_F]$ and the effect these deletions could have on the DFP of the inner code (see Section 6 of [21]).

4 The Recursive Construction

In the previous section we defined the manner in which we recursively concatenate our code to increase the message length at a negligible cost to the rate of the code and the complexity of its encoder and decoder. In this section we will construct the base of this recursion and set the parameters for the recursive steps.

4.1 The Base of the Recursion

When choosing the base of the recursion we are in essence transforming a lower bound on the capacity of the channel to an actual error correcting code. We will do this in a very inefficient manner, but as with Con and Shpilka's inner code, since this code will have message and block length of $O(1)$, this does not affect the asymptotic complexity of our construction.

In order to begin our recursion, we will need a base code with sufficiently low DFP and sufficiently large message length. By definition, a lower bound on the capacity of the channel is a proof that there exists a family of codes for the channel with $r \geq \text{capacity} - \varepsilon$ for any $\varepsilon > 0$. Therefore for any $k_0, \delta_0 > 0$ there are some codes in that family with message length $k > k_0$ and DFP $\delta < \delta_0$. Let κ be the smallest such block length. Since κ is determined by k_0 and δ_0, and since k_0 and δ_0 are constant parameters of our construction, $\kappa = f(k_0, \delta_0) = O(1)$ must also be constant.

We will enumerate over values of $k > k_0$ and for each of them we will attempt to construct a base code. This process will terminate when $k = \kappa$, so it will require only a finite number of iterations.

Since we are looking for an encoding map from some finite set of messages $\{0,1\}^k$ (where k is the message length) to some finite set of codewords $\{0,1\}^n$ (where $n < \frac{9}{1-p}k = O(1)$ is the block length), and a decoding map from $\{0,1\}^{\leq n} \to \{0,1\}^k$, there are only finitely many pairs of this form. By enumerating over all of these pairs and evaluating their DFP, we will be able to find a base code with message length k if one exists.

Since all of the steps in this process had a constant complexity, our construction of the inner code had a constant $O(1)$ complexity. It should be noted that this algorithm is extremely inefficient and that we do not even know how to bound its complexity (except that it is $O(1)$). We hope that future research will address this issue.

4.2 Connecting the Recursive Steps

All that remains now is to combine the recursive step shown in Section 3 with the base case constructed in the previous subsection.

Throughout most of the recursion we will use the recursive step defined in Theorem 8 in the following setting:

$$\begin{aligned} \delta_k &< \exp\left(-\frac{c_2}{2}k^{\frac{1}{6}}\right) \\ d_k &= k^{\frac{2}{3}} \\ t_k &= k^{\frac{2}{3}}. \end{aligned} \tag{1}$$

Applying the recursion Theorem 8, for sufficiently large k, we have:

$$\delta_{k^2} \leq \Pr\left[\text{Bin}\left(\delta^{c_1}, k+2t\right) > t\right] + c_3\left(k+2t\right)\exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right). \tag{2}$$

We use Theorem 6 to bound the first term in the new decoding failure probability for sufficiently small δ, by

$$\Pr\left[\mathrm{Bin}\left(\delta^{c_1}, k+2t\right) > t\right] \leq \exp\left(-\frac{c_1}{4}\log\left(\frac{1}{\delta}\right)t\right) \leq \frac{1}{2}\exp\left(-\frac{c_2}{2}t\right)$$

and the second term for sufficiently large k by

$$(k+2t)\exp\left(-c_2\min\left\{d, \frac{d^2}{k}\right\}\right) = \exp\left(-c_2 k^{\frac{1}{3}} + O\left(\log\left(k\right)\right)\right) \leq \frac{1}{2}\exp\left(-\frac{c_2}{2}k^{\frac{1}{3}}\right).$$

Combining these inequality gives us a bound on the new DFP:

$$\delta_{k^2} \leq \exp\left(-\frac{c_2}{2}k^{\frac{1}{3}}\right). \tag{3}$$

The overhead of the code can also be easily bounded. From the recursion theorem, we know that $n_{k^2} \leq \left(n_k + \frac{d_k}{1-p}\right)(k+2t_k)$. Since $n_k \geq \frac{k}{1-p}$ (from the upper bounds on the capacity of these channels) and $d_k = t_k = k^{\frac{2}{3}}$, we have:

$$\begin{aligned} n_{k^2} &\leq \left(1+k^{-\frac{1}{3}}\right)^2 \frac{k^2}{k} n_k \leq \left[\left(1+k^{-\frac{1}{3}}\right)\cdot\left(1+k^{-\frac{1}{6}}\right)\right]^2 \frac{k^2}{\sqrt{k}} n_{\sqrt{k}} = \\ &= \underbrace{\left[\left(1+k^{-\frac{1}{3}}\right)\cdot\left(1+k^{-\frac{1}{6}}\right)\cdot\ldots\cdot\left(1+\left(k_{\mathrm{base}}\right)^{-\frac{1}{3}}\right)\right]^2}_{=:X} \frac{k^2}{k_{\mathrm{base}}} n_{k_{\mathrm{base}}}. \end{aligned} \tag{4}$$

Since $n_{k_{\mathrm{base}}} = \frac{1}{r_{\mathrm{base}}}k_{\mathrm{base}}$ (where r_{base} is the rate of the base code), it is easy to see that $r_{k^2} = \frac{k^2}{n_{k^2}} = \frac{1}{X}r_{\mathrm{base}}$. By bounding the value of X, we can bound the increase in the overhead of the code due to the recursion.

$$\begin{aligned} X &= \left[\left(1+k^{-\frac{1}{3}}\right)\cdot\left(1+k^{-\frac{1}{6}}\right)\cdot\cdots\cdot\left(1+\left(k_{\mathrm{base}}\right)^{-\frac{1}{3}}\right)\right]^2 \\ &\leq \exp\left(2k^{-\frac{1}{3}} + 2k^{-\frac{1}{6}} + \cdots + 2\left(k_{\mathrm{base}}\right)^{-\frac{1}{3}}\right) \leq \exp\left(2\frac{\left(k_{\mathrm{base}}\right)^{-\frac{1}{3}}}{1-\left(k_{\mathrm{base}}\right)^{-\frac{1}{3}}}\right). \end{aligned} \tag{5}$$

For a sufficiently large k_{base}, it is clear that this value can be set arbitrarily close to 1, giving our code a nearly optimal rate.

However, this does not conclude our construction, since in each step of the recursion we assumed that $\delta_k \leq \exp\left(-\frac{c_2}{2}k^{\frac{1}{3}}\right)$, but our base construction only produced a code with an arbitrarily small δ_{base} and its relationship to k_{base} is unknown.

4.3 Completing the Construction

In order to bridge this gap, we apply the recursive step to the base code one more time, with slightly different parameters. We will denote by $k_0, \delta_0, d_0, t_0, n_0$ the parameters of the first application of the recursive step and by $k_{\mathrm{base}} = {k_0}^2, n_{\mathrm{base}}, \delta_{\mathrm{base}}$ the parameters of the resulting code.

As before, we will set $d = k^{\frac{2}{3}}$, but unlike the previous setting, since δ_0 is not necessarily as small as we would want it to be, we will need to set the value of t to be somewhat larger.

In particular, we will set $t_0 = \lceil {\delta_0}^{\frac{c_1}{2}} k_0 \rceil$. Theorem 6 shows that, for sufficiently small δ_0, the probability that a binomial variable with parameters $[\delta_0, k_0 + 2t_0]$ will be greater than t_0 is of the order of $\exp\left(-\Theta(k_0)\right)$. Therefore, for sufficiently large k_0, the DFP after the first step of the recursion would be:

$$\delta_{\mathrm{base}} < \exp\left(-\Theta(k_0)\right) + \exp\left(-c_2 {k_0}^{\frac{1}{3}} + o\left({k_0}^{\frac{1}{3}}\right)\right) < \exp\left(-\frac{c_2}{2}{k_{\mathrm{base}}}^{\frac{1}{6}}\right). \tag{6}$$

Finally, we need to bound the rate of the entire code. Using Equations (4) and (5), we are able to bound the rate:

$$\begin{aligned} r_{k^2} < X r_{\text{base}} &\leq \exp\left(2\frac{(k_{\text{base}})^{-\frac{1}{3}}}{1-(k_{\text{base}})^{-\frac{1}{3}}}\right) r_{\text{base}} \\ &\leq \exp\left(2\frac{{k_0}^{-\frac{1}{3}}}{1-{k_0}^{-\frac{1}{3}}}\right)\left(1+{\delta_0}^{\frac{c_1}{2}}\right) r_0. \end{aligned} \tag{7}$$

For sufficiently large k_0 and sufficiently small δ_0, this can be arbitrarily close to r_0 which in turn can be arbitrarily close to the capacity of the channel, obtaining an arbitrarily close to optimal rate for our code.

4.4 Decoding Complexity

The recursive step promises at most $k + 2t$ calls to the lower level of the construction and $\widetilde{O}(n')$ other operations. Let T_{base} be the encoding / decoding complexity of the base scenario, let T_k be the total complexity of the operation for our code with message length k and I_k be the complexity due to operations which are not part of the lower levels of the recursion (i.e. finding the delimiters, separating them from the inner codewords and decoding the Reed Solomon encoding).

Similar to our bound on the rate of the code, we can bound the decoding / encoding complexities by:

$$\begin{aligned} T_{k^2} &= (k+2t)T_k + I_{k^2} = I_{k^2} + \left(k + 2k^{\frac{2}{3}}\right) T_k \\ &= I_{k^2} + \left(k + 2k^{\frac{2}{3}}\right) I_k + \left(k + 2k^{\frac{2}{3}}\right)\left(\sqrt{k} + 2k^{\frac{1}{3}}\right) T_{\sqrt{k}} \\ &\leq I_{k^2} + \exp\left(2\frac{{k_0}^{-\frac{1}{3}}}{1-{k_0}^{-\frac{1}{3}}}\right)\frac{k^2}{k} I_k + \left(k + 2k^{\frac{2}{3}}\right)\left(\sqrt{k} + 2k^{\frac{1}{3}}\right) T_{\sqrt{k}} \leq \dots \\ \cdots &\leq \exp\left(2\frac{{k_0}^{-\frac{1}{3}}}{1-{k_0}^{-\frac{1}{3}}}\right)\left(\frac{k^2}{k} I_k + \frac{k^2}{\sqrt{k}} I_{\sqrt{k}} + \cdots + \frac{k^2}{k_{\text{base}}} I_{k_{\text{base}}} + \frac{k^2}{k_{\text{base}}} T_{\text{base}}\right). \end{aligned} \tag{8}$$

The recursive step can have at most a quasi-linear complexity on top of its calls to the inner code (see Theorem 8). Therefore, $I_k = \widetilde{O}(n)$. Inserting this into Equation (8), we are can see that $T_{k^2} = \widetilde{O}(n')$.

5 Adaptation to the Poisson Repeat Channel

In this section we will adapt the construction of our code for the BDC channel detailed in the last two sections, to the PRC channel. Adapting the recursive step will be fairly straightforward. However, adapting the base step will be a bit more tricky.

When working with the BDC, we used the fact that the received message could not be longer than the transmitted one. This allowed us to build a decoding table that can return some value for any of the possible received messages. However, the PRC could (with very low probability) expand a transmitted message to an arbitrarily long received message, and we will need to adapt our construction to address this issue.

5.1 Adapting the Recursive Step

▶ **Theorem 9** (Recursive-Step for the PRC). *There exist some constants $c_1, c_2, k_0, \delta_0 > 0$, such that for any $k > k_0$, $d > 0$, $2^{k-1} - k > t > 0$, $\delta_0 > \delta > 0$, $\lambda > 0$ and any error correcting code C for the PRC_λ channel with block length k, message length n and DFP δ, there exists an error correcting code C' for the PRC_λ with block length k^2, message length $n' \leq (k+2t)(n+\frac{d}{\lambda})$ and DFP $\delta' \leq \Pr\left[Bin\left(\delta^{c_1}, k+2t\right) > t\right] + (k+2t)\exp\left(-c_2 \min\left\{d, \frac{d^2}{k}\right\}\right)$.*

Furthermore, there exist an encoder and decoder for C' with time complexity $\widetilde{O}\left(n'\right)$ using up to $k+2t$ calls to the encoder and decoder of C.

We will construct the recursive step almost exactly as in Section 3, only changing $1-p$ to λ in our conversion of lengths of bits over the channel.

5.2 Adapting the Base of the Recursion

In this section, we will adapt our construction of the base of our recursion from Section 4 to the PRC. It is easy to see that for any $\lambda > 0$, there exists a family of codes for this channel with some non-negligible rate $\rho = \Theta(1)$ w.r.t the message length (for instance, by applying the jigsaw construction of [19] with the Morse code distribution). We will set k_0, δ_0 to be the minimal value of the message length k and DFP δ for which there exists such a base code that will suffice for our construction.

Unlike the previous construction, here we will only be able to approximate the DFP of our base code, so we will need to set two bounds. Let $\kappa_1 \geq k_0$ be the minimal message length for which there is a code in the family of codes such that it has a DFP of at most δ_0, and let $\kappa_2 \geq k_0$ be the minimal message length for which there is a code in the family of codes such that it has a DFP of at most $\frac{\delta_0}{2}$. Similar to the construction in Section 4.1, we do not have an explicit bound on κ_1, κ_2, but we know that they are bounded by some $O(1)$ constant.

As in Section 4.1, we will enumerate over values of $k \geq k_0$, but this time we only promise that our enumeration will end somewhere between κ_1 and κ_2. For each such k, we enumerate over all $n \leq \rho k$, encoders $C : \{0,1\}^k \to \{0,1\}^n$ and decoders $D : \{0,1\}^m \to \{0,1\}^k$, where m is the smallest integer for which:

$$\Pr\left[\text{Poisson}(\lambda n) \geq m\right] \leq \frac{\delta}{2}.$$

For each of these encoder-decoder pairs, we enumerate over all messages in $\{0,1\}^k$ and encode them using the encoder. For each codeword, we enumerate over all possible results of applying the channel to the codeword that have output length at most m.

We sum the probabilities of the eventualities where this process would result in a decoding failure (cases where the channel outputted more than m bits are counted as failures), and take the message with the highest DFP. This gives us an approximation of

$$\text{DFP}_{\text{actual}} \leq \text{DFP}_{\text{estimate}} \leq \text{DFP}_{\text{actual}} + \frac{\delta}{2}.$$

If the estimated DFP is at most δ, then we output the pair of encoder-decoder tables. It is easy to see that if the actual DFP is at most $\frac{\delta}{2}$ then the estimate DFP is at most δ and we will output it. Therefore our process either terminates before κ_2 or at κ_2 and must have a constant complexity.

6 Discussion

In this paper we presented several new techniques for constructing error-correcting–codes for asynchronous channels, and used them to prove a method of transforming lower bounds on the capacities of the BDC and PRC channels to efficient encoding and decoding algorithms. This answers the main question considered [6] and [9].

However, this construction is far from practical. For instance, the inner code at the basis of our recursion is constructed in a doubly exponential time in the size of the inner codes message length. Even our bound on the decoding failure of the inner code is $\delta^{\frac{1}{34}} = \exp\left(\frac{k^{\frac{1}{6}}}{34}\right)$ which is technically $o(1)$ but converges extremely slowly.

Furthermore, this work deals only with BDC and PRC channels. While these channels offer us a chance to model the behaviour of asynchronous channels, they do not represent the more realistic channels which contain both synchronisation errors and bit-flipping errors. To this end, the binary InsDel channel offers a more comprehensive model and it remains an open question whether the methods described here can be used to construct efficient codes for it as well.

Finally, now that we have a framework for efficient encoding and decoding for the BDC and PRC channels, we can return to the question of the capacity of these channels. We know that when $p \to 1$ this capacity scales proportionally to $1-p$, but the factor of this conversion is still unknown. Mitzenmacher and Drinea [19] showed that these capacities are at least $\frac{1-p}{9}$, and [8] gave an upper bound of $0.4143(1-p) \pm o_n(1)$, but this gap is far from closed.

References

1. Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.
2. GS Buller and RJ Collins. Single-photon generation and detection. *Measurement Science and Technology*, 21(1):012002, 2009.
3. Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 200–211. IEEE, 2018.
4. Mahdi Cheraghchi, Ryan Gabrys, Olgica Milenkovic, and Joao Ribeiro. Coded trace reconstruction. *IEEE Transactions on Information Theory*, 66(10):6084–6103, 2020.
5. Mahdi Cheraghchi and João Ribeiro. An overview of capacity results for synchronization channels. *IEEE Transactions on Information Theory*, 67(6):3207–3232, 2020.
6. Roni Con and Amir Shpilka. Explicit and efficient constructions of coding schemes for the binary deletion channel and the poisson repeat channel. *arXiv preprint*, 2019. `arXiv:1909.10177`.
7. Roni Con, Amir Shpilka, and Itzhak Tamo. Linear and reed solomon codes against adversarial insertions and deletions. *arXiv preprint*, 2021. `arXiv:2107.05699`.
8. Marco Dalai. A new bound on the capacity of the binary deletion channel with high deletion probabilities. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 499–502. IEEE, 2011.
9. Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Transactions on Information Theory*, 65(4):2171–2178, 2018.
10. Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.
11. Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 334–347. IEEE, 2019.
12. Bernhard Haeupler, Aviad Rubinstein, and Amirbehshad Shahrasbi. Near-linear time insertion-deletion codes and $(1+\varepsilon)$-approximating edit distance via indexing. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 697–708, 2019.

13 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the singleton bound. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 33–46, 2017.

14 Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.

15 Adam Kalai, Michael Mitzenmacher, and Madhu Sudan. Tight asymptotic bounds for the deletion channel with small deletion probabilities. In *2010 IEEE International Symposium on Information Theory*, pages 997–1001. IEEE, 2010.

16 Yonglong Li and Vincent YF Tan. On the capacity of channels with deletions and states. *IEEE Transactions on Information Theory*, 67(5):2663–2679, 2020.

17 Hugues Mercier, Vijay K Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys & Tutorials*, 12(1):87–96, 2010.

18 Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.

19 Michael Mitzenmacher and Eleni Drinea. A simple lower bound for the capacity of the deletion channel. *IEEE Transactions on Information Theory*, 52(10):4657–4660, 2006.

20 Henry D Pfister and Ido Tal. Polar codes for channels with insertions, deletions, and substitutions. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 2554–2559. IEEE, 2021.

21 Ittai Rubinstein. Explicit and efficient construction of (nearly) optimal rate codes for binary deletion channel and the poisson repeat channel. *arXiv preprint*, 2021. `arXiv:2111.00261`.

22 Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

23 Ido Tal, Henry D Pfister, Arman Fazeli, and Alexander Vardy. Polar codes for the deletion channel: Weak and strong polarization. *IEEE Transactions on Information Theory*, 2021.

24 Scott A Vanstone and Paul C Van Oorschot. *An introduction to error correcting codes with applications*, volume 71. Springer Science & Business Media, 2013.

25 Ramji Venkataramanan, Sekhar Tatikonda, and Kannan Ramchandran. Achievable rates for channels with deletions and insertions. *IEEE transactions on information theory*, 59(11):6990–7013, 2013.

Maximizing Non-Monotone Submodular Functions over Small Subsets: Beyond 1/2-Approximation

Aviad Rubinstein
Computer Science Department, Stanford University, CA, USA

Junyao Zhao
Computer Science Department, Stanford University, CA, USA

Abstract

In this work we give two new algorithms that use similar techniques for (non-monotone) submodular function maximization subject to a cardinality constraint. The first is an offline fixed-parameter tractable algorithm that guarantees a 0.539-approximation for all non-negative submodular functions. The second algorithm works in the random-order streaming model. It guarantees a $(1/2 + c)$-approximation for *symmetric* functions, and we complement it by showing that no space-efficient algorithm can beat 1/2 for asymmetric functions. To the best of our knowledge this is the first provable separation between symmetric and asymmetric submodular function maximization.

2012 ACM Subject Classification Theory of computation → Submodular optimization and polymatroids

Keywords and phrases Submodular optimization, Fixed-parameter tractability, Random-order streaming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.106

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2204.11149

Funding *Aviad Rubinstein*: Supported by NSF CCF-1954927, and a David and Lucile Packard Fellowship.
Junyao Zhao: Supported by NSF CCF-1954927.

1 Introduction

We study the algorithmic problem of selecting a small subset of k elements out of a (very) large ground set of n elements. In particular, we want the small subset to consist of k elements that are valuable *together*, as is captured by an objective function $f : 2^E \to \mathbb{R}_{\geq 0}$. Without any assumptions on f it is hopeless to get efficient algorithms; we make the assumption that f is *submodular*[1], one of the most fundamental and well-studied assumptions in combinatorial optimization.

Alas, even for submodular functions, strong impossibility results are known. Two of the most important frameworks we have for circumventing impossibility results are (i) approximation algorithms – look for solutions that are only approximately optimal; and (ii) paramerterized complexity – look for algorithms of which the runtime is efficient as a function of the large ground set n, but may have a worse dependence[2] on the smaller parameter

[1] I.e., functions where the marginal value of an element is decreasing as the set grows.

[2] Formally, an algorithm is said to be fixed-parameter tractable if it runs in time $h(k) \cdot \text{poly}(n)$ for *any* function h. Here h could be arbitrarily fast growing, e.g. doubly-exponential or Ackermann – this is asymptotically faster than the naive n^k. In this work we will be more ambitious (and closer to practice) and present algorithms that run in time $2^{\tilde{O}(k)} \cdot n$.

k. To appreciate the relevance of parameterized complexity in practice, it's important to note that in many applications of submodular maximization k is indeed quite small, e.g. in data summarization [4], we want an algorithm that, given a large image dataset chooses a representative subset of images that must be small enough to fit our screen.

Submodular optimization has been thoroughly studied under the lens of approximation algorithms; much less work has been done about its parameterized complexity (with the exception of [27], see discussion of related works). In either case, strong, tight hardness results are known. In this work we show that combining both approaches gives surprisingly powerful algorithms.

On the technical level, we develop novel insights to design and analyze fixed-parameter tractable (FPT) algorithms for (non-monotone) submodular function maximization. We instantiate these ideas to give new results in two settings: offline ("classical") algorithms for which we are interested in the running time and query complexity[3], and random-order streaming algorithms for which we mostly care about the memory cost.

Main result I: offline algorithms

Our first result is an (offline) FPT algorithm that guarantees an improved approximation ratio for submodular function maximization.

To compare our result with the approximation factors achievable by polynomial-time algorithms, we first briefly survey the existing algorithmic and hardness results.

On the algorithmic side, the current state-of-art polynomial-time algorithm for general non-monotone submodular functions achieves 0.385-approximation [5]. For sub-classes of submodular functions, better polynomial-time approximation algorithms are known: notable examples include monotone functions (the greedy algorithm achieves $(1 - 1/e)$-approximation [25]), and symmetric functions[4] (the state-of-the-art approximation factor is 0.432 [11]).

From the hardness perspective, known results rule out *polynomial* query complexity algorithms with approximation factors better than 1/2 or 0.491 for symmetric [10, 28] or asymmetric functions [14], respectively. It is also known that even FPT algorithms cannot beat $(1 - 1/e)$-approximation, and this holds even for monotone submodular functions [24].

In FPT time, the streaming algorithm of [2] implies a 1/2-approximation algorithm for general non-monotone functions (although it is not explicitly stated as an FPT algorithm in their paper), which slightly beats the aforementioned 0.491 bound for asymmetric functions. However, this result does not tell us whether FPT algorithms can beat the 1/2 bound for symmetric functions. A-priori, it was plausible that 1/2-approximation is the best achievable approximation by FPT algorithms for symmetric functions and hence general non-monotone functions. (In fact, prior to discovering our new algorithms, we had expected that the 1/2-approximation would indeed be the best that FPT algorithms can achieve.)

We are thus excited to report that we were able to design an FPT algorithm (Algorithm 4) that not only outperforms all the previous algorithms but also surpasses all the upper limits on the approximation factor in the existing hardness results, *by a significant margin, regardless of whether the function is symmetric or not*:

[3] I.e., the function f is given as a value oracle. The query complexity is number of queries made by the algorithm, which is clearly a lower bound of the runtime.

[4] I.e. functions that assign the same value to a set and its complement, which capture some of the most important applications of non-monotone submodular functions, including mutual information and cuts in (undirected) graphs and hypergraphs.

▶ **Theorem 1** (FPT algorithm). *There is a* 0.539*-approximation algorithm for cardinality constrained submodular maximization that has runtime and query complexity* $2^{\widetilde{O}(k)} \cdot n$.

Main result II: random-order streaming

Our FPT algorithm (Algorithm 4) uses a subroutine (Algorithm 1) which can be interpreted as a *random-order streaming algorithm*, and thus, in addition to our FPT result, we also hope to understand the power and the limit of Algorithm 1 in the random-order streaming model. In this model (see the detailed setup in Section 2), a streaming algorithm makes a single pass over a stream of elements arriving in a uniformly random order. An algorithm keeps a carefully chosen subset of the elements it has seen in a buffer of bounded size. At any point in the stream, the algorithm can make unlimited queries of the function values on subsets of the elements in the buffer. The goal is to obtain a good approximation of the offline optimum while keeping the buffer small (ideally, polynomial in k and independent of n).

We show that Algorithm 1 achieves 1/2-approximation for general non-monotone submodular functions[5] and beats 1/2-approximation for symmetric functions using $\widetilde{O}(k^2)$-size buffer[6]:

▶ **Theorem 2** (Random-order streaming algorithm). *For cardinality constrained submodular function maximization in the random-order streaming model, there is an algorithm using* $\widetilde{O}(k^2)$*-size buffer that achieves* 1/2*-approximation for general non-monotone submodular functions and* 0.5029*-approximation for symmetric submodular functions.*

We complement the algorithmic result with a tight 1/2-hardness result in the random-order streaming setting:

▶ **Theorem 3** (1/2-hardness for random-order streaming). *If* $n = 2^{o(k)}$*, any* $(1/2 + \varepsilon)$*-approximation algorithm for cardinality-constrained non-monotone submodular maximization in the random-order streaming model must use an* $\Omega(n/k^2)$*-size buffer. In fact, this hardness result holds against stronger algorithms that are not captured by the standard random-order streaming model for submodular maximization (see Remark 9).*

This hardness result is quite surprising because it shows in contrast to monotone submodular maximization, non-monotone submodular maximization in the random-order setting is *not any easier than* that in the worst-order setting where the elements arrive in the worst-case order – for worst-order streaming model, it is known that $\Omega(n/k^3)$-size buffer is required to beat 1/2-approximation even if the submodular function is monotone [13], but for random-order model, recent work by [1] gives a $(1-1/e-\varepsilon)$-approximate algorithm using $O(k/2^{\mathrm{poly}(\varepsilon)})$-size buffer, which is improved to $O(k/\varepsilon)$ by a simpler algorithm of [20].

Furthermore, notice that our algorithmic result and hardness result together exhibit a separation between symmetric and asymmetric submodular functions in the random-order streaming setting. This separation is interesting because in the literature, tight hardness result for general non-monotone functions often continues to hold for symmetric functions. For example, for unconstrained non-monotone submodular maximization, there is a family of symmetric functions for which $(1/2 + \varepsilon)$-approximation requires $2^{\Omega(n)}$ queries [10, 28], and there are efficient matching 1/2-approximation algorithms even for asymmetric functions [6].

[5] The 1/2-approximation of Algorithm 1 for general non-monotone functions is not interesting by itself – 1/2-approximation was achieved even if the elements arrive in the worst-case order [2]. However, Algorithm 1, which takes advantage of the random order, led us to the discovery of the 1/2-hardness in the random-order setting.

[6] These algorithmic results also hold for the (similar but incomparable) secretary with shortlists model [1].

To the best of our knowledge, our result is the *first provable separation of symmetric and asymmetric submodular maximization in any setting*, let alone a natural setting that gains a lot of interests recently. Although admittedly we would be more excited to see such separation in the more classic offline setting of constrained non-monotone submodular maximization, currently we are still far from figuring out whether there is a separation in this setting (because of the gap between the current best 0.432-approximation algorithm for symmetric functions and the current best 0.491-hardness for general asymmetric functions we mentioned earlier), and we hope our result can provide some insights on how to resolve this problem.

Future directions

Our FPT algorithm achieves significantly better-than-1/2 approximation for general non-monotone submodular functions in the offline setting, and its subrountine Algorithm 1 breaks the 1/2 hardness for symmetric submodular functions in the random-order streaming setting – but what is the best possible approximation ratio in those respective settings? We leave this as an open problem for future work.

We remark that no FPT (small-buffer resp.) algorithm can break the $1 - 1/e$ barrier for symmetric functions in the offline (random-order streaming resp.) setting. For asymmetric functions, this follows from the classic work of [24] for monotone submodular functions which we mentioned earlier. For symmetric submodular maximization, the monotone functions exhibiting $(1-1/e)$-hardness are obviously not symmetric; but in appendix of the full version, we are able to give a simple black-box approximation-preserving reduction from symmetric non-monotone to asymmetric monotone submodular function maximization that works in both the offline and random-order streaming settings:

▶ **Proposition 4.** *For cardinality-constrained symmetric submodular function maximization, any algorithm guaranteeing a $(1 - 1/e + \varepsilon)$-approximation must:*
Offline *use $n^{\Omega(k)}$ queries; or*
Random-order streaming *use $\Omega(n)$-buffer size.*

1.1 Additional related work

FPT submodular optimization

The study of parameterized complexity of submodular maximization was initiated by [27] who focused on monotone submodular functions. [27] gives an FPT approximation scheme for monotone submodular functions that are either p-separable or have a bounded ratio of total singleton contribution ($\sum_{e \in E} f(\{e\})$) to total value ($f(E)$). However, for general monotone submodular functions even FPT algorithms (in terms of query complexity) cannot break the classic $1 - 1/e$ barrier [24]. Furthermore, even for the special case of max-k-cover, no FPT algorithms can beat $1 - 1/e$ assuming gap-ETH [8, 21].

Streaming submodular optimization

Our work is related to recent works on maximizing submodular functions in random order streams [1, 20, 26], but all the latter focus on monotone functions. Submodular optimization in worst-order streaming models has also been extensively studied in recent years, e.g. [4, 7, 19, 9, 12, 23, 3, 17, 18, 22, 2, 15, 16, 13]. In the worst-order literature, most relevant to our work is [2] who gave a 1/2-approximation for general (non-monotone) submodular functions, and [13] who proved a matching inapproximability.

2 Preliminaries

▶ **Definition 5.** *Given a ground set of elements E, a function $f : 2^E \to \mathbb{R}_{\geq 0}$ is submodular if for all $S \subseteq T \subseteq E$ and $i \in E \setminus T$, $f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$. Moreover, we denote the marginal gain by $f(X|S) := f(X \cup S) - f(S)$.*

▶ **Definition 6.** *A function $f : 2^E \to \mathbb{R}_{\geq 0}$ is symmetric if for all $S \subseteq E$, $f(S) = f(E \setminus S)$.*

In this paper, we **always consider maximizing non-negative submodular functions** over n elements under a cardinality constraint k, i.e., $\max_{X \in E, |X| \leq k} f(X)$. The following lemma [10] for non-negative submodular functions will be useful.

▶ **Lemma 7.** *Let $f : 2^E \to \mathbb{R}_{\geq 0}$ be a submodular function. Further, let R be a random subset of $T \subseteq E$ in which every element occurs with probability at least p (not necessarily independently). Then, $\mathbf{E}[f(R)] \geq pf(T) + (1-p)f(\emptyset)$.*

Moreover, we are interested in the fixed-parameter tractable algorithms.

▶ **Definition 8.** *For submodular maximization over n elements with cardinality constraint k, we say an algorithm is fixed-parameter tractable (FPT) if it has runtime $h(k) \cdot poly(n)$, where h can be any finite function.*

Besides, we are also interested in studying low-memory algorithms for submodular maximization in the random-order streaming model, and in this setting, we only care about the memory cost but not the runtime. We follow the standard setup of streaming model for submodular maximization in the literature (see e.g., the model in the original work [4, Section 3] and more recent works [2, 16, 20]), and the only additional assumption we make is that the elements arrive in uniformly random order (which was studied in e.g., [1, 20]).

Random-order streaming model

In the random-order streaming model, an algorithm is given a single pass over a dataset in a streaming fashion, where the stream is a uniformly random permutation of the input dataset and each element is seen once. The algorithm is allowed to store the elements or any information in a memory buffer with certain size. To be precise, at any point during the runtime of the algorithm,

$$\text{memory cost} = \text{number of stored elements} + \text{number of bits of stored information}.$$

Note that the elements and information are treated separately. One can think of the elements as physical tokens[7], and the algorithm has a limited number of special slots to store the tokens. Besides these special slots, the algorithm has other limited space to store arbitrary information. The total memory cost should not exceed the algorithm's memory size.

Every time when a new element arrives, the algorithm can decide how to update its memory buffer, i.e., whether to store the new element or remove other elements in its memory, and what information to add or remove. At any time, the algorithm can make any number

[7] The standard streaming model for submodular maximization assumes the elements are stored like physical tokens rather than using arbitrary encoding, because the model eventually wants to restrict the algorithm's access to the value oracle. If we store elements using arbitrary encoding, it is not clear how to restrict oracle access for general submodular functions (although it is possible to define such model for some special applications). There is another model that allows elements to be stored in arbitrary encoding [13, Appendix B] - this model does not restrict oracle access at all, but instead it assumes that the elements appearing in the stream are a small part of the ground set.

of queries to the value oracle of the objective submodular function, but it is only allowed to query the value of any subset of the elements that are stored in its memory. For example, at some point during the stream, suppose the algorithm stores an element e, and it makes a query of the value of set $\{e\}$ and writes the result of the query in its memory in any format it prefers (e.g., "the value of $\{e\}$ is..."), and then it removes element e. After removing e, it will never be allowed to query the value of any set that contains e in the future, but it can still keep the information "the value of $\{e\}$ is...", which it wrote before, in its memory, as long as it wants.

At the end of the stream, the algorithm outputs a subset of elements that are stored in its memory as the solution set.

▶ Remark 9. Our streaming algorithm falls into the above model and has low memory cost (specifically, $\tilde{O}(k^2)$). Our hardness result in fact holds against stronger algorithms that are allowed to (i) store infinite bits of information (i.e., only the number of stored elements counts as memory cost) and (ii) output any size-($\leq k$) subset of elements as the solution set (i.e., during the stream, the algorithm is still only allowed to query any subset of elements stored in its memory, but at the end of the stream, it can output any size-($\leq k$) subset of the ground set as it wants[8]).

Finally, we note that all the missing proofs of this extended abstract can be found in the full version of our paper on arXiv (the URL is provided on the title page).

3 The core algorithm

In this section, we present the core algorithm of this work (Algorithm 1), which is actually our streaming algorithm. Our FPT algorithm will use this core algorithm as a subroutine. The goal of this section is to establish the common setup for the analysis of our FPT algorithm for general submodular functions and the analysis of the streaming algorithm for symmetric submodular functions. In this process, we will also do a warm-up that shows a 1/2-approximation for the core algorithm on general submodular functions.

Algorithm 1 SymmetricStream(f, E, k, ε).

Partition the first ε fraction of the random stream E into windows $w_1, \ldots, w_{3k}$ of equal size.
$S_0 \leftarrow \emptyset$
$H \leftarrow \emptyset$
for $i \leftarrow 1$ to $3k$ **do**
    $e_i \leftarrow \arg\max_{e \in w_i} f(e|S_{i-1})$
    $S_i \leftarrow S_{i-1} \cup \{e_i\}$
for $e \in E \setminus \{w_1, w_2, \ldots, w_{3k}\}$ **do**
    for $i = 1, 2, \ldots, 3k$ **do**
        if $f(e|S_{i-1}) > f(e_i|S_{i-1})$ and $|H| < 18k^2 \log k/\varepsilon$ **then**
            $H \leftarrow H \cup \{e\}$
return $\arg\max_{X \subseteq S_{3k} \cup H, |X| \leq k} f(X)$

[8] I.e., it can output something like "My solution set is {1,3,11,...}" even if elements $1, 3, 11$ are not in its memory.

At a high level, Algorithm 1 divides the first ε fraction of the stream into $3k$ windows[9] and greedily selects the element e_i with the best marginal gain in each window i. (Although we use the notation S_i in the pseudocode for clarity, we only need to keep track of one ordered solution set in the first for loop.) Then it freezes the solution set, and for the rest of the stream, it only selects the first $18k^2 \log k/\varepsilon$ elements that have better marginal gain than e_i conditioned on the $(i-1)$-th partial solution for some $0 \leq i \leq 3k-1$. Finally, it finds the best size-k solution from all the selected elements by brute force. Due to line 9, the memory usage is $O(k^2 \log k/\varepsilon)$. The runtime before the final brute-force search is $O(nk)$ because of the for loops, and the brute-force search takes time $k\binom{|S_{3k} \cup H|}{k} = 2^{\widetilde{O}(k)}$ as $|S_{3k}| = 3k$ and $|H| = O(k^2 \log k/\varepsilon)$, and hence, the total runtime is $O(nk) + 2^{\widetilde{O}(k)}$ (which is not polynomial in k, but in this paper, we focus on the memory bound for streaming setting and FPT algorithms for offline setting).

3.1 Warm-up

As a warm-up, we show the 1/2-approximation of our core algorithm for general submodular functions, which also helps set up the proof of our main algorithmic results. Before getting to the technical proof, we provide the intuition for Algorithm 1 in the following. First, we want to make sure that with high probability $|H|$ never meets the size threshold in the if condition at line 9, and thus the size threshold essentially does not affect our analysis. This follows by a standard argument (see Lemma 11).

Because the stream is in random order, most optimal elements will be visited during the for loop at line 7. A part of them O_H will be picked by the algorithm, and the other part O_L will not be selected. Consider the set $S_{|O_L|}$ defined in the algorithm. Because of the if condition at line 9, the elements in $S_{|O_L|}$ have better marginal contribution than O_L, and we can show that $f(S_{|O_L|}) \geq f(O_L|S_{|O_L|})$, which is somewhat similar to the classic greedy algorithm for monotone submodular maximization. Since $O_H \cup S_{|O_L|}$ and $S_{|O_L|}$ are two candidate solutions under the radar of the algorithm's final brute-force search, the algorithm achieves at least

$$\frac{f(O_H \cup S_{|O_L|}) + f(S_{|O_L|})}{2} \geq \frac{f(O_H \cup S_{|O_L|}) + f(O_L|S_{|O_L|})}{2} \geq \frac{f(O_H \cup O_L \cup S_{|O_L|})}{2}, \quad (1)$$

where the last inequality is by submodularity.

To complete the analysis, we observe that $f(O_H \cup O_L \cup S_{|O_L|})$ is not significantly worse than $f(O_H \cup O_L)$, and hence 1/2-approximation follows from (1). Indeed, because $S_{|O_L|}$ is chosen from a random ε fraction of E, it cannot hurt $O_H \cup O_L$ significantly – otherwise, there should be many other elements similar to $S_{|O_L|}$, and together they would hurt $O_H \cup O_L$ so much that would eventually contradict non-negativity. This is formally shown in Lemma 10. Using Lemma 10 and Lemma 11, we can prove the 1/2-approximation. The proof of Lemma 10 and Lemma 11 can be found in the full version.

▶ **Lemma 10.** *Let O denote the optimal size-k solution. Then, for any constants $\varepsilon \in (0, 1]$ and $\varepsilon' > 0$,*

- *with probability at least $1 - \varepsilon/\varepsilon'$, there does not exist a set Y of elements in the first ε fraction of the stream such that $f(Y|O) \leq -\varepsilon' f(O)$,*
- *and moreover, with probability at least $1 - 3\varepsilon/(\varepsilon')^2$, for all $\ell \in \{\varepsilon' k, 2\varepsilon' k, \ldots, k\}$, for any $S \subseteq S_{3\ell} \setminus S_{2\ell}$, $f(S|O \cup S_{2\ell}) \geq -\varepsilon' f(O)$.*

[9] The choice of $3k$ suffices to beat 1/2 approximation for symmetric submodular function, but it is conceivable that a larger budget might improve the final constant. For our FPT algorithm, dividing into k windows would also work, but that does not improve the runtime asymptotically.

▶ **Lemma 11.** *With probability* $1 - 3/k$, $|H| < 18k^2 \log k/\varepsilon$, *namely, Algorithm 1 never ignores any elements due to the memory threshold (line 9).*

▶ **Theorem 12.** *Algorithm 1 achieves* $(1/2 - O(\sqrt{\varepsilon}) - O(1/k))$*-approximation for non-negative non-monotone submodular functions in the random-order streaming model.*

Proof. Let O be the size-k optimal set, and let $O' \subseteq O$ be the subset of optimal elements that appear in the last $1 - \varepsilon$ fraction of the stream. We partition O' into O_L and O_H, where O_L are the optimal elements that are not selected by the algorithm, and O_H are those selected. Let S_L be short for solution $S_{|O_L|}$ in the algorithm. If the good event in the first bullet point of Lemma 10 with $\varepsilon' = \sqrt{\varepsilon}$ happens (which we denote by A_1), then $f(S_L|O_L \cup O_H) \geq f(S_L|O) \geq -\sqrt{\varepsilon} f(O)$, where the first inequality is by submodularity. If A_1 happens, we have that

$$\begin{aligned} f(O_H \cup S_L) + f(O_L|S_L) &= f(O_H|S_L) + f(S_L \cup O_L) \\ &\geq f(O_H|S_L \cup O_L) + f(S_L \cup O_L) \quad \text{(By submodularity)} \\ &= f(O_L \cup O_H \cup S_L) \\ &= f(O_L \cup O_H) + f(S_L|O_L \cup O_H) \\ &\geq f(O_L \cup O_H) - \sqrt{\varepsilon} f(O). \end{aligned} \tag{2}$$

Let $O_L = \{o_1, o_2, \ldots, o_{|L|}\}$. If the good event in Lemma 11, denoted by A_2, happens, then, we have the following simple observation by design of the algorithm.

▶ **Observation 13.** *If* A_2 *happens, then for any* $o \in O_L$, $f(o|S_{i-1}) \leq f(e_i|S_{i-1})$, *for all* $i \in [3k]$, *because* o *is skipped by the algorithm.*

Given A_2, using the above observation, we have that

$$\begin{aligned} f(O_L|S_L) &\leq \sum_{o \in O_L} f(o|S_L) && \text{(By submodularity)} \\ &\leq \sum_{i=1}^{|O_L|} f(o_i|S_{i-1}) && \text{(By submodularity)} \\ &\leq \sum_{i=1}^{|O_L|} f(e_i|S_{i-1}) && \text{(By Observation 13)} \\ &= f(S_L). && \text{(By telescoping sum)} \end{aligned} \tag{3}$$

Combining Eq. (2) and (3), we get

$$f(O_H \cup S_L) + f(S_L) \geq f(O_L \cup O_H) - \sqrt{\varepsilon} f(O). \tag{4}$$

By Lemma 7, $\mathbf{E}[f(O')] \geq (1 - \varepsilon) f(O)$. By a Markov argument,

$$\mathbf{E}[f(O') \mid A_1, A_2] \geq (1 - \varepsilon - \sqrt{\varepsilon} - 3/k) f(O), \tag{5}$$

and hence, by taking expectation for both sides of Eq. (4), we have that

$$\begin{aligned} \mathbf{E}[f(O_H \cup S_L) + f(S_L) \mid A_1, A_2] &\geq \mathbf{E}[f(O_L \cup O_H) \mid A_1, A_2] - \sqrt{\varepsilon} f(O) \\ &\geq (1 - \varepsilon - 2\sqrt{\varepsilon} - 3/k) f(O). \end{aligned}$$

Finally, $\mathbf{E}[f(O_H \cup S_L) + f(S_L)] \geq \Pr(A_1, A_2) \cdot \mathbf{E}[f(O_H \cup S_L) + f(S_L) \mid A_1, A_2] \geq (1 - \sqrt{\varepsilon} - 3/k)(1 - \varepsilon - 2\sqrt{\varepsilon} - 3/k) f(O) = (1 - O(\sqrt{\varepsilon} + 1/k)) f(O)$. The proof finishes because $O_H \cup S_L$ and S_L are both subsets of $S_{3k} \cup H$, so one of them must achieve at least half of $(1 - O(\sqrt{\varepsilon} + 1/k)) f(O)$. ◀

3.2 Our plan for the algorithmic results

The proof of our main algorithmic results (Theorem 17, Theorem 14 and Theorem 15) is based on factor-revealing convex programs[10]. We know that factor-revealing programs are not intuitive and hence not easy to understand, although they are effective tools for formalizing the proof. Therefore, in the future sections of this extended abstract, instead of formally proving the main results, we will provide the intuition and interpretable (but less formal) analysis for our algorithmic results. **We recommend the readers read the intuition and informal interpretable analysis in this extended abstract, and then check the formal proofs that involve factor-revealing programs starting from Section 3.2 in the full version.**

For convenience, in the informal interpretable analysis in this extended abstract, we will **continue using the notations** O, O_L, O_H that have appeared in this section.

4 FPT algorithms for non-monotone submodular functions

In this section, building on Algorithm 1, we give FPT algorithms that achieves better-than-1/2 approximation for general non-monotone submodular functions. We first present a basic FPT algorithm that achieves 0.512-approximation to show the main ideas, then we discuss how to improve the basic algorithm to get 0.539-approximation.

4.1 The basic FPT algorithm

Essentially, after running Algorithm 1, our basic FPT algorithm (Algorithm 2) searches for O_H by brute force, and then starting with O_H as the initial solution set, it runs classic greedy algorithm to construct a size-k solution set, and it repeats this step many times without replacement, i.e., each time the elements selected by greedy algorithm are removed from ground set, and finally it returns the best size-k solution set among all the repetitions. The pseudocode is given in Algorithm 2.

Algorithm 2 FPT$(f, E, k, \varepsilon, T)$.

Initialize an empty set X^*.
Run Algorithm 1 on the input[11] (f, E, k, ε) and keep S_{3k} and the final version of H in Algorithm 1.
for each size-$(\leq k)$ subset $O_H^{\text{guess}} \subseteq H$ **do**
    Initialize an empty set of elements I.
    for $i = 1, 2, \ldots, T$ **do**
        Run greedy algorithm with O_H^{guess} as the initial solution set[12] to build a size-k solution set $O_H^{\text{guess}} \cup X_i$. Add X_i to I and remove X_i from E.
    Let $X' = \arg\max_{X \subseteq S_{3k} \cup H \cup I, |X| \leq k} f(X)$ and let $X^* = X'$ if $f(X') > f(X^*)$.
    Add I back to E.
return X^*

[10] The certificates for these convex programs can be found in the full version.
[11] Randomly permute E if it is not in random order.
[12] Specifically, the greedy algorithm starts with solution set $X = O_H^{\text{guess}}$ and runs in $k - |X|$ iterations. In each iteration, it selects the element e in E that maximizes $f(e|X)$ and add e to X. (**We assume without loss of generality** that the maximal $f(e|X)$ is always non-negative. Otherwise, we can add dummy elements to E.)

Algorithm 2 runs in fixed-parameter polynomial time. Indeed, because $|H| = \widetilde{O}(k^2)$ by design of Algorithm 1, the outer loop has less than $k\binom{\widetilde{O}(k^2)}{k} = 2^{\widetilde{O}(k)}$ iterations, and for the inner loop, we will only need T to be an arbitrarily large constant. Greedy algorithm runs in $O(kn)$ time. Moreover, since $|I| \leq Tk$ and $|S_{3k} \cup H| = \widetilde{O}(k^2)$ with high probability, the brute-force step (Line 7) in Algorithm 2 takes time $k\binom{\widetilde{O}(k^2)}{k} = 2^{\widetilde{O}(k)}$. Furthermore, the runtime of Algorithm 1 excluding the exhaustive search in its last step is polynomial. Hence, the total runtime is $n \cdot 2^{\widetilde{O}(k)}$.

Instead of formally proving the approximation ratio for Algorithm 2 (which we will do in the full version), we give an interpretable analysis for the better-than-1/2 approximation. At very high level, the intuition is that if Algorithm 1 only gets 1/2 approximation, then it must be the case that $f(O_H) = \frac{f(O_H \cup O_L)}{2}$. Now we consider the candidate solution $O_H \cup X_i$ for $i \in \{1, 2, 3\}$, and we can argue that if $f(X_i|O_H) = 0$, i.e., the candidate solution does not beat 1/2, then X_i must hurt $O_H \cup O_L$ a lot. Moreover, By submodularity, $X_1 \cup X_2 \cup X_3$ hurts $O_H \cup O_L$ by at least the sum of how much each X_i ($i \in \{1, 2, 3\}$) hurts. Together, we show that this would contradict non-negativity of the function. Now we explain this intuition in more details.

Informal interpretable analysis

The starting point is the analysis of Theorem 12. We can show that for the instance to be hard, in the sense that Algorithm 1 is only able to get 1/2 approximation, then it requires $f(O_H) = \frac{f(O_H \cup O_L)}{2}$ (this will be explained with more details in the interpretable analysis provided before Theorem 17, but for now, let us take this as given). Because O_H is selected by Algorithm 1, in the outer iteration when Algorithm 2 guesses O_H correctly, it runs classic greedy algorithm many times based on O_H without replacement. Consider the set X_1 selected in the first run of greedy algorithm. By standard analysis of greedy algorithm, we can derive that $f(X_1|O_H) \geq f(O_L|O_H \cup X_1)$. If $f(X_1|O_H) = 0$ (otherwise $X_1 \cup O_H$ beats 1/2), then $f(O_L|O_H \cup X_1) \leq 0$, which implies $f(O_L \cup O_H \cup X_1) \leq f(O_H \cup X_1) = f(O_H) = \frac{f(O_H \cup O_L)}{2}$. Hence $f(X_1|O_L \cup O_H) \leq -\frac{f(O_H \cup O_L)}{2}$. WLOG, the first run of greedy did not select most of O_L, because otherwise $f(X_1|O_H)$ should be significantly large. Therefore, similarly, we can derive that if $f(X_2|O_H) = 0$, where X_2 is selected in the second run of greedy, then $f(X_2|O_L \cup O_H) \leq -\frac{f(O_H \cup O_L)}{2}$. By submodularity, $f(X_1 \cup X_2|O_L \cup O_H) \leq f(X_1|O_L \cup O_H) + f(X_2|O_L \cup O_H) \leq -f(O_H \cup O_L)$. Notice that this implies $f(X_1 \cup X_2 \cup O_L \cup O_H) \leq 0$. Hence, the third run of greedy algorithm must obtain very large $f(X_3|O_H)$ (and hence $X_3 \cup O_H$ beats 1/2), because otherwise we can repeat above argument and show that $f(X_1 \cup X_2 \cup X_3 \cup O_L \cup O_H) < 0$, which violates non-negativity of the function f. Furthermore, by running greedy many times, we are able to extract even more value from O_L, which is formally formulated by the factor-revealing programs in the proof in the full version.

▶ **Theorem 14.** *For sufficiently large constant T and sufficiently small constant ε, Algorithm 2 achieves* 0.512*-approximation for non-negative non-monotone submodular maximization with a cardinality constraint.*

4.2 Improved FPT algorithm

Algorithm 2 can be improved to achieve better approximation ratio. In this subsection, we present the improved algorithm, the pseudocode of which is given in Algorithm 4. In the full version, we will provide the intuition behind this algorithm (which involves the formal proof of Theorem 14) and the formal proof for the improved approximation factor.

Algorithm 3 RECURSIVE(f, E, k, I, t, T).

Initialize empty sets I' and X^*.
for each size-($\leq k$) subset $O_t^{\text{guess}} \subseteq I$ **do**
  for $i = 1, 2, \ldots, T$ **do**
    Run greedy algorithm on E with O_t^{guess} as the initial solution set to build a size-k solution set $O_t^{\text{guess}} \cup X_i$.
    Add X_i to I' and remove X_i from E.
  Add I' back to E.
  if $t \leq T$ **then**
    Run Algorithm 3 on input $(f, E, k, I \cup I', t+1, T)$, which returns solution set X'.
    Let $X^* = X'$ if $f(X') > f(X^*)$.
  else
    Let $X^* = \arg\max_{X \subseteq I, |X| \leq k} f(X)$.
return X^*

Algorithm 4 FPT$^+$(f, E, k, ε, T).

Initialize an empty set of elements I.
Run Algorithm 1 on input (f, E, k, ε) and keep S_{3k} and the final version of H in Algorithm 1.
Run Algorithm 3 on input $(f, E, k, S_{3k} \cup H, 1, T)$, which returns X^*.
return X^*

▶ **Theorem 15.** *For sufficiently large constant T and sufficiently small constant ε, Algorithm 4 achieves 0.539-approximation for non-negative non-monotone submodular maximization with a cardinality constraint.*

5 $(1/2 + c)$-approximation for random-order streaming symmetric submodular maximization

In this section, we show that Algorithm 1 *beats* 1/2-approximation for symmetric non-monotone submodular functions using $\widetilde{O}(k^2)$ memory. Together with our lower bound result (Theorem 18), this separates the symmetric non-monotone submodular functions from general non-monotone submodular functions in the random-order streaming model. To our best knowledge, this is first such separation.

The following lemma is the key feature of symmetric submodular functions which we will take advantage of, and it basically says for symmetric submodular function, a set can not hurt another set by more than its own value.

▶ **Lemma 16.** *For any non-negative symmetric submodular function $f : V \to \mathbb{R}_{\geq 0}$, for any disjoint $X, Y \subseteq V$, $f(X|Y) \geq -f(X)$.*

Proof. By submodularity, $f(X|Y) \geq f(X|V \setminus X) = f(V) - f(V \setminus X)$, and by symmetry and non-negativity, $f(V) - f(V \setminus X) = f(\emptyset) - f(X) \geq -f(X)$. ◀

Instead of showing the technical proof of the better-than-1/2 approximation (which is provided in the full version), we give the interpretable analysis for why Algorithm 1 can beat 1/2 for symmetric submodular functions. The interpretable analysis is still a little lengthy and technical. At very high level, the idea is if none of $S_{|O_L|}$ and $O_H \cup S_{|O_L|}$ and

$O_H \cup (S_{2|O_L|} \setminus S_{|O_L|})$ beats 1/2, then we can show that (i) $f(O_H) = f(O_L) = \frac{f(O_H \cup O_L)}{2}$, (ii) $f(O_H|S_{2|O_L|}) = -f(O_H)$, and (iii) $f(S_{3|O_L|} \setminus S_{2|O_L|}|S_{2|O_L|}) \geq (1 - 1/e)f(O_L)$. The punchline is that using (ii), we can further show that (iv) $f(S_{3|O_L|} \setminus S_{2|O_L|}|O_H) \geq f(S_{3|O_L|} \setminus S_{2|O_L|}|S_{2|O_L|})$ (basically, the argument is if O_H hurts $S_{2|O_L|}$ a lot, then by Lemma 16, which is due to symmetry, one can argue O_H can not hurt $S_{3|O_L|}$ more than how much it hurts $S_{2|O_L|}$). Therefore, by (i) and (iii) and (iv), we know that $O_H \cup (S_{3|O_L|} \setminus S_{2|O_L|})$ beats 1/2 approximation. Now we explain this idea in more details.

Informal interpretable analysis

The starting point is the analysis for Theorem 12. The reader can first review the intuition given in the beginning of the subsection of Theorem 12. There we argued that the algorithm achieves half of $f(O_H \cup S_{|O_L|}) + f(S_{|O_L|}) \geq f(O_H \cup O_L)$, and hence for the instance to be hard (in the sense that the algorithm only gets 1/2 approximation), it requires $f(O_H \cup S_{|O_L|}) = f(S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$. This implies $f(O_H|S_{|O_L|}) = 0$, and by submodularity $f(O_H|O_L \cup S_{|O_L|}) \leq 0$, and hence $f(O_H \cup O_L \cup S_{|O_L|}) \leq f(O_L \cup S_{|O_L|})$. Recall that we argued $S_{|O_L|}$ can not hurt $O_H \cup O_L$ significantly, and thus, $f(O_H \cup O_L) \leq f(O_H \cup O_L \cup S_{|O_L|}) \leq f(O_L \cup S_{|O_L|})$, but since $f(S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$, we have that $f(O_L|S_{|O_L|}) = f(S_{|O_L|})$, which implies[13] $f(S_{|O_L|}) \geq f(O_L)$. Moreover, since $f(S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$, we have $f(O_L) \leq \frac{f(O_H \cup O_L)}{2}$ and hence $f(O_H) \geq \frac{f(O_H \cup O_L)}{2}$, but we also have $f(O_H) \leq \frac{f(O_H \cup O_L)}{2}$ because otherwise $O_H \cup S_{|O_L|}$ should have beaten 1/2-approximation as $S_{|O_L|}$ does not hurt O_H significantly, and therefore it holds that $f(O_H) = f(O_L) = \frac{f(O_H \cup O_L)}{2}$.

Now consider the set $S_{2|O_L|} \setminus S_{|O_L|}$. If $f(S_{2|O_L|} \setminus S_{|O_L|}|O_H) = 0$ (otherwise $S_{2|O_L|} \setminus S_{|O_L|} \cup O_H$ beats 1/2), then by submodularity and the fact that $S_{2|O_L|} \setminus S_{|O_L|}$ does not hurt anything significantly (which is yet another application of Lemma 10), we have $f(S_{2|O_L|} \setminus S_{|O_L|}|O_H \cup S_{|O_L|}) = 0$ and hence $f(O_H \cup S_{2|O_L|}) = f(O_H \cup S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$. Notice that $f(O_L|O_H \cup S_{2|O_L|}) = f(O_L \cup O_H \cup S_{2|O_L|}) - f(O_H \cup S_{2|O_L|}) \geq f(O_L \cup O_H) - f(O_H \cup S_{2|O_L|}) = \frac{f(O_H \cup O_L)}{2}$, where the inequality is again due to the fact that $S_{2|O_L|}$ does not hurt. Therefore, similar to how we argued $f(S_{|O_L|}) \geq f(O_L)$, we can show that $f(S_{2|O_L|} \setminus S_{|O_L|}|S_{|O_L|}) \geq f(O_L)$. Since $f(S_{2|O_L|}) \geq f(S_{2|O_L|} \setminus S_{|O_L|}|S_{|O_L|}) + f(S_{|O_L|}) \geq 2f(O_L) = 2f(O_H)$ and $f(O_H \cup S_{2|O_L|}) = \frac{f(O_H \cup O_L)}{2} = f(O_H)$, we have that $f(O_H|S_{2|O_L|}) \leq -f(O_H)$, and together with Lemma 16, we have that $f(O_H|S_{2|O_L|}) = -f(O_H)$.

Here comes the final punchline – If $S_{3|O_L|}/S_{2|O_L|}$ has significant marginal contribution to $S_{2|O_L|}$, then $S_{3|O_L|}/S_{2|O_L|}$ must have at least the same marginal contribution to O_H (and therefore, $O_H \cup S_{3|O_L|}/S_{2|O_L|}$ will beat 1/2). Specifically, this follows from

$$
\begin{aligned}
f(S_{3|O_L|}/S_{2|O_L|}|O_H) &\geq f(S_{3|O_L|}/S_{2|O_L|}|O_H \cup S_{2|O_L|}) \\
&\qquad \text{(By submodularity)} \\
&= f(O_H|S_{3|O_L|}) - f(O_H|S_{2|O_L|}) + f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|}) \\
&\geq -f(O_H) - f(O_H|S_{2|O_L|}) + f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|}) \\
&\qquad \text{(By Lemma 16)} \\
&= f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|}) \\
&\qquad \text{(By } f(O_H|S_{2|O_L|}) = -f(O_H)\text{)}
\end{aligned}
$$

[13] Intuitively, by the if condition at line 9 of Algorithm 1, we can show that the marginal contribution of each iterate of $S_{|O_L|}$ is at least $\frac{f(O_L|S_{|O_L|})}{|O_L|}$, but because $f(O_L|S_{|O_L|}) = f(S_{|O_L|})$, each iterate actually makes the same marginal contribution. Notice that the first iterate should make contribution more than any element in O_L by the if condition.

(One can check the first equality is true by expanding both sides of the equality.) It remains to show $f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|})$ is indeed significantly large. This is essentially due to $f(O_L|O_H \cup S_{2|O_L|}) \geq \frac{f(O_H \cup O_L)}{2}$, which we argued earlier, and the if condition at line 9. In particular, we can show $f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|})$ is at least $1-1/e$ fraction of $f(O_L|S_{2|O_L|})$ by the standard analysis of the classic greedy algorithm for monotone submodular maximization.

In the full version, we will formally prove the better-than-1/2 approximation of Algorithm 1 using the factor-revealing programs.

▶ **Theorem 17.** *For sufficiently small ε and large k, Algorithm 1 achieves strictly better-than-1/2 approximation for non-negative non-monotone symmetric submodular functions in the random-order streaming model.*

On a side note, the constant we get here is by no means tight. (Indeed, we have an improvement, which may also improve the constant for our FPT algorithm, but it requires numerically solving non-convex programs rather than the convex programs in our proofs.) What is interesting is the separation between symmetric and general submodular functions in the random-order streaming model. Also, it is tempting to conjecture that Algorithm 1 achieves optimal $1-1/e$ approximation for monotone submodular functions, given its success in the non-monotone regime. Nonetheless, we have a hard instance that refutes this conjecture. The details would be made available to the interested reader upon request.

6 Tight 1/2 hardness for random-order streaming non-monotone submodular maximization

In this section, we present the lower bound result for non-monotone submodular maximization in the random-order streaming model (described in Section 2). The approximation factor in the lower bound result is tight because of the upper bound in Theorem 12 for example.

▶ **Theorem 18.** *Assuming $n = 2^{o(k)}$, any $(1/2+\varepsilon)$-approximation algorithm for non-monotone submodular maximization in the random-order streaming model must use $\Omega(n/k^2)$ memory. In fact, this hardness result holds against even stronger algorithms (see Remark 9) that are beyond the scope of the standard random-order streaming model for submodular maximization.*

The formal proof is provided in the full version. Here we give the construction of the hard instance and explain the main idea.

Construction of the hard instance

The function f we construct here is essentially a cut function on an unweighted bipartite directed hypergraph[14] plus a modular function. The ground set $V := A_1 \cup A_2$ for f is the set of n vertices of the graph, where A_1 and A_2 denote the two parts respectively. Specifically, $A_2 := \{u_1, \ldots, u_{\varepsilon k}\}$, and A_1 is partitioned into $\ell := (n-\varepsilon k)/b$ buckets of vertices $B_1, \ldots, B_\ell$, each of size $b := k - \varepsilon k$. Now we describe a random generating procedure that generates the hyperedges in the graph:

1. First, for each $i \in [\ell]$, we sample a random subset of vertices $N_i \subset A_2$ of size $|N_i| = \varepsilon^2 k$, and for each $u_j \in N_i$, we create a directed hyperedge from B_i to u_j.

[14] A directed hyperedge in a directed hypergraph is represented by some (U, v), where U is a subset of vertices, and $v \notin U$ is a vertex. For any subset of vertices S, a hyperedge (U, v) is cut by S iff $|U \cap S| > 0$ and $v \notin S$. It is well-known that such cut function is submodular.

2. Then, we slightly modify the graph generated in step 1 as follows: We sample a uniformly random $g \in [\ell]$. For each $u_j \in N_g$, we remove the hyperedge from B_g to u_j, and instead, for each $v \in B_g$, we create a directed hyperedge from $\{v\}$ to u_j. (That is, for each $u_j \in N_g$, we replace the hyperedge from B_g to u_j with individual edges from each $v \in B_g$ to u_j.)

The final submodular function $f : V \to \mathbb{R}_{\geq 0}$ is the sum of the cut function on the above generated hypergraph plus the modular function $c(S) := (\varepsilon^2 k^2) \cdot \frac{|S \cap A_2|}{|A_2|}$. See Figure 1 for an illustration.

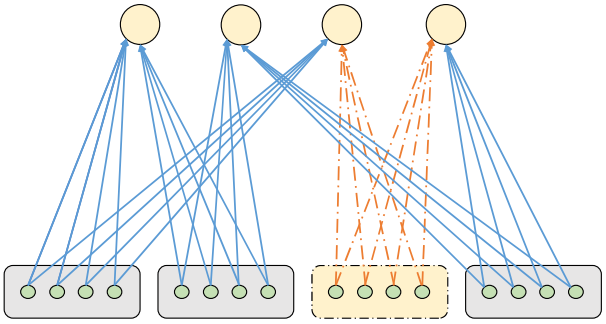


Figure 1 An illustration of our hard instance: At the top we have all εk vertices of A_2, each of which has value εk. At the bottom we have all $(n - \varepsilon k)$ vertices of A_1 that are separated into $\ell = (n - \varepsilon k)/b$ buckets, each of which has $b = k - \varepsilon k$ vertices. We choose a bucket B_g (with yellow filling and dashed outline) uniformly at random. There are εk individual edges (orange and dashed) from each vertex in bucket B_g to B_g's neighborhood N_g, and there are εk hyperedges (blue and solid) from every other bucket B_i (with gray filling and solid outline) to its neighborhood N_i.

Basically, in the above hard instance, we constructed a single good bucket B_g (which is incident to independent individual edges and hence can contribute high value in a cut) and lots of bad buckets B_i (which is incident to hyperedges and hence can only contribute low value in a cut) for all $i \in [\ell] \setminus \{g\}$.

The optimal solution for this hard instance is $B_g \cup (A_2 \setminus N_g)$, i.e., the union of the vertices in the good bucket B_g and the vertices in A_2 except B_g's neighborhood N_g, and specifically, the edges incident to B_g contribute half of the optimal value due to the cut function, and $A_2 \setminus N_g$ contributes the other half due to the modular function c.

However, it is hard for an $o(n/k^2)$-memory algorithm to find out which bucket is the good bucket. Intuitively, to tell whether a bucket B_i is the good bucket, the algorithm needs to query the value of a set that contains at least two vertices of B_i, because otherwise, the value of the set does not depend on whether B_i is incident to individual edges or hyperedges. In order to query the value of a set that contains at least two vertices of B_i, the algorithm has to store at least two vertices of B_i in its memory, because of the restriction in the random-order streaming model (see Section 2). Since the algorithm has low memory, it cannot store two elements for many B_i's at the same time. Using this observation, we can prove w.h.p. the algorithm cannot find the good bucket B_g during the entire stream. Furthermore, we can show by standard concentration inequality that w.h.p. any size-$(\leq k)$ solution set that does not contain enough vertices of B_g has at most half of the optimal value.

Although it was easy to describe the basic idea above, formalizing it requires nontrivial efforts, and we believe that the techniques in our formal proof, which we provide in the full version, are interesting and could be applied for proving hardness of other problems in the random-order streaming setting.

References

1 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular Secretary Problem with Shortlists. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 1:1–1:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.1.

2 Naor Alaluf, Alina Ene, Moran Feldman, Huy L Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *ICALP*, 2020.

3 Naor Alaluf and Moran Feldman. Making a Sieve Random: Improved Semi-Streaming Algorithm for Submodular Maximization under a Cardinality Constraint. *arXiv:1906.11237 [cs]*, June 2019. arXiv:1906.11237.

4 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming Submodular Maximization: Massive Data Summarization on the Fly. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680. ACM, 2014. doi:10.1145/2623330.2623637.

5 Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.

6 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A Tight Linear Time (1/2)-Approximation for Unconstrained Submodular Maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015. doi:10.1137/130929205.

7 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2015. doi:10.1007/978-3-662-47672-7_26.

8 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 42:1–42:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.42.

9 Ethan R. Elenberg, Alexandros G. Dimakis, Moran Feldman, and Amin Karbasi. Streaming weak submodularity: Interpreting neural networks on the fly. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4044–4054, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/c182f930a06317057d31c73bb2fedd4f-Abstract.html.

10 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing Non-monotone Submodular Functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011. doi:10.1137/090779346.

11 Moran Feldman. Maximizing symmetric submodular functions. *ACM Trans. Algorithms*, 13(3):39:1–39:36, 2017. doi:10.1145/3070685.

12 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 730–740, 2018. URL: http://papers.nips.cc/paper/7353-do-less-get-more-streaming-submodular-maximization-with-subsampling.

13 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The One-way Communication Complexity of Submodular Maximization with Applications to Streaming and Robustness. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1363–1374. ACM, 2020. doi:10.1145/3357713.3384286.

14 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1098–1116. SIAM, 2011.

15 Ran Haba, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. Streaming submodular maximization under a k-set system constraint. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3939–3949. PMLR, 2020. URL: http://proceedings.mlr.press/v119/haba20a.html.

16 Chien-Chung Huang, Naonori Kakimura, Simon Mauras, and Yuichi Yoshida. Approximability of Monotone Submodular Function Maximization under Cardinality and Matroid Constraints in the Streaming Model. *arXiv:2002.05477 [cs]*, February 2020. arXiv:2002.05477.

17 Piotr Indyk and Ali Vakilian. Tight Trade-offs for the Maximum k-Coverage Problem in the General Streaming Model. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 200–217. ACM, 2019. doi:10.1145/3294052.3319691.

18 Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular Streaming in All its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3311–3320. PMLR, 2019.

19 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast Greedy Algorithms in MapReduce and Streaming. *TOPC*, 2(3):14:1–14:22, 2015. doi:10.1145/2809814.

20 Paul Liu, Aviad Rubinstein, Jan Vondrák, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. *Advances in Neural Information Processing Systems*, 34, 2021.

21 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 62–81. SIAM, 2020. doi:10.1137/1.9781611975994.5.

22 Andrew McGregor and Hoa T. Vu. Better Streaming Algorithms for the Maximum Coverage Problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019. doi:10.1007/s00224-018-9878-x.

23 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1379–1386. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17014.

24 George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978.

25 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions - I. *Math. Program.*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

26 Mohammad Shadravan. Improved submodular secretary problem with shortlists. *CoRR*, abs/2010.01901, 2020. arXiv:2010.01901.

27 Piotr Skowron. FPT approximation schemes for maximizing submodular functions. *Inf. Comput.*, 257:65–78, 2017. doi:10.1016/j.ic.2017.10.002.

28 Jan Vondrák. Symmetry and Approximability of Submodular Maximization Problems. *SIAM J. Comput.*, 42(1):265–304, 2013. doi:10.1137/110832318.

Approximate Triangle Counting via Sampling and Fast Matrix Multiplication

Jakub Tětek
Basic Algorithms Research Copenhagen, University of Copenhagen, Denmark

Abstract

There is a simple $O(\frac{n^3}{\epsilon^2 T})$ time algorithm for $1\pm\epsilon$-approximate triangle counting where T is the number of triangles in the graph and n the number of vertices. At the same time, one may count triangles exactly using fast matrix multiplication in time $\tilde{O}(n^\omega)$. Is it possible to get a negative dependency on the number of triangles T while retaining the state-of-the-art n^ω dependency on n? We answer this question positively by providing an algorithm which runs in time $O\left(\frac{n^\omega}{T^{\omega-2}}\right) \cdot \text{poly}(n^{o(1)}/\epsilon)$. This is optimal in the sense that as long as the exponent of T is independent of n, T, it cannot be improved while retaining the dependency on n. Our algorithm improves upon the state of the art when $T \gg 1$ and $T \ll n$.

We also consider the problem of approximate triangle counting in sparse graphs, parameterized by the number of edges m. The best known algorithm runs in time $\tilde{O}_\epsilon\left(\frac{m^{3/2}}{T}\right)$ [Eden et al., SIAM Journal on Computing, 2017]. An algorithm by Alon et al. [JACM, 1995] for exact triangle counting that runs in time $\tilde{O}(m^{2\omega/(\omega+1)})$. We again get an algorithm whose complexity has a state-of-the-art dependency on m while having negative dependency on T. Specifically, our algorithm runs in time $O\left(\frac{m^{2\omega/(\omega+1)}}{T^{2(\omega-1)/(\omega+1)}}\right) \cdot \text{poly}(n^{o(1)}/\epsilon)$. This is again optimal in the sense that no better constant exponent of T is possible without worsening the dependency on m. This algorithm improves upon the state of the art when $T \gg 1$ and $T \ll \sqrt{m}$.

In both cases, algorithms with *time* complexity matching *query* complexity lower bounds were known on some range of parameters. While those algorithms have optimal *query* complexity for the whole range of T, the time complexity departs from the query complexity and is no longer optimal (as we show) for $T \ll n$ and $T \ll \sqrt{m}$, respectively. We focus on the time complexity in this range of T. To the best of our knowledge, this is the first paper considering the discrepancy between query and time complexity in graph parameter estimation.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Approximate triangle counting, Fast matrix multiplication, Sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.107

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2104.08501`

Funding The author was supported by the VILLUM Foundation grant 16582 and by the Bakala Foundation.

Acknowledgements I would like to thank several people: Václav Rozhoň, who gave early feedback and who suggested using a recursive approach; Rasmus Pagh for helping improve a draft of this paper; my supervisor Mikkel Thorup for helpful discussions and for his support; and Talya Eden for advice regarding related work.

1 Introduction

The problem of counting triangles in a graph is both a fundamental problem in graph algorithms and a problem with many applications, for example in network science [5], biology [8, 19, 21] or sociology [25, 15]. Triangle counting is, also for these reasons, one of the basic procedures in graph mining. Consequently, triangle counting has received a lot of attention both in the theoretical and applied communities.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 107; pp. 107:1–107:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are algorithms for *exact* triangle counting that have time complexity $\tilde{O}(n^\omega)$ [1] and $\tilde{O}(m^{2\omega/(\omega+1)})$ [3] where ω is the smallest constant such that two $n \times n$ matrices can be multiplied in time $n^{\omega+o(1)}$.

A natural variant of this problem is *approximate triangle counting*. In this problem, we are given a graph with T triangles and want to output $\hat{T} = (1 \pm \epsilon)T$ with probability 2/3. Several algorithms are known for approximate triangle counting that have *negative dependence* on the number of triangles T. Assuming the algorithm may sample random vertices and edges, there are algorithms that run in time $O_\epsilon(\frac{n^3}{T})$ [2] and $\tilde{O}_\epsilon(\frac{m^{3/2}}{T})$ [9][3]. However, no algorithm is known that would get the best of both worlds – algorithms with state-of-the-art dependency on n or m while being negatively dependent on T. We show two such algorithms in this paper. Moreover, our algorithms are optimal in the following sense: the dependence on T in the time complexity of our algorithms cannot be improved without worsening the dependency on n or m as long as the exponent of T is constant (i.e., is independent of n, m, T). This optimality follows from the lower bound shown by Eden, Levi, Ron, and Seshadhri [9], as we show in the full version of this paper.

The main contribution of our paper is an algorithm for approximate triangle counting that runs in time $\tilde{O}_\epsilon\left(\frac{n^\omega}{T^{\omega-2}}\right)$. We then use it to give an algorithm for the same problem running in time $\tilde{O}_\epsilon\left(\frac{m^{2\omega/(\omega+1)}}{T^{2(\omega-1)/(\omega+1)}}\right)$. This improves upon our first algorithm for sparse graphs. Note that both time complexities are sublinear for sufficiently large values of T. To the best of our knowledge, this is the first work that uses fast matrix multiplication in sublinear-time algorithms. Assuming constant ϵ, our algorithms improve upon the state-of-the-art when $T \ll n$, and $T \gg 1$ or $T \ll \sqrt{m}$ and $T \gg 1$ (for Algorithm 4 and Algorithm 5, respectively). In other words, our algorithms improve upon the state of the art when the time complexity (of both our and the state-of-the-art algorithms) is $\gg n^2$ and $\gg m$, respectively. Figure 1 shows a plot of the complexity of our algorithms, in comparison to other algorithms for approximate triangle counting.

The basic approach we use is to sample vertices, recursively count triangles in the subgraph induced by these sampled vertices, and estimate the total number of triangles based on this count. The main hurdle in this approach is that when a vertex is contained in many triangles (we call such a vertex triangle-heavy, and triangle-light otherwise), this results in a poor concentration of the number of triangles in the induced subgraph as the inclusion of one vertex can make a large difference in the number of triangles. We get around this by introducing a procedure based on fast matrix multiplication which finds all vertices contained in many triangles, and a procedure for approximately counting triangles that contain at least one such vertex. We count triangles that consist of light vertices separately. The fact that the vertices are light helps us ensure concentration of our estimate. These triangles are counted using a recursive approach. An obstacle to overcome is that in some cases the time spent in each successive level of recursion may grow exponentially.

---

[1] We use $\tilde{O}(f(x))$ with the meaning that it ignores an $f(x)^{o(1)}$ factor; this is necessary as ω only determines the time complexity up to $n^{o(1)}$. This differs from the usual use in which $\tilde{O}(f(x)) = \cup_{c>0} O(f(x)\log^c f(x))$. $\tilde{O}_\epsilon$ in addition ignores $\text{poly}(\frac{1}{\epsilon})$.

[2] This algorithm follows from a paper of Lipton, Naughton, Schneider, and Seshadhri [20]. In this paper, the authors show an algorithm that gives a relative approximation to the bias of a Bernoulli trial. One may reduce approximate triangle counting to this problem by picking at random three vertices and checking whether they form a triangle. See also the presentation of Watanabe [24].

[3] The main result of the paper is an algorithm and lower bound in a slightly different setting. This algorithm is only mentioned in the paper briefly.

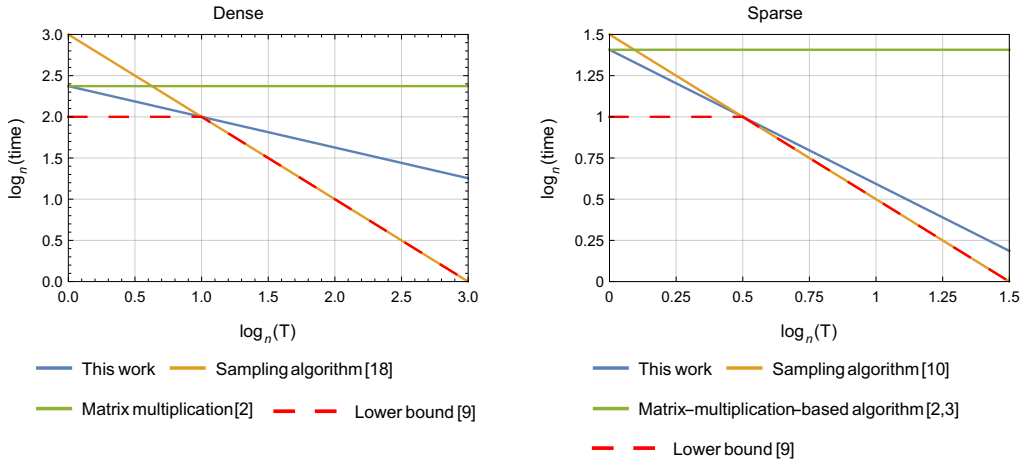


Figure 1 Comparison of the time complexities, assuming ϵ is constant. This plot assumes the best currently known bound $\omega < 2.3728596$ from [2]. For more detail, see Table 1.

We find it surprising that it is sufficient to use matrix multiplication to find the vertices contained in many triangles. That is, the matrix multiplication is only used to decide which triangles are going to be counted by which subroutine, while the counting itself is done without the help of matrix multiplication.

The setting. When an algorithm runs in sublinear time, it is of importance how the algorithm is allowed to access the graph. This is because the algorithm does not have the time to pre-process the whole graph. We assume the following standard setting: the algorithm may in constant time (1) get the i-th vertex, (2) get the j-th neighbor of a given vertex, (3) query the degree of a given vertex, and (4) given two vertices, tell whether they are adjacent or not (this is called the pair query). In the second part of this paper, we assume a setting where instead of the query (1), we can in constant time (1') get the i-th edge. This setting is also standard in the area of sublinear algorithms. As the main contribution of this paper is in the superlinear regime (as this is where we improve upon the state of the art), we do not describe the settings in more detail. See, for example, [13] for a more detailed description.

Time complexity vs. query complexity. In the area of sublinear algorithms, it is customary to focus on the *query complexity* (that is the number of queries, in the above sense, performed) of an algorithm, as opposed to the time complexity. The time complexity is then usually (near-)linear in the query complexity. However, this is the case only in the sublinear regime, and it breaks down when the time complexity is superlinear[4]. In this case, one may always read the whole graph in $O(n+m)$ queries. The time complexity is then usually significantly larger than the query complexity. This regime is our main focus.

Specifically, an algorithm is described in [9] that uses the edge access query (1') and has query complexity $\tilde{O}_\epsilon(\min(n+m, m^{3/2}/T))$. This is known to be near-optimal[5] [11]. In the sublinear regime (that is, for T large enough), the asymptotic time complexity is the same

[4] It may seem confusing talking about a sublinear-time algorithm having superlinear time complexity. The reason is that an algorithm is usually said to be sublinear time if its complexity is sublinear for some setting of the parameters (but not necessarily for all).

[5] Whenever we talk about optimality, we consider the case of ϵ being constant.

as the query complexity and it is, therefore, also near-optimal. However, in the superlinear regime, the time complexity is polynomially larger than the query complexity. Up until now, nothing was known about whether the time complexity can be improved in this case. We answer this question positively. In doing this, we give, to the best of our knowledge, the first algorithm answering a question of this type (that is, a question regarding the discrepancy between the time and query complexities in the superlinear regime).

1.1 Our results

The main contributions of this paper are the following two algorithms for approximate triangle counting. The first algorithm (Algorithm 4) is suitable for dense graphs and its time complexity is parameterized by n, T, and ϵ. The second algorithm (Algorithm 5) is suitable for sparse graphs and its time complexity is parameterized by m and T. Both of our algorithms have a state-of-the-art dependency on n, m while being negatively dependent on T. Our algorithms improve upon the existing algorithms when $1 \ll T \ll n$ and $1 \ll T \ll \sqrt{m}$, respectively.

We compare these results to the previous work in the following table. These time complexities are also plotted in Figure 1.

Table 1 Algorithms for approximate triangle counting. Algorithms with complexity independent of ϵ in fact solve exact triangle counting. (Note that algorithms with dependency on ϵ have the dependency hidden in O_ϵ in this table.)

	Dense	Sparse
This work	$\tilde{O}_\epsilon\left(\frac{n^\omega}{T^{\omega-2}}\right)$	$\tilde{O}_\epsilon\left(\frac{m^{2\omega/(\omega+1)}}{T^{2(\omega-1)/(\omega+1)}}\right)$
Sampling algorithm [20]	$O_\epsilon(\frac{n^3}{T})$	
Fast matrix multiplication	$\tilde{O}(n^\omega)$	
Alon, Yuster, and Zwick[3]		$\tilde{O}(m^{2\omega/(\omega+1)})$
Eden et. al. [9]	$\Omega\big(\min(n^2, \frac{n^3}{T})\big)$	$\tilde{O}_\epsilon(\frac{m^{3/2}}{T})$, $\Omega\big(\min(m, \frac{m^{3/2}}{T})\big)$

1.2 Related work

Algorithms. The number of triangles in a graph can be trivially counted in time $O(n^3)$. Itai and Rodeh [16] obtained an algorithm that runs in time $O(m^{3/2})$, a significant improvement for sparse graphs. A simpler algorithm with the same complexity, based on bounds on arboricity has been given by Chiba and Nishizeki [7]. A more efficient algorithm for *approximate* triangle counting has been obtained by Kolountzakis, Miller, Peng, and Tsourakakis [18]. In their paper, the authors presented an algorithm that runs in time $\tilde{O}_\epsilon(m + \frac{m^{3/2}}{T})$. This has been later improved by Eden et. al. [9] to $\tilde{O}_\epsilon(\frac{n}{T^{1/3}} + \frac{m^{3/2}}{T})$ when having vertex queries and not edge queries (having query (1) but not (1′)) and to $\tilde{O}_\epsilon(\frac{m^{3/2}}{T})$ when having edge queries (query (1′)).

While fast matrix multiplication gives a $\tilde{O}(n^\omega)$ algorithm for exact triangle counting, it is not immediately clear an improvement can be achieved in sparse graphs. In their paper, Alon, Yuster, and Zwick [3] show an algorithm for exact triangle counting that runs in time $\tilde{O}(m^{2\omega/(\omega+1)})$. This algorithm works by counting triangles whose all vertices have high degree using matrix multiplication while using a naïve algorithm for the rest of the graph. We use a variant of this approach for approximate triangle counting in sparse graphs in Section 3.

Sampling from graphs in order to estimate the number of triangles has been considered many times [23, 22, 17, 6]; for more details, see the introduction of [17]. The bounds then often depend on an upper bound on the number of triangles containing any single edge or similar parameters, for example in [22] or [23]. In this paper, we go further in the sense that we use a similar sampling approach (in fact, we use the same sampling scheme that was used in [6] but use a finer analysis) but explicitly ensure the bound on the number of triangles containing any single vertex. In other papers using similar sampling schemes, it is just assumed that this or a similar bound is already satisfied for the input instance.

Lower bounds. From the side of lower bounds, Eden et. al. [9] have proven a lower bound on the query complexity, thus also implying a lower bound on the time complexity. This lower bound is presented for the setting when random edge queries are not allowed, but the lower bound of $\Omega(\min(m, m^{3/2}/T))$ also holds in the setting when they are. This was made explicit by Eden and Rosenbaum [12] who presented a significantly simpler proof based on communication complexity.

▶ **Theorem 1** (Theorem 4.7 of [12]). *For any $n, m \in O(n^2), T \in O(m^{3/2})$, it holds that any multiplicative-approximation algorithm for the number of triangles in a graph must perform $\Omega\left(\min\left(m, \frac{m^{3/2}}{T}\right)\right)$ queries, where the allowed queries are degree queries, pair queries, random neighbor queries, random vertex queries, and random edge queries.*

We use this lower bound to prove the claim that the exponent of T in Theorem 12 is optimal, out of all algorithms running in time $\tilde{O}(n^\omega/T^c)$ for some constant c and similarly the exponent in Theorem 15 is optimal out of all algorithms running in time $\tilde{O}(m^{2\omega/(\omega+1)}/T^c)$.

By choosing $m = \Theta(n^2)$ in the above theorem, it follows that

▶ **Corollary 2.** *For any n and $T \in O(n^3)$, it holds that any multiplicative-approximation algorithm for the number of triangles in a graph must perform $\Omega\left(\min\left(n^2, \frac{n^3}{T}\right)\right)$ queries, where the allowed queries are degree queries, pair queries, neighbor queries, random vertex queries, and random edge queries.*

Fast matrix multiplication. There is a large literature on fast matrix multiplication. We only mention here the asymptotically most efficient matrix multiplication algorithm which is currently the algorithm by Alman and Williams [2] which runs in time $\tilde{O}(n^\omega)$ for some $\omega < 2.3728596$.

1.3 Technical overview

1.3.1 Triangle counting in dense graphs

Sampling and (lack of) concentration. Suppose we sample each vertex independently with some probability p. The expected number of triangles in the subgraph induced by the sampled vertices is then p^3T. If we were able to show concentration around the expected value, we could count the number of triangles in this subgraph and based on that, estimate the number of triangles in the original graph.

Unfortunately, the number of triangles is not concentrated; for example, in the case when there is one vertex that is contained in all triangles. We show that we get concentration if we limit the number of triangles containing any single vertex. We, therefore, have some threshold τ and call all vertices that are contained in more than τ triangles triangle-heavy. Assume that the graph does not have any triangle-heavy vertices. By choosing τ sufficiently

small and by choosing p sufficiently large, we may get a good estimate of the number of triangles in the original graph, based on the number of triangles in the subgraph induced by the sampled vertices.

In the final algorithm, we separately count triangles that contain at least one triangle-heavy vertex and triangles that do not. For this, we need to be able to find all triangle-heavy vertices in the graph and then estimate the number of triangles that contain at least one triangle-heavy vertex. We efficiently count triangles with no triangle-heavy vertex by reducing the problem to one on a smaller graph by the sampling procedure we just described.

Finding triangle-heavy vertices by matrix multiplication. We now sketch how to find the set of vertices that are contained in many triangles. We sample each vertex with some small probability p'. This gives a subset of vertices of size $\approx p'n$. We find all vertices that are contained in a triangle in the subgraph induced by the sampled vertices in time $O\big((p'n)^\omega\big)$ by using matrix multiplication. We repeat this experiment an appropriate number of times. We report vertices that were contained in at least one triangle in at least one repetition of the experiment. If a vertex is contained in few triangles, the probability that it is reported can be bounded by using union bound over all triangles it is contained in and over all iterations. The intuition for why vertices contained in many triangles are reported with good probability is the following. We bound the variance of the total number of triangles that the vertex will be in over all repetitions. The repetitions are independent, but there are correlations between triangles within one repetition. If we set the probability p' to be small enough, we can show that these correlations are small. This allows us to show concentration of the number of triangles containing any single vertex that are observed in the sampled subgraphs.

Counting triangles with triangle-heavy vertices. Suppose we are given a subset of vertices such that each of those vertices is contained in many triangles. We now sketch how to give a factor $3+\epsilon$ approximation to the number of triangles that contain at least one of these vertices. In the body of the paper, we then give a $(1+\epsilon)$-approximation by a more careful argument.

We estimate separately for each vertex the number of triangles it is contained in. The fact that each vertex individually is in many triangles allows us to employ a concentration argument separately for each vertex. We then use union bound over all vertices. Specifically, for each vertex v from the set of vertices that is given to us, we perform the following experiment many times: we sample two vertices and we check whether they form a triangle with v. We use the proportion of the experiments that ended up with a triangle to estimate the number of triangles v is contained in. We add together the estimates for all vertices in the set. This is where the factor 3 comes from – a triangle may be counted from all its 3 vertices.

Recursive algorithm. The last trick we introduce is that we use recursion to approximately count the triangles in the sampled subgraph. Doing this in such a way to give a correct algorithm with the desired complexity is the technically most challenging part of our paper. An obstacle is that we need the precision to be increasing in the depth of recursion. Specifically, in our algorithm, the allowed error decreases at an exponential rate in the depth of recursion. Moreover, we need to use probability amplification, meaning that the number of recursive calls also grows exponentially with the depth of recursion. This leads, in some situations, to the time complexity of the k-th level of recursion increasing exponentially in k; we bound this time.

1.3.2 Triangle counting in sparse graphs

We set a parameter θ and say a vertex is degree-heavy if its degree is at least θ and degree-light otherwise. We count separately *(a)* triangles on the subgraph induced by the degree-heavy vertices and *(b)* triangles that contain at least one degree-light vertex. This is basically the idea used by Alon, Yester, and Zwick [3] for their exact triangle-counting algorithm. We use our algorithm for approximate triangle counting in dense graphs to count the triangles from case *(a)* and sampling to count the triangles *(b)*. The sampling-based estimation is based on ideas from [7, 9]. Specifically, we assign each triangle to its vertex with the lowest degree (breaking ties arbitrarily). We then repeatedly perform the following experiment: pick an edge uniformly at random; consider its endpoint v with the lower degree; pick one of its neighbors at random; check whether it forms a triangle together with the edge we have sampled and whether the triangle is assigned to v. If so, return $d(v)$, otherwise return 0. We estimate the number of triangles based on the average of the returned values of the experiments.

1.4 Notation

Throughout the paper, we denote the number of vertices and edges of a graph by n and m, respectively. We use T to denote the number of triangles. We define $a \pm b = [a - b, a + b]$. This can be used for example in $(1 \pm \epsilon)a$ which is then equal to $[(1 - \epsilon)a, (1 + \epsilon)a]$. We call $(1 \pm \epsilon)$-approximation to a number a any number b such that $b \in (1 \pm \epsilon)a$. Similarly, we call *additive* $\pm c$ approximation to a number a any number b such that $b \in a \pm c$. We call *diamond* a graph isomorphic to  and *butterfly* a graph isomorphic to  . By "the number of diamonds in G" we mean the number of, not necessarily induced, subgraphs of G isomorphic to a diamond. We similarly talk about "the number of butterflies in a graph". We denote the subgraph of G induced by $V' \subseteq V$ by $G[V']$.

2 Triangle counting in dense graphs

In this section, we present the main result of our paper – an algorithm for approximate triangle counting, parameterized by n and the number of triangles T. In the lemmas that follow, we prove, among other things, bounds on the expectation of the estimators. We do this because our algorithm requires some "advice" and having a bound on the expectation will allow us to remove the need for this advice, as we describe in Section 2.5. We now define some notation that we will need.

We call a vertex *triangle-dense* for a parameter τ if there are at least τ triangles that contain the vertex and *triangle-light* if at most $\tau/20$ triangles contain it (there can thus be vertices that are neither triangle-light nor triangle-heavy). We define T_v to be the number of triangles containing the vertex v. We abuse notation and also denote by T_v the set of all triangles containing v. We use $T_v(G)$ instead when we want to explicitly specify the graph. T denotes the number of triangles in G but we again abuse notation and use this to also denote the set of triangles. Again, if we want to specify the graph, we use $T(G)$.

We now prove a simple lemma on fractional moments of the binomial distribution. We will use this to bound the time complexity of matrix multiplication executed on a random subgraph of some given graph. Specifically, we will use it on a subgraph obtained by keeping each vertex with some appropriately chosen probability and removing the other vertices. The lemma also holds for other exponents than ω, but fixing the exponent allows for a simpler proof.

▶ **Lemma 3.** *Assume $p \geq 1/n$. It holds that $E(Bin(n,p)^\omega) = O(n^\omega p^\omega)$.*

Proof. $Bin(n,p)$ has its third moment equal to

$$(n(n-1)(n-2)p^3) + 3(n(n-1)p^2) + np = O(n^3p^3)$$

where the equality holds because $p \geq 1/n$. Since $\omega < 3$, the function $x^{\omega/3}$ is concave. By the Jensen's inequality, we then get $E(Bin(n,p)^\omega) \leq \big(E(Bin(n,p)^3)\big)^{\omega/3} = O(n^\omega p^\omega)$. ◀

2.1 Triangle-light subgraph

We show a reduction from the problem of approximately counting triangles in a graph with no triangle-heavy vertices to the problem of approximately counting triangles in a smaller graph. Before that, we need to prove the following lemma.

▶ **Lemma 4.** *Let G be a graph with T triangles such that any vertex is contained in at most τ triangles and let D, B be the number of diamonds and butterflies in G, respectively. Then $D + B \leq \frac{3}{2}\tau T$.*

Proof. Let S be the number of pairs $(\triangle_1, \triangle_2)$ of triangles in G such that $\triangle_1$ and $\triangle_2$ share at least one vertex. It holds $D + B \leq S/2$ (the factor of 2 is there because the pairs are ordered). Consider a fixed triangle $\triangle_1 = uvw$. How many triangles are there that share a vertex with $\triangle_1$? Each of the vertices u, v, w can be contained in at most $\tau - 1$ other triangles. There can be, therefore, at most $3(\tau - 1)$ incidences between $\triangle_1$ and other triangles containing at least one of the vertices u, v, w. Therefore $D + B \leq S/2 \leq 3T(\tau - 1)/2$. ◀

We are now ready to prove the following lemma. Note that to use it (in order to set the probability p), one has to have a lower bound on the number of triangles. We resolve this issue later.

▶ **Lemma 5.** *Let $G' = (V', E')$ be the induced subgraph of G resulting from keeping each vertex with probability $p \geq \max(5\frac{1}{\sqrt[3]{\epsilon^2 T}}, 10\frac{1}{\epsilon\sqrt{T/\tau}})$ and removing it otherwise. Let $\hat{T} = |T(G')|/p^3$. Then $\hat{T}$ is an unbiased estimate of T. Moreover if any vertex $v \in G$ is contained in at most τ triangles, then $\hat{T} \in (1 \pm \epsilon)T$ with probability at least $19/20$.*

Proof. Let $\triangle$ be a triangle in G. Let $X_\triangle = 1$ if $\triangle \in T(G')$ and 0 otherwise. Let $T' = T(G')$. It holds that

$$E(\hat{T}) = E(T'/p^3) = \sum_{\triangle \in T} E(X_\triangle)/p^3 = T$$

and the estimate is, therefore, unbiased.

We now bound $Var(T')$. Recall that $T(G)$ is the set of triangles in G. Let $D(G)$ the set of diamonds in G, $B(G)$ the set of butterflies in G, and $\ddot{T}(G)$ the set of disjoint (unordered) pairs of triangles in G. Let $D = |D(G)|$, $B = |B(G)|$, and $\ddot{T} = |\ddot{T}(G)|$. By $\{\triangle_1, \triangle_2\} \in D(G)$ we mean that $\triangle_1 \cup \triangle_2 = A$ for some $A \in D(G)$ and analogously for $B(G)$ and $\ddot{T}(G)$.

We first bound the second moment of T'

$$\begin{aligned} E(T'^2) =& E\left(\Big(\sum_{\triangle \in T(G)} X_\triangle\Big)^2\right) \\ =& E\Big(\sum_{\triangle \in T(G)} X_\triangle^2\Big) + E\Big(\sum_{\substack{\triangle_1, \triangle_2 \in T(G) \\ \triangle_1 \neq \triangle_2}} X_{\triangle_1} X_{\triangle_2}\Big) \end{aligned}$$

$$
\begin{aligned}
&= E\Big(\sum_{\triangle \in T(G)} X_{\triangle}^2\Big) + E\Big(2 \sum_{\{\triangle_1,\triangle_2\} \in D(G)} X_{\triangle_1} X_{\triangle_2}\Big) \\
&\quad + E\Big(2 \sum_{\{\triangle_1,\triangle_2\} \in B(G)} X_{\triangle_1} X_{\triangle_2}\Big) + E\Big(2 \sum_{\{\triangle_1,\triangle_2\} \in \ddot{T}(G)} X_{\triangle_1} X_{\triangle_2}\Big) \\
&= p^3 T + 2p^4 D + 2p^5 B + 2p^6 \ddot{T} \\
&\leq p^3 T + 2(D+B)p^4 + p^6 T^2 \\
&\leq p^3 T + 3p^4 \tau T + p^6 T^2
\end{aligned}
$$

where we have used that $D + B \leq \frac{3}{2}\tau T$ (by Lemma 4) and $2\ddot{T} \leq T^2$. At the same time, $E(T') = p^3 T$. Therefore,

$$Var(T') = E(T'^2) - E(T')^2 \leq p^3 T + 3p^4 \tau T.$$

This means that $E(\hat{T}) = T$ and $Var(\hat{T}) = Var(\frac{1}{p^3}T') = \frac{1}{p^3}T + \frac{3}{p^2}\tau T$. Since $p \geq \max(5\frac{1}{\sqrt[3]{\epsilon^2 T}}, 10\frac{1}{\epsilon\sqrt{T/\tau}})$, it holds $Var(\hat{T}) \leq (1/5^3 + 3/10^2)\epsilon^2 T^2 < \frac{1}{20}\epsilon^2 T^2$. It, therefore holds by the Chebyshev inequality that $P(|\hat{T} - T| \geq \epsilon T) \leq 1/20$. ◀

2.2 Triangle-heavy subgraph

We now show an algorithm that approximately counts triangles that contain at least one vertex from set V_H where V_H is some given set that does not contain any triangle-light vertices.

Algorithm 1 Count $(1 \pm \epsilon)$-approximately triangles in G containing a vertex from V_H.

```
for v ∈ V_H do
    T̂_v ← 0
    repeat 360 n² log n / (ε²τ) times
        Sample u, w ∈ V
        Let ℓ = |{u, v, w} ∩ V_H|
        If uvw forms a triangle, increment T̂_v by ε²τ / (360ℓ log n)
return Σ_{v∈V_H} T̂_v
```

▶ **Lemma 6.** *Given a set V_H, Algorithm 1 returns an unbiased estimate of the number of triangles containing at least one vertex from V_H.*

Assume V_H contains all triangle-heavy vertices of G and no triangle-light vertices. Then Algorithm 1 returns $(1 \pm \epsilon)$-approximation of the number of triangles that contain at least one triangle-heavy vertex with probability at least $1 - O(\frac{1}{n})$. It runs in time $O(\frac{Tn^2 \log n}{\epsilon^2 \tau^2})$.

Proof. We introduce charges on vertices. For any triangle $\triangle$ that contains at least one vertex that is in V_H, we divide and charge single unit to the vertices in $\triangle \cap V_H$, dividing it fairly (if there are, e.g., two vertices from V_H in the triangle, they are both charged 1/2). Let χ_v be the charge on vertex v. The total amount charged (that is, $\sum_{v \in V_H} \chi_v$) is equal to the number of triangles that contain at least one vertex from V_H. Note that this is the quantity we want to estimate.

The algorithm considers $v \in V_H$ and samples one pair of vertices u, w. Consider the amount charged by uvw to v; call it χ'_v. It holds that $E(\chi'_v) = \chi_v/n^2$. When uvw form a triangle, the algorithm increments $\hat{T}_v$ by $\frac{\epsilon^2\tau}{360\ell \log n} = \frac{\chi'_v \epsilon^2 \tau}{360 \log n}$. Therefore, the increment is, in expectation, equal to $\frac{\chi_v \epsilon^2 \tau}{360 n^2 \log n}$. Since there are $\frac{n^2 \log n}{360\epsilon^2\tau}$ repetitions, it holds

$$E(\hat{T}_v) = \frac{360n^2 \log n}{\epsilon^2\tau} \cdot \frac{\chi_v \epsilon^2 \tau}{360 n^2 \log n} = \chi_v.$$

By the linearity of expectation, we have $E(\sum_{v \in V_H} \hat{T}_v) = \sum_{v \in V_H} \chi_v$ which we said above is equal to the number of triangles with empty intersection with V_H.

We now prove concentration around mean. For each vertex, we have $\frac{360n^2 \log n}{\epsilon^2\tau}$ independent random variables corresponding to the increments in $\hat{T}$. These random variables have values between 0 and $\frac{\epsilon^2\tau}{360 \log n}$ and their sum has expectation $\chi_v \geq \frac{T_v}{3} \geq \frac{\tau}{60}$ where the second inequality holds from the assumption that V_H contains no triangle-light vertices. By the Chernoff bound,

$$P(|\hat{T}_v - T_v| \geq \epsilon T_v) \leq 2 \exp\Big(- \frac{\epsilon^2\tau/60}{3\epsilon^2\tau/(360 \log n)} \Big) \leq \frac{2}{n^2}.$$

By the union bound, it holds for all v that $\hat{T}_v$ is an $(1 \pm \epsilon)$-approximate of T_v, with probability at least $1 - \frac{2}{n}$. On this event, the algorithm correctly gives a $(1 \pm \epsilon)$-approximation of the number of triangles containing at least one triangle-heavy vertex. ◀

2.3 Finding the triangle-heavy subgraph

In this section, we show how to find a set of vertices that contains each triangle-heavy vertex with probability at least 2/3 and each triangle-light vertex with probability at most 1/3. This guarantee may be strengthened by probability amplification (applying the probability amplification separately to each vertex) to make sure that, with high probability, all triangle-heavy vertices are reported, and none of the triangle-light vertices are. This only adds a $O(\log n)$ factor to the time complexity of this subroutine.

We solve separately the case $\tau \leq n$ and $\tau \geq n$. On the range $\tau \leq n$, we get an algorithm running in time $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$. On the range $\tau \geq n$, we get time complexity $O(\frac{n^3}{\tau}) \subseteq \tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$ (the inclusion holds on this range of τ). This means that on this range, our bound is not tight. However, this is the case only on the range of T for which a near-optimal algorithm was already known, and we only show this for completeness; the reader may wish to skip the case of $\tau \geq n$. Putting the guarantees for the two ranges together, it follows that

▶ **Corollary 7.** *There is an algorithm that with probability at least $1 - O(\frac{1}{n})$ reports all triangle-heavy vertices and no triangle-light vertices while having time complexity $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$.*

2.3.1 The case $\tau \leq n$

▶ **Lemma 8.** *Assuming $\tau \leq n$, Algorithm 2 lists any triangle-heavy vertex with probability at least 2/3 and any triangle-light vertex with probability at most 1/3. It has expected time complexity $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$.*

---

[6] This may be done as follows. Raise the adjacency matrix of $G[V_i]$ to the third power. Each diagonal element in this matrix corresponds to a vertex (as rows and columns correspond to vertices and diagonal vertices have both the row and column corresponding to the same vertex). Consider the vertices that have a non-zero value on the corresponding diagonal position. These are exactly the vertices contained in some triangle by standard equivalence between taking powers of adjacency matrices and between counting walks in graphs.

Algorithm 2 Distinguish triangle-heavy vertices from triangle-light vertices.

$\mathcal{M} \leftarrow \emptyset$
for $i \in [k = 6\tau^2]$ **do**
  $V_i \leftarrow$ sample each vertex $v \in V$ independently with probability $p = \tau^{-1}$
  $V_i' \leftarrow$ find all vertices contained in a triangle in $G[V_i]$ in time $\tilde{O}(|V_i|^\omega)$ [6]
  $\mathcal{M} \leftarrow \mathcal{M} \cup V_i'$
return $\mathcal{M}$

Proof. Consider a triangle-light vertex v. That is, there are at most $\tau/20$ triangles in G containing v. The probability that a triangle is discovered in one iteration is, by the union bound, at most $\tau p^3/20$. Taking a union bound over the k iterations, the probability that a triangle containing v is found (and thus v reported) is at most $k\tau p^3/20 < 1/3$.

Consider a triangle-heavy vertex v. Let X_i for $i \in [k]$ be the number of triangles containing v discovered in the i-th iteration. For $\triangle \in T$, let $X_{\triangle,i}$ be an indicator that triangle $\triangle$ is discovered in i-th iteration. It now holds $X_i = \sum_{\triangle \in T, v \in \triangle} X_{\triangle,i}$. Let $X = \sum_{i=1}^{k} X_i$.

By the linearity of expectation, it holds that

$$E(X) = E\Big(\sum_{i=1}^{k} X_i\Big) = kp^3 T_v = 6T_v/\tau.$$

We now bound the variance of X. We first bound

$$Var(X_i) \leq E(X_i^2) = E\Bigg(\sum_{\triangle_1, \triangle_2 \in T_v} X_{\triangle_1,i} X_{\triangle_2,i}\Bigg) \leq p^3 T_v + p^4 T_v^2$$

where the last inequality holds because there are $\leq T_v^2$ terms (i.e. pairs of triangles containing v) that are 1 with probability $\leq p^4$ (such pairs have at least 4 vertices) and there are T_v terms (i.e. pairs of triangles containing v) that are 1 with probability p^3 (these are pairs that have $\triangle_1 = \triangle_2$ and the pair thus has 3 vertices). Since the iterations are independent, it holds that

$$Var(X) = Var\Big(\sum_{i=1}^{k} X_i\Big) = kVar(X_i) \leq 6\tau^2(p^3 T_v + p^4 T_v^2) = \frac{6}{\tau} T_v + \frac{6}{\tau^2} T_v^2 \leq \frac{12 T_v^2}{\tau^2}$$

where the last inequality holds because $T_v \geq \tau$. It, therefore, holds by the Chebyshev inequality that

$$P(X = 0) \leq \frac{Var(X)}{E(X)^2} \leq \frac{12T_v^2/\tau^2}{(6T_v/\tau)^2} = 1/3.$$

We now argue the time complexity. It holds that $|V_i| \sim Bin(n, p)$. By Lemma 3, each iteration has expected time complexity $\tilde{O}(n^\omega p^\omega)$ because of the assumption $p = 1/\tau \geq 1/n$. There are $O(p^{-2})$ iterations. The total time complexity is thus $\tilde{O}(p^{-2} p^\omega n^\omega) = \tilde{O}\Big(\frac{n^\omega}{\tau^{\omega-2}}\Big)$. ◀

2.3.2 The case $\tau > n$

We repeat that this case only applies to the range where an optimal algorithm is already known (see the beginning of Section 2.3) and we only show this for completeness; the reader may wish to skip this part.

Algorithm 3 Distinguish triangle-heavy vertices from triangle-light vertices.

```
1 M ← ∅
2 for v ∈ V do
3     repeat k = 2n²/τ times
4         Sample u, w ∈ V
5         if uvw ∈ T then
6             M ← M ∪ {v}
7 return M
```

▶ **Lemma 9.** *Algorithm 3 lists any triangle-heavy vertex with probability at least* $2/3$ *and any triangle-light vertex with probability at most* $1/3$*. It runs in time* $\tilde{O}(\frac{n^3}{\tau})$*.*

Proof. Consider the probability that a triangle-light vertex v is reported. Similarly to the case $\tau \leq n$, it holds by the union bound over all incident triangles and over the k iterations that this probability is at most $\tau/20 \cdot k \cdot 1/n^2 \leq 1/3$.

Consider now the probability a triangle-heavy vertex v is reported. In each iteration, the probability that we find an incident triangle is $T_v/n^2 \geq \tau/n^2$. The probability that we report v is now at least $1 - (1 - \tau/n^2)^{2n^2/\tau} \geq 1 - 1/e^2 \geq 2/3$. ◀

2.4 Recursive algorithm

We now take the subroutines presented above and put them together into one recursive algorithm. Our proof of correctness works by induction on the depth of recursion. For this reason, the statement assumes the input graph can be random, as we use the inductive hypothesis on a sampled subgraph of the input graph. This algorithm requires advice $\tilde{T}$ such that $\tilde{T} \leq E(T)$ (note that T is a random variable as the input graph may be random); we will remove the need for advice later. As we already mentioned, for the advice removal, we will need to prove a bound on the expectation of the estimate.

We show an algorithm that gives additive approximation. The reason is that this is better suited for performing recursive calls. Specifically, the time complexity of getting relative approximation is worse when T is small. If it happened that most triangles contained a triangle-heavy vertex, we would recurse on a subgraph with few triangles, making it more costly to get the relative approximation. We then show how this algorithm can be turned into an algorithm that gives relative approximation.

Note that line 11 can be efficiently implemented as follows: We pick $X \sim Bin(n, p)$, then sample X vertices without replacement, keep only the vertices from $V \setminus V_H$.

▶ **Lemma 10.** *Suppose* G *is a (possibly random) graph. Given parameters* A*,* $\tilde{T}$*, Algorithm 4 returns* $\hat{T}$ *such that* $E(\hat{T}) \leq E(T)$ [7]*. Moreover, if* $\tilde{T} \geq E(T)$*, Algorithm 4 returns* $\hat{T} \in T \pm A$ *with probability at least* $4/5$*. It runs in expected time* $\tilde{O}\big(\frac{n^\omega}{(A^2/\tilde{T})^{\omega-2}} + \frac{\tilde{T}^{4+1/3}Tn^2}{A^6}\big)$*.*

Proof. The structure of the proof is as follows. We first prove that $E(\hat{T}) \leq E(T)$. We prove this by induction. We then go on to prove correctness (that is, that the returned answer is $(1 \pm \epsilon)$-approximation with probability $4/5$), also by induction. In both cases, the induction is on the depth of recursion. Note that, while the instances on which the algorithm

[7] Since the graph can be random, T is a random variable

Algorithm 4 Estimate the number of triangles in G, advice $\tilde{T}$, and error parameter A.

1 $q \leftarrow 1/\log(nT/A)$
2 **if** $A^2 \leq \frac{10^7}{q}\tilde{T}$ **then**
3 **return** *number of triangles in G, computed using matrix multiplication*
4 **if** $\tilde{T} < 1/5$ **then**
5 **return** 0
6 $q \leftarrow 1/\log\left(\frac{nT}{A}\right)$
7 $\tilde{T}' = 20\tilde{T}$
8 $\tau \leftarrow \frac{A^2q^2}{40000\tilde{T}'}$
9 $V_H \leftarrow$ triangle-heavy vertices w.r.t. τ using Corollary 7
10 $\hat{T}_H \leftarrow$ estimate number of triangles with non-empty intersection with V_H using Algorithm 1 with $\epsilon = qA/(2\tilde{T}')$
11 $V' \leftarrow$ sample each vertex from $V \setminus V_H$ independently with probability $p = \frac{20\sqrt{\tau\tilde{T}'}}{qA} = 1/10$
12 $\hat{T}_{V'} \leftarrow$ estimate number of triangles in $G[V']$ by a recursive call with $A = (1-q)p^3A$ and with $\tilde{T} = \tilde{T}p^3$; amplify success probability to 19/20 by taking median of 7 independent executions
13 **return** $\hat{T}_H + \hat{T}_{V'}/p^3$

is called in the recursive calls are random, the recursion tree is deterministic. This means that performing induction on the depth of recursion is formally correct. Finally, we prove the time complexity; in this part, we analyze the whole recursion tree instead of using induction.

Estimate's expected value. We now prove $E(\hat{T}|G) \leq E(T|G)$. We prove this by induction on the depth of recursion. If the condition on line 2 is satisfied, then $\hat{T} = T$ and the inequality thus holds. If the condition on line 4 is satisfied, then $\hat{T} = 0$ and the inequality also holds. Consider now the case when neither of these conditions are satisfied. By the inductive hypothesis, $E(\hat{T}_{V'}|V', G) \leq E(T_{V'}|V', G)$. Moreover, by Lemma 5, $E(T_{V'}/p^3|V_H, G) = E(T(G[V \setminus V_H])|V_H, G)$ and therefore

$$\begin{aligned} E(\hat{T}_{V'}/p^3|V_H, G) &= E(E(\hat{T}_{V'}/p^3|V', G)|V_H, G) \\ &\leq E(E(T_{V'}/p^3|V', G)|V_H, G) \\ &= E(T_{V'}/p^3|V_H, G) = E(T(G[V \setminus V_H])|V_H, G). \end{aligned}$$

It follows from Lemma 6 that $E(\hat{T}_H|V_H, G)$ is an unbiased estimate of the number of triangles in G containing at least one vertex from V_H. This implies that $E(\hat{T}_H|G) = T - T(G[V \setminus V_H])$. We now put this all together:

$$\begin{aligned} E(\hat{T}) &= E\big(E(\hat{T}_H|V_H, G) + E(\hat{T}_{V'}/p^3|V_H, G)\big) \\ &\leq E\big(E(\hat{T}_H|V_H, G)\big) + E\big(E(T(G[V \setminus V_H])|V_H, G)\big) \\ &= E(T - T(G[V \setminus V_H])) + E(T(G[V \setminus V_H])) = E(T). \end{aligned}$$

Note that this bound on the expectation is not using in any way that V_H contains all triangle-heavy vertices and no triangle-light vertices.

Correctness. We first focus on the base case. Consider the case $A^2 \leq \tilde{T}$. In this case, the condition on line 2 is satisfied. The algorithm is then clearly correct.

We now focus on the case when the condition on line 4 is satisfied. Consider one call of the algorithm in the tree of recursion where this is the case. Let k be the depth of this call in the tree. We denote by $G, T, n, A, \tilde{T}, \epsilon$ the respective values in this specific call. We use the subscript $_0$ to denote the respective values in the original call. For example, T_0 would be the number of triangles in the whole graph on which the algorithm was executed.

It holds that $E(T) \leq 1/10^{3k} T_0 \leq 1/10^{3k} \tilde{T}_0 = \tilde{T}$. It holds by the Markov's inequality that $P(5\tilde{T} \geq T) \geq 4/5$. When $\tilde{T} < 1/5$ (this is the condition on line 4), this means that $P(T = 0) \geq 4/5$, meaning that the algorithm is correct in this case and runs in time $O(1)$.

We now prove the inductive step. Consider one recursive call, regardless of the level of recursion. Again, we use $G, T, n, A, \tilde{T}, \epsilon$ to denote the respective values. We assume that $\tilde{T}' \geq T$ in this call. This holds with probability at least 19/20 by the Markov inequality as $\tilde{T}' = 20\tilde{T} \geq 20E(T)$. We now bound the error probability by 3/20. Together with the fact that $P(\tilde{T}' \geq T) \geq 19/20$, this gives a bound on the probability of the call returning an invalid answer of 4/20. Let $\epsilon = \frac{qA}{2\tilde{T}'}$. It then holds $\epsilon \leq \frac{qA}{2T}$. We have $p = 1/10$. We show that $p \geq \max(5\frac{1}{\sqrt[3]{\epsilon^2 T}}, 10\frac{1}{\epsilon\sqrt{T/\tau}})$ (this is the assumption of Lemma 5). It holds $\frac{1}{10} \geq 10\frac{1}{\epsilon\sqrt{T/\tau}}$ from the way we set τ. Because $A^2 \geq \frac{10^7}{q}\tilde{T}$, it also holds

$$5\frac{1}{\sqrt[3]{\epsilon^2 T}} \leq \frac{5}{\sqrt[3]{qA^2/(4\tilde{T}')}} \leq \frac{5}{\sqrt[3]{q\frac{10^7}{q}\tilde{T}/(4\tilde{T}')}} = \frac{1}{10}.$$

By Lemma 5, it holds with probability at least 19/20 that

$$T(G[V']) / p^3 \in (1 \pm \epsilon) T(G[V \setminus V_H]) \subseteq T(G[V \setminus V_H]) \pm qA/2.$$

By the induction hypothesis, it holds that with probability at least 19/20, $\hat{T}_{V'} = T(G[V']) \pm (1-q)p^3 A$. Note that $E(T_{k+1}) = E(E(T_{V'}|G)) \leq E(T/10^3) \leq \tilde{T}/10^3 = \tilde{T}_{k+1}$, where the subscript $_{k+1}$ refers to the respective values at a call on recursion depth $k+1$. This means that the assumption on $\tilde{T}$ is satisfied. Also note that to get success probability 19/20, it does suffice to take the median of 7 independent executions (as can be easily checked, $P(Bin(7, 1/5) \geq g) < 1/20$). Putting this together, with probability at least 18/20, it holds that

$$\hat{T}_{V'}/p^3 \in T(G[V'])/p^3 \pm (1-q)A \subseteq T(G[V \setminus V_H]) \pm (1 - q/2)A.$$

Moreover, with probability at least 19/20, $\hat{T}_H = T(V_H) \pm \epsilon T(V_H) \subseteq T(V_H) \pm qA/2$ by Lemma 6 where the inclusion holds because we have set $\epsilon = qA/(2\tilde{T}') \leq qA/(2T) \leq qA/(2T(V_H))$. By the union bound, with probability at least 17/20, $|\hat{T}_H - T(V_H)| \leq qA/2$ and $|T(V')/p^3 - T(V \setminus V_H)| \leq (1 - q/2)A$, in which case $\hat{T}_H + \hat{T}_L/p^3 = T \pm A$ – the resulting answer is correct. Including the probability that $\tilde{T}' \leq T$ (we assumed in the analysis that this is not the case), we get that the answer is correct with probability at least 16/20.

Time complexity. We again consider one recursive call and use $G, T, n, A, \tilde{T}, \epsilon$ to denote the respective values in this one call, and the subscript $_0$ is used to refer to the respective values in the original call. We first show that we may ignore in the analysis the time spent on line 3. Counting triangles exactly by using fast matrix multiplication takes $\tilde{O}(n^\omega) \subseteq \tilde{O}\big(\frac{n^\omega}{\tau^{\omega-2}} + \frac{\tilde{T}^4 T n^2}{A^6}\big)$ time (the inclusion holds thanks to the assumption $A^2 \leq \tilde{T}$ which is satisfied on line 3). This is equal to the bound on the time complexity of the call that we use below. In the rest of the proof, we therefore ignore the time spent on line 3.

The time complexity of line 9 is $\tilde{O}(\frac{n^\omega}{\tau^{\omega-2}})$ and complexity of line 10 is $\tilde{O}(\frac{Tn^2}{\epsilon^2\tau^2}) = \tilde{O}(\frac{\tilde{T}'^4Tn^2}{q^2A^6})$ by Corollary 7 and Lemma 6, respectively (note that ϵ is defined on line 10). Since $p = 1/10$, the number of vertices in a depth k recursive call is stochastically dominated by $Bin(n, 1/10^k)$ [8] while the value of A is in a depth k recursive call multiplied by factor of $\big((1-q)p^3\big)^k = \big((1-q)/10^3\big)^k$ and $\tilde{T}' = 1/10^{3k}\tilde{T}'_0$. In each recursive call, we make 7 recursive calls for the probability amplification. The number of calls at recursion depth k is then 7^k. The expected time spent in the depth k recursive calls on line 9 is

$$E\Big(7^k\frac{n^\omega\log n}{\tau^{\omega-2}}\Big) \le \tilde{O}\Big(7^k\frac{n_0^\omega/10^{\omega k}}{(q^2\epsilon_0^2(1-q)^k * \tilde{T}'_0/10^{3k})^{\omega-2}}\Big) = \tilde{O}\Big(q^{4-2\omega}\Big(\frac{7\cdot 10^{3(\omega-2)}}{(1-q)^{\omega-2}10^\omega}\Big)^k\cdot\frac{n_0^\omega}{\epsilon_0^2T_0^{\omega-2}}\Big).$$

This decreases exponentially since $\omega \le 2.5$ [9]and, therefore, the time on line 9 is dominated by the first call. (By changing constants in the algorithm, this argument can be made to work even for asymptotically slower sub-cubic matrix multiplication algorithms.) However, the time spent on line 10 is

$$E\Big(7^k\frac{\tilde{T}'^4Tn^2}{q^2A^6}\Big) = 7^k\frac{\tilde{T}'^4E(T)n^2}{q^2A^6} \le 7^k\frac{\tilde{T}'^4_0/10^{4\cdot3k}\cdot T_0/10^{3k}\cdot n^2/10^{2k}}{q^2A_0^6(1-q)^{6k}/10^{6\cdot3k}} = \Big(\frac{10}{1-q}\Big)^k\cdot\frac{\tilde{T}'^4_0T_0n_0^2}{q^2A_0^6}$$

which increases at an exponential rate. We now upper bound the number of recursive calls (in other words, we upper bound k). In each subsequent recursive call, $\tilde{T}$ decreases by a factor of $1/p^3 = 10^3$. This means that after $\log_{1000}\tilde{T} + O(1)$ recursive calls, it will hold that $\tilde{T} \le 1/20$, in which case the algorithm finishes on line 4 in time $O(1)$ with no additional recursive calls. Therefore $k \le \log_{1000}\tilde{T} + O(1)$. Since the amount of work at each level of recursion increases exponentially, the work is dominated by the last level of recursion. Thanks to our bound on k, the amount of time spent on line 8 is

$$\begin{aligned}O\Big(\Big(\frac{10}{1-q}\Big)^{\log_{1000}\tilde{T}_0+O(1)}\cdot\frac{\tilde{T}_0^4T_0n^2}{q^2A_0^6}\Big) &= O\Big((1-q)^{-\log_{1000}\tilde{T}_0}\frac{\tilde{T}_0^{4+1/3}T_0n^2}{q^2A_0^6}\Big)\\ &= O\Big(\frac{\tilde{T}_0^{4+1/3+\log_{1000}(1/(1-q))}T_0n^2}{q^2A_0^6}\Big)\end{aligned}$$

and substituting $q = 1/\log(nT/A)$, we get that the total time complexity is

$$\tilde{O}\left(\frac{q^{4-2\omega}n^\omega}{\epsilon^2T^{\omega-2}} + \frac{\tilde{T}^{4+1/3+\log_{1000}(1/(1-q))}Tn^2}{q^2A^6}\right) = \tilde{O}\left(\frac{n^\omega}{\epsilon^2T^{\omega-2}} + \frac{\tilde{T}^{4+1/3}Tn^2}{A^6}\right). \quad ◀$$

We now get the following claim, which guarantees relative approximation, unlike Lemma 10.

▶ **Proposition 11.** *There is an algorithm that, given $\epsilon > 0$ and $\tilde{T}$ returns an estimate $\hat{T}$ of T, such that $E(\hat{T}) \le E(T)$ of T and if, moreover, $T \le \tilde{T} \le 2T$, it holds $\hat{T} \in (1\pm\epsilon)T$ with probability at least $4/5$. It runs in expected time $\tilde{O}(\frac{n^\omega}{(\epsilon^2T)^{\omega-2}} + \frac{n^2}{\epsilon^6T^{2/3}})$.*

Proof. By substituting $A = \epsilon\tilde{T}/2$, it is sufficient to calculate approximation with additive error A. This can be done by Lemma 10, giving us the desired bounds. ◀

[8] This is the case as in each level of recursion, we keep each triangle-light vertex with probability $p = 1/10$ while removing all other vertices with high probability.

[9] The argument works for $\omega > 2.5$ if some constants in the algorithms are modified.

2.5 Removing advice

We remove the dependency on $\tilde{T}$ by performing a geometric search. This method has been used many times before, for example in [9, 14, 1, 10, 4, 11]. In our case, it is slightly more complicated in that our algorithm requires both a lower and an upper bound on T. For this reason, we describe the method in completeness. We also prove a variant that gives a guarantee of absolute approximation. We will need this for triangle counting in sparse graphs.

▶ **Theorem 12.** *There is an algorithm that, given $\epsilon > 0$ (respectively A) returns $\hat{T} \in (1 \pm \epsilon)T$ (respectively $\hat{T} \in T \pm A$) with probability at least $2/3$. It runs in expected time $\tilde{O}(\frac{n^\omega}{(\epsilon^2 T)^{\omega-2}} + \frac{n^2}{\epsilon^6 T^{2/3}})$ (respectively $\tilde{O}(\frac{n^\omega}{(A^2/\tilde{T})^{\omega-2}} + \frac{T^{5+1/3}n^2}{A^6})$).*

Proof. We present the argument for the relative approximation. The exact same argument gives the absolute approximation by using Lemma 10 in place of Proposition 11.

We start with $\tilde{T} = \binom{n}{3}$ and, in each step, divide $\tilde{T}$ by a factor of two. When it holds that $\hat{T} \geq \tilde{T}$ where $\hat{T}$ is the estimate returned by the algorithm, we return $\hat{T}$ as the final estimate of T.

We now argue correctness. The estimate $\hat{T}$ given by the algorithm is always non-negative and it holds $E(\hat{T}) \leq T$. By Markov's inequality, $P(\hat{T} \geq 2T) \leq 1/2$, regardless of the choice of $\tilde{T}$. We amplify this probability to $\Theta(1/\log n)$ by taking the median of $\Theta(\log\log n)$ independent executions. When $\hat{T} \geq \tilde{T}$ (this is when we return $\hat{T}$ as the final estimate), it holds with probability at least $1 - O(1/\log n)$ that $\tilde{T} \leq 2T$ as otherwise, it would be the case that $\hat{T} \geq \tilde{T} \geq 2T$ which holds with probability $O(1/\log n)$. Conditioned on $\tilde{T} \leq 2T$, the algorithm gives a $(1 \pm \epsilon)$-approximate estimate (respectively additive $\pm A$ estimate) by Proposition 11 (respectively Lemma 10). By the union bound, the failure probability is then arbitrarily small if we make the constants in Θ sufficiently small.

We now argue the time complexity. The probability amplification only increases the time complexity by a $O(\log\log n)$ factor. When $\tilde{T} \leq T$, with probability $1 - O(1/\log n)$ we return the estimate and quit. We now consider the calls of the algorithm with $\tilde{T} \leq T$. The time complexity increases exponentially (as T is decreasing exponentially) while the probability of performing a call is decreasing at a rate faster than exponential. Therefore, the time complexity is dominated by the first call when $\tilde{T} \leq T$. This time complexity is as claimed by Proposition 11 (respectively Lemma 10). ◀

2.6 Optimality

The lower bound of Eden et. al. [9] implies that our algorithm is in certain sense optimal. Specifically, the exponent in the dependency on T cannot be improved without worsening the dependency on n, as long as the exponent of T is constant. That still leaves open the possibility of improving the dependency on n and getting an algorithm with a non-constant exponent of T.

▶ **Proposition 13.** *Suppose there is an algorithm which runs in time $\tilde{O}\left(\frac{n^\omega}{T^\delta}\right)$ for some constant δ, and returns a constant-factor approximation of T with probability at least $2/3$. Then $\delta \leq \omega - 2$.*

Proof. Suppose $\delta > \omega - 2$. Then, for $T = \Theta(n)$, the algorithm runs in time $o(n)$. This is in contradiction to the lower bound of from [9] which states that any such algorithm has to run in time $\Omega(n)$. ◀

3 Triangle counting in sparse graphs

We now show how Algorithm 4 can be used to get an efficient algorithm for counting triangles in a sparse graph. The algorithm finds all vertices with degree at least some parameter θ, then it uses Algorithm 4 to count triangles in the subgraph induced by these vertices and uses sampling in the rest of the graph. We set $\theta = \frac{m^{(\omega-1)/(\omega+1)}}{T^{(\omega-3)/(\omega+1)}\epsilon^{(2\omega-6)/(\omega+1)}}$.

For sake of simplicity, we show an algorithm with an additional $O(m)$ term in its complexity. As the time complexity of the algorithm from [4] is lower in the sublinear regime than what we claim, we may obtain an algorithm with the desired complexity by running the two algorithms in parallel and terminating when one of them finishes. In this section, we use a total order $\prec$ defined as $u \prec v$ if $d(u) \leq d(v)$ and ties broken arbitrarily in a consistent way.

Algorithm 5 Estimate the number of triangles in G and advice $\tilde{T}$.

1 $\theta \leftarrow \frac{m^{(\omega-1)/(\omega+1)}}{\tilde{T}^{(\omega-3)/(\omega+1)}\epsilon^{(2\omega-6)/(\omega+1)}}$
2 $V_H \leftarrow$ find all vertices v with $d(v) \geq \theta$
3 $\hat{T}_H \leftarrow$ count triangles in $G[V_H]$ using Algorithm 4 with error $\pm\epsilon\tilde{T}/2$; amplify success probability to $\frac{5}{6}$
4 $M \leftarrow 0$
5 **repeat** $k = 12\frac{\theta m}{\epsilon^2\tilde{T}}$ **times**
6 $\quad e \leftarrow$ pick an edge uniformly at random
7 $\quad uv \leftarrow e$, such that $u \succ v$
8 $\quad w \leftarrow$ pick a random neighbor of v
9 $\quad$ **if** $v \notin V_H$ *and* $w \succ v$ *and* $uvw \in T$ **then**
10 $\quad\quad M \leftarrow M + d(v)$
11 $\hat{T}_L = \frac{m}{2k}M$
12 **return** $\hat{T}_H + \hat{T}_L$

We now prove the main theorem for triangle counting in sparse graphs. The last part in the following theorem is there to enable us to remove advice like we did in Section 2.5.

▶ **Lemma 14.** *Let us have a graph G. Given parameters $1 > \epsilon > 0$ and advice $\tilde{T}$, Algorithm 5 returns $\hat{T}$ such that $P(\hat{T} \geq 2T) \leq 2/3$. Moreover, if $\tilde{T} \leq T$, Algorithm 5 returns $\hat{T} \in (1 \pm \epsilon)T$ with probability at least $4/5$. It runs in expected time*

$$\tilde{O}\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}} + \frac{m^{4/(\omega+1)}}{\epsilon^{2(9+\omega)/(\omega+1)}T^{4(5-\omega)/(3(\omega+1))}} + m\Big).$$

With the current best bound on ω, we get running time of $\tilde{O}(\frac{m^{1.408}}{\epsilon^{1.628}T^{0.814}} + \frac{m^{1.186}}{\epsilon^{6.744}T^{1.038}})$. For constant ϵ, the second term is significantly smaller than the first one.

Proof. Let T_L be the set of triangles that are not contained in $G[V_H]$ – or equivalently, that have their $\prec$-minimal vertex outside of V_H – and, by an abuse of notation, also the number of such triangles. Let T_H be the number of triangles contained in $G[V_H]$.

Let X_i be the increment in M in the i-th iteration of the loop on line line 5. We now calculate the expectation of M after all k iterations. Let us fix a triangle $\triangle \in T$ and define $X_i' = X_i$ if $uvw = \triangle$ in the i-th iteration and $X_i' = 0$ otherwise (note that u, v, w are defined in the algorithm). If a is the vertex of $\triangle$ minimal w.r.t. $\prec$, it now holds that

$E(X_i') = d(a)P(X_i' = d(a))$. It holds that $X_i' = d(a)$ if and only if $v = a$ and $uvw = \triangle$. For this to happen, the algorithm has to sample in the i-th iteration the edge uv or wv. This happens with probability $2/m$. Then, it has to be the case that the random neighbor sampled on line 7 is the single vertex of $\triangle$ not in e. This happens with probability $1/d(a)$. This gives us that $E(X_i') = \frac{d(a)}{md(a)} = 2/m$. By summing over all triangles in T_L and by linearity of expectation, it holds that $E(X_i) = 2T_L/m$. Therefore, the estimate $\hat{T}_L = \frac{m}{2k}M = \frac{m}{2k}\sum_{i=1}^{k} X_i$ is an unbiased estimate of T_L.

We now bound the variance. We use the inequality[10] $Var(X) \leq \sup(X)E(X)$ to get $Var(X_i) \leq \theta E(X_i) = 2\theta T_L/m$. Therefore $Var(mX_i/2) = \frac{1}{2}\theta m T_L$. Taking average of $12\frac{\theta m}{\epsilon^2 \tilde{T}}$ independent trials, we get variance $\frac{1}{24}\epsilon^2 \tilde{T} T_L \leq \frac{1}{24}\epsilon^2 T^2$. By the Chebyshev's inequality, it therefore holds that

$$P(|\hat{T}_L - T_L| \geq \epsilon T/2) \leq \frac{Var(\hat{T}_L)}{(\epsilon T/2)^2} \leq 1/6.$$

At the same time, by Lemma 10, it holds that $\hat{T}_H \in T_H \pm \epsilon\tilde{T}/2 \subseteq T_H \pm \epsilon T/2$ with probability at least $5/6$ (note that we have amplified the success probability). By the union bound, with probability at least $2/3$, it holds that both $|\hat{T}_L - T_L| \leq \epsilon T/2$ and $|\hat{T}_H - T_H| \leq \epsilon T/2$. Therefore, the algorithm returns a $(1 \pm \epsilon)$-approximate answer with probability at least $2/3$.

We now argue the time complexity. We spend $O(n)$ time on line 2. There are no more than m/θ vertices with degree at least θ. Triangles in $G[V_H]$ are therefore counted, by Theorem 12, in time

$$\tilde{O}\Big(\frac{(m/\theta)^\omega}{(A^2/\tilde{T})^{\omega-2}} + \frac{T_H^{5+1/3}(m/\theta)^2}{A^6}\Big) \leq \tilde{O}\Big(\frac{(m/\theta)^\omega}{(\epsilon^2\tilde{T})^{\omega-2}} + \frac{(m/\theta)^2}{\epsilon^6 T^{2/3}}\Big)$$
$$= \tilde{O}\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}} + \frac{m^{4/(\omega+1)}}{\epsilon^{2(\omega+1)/(\omega+1)}T^{(7-\omega)/(\omega+1)}}\Big).$$

The time spent in each iteration of the loop on line 5 is $O(1)$. The total time spent in the loop is therefore

$$O\Big(\frac{\theta m}{A^2/\tilde{T}}\Big) = O\Big(\frac{\theta m}{\epsilon^2\tilde{T}}\Big) = O\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}}\Big).$$

Putting these two time complexities together, we get the desired running time.

We now prove that $P(\hat{T} \geq 2T) \leq 2/3$ [11]. Since $\epsilon < 1$, it holds that $P(\hat{T}_H \geq 2T_H) \leq P(\hat{T}_H \geq (1+\epsilon)T_H) \leq 1/6$ as we have amplified the success probability of Algorithm 4 to $\frac{5}{6}$. Moreover, as we have shown, $E(\hat{T}_L) = T_L$ and, therefore, $P(\hat{T}_L \geq 2T_L) \leq 1/2$ ◀

By the same argument as in Section 2.5, we may remove the need for advice [12]. We run the algorithm in parallel with the algorithm from [4], resulting in an algorithm with its time complexity being the minimum of the complexities of the two respective algorithms. This gives us the following claim.

---

[10] $\sup(X)$ is the smallest x such that $P(X > x) = 0$.

[11] We cannot use the Markov's inequality in a straightforward way. The reason is that it is not clear how to bound the expectation of the estimate coming from Theorem 12. While we do have a bound on the expectation of the estimate given by, Algorithm 4 it is not clear a bound of this type carries over when we perform the advice removal.

[12] Instead of directly using the Markov inequality in the argument, we may use the last part of Lemma 14 to get the same guarantee.

▶ **Theorem 15.** *There is an algorithm that returns a* $(1 \pm \epsilon)$*-approximation of the number of triangles with probability at least* $2/3$*. It runs in time*

$$\tilde{O}\Big(\frac{m^{2\omega/(\omega+1)}}{\epsilon^{4(\omega-1)/(\omega+1)}\tilde{T}^{2(\omega-1)/(\omega+1)}} + \frac{m^{4/(\omega+1)}}{\epsilon^{2(9+\omega)/(\omega+1)}T^{4(5-\omega)/(3(\omega+1))}}\Big).$$

3.1 Optimality

Like in the case of dense graphs, we show that our algorithm is in certain sense optimal.

▶ **Proposition 16.** *Suppose there is an algorithm that uses both random vertex and random edge queries, which runs in time* $\tilde{O}(\frac{m^{2\omega/(\omega+1)}}{T^\delta})$ *for some constant* δ*, and returns a constant-factor approximation of* T *with probability at least* $2/3$*. Then* $\delta \leq 2(\omega-1)/(\omega+1)$*.*

Proof. Suppose $\delta > 2(\omega-1)/(\omega+1)$. Then, for $T = \Theta(\sqrt{m})$, the algorithm runs in time $o(m)$. This is in contradiction to the lower bound from [9] which states that any such algorithms has to run in time $\Omega(m)$. ◀

References

1. Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling. *Algorithmica*, 80(2):668–697, February 2018. doi:10.1007/s00453-017-0287-3.
2. Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 522–539. Society for Industrial and Applied Mathematics, January 2021. doi:10.1137/1.9781611976465.32.
3. N. Alon, R. Yuster, and U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica (New York)*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
4. Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. *Leibniz International Proceedings in Informatics, LIPIcs*, 124, November 2019. doi:10.4230/LIPIcs.ITCS.2019.6.
5. Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016. URL: http://barabasi.com/networksciencebook/.
6. Amartya Shankha Biswas, T. Eden, Q. Liu, Slobodan Mitrović, and R. Rubinfeld. Parallel algorithms for small subgraph counting. *ArXiv*, abs/2002.08299, 2020. arXiv:2002.08299.
7. N. Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.
8. Fan Chung and Linyuan Lu. *Complex Graphs and Networks (Cbms Regional Conference Series in Mathematics)*. American Mathematical Society, USA, 2006.
9. Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017. doi:10.1137/15M1054389.
10. Talya Eden, Dana Ron, and C. Seshadhri. Sublinear Time Estimation of Degree Distribution Moments: The Degeneracy Connection. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.7.
11. Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 722–734, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188810.

12 Talya Eden and Will Rosenbaum. Lower Bounds for Approximating Graph Parameters via Communication Complexity. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.11.

13 Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

14 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. In Josep Díaz, Klaus Jansen, José D. P. Rolim, and Uri Zwick, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 363–374, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

15 Paul W Holland and Samuel Leinhardt. A method for detecting structure in sociometric data. In *Social networks*, pages 411–432. Elsevier, 1977.

16 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 1–10, New York, NY, USA, 1977. Association for Computing Machinery. doi:10.1145/800105.803390.

17 John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1778–1797, USA, 2017. Society for Industrial and Applied Mathematics.

18 Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012. doi:10.1080/15427951.2012.625260.

19 Tong Ihn Lee, Nicola J Rinaldi, François Robert, Duncan T Odom, Ziv Bar-Joseph, Georg K Gerber, Nancy M Hannett, Christopher T Harbison, Craig M Thompson, Itamar Simon, et al. Transcriptional regulatory networks in saccharomyces cerevisiae. *science*, 298(5594):799–804, 2002.

20 Richard J. Lipton, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116(1):195–226, 1993. doi:10.1016/0304-3975(93)90224-H.

21 Duncan T Odom, Nora Zizlsperger, D Benjamin Gordon, George W Bell, Nicola J Rinaldi, Heather L Murray, Tom L Volkert, Jorg Schreiber, P Alexander Rolfe, David K Gifford, et al. Control of pancreas and liver gene expression by hnf transcription factors. *Science*, 303(5662):1378–1381, 2004.

22 Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Inf. Process. Lett.*, 112(7):277–281, March 2012. doi:10.1016/j.ipl.2011.12.007.

23 Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. Doulion: Counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 837–846, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1557019.1557111.

24 Osamu Watanabe. Sequential sampling techniques for algorithmic learning theory. *Theoretical Computer Science*, 348(1):3–14, 2005. Algorithmic Learning Theory (ALT 2000). doi:10.1016/j.tcs.2005.09.003.

25 Andreas Wimmer and Kevin Lewis. Beyond and below racial homophily: Erg models of a friendship network documented on facebook. *American Journal of Sociology*, 116(2):583–642, 2010. URL: http://www.jstor.org/stable/10.1086/653658.

Polynomial-Time Approximation of Zero-Free Partition Functions

Penghui Yao
State Key Laboratory for Novel Software Technology, Nanjing University, China

Yitong Yin
State Key Laboratory for Novel Software Technology, Nanjing University, China

Xinyuan Zhang
State Key Laboratory for Novel Software Technology, Nanjing University, China

Abstract

Zero-free based algorithms are a major technique for deterministic approximate counting. In Barvinok's original framework [4], by calculating truncated Taylor expansions, a quasi-polynomial time algorithm was given for estimating zero-free partition functions. Patel and Regts [29] later gave a refinement of Barvinok's framework, which gave a polynomial-time algorithm for a class of zero-free graph polynomials that can be expressed as counting induced subgraphs in bounded-degree graphs.

In this paper, we give a polynomial-time algorithm for estimating classical and quantum partition functions specified by local Hamiltonians with bounded maximum degree, assuming a zero-free property for the temperature. Consequently, when the inverse temperature is close enough to zero by a constant gap, we have a polynomial-time approximation algorithm for all such partition functions. Our result is based on a new abstract framework that extends and generalizes the approach of Patel and Regts.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases partition function, zero-freeness, local Hamiltonian

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.108

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2201.12772

Funding This research was supported by National Natural Science Foundation of China (Grant No. 61972191) and the Program for Innovative Talents and Entrepreneur in Jiangsu and Anhui Initiative in Quantum Information Technologies Grant No. AHY150100.

1 Introduction

Let $\Omega = [q]^V$ be a finite space of configurations, where V is a set of n variables. Let $H_1, \ldots, H_m$ be a collection of local constraints, where each $H_j : \Omega \to \mathbb{C}$ is independent of all but a small subset of variables, and let $H = \sum_{j=1}^{m} H_j$. The *partition function* of the given system is defined by

$$Z_H(\beta) = \sum_{\sigma \in \Omega} \exp(-\beta \cdot H(\sigma)), \tag{1}$$

where the parameter β is usually called the *inverse temperature*.

The computational complexity of partition functions is one of the central topics in theoretical computer science, which has been found wide applications in computational counting, combinatorics, and statistical physics. To date, numerous algorithms as well as hardnesses of approximation for the partition functions of various systems have been established, to list a few [20, 32, 15, 21, 39, 37, 35, 36, 25, 34, 13, 26, 6, 10, 7]. The most important question here is, what property captures the approximability of partition functions.


It is widely believed that for various classes of partition functions of interests, the hardness of approximation is captured by the locus of complex zeros. The study of the locus of complex zeros has a rich history in statistical physics, for example, in the famous Lee-Yang theorem [24]. In computer science, the absence of complex zeros may imply efficient approximation algorithms for partition functions [4, 29, 31, 19, 27, 26, 30, 12, 17, 16, 18, 33, 9, 11]. This line of research was initiated by Barvinok's pioneering works [1, 2, 3, 4, 5], which used truncated Taylor expansions to approximate non-vanishing polynomials and established quasi-polynomial time approximations of partition functions with no complex zeros within a region. Later in a seminal work of Patel and Regts [29], this quasi-polynomial running time was improved to polynomial time for a class of graph polynomials which can be expressed as induced subgraph sums in graphs of constant maximum degree. And this polynomial-time framework was further extended by Liu, Sinclair and Srivastava [27] to hypergraph 2-spin systems with no complex-zeros for the external field.

For the quantum version, several (classical) algorithms have been proposed, including [23, 18, 28], to estimate the quantum generalization of the partition function defined as in (1) where H is the *Hamiltonian*. Yet, an important question remains to answer is the polynomial-time approximability of the quantum or classic partition function in the form of (1) assuming its zero-freeness.

1.1 Our results

We show the polynomial-time approximability of partition functions assuming zero-freeness, for both classical and quantum partition functions.

Let V be a set of n *sites* (also called *vertices* or *particles*). Let $q \geq 2$ be an integer. Throughout the paper, we assume that each site $u \in V$ is associated with a q-dimensional Hilbert space $\mathcal{V}_u$, and let $\mathcal{V} = \bigotimes_{u \in V} \mathcal{V}_u$. A *Hamiltonian* H is a Hermitian matrix in $\mathcal{V}$. The support of a Hamiltonian H, denoted by $\mathrm{supp}(H)$, is the minimal set of sites on which H acts non-trivially. Given a Hamiltonian H, $\exp(H)$ is defined by $\exp(H) = \sum_{\ell=1}^{\infty} \frac{1}{\ell!} H^{\ell}$, and the *partition function* $Z_H : \mathbb{C} \to \mathbb{C}$ induced by H is defined as follows:

$$\forall \beta \in \mathbb{C}, \quad Z_H(\beta) \triangleq \mathbf{Tr}\left[\exp(-\beta H)\right]. \tag{2}$$

We are interested in partition functions induced by local Hamiltonians with bounded maximum degrees.

▶ **Definition 1** (local Hamiltonian). *A Hamiltonian $H \in \mathcal{V}$ is said to be* k-local *if H can be expressed as*

$$H = \sum_{j=1}^{m} H_j,$$

where each H_j acts non-trivially on at most k sites, i.e. $|\mathrm{supp}(H_j)| \leq k$. A Hamiltonian $H \in \mathcal{V}$ called a (k, d)-Hamiltonian *if H is k-local and for every $v \in V$, $\deg(v) \triangleq |\{j \mid v \in \mathrm{supp}(H_j)\}| \leq d$.*

Notice that if all H_j's are diagonal, then H is diagonal as well. The quantum partition function $Z_H(\beta)$ then degenerates to the classical partition function defined in (1). Indeed, in such diagonal case we have

$$Z_H(\beta) = \sum_{\sigma \in [q]^V} \exp(-\beta \cdot H(\sigma)),$$

where $H(\sigma) = \sum_{j=1}^{m} H_j(\sigma)$ and $H_j(\sigma)$ represents the σ-th diagonal entry of H_j. Since each H_j is diagonal and acts non-trivially on subset $\mathrm{supp}(H_j)$ of at most k sites, the value of $H_j(\sigma)$ is determined by the variables in $\mathrm{supp}(H_j)$. Hence the $Z_H(\beta)$ above is precisely the classical partition function defined in (1).

The quantum partition functions encode rich information about quantum systems, e.g. the free energy and ground state energy. Meanwhile, the non-diagonal property, especially the non-commutativity of multiplication for non-diagonal matrices, imposes great challenges for the computation of partition functions.

We prove the following zero-free based approximability of quantum partition functions.

▶ **Theorem 2** (Theorem 15, informal). *Let $\Omega \subseteq \mathbb{C}$ be a "well-shaped" complex region (formalized by Definition 12) and $k, d \geq 1$ be constants. There is a deterministic algorithm which takes a (k, d)-Hamiltonian H on n sites and a β from interior of Ω as input, and outputs an estimation of the quantum partition function $Z_H(\beta)$ in polynomial time in n, if Z_H satisfies the zero-free property such that $|\log Z_H| \leq \mathrm{poly}(n)$ on Ω.*

The formal statement (Theorem 15) is more general: it further takes into account the measurement of the quantum system. Such generalization may encode broader classes of partition functions, e.g. the ones with external fields, and also enable sampling from Gibbs state. Following a recent major advance for quantum zero-freeness of Harrow et al. [18], we give concrete applications (in Section 5), namely, polynomial-time algorithms for approximating the quantum partition function (Theorem 21) and sampling from the Gibbs state (Theorem 25) in a high-temperature regime (where β is close to zero by a constant gap), improving the quasi-polynomial-time algorithms in [18]. A polynomial-time approximation of the quantum partition functions in a slightly bigger high-temperature regime was obtained in [28] using the cluster expansion technique [19], by transforming the quantum partition function to a polymer model and showing the convergence of the cluster expansion assuming high temperature.

We prove polynomial-time approximability of the quantum partition function directly from a black-box property of zero-freeness, without further restricting the parameters of the model. Moreover, our result is proved in a new abstract framework, namely, functions specified by abstract neighborhood structures called *dependency graphs*. We prove the following general result.

▶ **Theorem 3** (Theorem 14, informal). *Suppose that functions $\{f_G\}$ specified by dependency graphs G satisfies certain boundedness property of its Taylor coefficients (formalized in Definition 10). Let $\Omega \subseteq \mathbb{C}$ be a "well-shaped" complex region. There is a deterministic algorithm which takes a dependency graph G of $O(1)$ max-degree and x from the interior of Ω as input, and outputs an estimation of $f_G(x)$ in polynomial time in size n of G, if $f_G(0)$ is easy to compute and f_G satisfies the zero-free property such that $|\log f_G| \leq \mathrm{poly}(n)$ on Ω.*

The abstract framework is described in Definition 10. As verified in Section 3, our framework subsumes previous polynomial-time frameworks for zero-free based algorithms ([29] and [27]) as special cases, and more importantly, it extends the previous frameworks to become compatible with infinite-degree polynomials and non-commutative systems, which are crucial for quantum partition functions.

2 Preliminaries

2.1 Local Hamiltonians

Given a Hamiltonian H in $\mathcal{V}$, we use $\mathrm{supp}(H)$ to denote the *support* of H, the minimal set of sites on which H acts non-trivially. Formally, if S is the support of H, then S is the minimal subset of V satisfying that $H = H_S \otimes I_{V\setminus S}$, where H_S is a Hamiltonian in the space $\bigotimes_{v\in S}\mathcal{V}_v$ and $I_{V\setminus S}$ is the identity matrix in the space $\bigotimes_{v\in V\setminus S}\mathcal{V}_v$. Readers may refer to [14, 22] for a thorough treatment.

2.2 Basic facts about complex functions

A complex-valued function $f : \Omega \to \mathbb{C}$ defined on a complex domain $\Omega \subseteq \mathbb{C}$ is called *holomorphic* if for every point $z \in \Omega$, the complex derivative exists in a neighborhood of z. A holomorphic function $f : \Omega \to \mathbb{C}$ is infinitely differentiable and equals locally to its Taylor series. A *biholomorphic* function is a bijective holomorphic function whose inverse is also holomorphic Furthermore, a function $f : \mathbb{C} \to \mathbb{C}$ is called an *entire* function if it is holomorphic on $\mathbb{C}$. A region $\Omega \subseteq \mathbb{C}$ is simply connected if $\overline{\mathbb{C}} \setminus \Omega$ is connected, where $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$ denotes the extended complex plane.

The logarithm of a complex-valued function f, denoted by $g = \log f$, is a function such that $f(z) = \mathrm{e}^{g(z)}$. For holomorphic function $f : \Omega \to \mathbb{C} \setminus \{0\}$ on simply connected region $\Omega \subseteq \mathbb{C}$, such $\log f$ always exists (see e.g. [38]). Specifically, for an arbitrarily fixed pair $z_0, c_0 \in \mathbb{C}$ satisfying that $f(z_0) = \mathrm{e}^{c_0}$, we have

$$\forall z \in \Omega, \quad \log f(z) = \int_P \frac{f'(w)}{f(w)}\, dw \; + c_0, \tag{3}$$

where P is an arbitrary path in Ω connecting z and z_0. Throughout the paper, we mainly deal with such holomorphic f on simply connected Ω that $0 \in \Omega$ and $f(0) \in \mathbb{R}^+$. For such case, the definition of $\log f$ is uniquely determined by $z_0 = 0$ and the real $c_0 = \ln(f(0))$.

2.3 Approximation of non-vanishing function

We now recap the polynomial interpolation technique of Barvinok [4] to estimate values of non-vanishing holomorphic functions.

For $b \in \mathbb{R}^+$, we use $\mathbb{D}_b$ to denote the complex disc of radius b centered at the origin. Formally,

$$\mathbb{D}_b = \{z \in \mathbb{C} | \, |z| < b\}\,.$$

In particular, let $\mathbb{D} = \mathbb{D}_1$ denote the unit disc.

For $\beta \in \mathbb{C}$ and $\delta \in \mathbb{R}^+$, we use $S_{\beta,\delta}$ to denote δ-strip of interval $[0, \beta] = \{t\beta \mid t \in [0, 1]\}$. Formally,

$$S_{\beta,\delta} = \{z \in \mathbb{C} \mid \mathrm{dist}(z, [0, \beta]) < \delta\}\,.$$

where $\mathrm{dist}(\cdot, \cdot)$ denotes Euclidean distance. It is clear that both $\mathbb{D}_b$ and $S_{\beta,\delta}$ are simply connected.

The following is the key property of zero-freeness for complex-valued functions.

▶ **Definition 4** (zero-freeness). *Let $M > 0$ be finite positive real. A holomorphic function f on a simply connected region $\Omega \subseteq \mathbb{C}$ is* M-zero-free *on Ω if $|\log f(z)| \le M$ for all $z \in \Omega$.*

Notice that the zero-freeness of f on Ω implies that f is non-vanishing on the same region. A definition of $\log f$ is assumed in the context when this concept is used.

For any polynomial $p \in \mathbb{C}[z]$ that does not vanish on $\mathbb{D}$, the polynomial p automatically exhibits the above zero-freeness property with a bounded gap on $\mathbb{D}_b$ for any $b \in (0,1)$.

▶ **Lemma 5.** *Let $p \in \mathbb{C}[z]$ be a polynomial of degree d, and let $b \in (0,1)$. If $p(z) \neq 0$ for all $z \in \mathbb{D}$, then p is M-zero-free on $\mathbb{D}_b$ for $M = d \ln \frac{1}{1-b} + |\log p(0)|$.*

Proof. Let $\zeta_1, \zeta_2, \ldots, \zeta_d \in \mathbb{C}$ denote the roots of polynomial p. For any $z \in \mathbb{D}_b$,

$$\begin{aligned} |\log p(z)| &= \left| \log p(0) + \sum_{j=1}^{d} \log\left(1 - \frac{z}{\zeta_j}\right) \right| \\ &\leq |\log p(0)| - \sum_{j=1}^{d} \ln\left(1 - \left|\frac{z}{\zeta_j}\right|\right) \\ &\leq |\log p(0)| + d \ln \frac{1}{1-b}. \end{aligned}$$

The inequalities are due to that all $|\zeta_j| > 1$ since $p(z) \neq 0$ for all $z \in \mathbb{D}$. ◀

The following lemma of Barvinok says that any holomorphic function on $\mathbb{D}$ can be approximated by its truncated Taylor expansion if it is zero-free on $\mathbb{D}$.

▶ **Lemma 6** (Barvinok [4])**.** *Let $g : \mathbb{D} \to \mathbb{C}$ be holomorphic and $M > 0$. If $|g(z)| \leq M$ for all $z \in \mathbb{D}$, then for any $z \in \mathbb{D}$ and any $m \in \mathbb{N}^+$,*

$$\left| g(z) - \sum_{k=0}^{m} \frac{g^{(k)}(0)}{k!} z^k \right| \leq \frac{M}{\delta} (1-\delta)^{m+1},$$

where $\delta = \operatorname{dist}(z, \partial\mathbb{D})$ denotes the Euclidean distance between z and boundary of unit disc.

When Lemma 6 is applied to $g = \log f$ for some holomorphic $f : \mathbb{D} \to \mathbb{C} \setminus \{0\}$, one can obtain a multiplicative approximation of f on $\mathbb{D}$ assuming zero-freeness of f on $\mathbb{D}$. To make such approximation effective, we should be able to compute the Taylor coefficients of $g = \log f$.

The following *Newton's identity* relates the Taylor coefficients of $g = \log f$ to those of f.

▶ **Lemma 7** (Newton's identity)**.** *Let $f(z) = \sum_{k=0}^{+\infty} f_k z^k$ be an entire function such that $f(z) \neq 0$ for all $z \in \mathbb{D}$. Then $g(z) = \log f(z) = \sum_{k=0}^{+\infty} g_k z^k$ is well-defined on $\mathbb{D}$, and*

$$n g_n = n f_n - \sum_{k=1}^{n-1} k g_k f_{n-k}.$$

Proof. By the definition of $g = \log f$, we have $f' = g' f$. Therefore,

$$\begin{aligned} n f_n = \frac{1}{(n-1)!} f^{(n)}(0) &= \frac{1}{(n-1)!} \sum_{k=0}^{n-1} \binom{n-1}{k} f^{(k)}(0) g^{(n-k)}(0) \\ &= \sum_{k=0}^{n-1} (n-k) g_{n-k} f_k \\ &= \sum_{k=1}^{n} k g_k f_{n-k}. \end{aligned}$$

◀

When the zero-free region is not unit disc, some preprocessing is needed. The following polynomial transformation from $\mathbb{D}$ to any $S_{\beta,\delta}$ is a consequence of [4, Lemma 2.2.3].

▶ **Lemma 8.** *For any $\beta \in \mathbb{C}$, $\delta \in (0,1)$, there is an explicitly constructed polynomial $p_{\beta,\delta}$ of degree $d = d(\beta,\delta)$ satisfying*

- *$p_{\beta,\delta}(0) = 0$ and $p_{\beta,\delta}(1-\delta_0) = \beta$ for some $\delta_0 \in (0,1)$;*
- *$p_{\beta,\delta}(\mathbb{D}) \subseteq S_{\beta,\delta}$;*

The following proposition was proved in [4, Lemma 2.2.3], which explicitly gave a polynomial mapping a disk with radius slightly larger than 1 to a strip.

▶ **Proposition 9.** *Let $\delta \in (0,1)$ be a constant, and let $q_\delta \in \mathbb{C}[z]$ be defined as follows:*

$$q_\delta(z) = \frac{1}{\sum_{k=1}^{n} \frac{C^k}{k}} \sum_{k=1}^{n} \frac{(Cz)^k}{k},$$

where $C = 1 - \exp\left(-\frac{1}{\delta}\right)$, $n = \left\lfloor \left(1+\frac{1}{\delta}\right)\exp(1+\frac{1}{\delta}) \right\rfloor$.

Then for all $|z| < \rho$, where $\rho = \frac{1-\exp\left(-1-\frac{1}{\delta}\right)}{1-\exp\left(-\frac{1}{\delta}\right)} > 1$,

1. *$q_\delta(0) = 0$, $q_\delta(1) = 1$,*
2. *$\mathbf{Re}\left(q_\delta(z)\right) \in [-\delta, 1+2\delta]$ and $|\mathbf{Im}\left(q_\delta(z)\right)| \le 2\delta$.*

Proof of Lemma 8. We now prove Lemma 8 using Proposition 9. Without loss of generality, we assume that $\beta \neq 0$. Note that the polynomial q_δ defined in Proposition 9 satisfies $q_\delta(0) = 0$, $q_\delta(1) = 1$, and $q_\delta(\mathbb{D}_\rho) \subseteq S_{1,4\delta}$.

Therefore, we can set $p_{\beta,\delta}(z) = \beta q_{\delta'}(\rho' z)$, where $\delta_0 = \frac{\delta}{4|\beta|}$ and $\rho' = \frac{1-\exp\left(-1-\frac{1}{\delta'}\right)}{1-\exp\left(-\frac{1}{\delta'}\right)}$. We conclude our proof by observing that $p_{\beta,\delta}(\mathbb{D}) \subseteq S_{\beta,\delta}$ and $p_{\beta,\delta}(0) = 0$, $p_{\beta,\delta}\left(\frac{1}{\rho'}\right) = \beta$. ◀

3 Approximation of Zero-Free Holomorphic Function

We now introduce an abstraction for partition functions, namely, multiplicative holomorphic functions specified by a class of abstract structures called dependency graphs.

A *dependency graph* is a vertex-labeled graph $G = (V, E, \mathcal{L})$, where (V, E) is an undirected simple graph, and $\mathcal{L} = (L_v)_{v \in V}$ assigns each vertex $v \in V$ a label L_v. Two labeled graphs $G = (V, E, \mathcal{L})$ and $G' = (V', E', \mathcal{L}')$ are isomorphic if there is a bijection $\phi : V \to V'$ such that the two simple graphs (V, E) and (V', E') are isomorphic under ϕ and $L_v = L'_{\phi(v)}$ for all $v \in V$. Furthermore, we say that two labeled graphs $G = (V, E, \mathcal{L})$ and $G' = (V', E', \mathcal{L}')$ are disjoint if $V \cap V' = \varnothing$. A family $\mathcal{G}$ of dependency graphs is called *downward-closed* if for any $G = (V, E, \mathcal{L}) \in \mathcal{G}$ and any $S \subseteq V$ we have $G[S] \in \mathcal{G}$, where $G[S]$ stands for the subgraph of G induced by subset $S \subseteq V$ preserving labels.

We use $f.$ to denote an operator that maps each dependency graph G in $\mathcal{G}$ to an entire function $f_G : \mathbb{C} \to \mathbb{C}$ (i.e. f_G is holomorphic on $\mathbb{C}$), such that f_G gives the same entire function for isomorphic dependency graphs G. Such an $f.$ is *multiplicative* if for any G that is disjoint union of G_1, G_2, we have $f_G = f_{G_1} f_{G_2}$.

▶ **Definition 10** (boundedness). *Let $\mathcal{G}$ be a downward-closed family of dependency graphs. Let $\alpha, \beta \ge 1$. A multiplicative $f.$ is called (α,β)-*bounded *on $\mathcal{G}$ if for any $G = (V, E, \mathcal{L}) \in \mathcal{G}$, we have $f_G(0) \in \mathbb{R}^+$ and*

$$f_G(z) = f_G(0) + \sum_{\ell=1}^{+\infty} \left(\sum_{S \subseteq V} \lambda_{G[S],\ell} \right) z^\ell,$$

where the complex coefficients $(\lambda_{H,\ell})_{H\in\mathcal{G},\ell\in\mathbb{N}^+}$ *are invariant for isomorphic dependency graphs* H*, and satisfy that* $\lambda_{H,\ell}\neq 0$ *only if* $|V_H|\leq\alpha\ell$*, and each* $\lambda_{H,\ell}$ *can be calculated within* $\beta^\ell\cdot\mathrm{poly}(\ell)$ *time.*

For (α,β)-bounded $f.$, it always holds that $f_G(0)\in\mathbb{R}^+$. Then we always fix the definition of $\log f$ to be the one uniquely defined by Eq.(3) with $z_0=0$ and $c_0=\ln(f(0))$ being real. Such $\log f$ is well defined within a neighborhood of the origin.

As we will formally verify in Section 3.2, this notion of bounded holomorphic functions specified by dependency graphs generalizes the bounded induced graph counting polynomials (BIGCPs) of Patel and Regts [29]. A major difference here is that f_G may not be a polynomial of finite degree.

We show that for (α,β)-bounded $f.$, the approach of Patel and Regts [29] based on Newton's identity and local enumeration of connected subgraphs can efficiently compute Taylor coefficients of $\log f_G$, even though the function f_G can now encode problems far beyond counting subgraphs.

▶ **Theorem 11** (efficient coefficient computing)**.** *Let* $\mathcal{G}$ *be a downward-closed family of dependency graphs, and* $f.$ *be* (α,β)*-bounded on* $\mathcal{G}$ *for* $\alpha,\beta\geq 1$*. There exists a deterministic algorithm which given any* $G\in\mathcal{G}$ *and* $\ell\in\mathbb{N}^+$ *as input, outputs the* ℓ*-th coefficient of the Taylor series of* $\log f_G$ *at the origin in time* $\widetilde{O}\left(n(8\mathrm{e}\beta\Delta)^{\alpha\ell}\right)$*, where* n *is the number of vertices,* Δ *is the maximum degree of* G*, and* $\widetilde{O}(\cdot)$ *hides* $\mathrm{poly}(\Delta,\ell,\log(n))$*.*

The theorem will be proved in Section 4.

Due to Riemann mapping theorem in complex analysis, for any proper and simply connected region $\Omega\subset\mathbb{C}$ and any point $z_0\in\Omega$, there is a biholomorphic function h from $\mathbb{D}$ to Ω such that $h(0)=z_0$. We are interested in those good regions $\Omega\subseteq\mathbb{C}$ such that a transformation h from $\mathbb{D}$ to Ω does not only exist but also has efficiently computable Taylor coefficients.

▶ **Definition 12** (good region)**.** *Let* $\gamma\geq 1$*. A simply connected region* $\Omega\subseteq\mathbb{C}$ *is* γ-good *if* $0\in\Omega$ *and given any* $x\in\Omega$*, there exists a holomorphic function* h_x *on* $\mathbb{D}$ *along with a* $z_x\in\mathbb{D}$ *such that:*

1. $h_x(\mathbb{D})\subseteq\Omega$*,* $h_x(0)=0$ *and* $h_x(z_x)=x$*;*
2. *for every* $\ell\in\mathbb{N}^+$*, the* ℓ*-th Taylor coefficient* $h_{x,\ell}$ *of* h_x *at 0 can be determined in* $\gamma^\ell\mathrm{poly}(\ell)$ *time.*

Given a γ*-good region* $\Omega\subseteq\mathbb{C}$ *and* $\delta\in(0,1)$*, we use* Ω_δ *to denote the set of all* $x\in\Omega$ *with* $z_x\in\mathbb{D}_{1-\delta}$*.*

Any convex region is 1-good, given access to an oracle that determines the distance to its boundary.

▶ **Fact 13.** *Let* $\Omega\subseteq\mathbb{C}$ *be a convex region. Suppose that for any* $z\in\Omega$*,* $\mathrm{dist}(z,\partial\Omega)$ *can be determined in* $O(1)$ *time. Then* Ω *is* 1*-good.*

Proof of Fact 13. The convexity of Ω implies that $\mathrm{dist}([z_l,z_r],\partial\Omega)=\min(\mathrm{dist}(z_l),\mathrm{dist}(z_r))$ for arbitrary complex values $z_l,z_r\in\Omega$, where $[z_l,z_r]=\{z_l+t(z_r-z_l)\mid t\in[0,1]\}$. Hence, for each $x\in\Omega$, we set $f_x=p_{x,\delta}$, a polynomial defined in Proposition 9. We conclude the proof by observing that the k-th coefficient of $p_{x,\delta}$ can be determined in $O(k)$ time. ◀

Applying our Theorem 11 to $\log f_G$, combining with Barvinok's approximation (Lemma 6) and our notion of good region, we obtain the following theorem for multiplicative approximation of zero-free f_G's.

▶ **Theorem 14** (efficient ε-approximation). *Let $\alpha, \beta, \gamma \geq 1$. Let $\mathcal{G}$ be a downward-closed family of dependency graphs, and $f_\cdot$ be (α, β)-bounded on $\mathcal{G}$. Let $\Omega \subset \mathbb{C}$ be a γ-good region.*

For any $\delta \in (0,1)$, there is a deterministic algorithm which takes $G \in \mathcal{G}$, $x \in \Omega_\delta$ and an error bound $\varepsilon \in (0,1)$ as input, and outputs an estimation $\widehat{f_G(x)}$ of $f_G(x)$ within ε-multiplicative error:

$$1 - \varepsilon \leq \frac{\left|\widehat{f_G(x)}\right|}{|f_G(x)|} \leq 1 + \varepsilon,$$

in $\widetilde{O}\left(n \left(\frac{M}{\delta\varepsilon}\right)^C\right)$ time with $C = \frac{\alpha}{\delta}\ln(8\mathrm{e}\Delta(\beta+\gamma))$, where Δ is the maximum degree of G, if provided that f_G is M-zero-free on $\Omega \subset \mathbb{C}$ and the value of $f_G(0)$ is also provided to the algorithm.

Note that in above theorem, the zero-freeness property can be verified on any particular class of dependency graphs, although the boundedness property should be guaranteed on a downward-closed family. When applying this theorem, the value of $f_G(0)$ is usually trivial to compute (e.g. $f_G(0) = 1$), and the M-zero-freeness is usually established for some $M = \mathrm{poly}(n)$ (e.g. $M = O(n)$) on n-vertex dependency graphs. In such typical cases, the running time in Theorem 14 is bounded as $\left(\frac{n}{\delta\varepsilon}\right)^{O\left(\frac{1}{\delta}\log\Delta\right)}$.

Proof of Theorem 14. Let $h = h_x$ be a holomorphic function that transforms $\mathbb{D}$ to the γ-good region Ω with $h(z_x) = x$, where $z_x \in \mathbb{D}_{1-\delta}$ since $x \in \Omega_\delta$. And let $f_G^h = f_G \circ h$.

First, observe that f_G^h is M-zero-free on $\mathbb{D}$, because $\left|\log f_G^h(z)\right| = \left|\log f_G(h(z))\right| \leq M$ holds for all $z \in \mathbb{D}$ since $h(\mathbb{D}) \subseteq \Omega$ and f_G is M-zero-free on Ω. Then by Lemma 6, for any $z \in \mathbb{D}$, the difference between $\log f_G^h(z)$ and the truncated Taylor expansion at 0 is bounded by

$$\left|\log f_G^h(z) - \sum_{k=0}^{m} \frac{\left(\log f_G^h\right)^{(k)}(0)}{k!} z^k\right| \leq \frac{M}{\delta}(1-\delta)^{m+1} < \varepsilon, \tag{4}$$

for $m = \lceil \frac{1}{\delta}\ln\frac{M}{\delta\varepsilon} \rceil$.

It remains to verify that $f_\cdot^h$ is $(\alpha, \beta+\gamma)$-bounded on $\mathcal{G}$. By Theorem 11, this will prove the theorem.

For $\ell \in \mathbb{N}^+$, let $h_k^{(\ell)}$ denote the k-th Taylor coefficient of $h(z)^\ell$ at $z = 0$. Since $h(0) = 0$, we have

$$h(z)^\ell = \left(\sum_{k=1}^{+\infty} h_k z\right)^\ell = \sum_{k=\ell}^{+\infty} h_k^{(\ell)} z^k$$

Since f_G is (α, β)-bounded and $h(0) = 0$, we have

$$\begin{aligned}
f_G^h(z) &= f_G(0) + \sum_{\ell=1}^{+\infty}\left(\sum_{S\subseteq V}\lambda_{G[S],\ell}\right) h(z)^\ell \\
&= f_G(0) + \sum_{\ell=1}^{+\infty}\left(\sum_{S\subseteq V}\lambda_{G[S],\ell}\right)\left(\sum_{k=\ell}^{+\infty} h_k^{(\ell)} z^k\right) \\
&= f_G(0) + \sum_{k=1}^{+\infty}\sum_{S\subseteq V}\left(\sum_{\ell=1}^{k} h_k^{(\ell)}\lambda_{G[S],\ell}\right) z^k \\
&= f_G(0) + \sum_{k=1}^{+\infty}\left(\sum_{S\subseteq V}\lambda_{G[S],k}^h\right) z^k,
\end{aligned}$$

where $\lambda^h_{H,k}$ for any $H \in \mathcal{G}$ and $k \in \mathbb{N}^+$ is defined as

$$\lambda^h_{H,k} \triangleq \sum_{\ell=1}^{k} h_k^{(\ell)} \lambda_{H,\ell}.$$

Clearly, $\lambda^h_{H,k} = 0$ if $|V_H| > \alpha k$, where V_H denotes the vertex set of H, since $\lambda_{H,\ell} = 0$ if $|V_H| > \alpha\ell$. And it can be verified that any $\lambda^h_{H,k}$ can be determined within $(\beta+\gamma)^k \mathrm{poly}(k)$ time. This is because within $\beta^k \mathrm{poly}(k)$ time, one can list all $\lambda_{H,1}, \ldots, \lambda_{H,k}$, and for each $1 \le \ell \le k$, $h_k^{(\ell)}$ is just the coefficient of z^k in $\left(h_1 z + h_2 z^2 + \cdots + h_k z^k\right)^\ell$, which can be calculated in $\mathrm{poly}(k)$ time given all $h_1, \ldots, h_k$, which can be listed beforehand in $\gamma^k \mathrm{poly}(k)$ time. Overall, this takes at most $(\beta+\gamma)^k \mathrm{poly}(k)$ time.

Therefore, $f^h_\cdot$ is $(\alpha, \beta+\gamma)$-bounded. ◀

3.1 Quantum partition functions

We formally prove Theorem 2. Recall the definition of quantum partition function Z_H in (2). We extend this definition by considering measurement.

Recall that $\mathcal{V} = \bigotimes_{u \in V} \mathcal{V}_u$ where V is the set of n sites and each $\mathcal{V}_u$ is a q-dimensional Hilbert space. A *measurement* $\mathcal{O}$ is a positive operator in $\mathcal{V}$. The quantum partition function induced by Hamiltonian H under measurement $\mathcal{O}$, both in $\mathcal{V}$, is defined by

$$Z_{H,\mathcal{O}}(\beta) \triangleq \mathbf{Tr}\left[\exp(\beta H)\mathcal{O}\right]. \tag{5}$$

Furthermore, a measurement $\mathcal{O}$ is *tensorized* if $\mathcal{O} = \bigotimes_{v \in V} \mathcal{O}_v$ where $\mathrm{supp}(\mathcal{O}_v) = \{v\}$.

We show that under tensorized measurement $\mathcal{O}$, the quantum partition functions $Z_{H,\mathcal{O}}$ defined by local Hamiltonians with $O(1)$ maximum degree are $(1, O(1))$-bounded. Together with Theorem 14 we obtain the following theorem.

▶ **Theorem 15.** *Let $\Omega \subset \mathbb{C}$ be a γ-good region for $\gamma \ge 1$. For any $\delta \in (0,1)$, there is a deterministic algorithm such that given any (k,d)-Hamiltonian H and tensorized measurement $\mathcal{O}$, provided that $\frac{1}{\mathbf{Tr}[\mathcal{O}]} Z_{H,\mathcal{O}}$ is M-zero-free on Ω, for any temperature $\beta \in \Omega_\delta$ and error bound $\varepsilon \in (0,1)$, the algorithm outputs an estimation of $Z_{H,\mathcal{O}}(\beta)$ within ε-multiplicative error in $\widetilde{O}\left(n \left(\frac{M}{\delta\varepsilon}\right)^C\right)$ time with $C = \frac{1}{\delta} \ln\left(8ed\left(2q^{3k}+\gamma\right)\right)$.*

Note that when the measurement $\mathcal{O}$ is the identity, $Z_{H,\mathcal{O}}$ is precisely the partition function Z_H, which implies Theorem 2. As discussed in the introduction, Theorem 15 covers all classical partition functions (with or without external field) when temperature is the complex variable.

Following a recent work of Harrow, Mehraban and Soleimanifar [18], Theorem 15 gives polynomial-time approximations of quantum partition functions defined by local Hamiltonians with $O(1)$ maximum degree when the inverse temperature β is close to zero. And following a standard routine of self-reduction, in the same regime, we have a polynomial-time approximate sampler from the quantum Gibbs state after the measurement in the computational basis. These applications are given in Section 5.

Proof of Theorem 15. Given a (k,d)-Hamiltonian $H = \sum_{j=1}^m H_j$, we can construct a dependency graph $G_H = (U, E, \mathcal{L})$ as follows:

1. $U = [m]$ is the vertex set;
2. $E = \left\{\{x,y\} \in \binom{U}{2} \mid \mathrm{supp}(H_x) \cap \mathrm{supp}(H_y) \neq \varnothing\right\}$;
3. for any $x \in U$, its label is given by $L_x = H_x$.

Let $\mathcal{G}_k$ denote the family of all such G_H where H is a k-local Hamiltonian. It is obvious that such $\mathcal{G}_k$ is downward-closed.

Let $\mathcal{O} = \bigotimes_{v\in V} \mathcal{O}_v$ be a tensorized measurement. For any $G = G_H \in \mathcal{G}_k$, where $H = \sum_{j=1}^{m} H_j$, define:

$$f_G(z) = \frac{1}{\mathbf{Tr}\left[\mathcal{O}\right]} Z_{H,\mathcal{O}}(z) = \frac{1}{\mathbf{Tr}\left[\mathcal{O}\right]} \mathbf{Tr}\left[\exp\left(-z\sum_{j=1}^{m} H_j\right)\mathcal{O}\right].$$

The rest of the proof verifies that such $f_\cdot$ is $(1, 2q^{3k})$-bounded on $\mathcal{G}_k$, which is sufficient to prove the theorem by Theorem 14.

We first verify that such $f_\cdot$ is multiplicative. For any $G = (U, E, \mathcal{L}) \in \mathcal{G}_k$ that is the disjoint union of $G_1 = (U_1, E_1, \mathcal{L}_1)$ and $G_2 = (U_2, E_2, \mathcal{L}_2)$, there exists a bipartition $V = V_1 \uplus V_2$ such that $\mathrm{supp}(H_x) \subseteq V_1$ for all $x \in U_1$ and $\mathrm{supp}(H_y) \subset V_2$ for all $y \in U_2$. Let $H_{U_i} = \sum_{x\in U_i} H_x$ for $i = 1, 2$. We have,

$$\begin{aligned}
f_G(z) &= \frac{1}{\mathbf{Tr}[\mathcal{O}]} \mathbf{Tr}\left[\exp\left(-z(H_{U_1} + H_{U_2})\right)\mathcal{O}\right] \\
&= \frac{1}{\mathbf{Tr}[\mathcal{O}]} \mathbf{Tr}[\exp(-zH_{U_1})\exp(-zH_{U_2})\mathcal{O}] \\
&= \frac{1}{\mathbf{Tr}[\mathcal{O}]} \mathbf{Tr}_{V_1}\left[\exp(-zH_{U_1})\bigotimes_{v\in V_1}\mathcal{O}_v\right] \mathbf{Tr}_{V_2}\left[\exp(-zH_{U_2})\bigotimes_{v\in V_2}\mathcal{O}_v\right] \\
&= \frac{1}{\mathbf{Tr}^2[\mathcal{O}]} \mathbf{Tr}[\exp(-zH_{U_1})\mathcal{O}]\mathbf{Tr}[\exp(-zH_{U_2})\mathcal{O}] \\
&= f_{G_1}(z) f_{G_2}(z)
\end{aligned}$$

Here the subscripts V_1, V_2 in $\mathbf{Tr}_{V_1}, \mathbf{Tr}_{V_2}$ indicates the sets of sites that the operators act on. Therefore, $f_\cdot$ is multiplicative.

For any $G = (U, E, \mathcal{L}) \in \mathcal{G}_k$ and any $\ell \in \mathbb{N}^+$, define

$$\lambda_{G,\ell} = \frac{1}{\ell!}\frac{1}{\mathbf{Tr}[\mathcal{O}]} \sum_{f:[\ell]\overset{\text{onto}}{\to} U} \mathbf{Tr}\left[\left(\prod_{j=1}^{\ell} H_{f(j)}\right)\mathcal{O}\right]. \tag{6}$$

Observe that $\lambda_{G,\ell} = 0$ if $|U| > \ell$ as there is no surjection from $[\ell]$ to U. Moreover, for $H = \sum_{x\in U} H_x$,

$$\begin{aligned}
f_G(z) = 1 + \sum_{\ell=1}^{+\infty} \frac{\beta^\ell}{\ell!}\mathbf{Tr}\left[H^\ell\mathcal{O}\right] &= 1 + \sum_{\ell=1}^{+\infty} \frac{z^\ell}{\ell!} \sum_{x_1,x_2,\ldots,x_\ell\in U} \mathbf{Tr}\left[\left(\prod_{j=1}^{\ell} H_{x_j}\right)\mathcal{O}\right] \\
&= 1 + \sum_{\ell=1}^{+\infty}\left(\sum_{S\subseteq U}\lambda_{G[S],\ell}\right) z^\ell.
\end{aligned}$$

It remains to show that $\lambda_{G,\ell}$ can be determined within $(2q^{3k})^\ell \mathrm{poly}(\ell)$ time.

Fix a $G = (U, E, \mathcal{L}) \in \mathcal{G}_k$. For any $S \subseteq U$, define

$$H_{S,\ell} \triangleq \sum_{f:[\ell]\overset{\text{onto}}{\to} S} \prod_{j=1}^{\ell} H_{f(j)}.$$

Clearly, $\lambda_{G,\ell} = \frac{1}{\ell!}\frac{1}{\mathbf{Tr}[\mathcal{O}]}\mathbf{Tr}\left[H_{U,\ell}\mathcal{O}\right]$. Moreover, the following recurrence holds for $H_{S,\ell}$:

$$H_{S,\ell} = \sum_{j\in S} H_j \left(H_{S,\ell-1} + H_{S\setminus\{j\},\ell-1}\right), \tag{7}$$

where the boundary cases are given by $H_{\varnothing,0} = I$, and $H_{S,\ell} = \mathbf{0}$ (the 0-matrix) if $\ell < |S|$, or $S = \varnothing$ but $\ell > 0$. Note that $H_{S,\ell}$ acts non-trivially on at most $k\,|S|$ sites, where each site corresponds to a q-dimensional Hilbert space, thus $H_{S,\ell}$ can be represented as a matrix of size at most $q^{k|S|} \times q^{k|S|}$ and the recursion step (7) can be evaluated in time $O(|S|q^{3k|S|})$. Therefore, for any $S \subseteq U$ that $1 \le |S| \le \ell$, $H_{S,\ell}$ can be computed in time $O(2^{|S|}\ell|S|q^{3k|S|}) = O(\ell^2 2^\ell q^{3k\ell})$ by a dynamic programming that constructs a $2^S \times [\ell]$ table according to the recurrence (7). And finally, $\lambda_{G,\ell} = \frac{1}{\ell!}\frac{1}{\mathbf{Tr}[\mathcal{O}]}\mathbf{Tr}\left[H_{U,\ell}\mathcal{O}\right]$ can be computed from $H_{U,\ell}$ in $O(q^{3k\ell})$ time because $H_{U,\ell}$ acts non-trivially on at most $k|U| \le k\ell$ sites and $\mathcal{O}$ is tensorized. ◀

3.2 Induced subgraph counting

Our framework (Definition 10) subsumes bounded induced graph counting polynomials (BIGCP) defined by Patel and Regts [29].

A BIGCP $p.$ defines multiplicative graph polynomials p_G for all graphs $G = (V, E)$. Moreover, there exists integer $\alpha \ge 1$ and sequence $\lambda_{H,\ell}$ of complex values such that the following conditions are satisfied.

1. For any graph $G = (V, E)$, p_G can be expressed as

$$p_G(z) = 1 + \sum_{\ell=1}^{m(G)} \left(\sum_{\substack{H=(V_H,E_H)\\ |V_H|\le\alpha\ell}} \lambda_{H,\ell}\cdot \mathrm{ind}(H,G) \right) z^\ell,$$

   where $\mathrm{ind}(H, G)$ represents the number of induced subgraphs $G[S]$, $S \subseteq V$, isomorphic to H.
2. $\lambda_{H,\ell}$ can be determined in $O(\beta^\ell)$ time for some constant $\beta \ge 1$.

For any $G = (V, E)$, we define a dependency graph $G^* = (V, E, \mathcal{L})$ where $\mathcal{L}$ labels every $v \in V$ with a trivial symbol $*$. Let $\mathcal{G}$ denote the family of all G^*, which is clearly downward-closed. We define $f_{G^*} = p_G$.

Note that

$$\sum_{\substack{H=(V_H,E_H)\\ |V_H|\le\alpha\ell}} \lambda_{H,\ell}\mathrm{ind}(H,G) = \sum_{\substack{S\subseteq V\\ |S|\le\alpha\ell}} \lambda_{G[S],\ell}.$$

Therefore, any BIGCP $p.$ corresponds to an $f.$ that is (α, β)-bounded on $\mathcal{G}^*$.

3.3 Boolean CSP with external field

A Boolean-variate constraint satisfaction problem (Boolean CSP) is specified by a $\Phi = (V, E, \boldsymbol{\phi})$, where $H = (V, E)$ is a hypergraph and $\boldsymbol{\phi} = (\phi_e)_{e\in E}$ such that each $\phi_e : \{0,1\}^e \to \mathbb{C}$ is a Boolean-variate complex-valued constraint function. Furthermore, $\Phi = (V, E, \boldsymbol{\phi})$ is a (k, d)-formula if $|e| \le k$ for every $e \in E$ and $\deg(v) = |\{e \in E \mid v \in e\}| \le d$ for every $v \in V$.

The partition function for a Boolean CSP $\Phi = (V, E, \boldsymbol{\phi})$ of external field λ is defined as:

$$Z_\Phi(\lambda) = \sum_{\sigma\in\{0,1\}^V} \left(\prod_{e\in E} \phi_e(\sigma|_e)\right) \lambda^{\|\sigma\|_1}.$$

In [27], Liu, Sinclair and Srivastava formulated the above partition function as counting hypergraph insects and gave a polynomial-time algorithm for such a partition function assuming its zero-freeness. Such partition functions are also subsumed in our framework.

Given a Boolean CSP $\Phi = (V, E, \boldsymbol{\phi})$, a dependency graph $G_\Phi = (V_\Phi, E_\Phi, \mathcal{L}_\Phi)$ can be constructed as follows:

1. $V_\Phi = V$;
2. for any distinct $u, v \in V_\Phi = V$, $\{u, v\} \in E_\Phi$ iff $\{u, v\} \subseteq e$ for some $e \in E$;
3. for any $v \in V_\Phi = V$, its label $L_v = (\phi_e)_{e\in E, v\in e}$.

Note that each constraint ϕ_e appears in labels of all $v \in e$, and the maximum degree of G_Φ is bounded by $\Delta \le (k-1)d$ for a (k, d)-formula Φ.

Let $\mathcal{G}_{k,d}$ denote the family of all such dependency graphs G_Φ, where $\Phi = (V, E, \boldsymbol{\phi})$ is a (k, d)-formula, and all their induced subgraphs. Obviously, such $\mathcal{G}_{k,d}$ is downward-closed.

Let $G \in \mathcal{G}_{k,d}$. Without loss of generality, suppose that $G = G_\Phi[U]$ is the subgraph of the dependency graph G_Φ induced by $U \subseteq V$, where $\Phi = (V, E, \boldsymbol{\phi})$ is a Boolean CSP.

We define

$$f_G(\lambda) = \sum_{\sigma\in\{0,1\}^U} \prod_{\substack{e\in E\\ e\cap U\neq\varnothing}} \phi_e^U(\sigma|_{U\cap e})\lambda^{\|\sigma\|_1},$$

where $\phi_e^U : U \cap e \to \mathbb{C}$ is defined as that for any $\tau \in \{0, 1\}^{U\cap e}$,

$$\phi_e^U(\tau) = \phi_e(\tau^*),$$

where $\tau^* \in \{0, 1\}^e$ extends τ and assigns all $v \in e \setminus U$ with 0. It is easy to verify that such definition f_G uses only the information stored in the dependency graph G, thus it is well defined. Meanwhile, it is also easy to verify that $f_\cdot$ is multiplicative and $f_G(\lambda) = Z_\Phi(\lambda)$ if $G = G_\Phi$.

For $G = G_\Phi[U]$ where $\Phi = (V, E, \boldsymbol{\phi})$ and $U \subseteq V$, define $\lambda_{G,\ell}$, as

$$\lambda_{G,\ell} = \begin{cases} \prod_{\substack{e\in E\\ e\cap U\neq\varnothing}} \phi_e^U(\mathbf{1}_{U\cap e}), & |U| = \ell \\ 0, & o.w. \end{cases}$$

Each $\lambda_{G,\ell}$ can be determined in $\mathrm{poly}(k, d, \ell)$ time.

Observe that,

$$f_G(\lambda) = 1 + \sum_{\ell=1}^{|U|} \sum_{\substack{\sigma\in\{0,1\}^U\\ \|\sigma\|_1=\ell}} \prod_{\substack{e\in E\\ e\cap U\neq\varnothing}} \phi_e^U(\sigma|_{U\cap e})\lambda^\ell = 1 + \sum_{\ell=1}^{|U|} \sum_{\substack{S\subseteq U\\ |S|=\ell}} \prod_{\substack{e\in E\\ e\cap S\neq\varnothing}} \phi_e^S(\mathbf{1}_{S\cap e})\lambda^\ell$$
$$= 1 + \sum_{\ell=1}^{+\infty} \left(\sum_{S\subseteq U} \lambda_{G[S],\ell}\right) \lambda^\ell.$$

Therefore, $f_\cdot$ is a $(1, 1)$-bounded on $\mathcal{G}_{k,d}$.

Applying Theorem 14, we immediately obtain the following corollary. Similar bound has been proved in [27], but here we only need to encode the problem in our framework.

▶ **Corollary 16.** *Let $\Omega \subseteq \mathbb{C}$ be a γ-good region for $\gamma \geq 1$. For any $\delta \in (0,1)$, there is a deterministic algorithm such that given any (k,d)-formula Φ for Boolean CSP, provided that Z_Φ is M-zero-free on Ω, for any external field $\lambda \in \Omega_\delta$ and error bound $\varepsilon \in (0,1)$, the algorithm outputs an estimation of $Z_\Phi(\lambda)$ within ε-multiplicative error in $\widetilde{O}\left(n\left(\frac{M}{\delta\varepsilon}\right)^C\right)$ time with $C = \frac{1}{\delta}\ln(8ekd(1+\gamma))$.*

Since such $Z_\Phi(\lambda)$ is a polynomial with finite degree, by Fact 13 and Lemma 5, if $\Omega \subseteq \mathbb{C}$ is a convex region and Z_Φ does not vanish on a slightly larger region $\Omega' = \{(1+\delta)z \mid z \in \Omega\}$ for some constant gap $\delta \in \mathbb{R}^+$, then $M = O_\delta(n)$ and hence the algorithm in Corollary 16 runs in $\left(\frac{n}{\varepsilon}\right)^{O(\ln(kd))}$ time.

3.4 A barrier to non-multiplicative functions

Our framework based on functions f_G induced by dependency graphs G is fairly expressive. However, the current technique crucially relies on the multiplicative property of f_G. The current method would meet a barrier when dealing with systems lacking such property.

We explain this using a concrete example. Consider the following generalization of (1):

$$\forall \beta \in \mathbb{C}, \quad Z_{H,H'}(\beta) \triangleq \mathbf{Tr}\left[\exp(-\beta H + H')\right]. \tag{8}$$

Here, H and H' are two Hamiltonians in $\mathcal{V}$. It encompasses the transverse Ising model and XXZ model. Unfortunately, this partition function fails to fit in our framework due to its non-multiplicative nature, even when H' is a tensorized operator. For Hamiltonians H_1, H_2 such that $\mathrm{supp}(H_1) \cap \mathrm{supp}(H_2) = \varnothing$ and a tensorized operator $H' = \bigotimes_{v\in V} H'_v$, the following equality fails to hold in general: $Z_{H_1+H_2,H'}(\beta) = \frac{1}{\mathbf{Tr}[\exp(H')]} Z_{H_1,H'}(\beta) Z_{H_2,H'}(\beta)$.

For example, let

$$H_1 = I \bigotimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix},\ H_2 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \bigotimes I,\ H' = -\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \bigotimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

For $\beta = 1$, it holds that $Z_{H_1+H_2,H'}(\beta) = \mathbf{Tr}\left[\exp(-H_1 - H_2 + H')\right] = 3\mathrm{e}^{-1} + \mathrm{e}^{-2}$ but $\mathbf{Tr}\left[\exp(H')\right] = \mathrm{e}^{-1} + 3$ and $Z_{H_i,H'}(\beta) = \mathbf{Tr}\left[\exp(-H_i + H')\right] = 3\mathrm{e}^{-1} + 1$ for $i \in \{1,2\}$.

The main obstacle comes from the non-commutativity of Hamiltonians and it remains open to design a polynomial-time algorithm for such partition function assuming only zero-freeness.

4 Efficient Coefficient Computing

In this section we prove Theorem 11. First we need to establish the following lemma.

▶ **Lemma 17.** *Let $\mathcal{G}$ be a downward-closed family of dependency graphs, and f be (α,β)-bounded on $\mathcal{G}$ for $\alpha, \beta \geq 1$. Recursively define the sequence $(\zeta_{H,i})_{H\in\mathcal{G},\ell\in\mathbb{N}^+}$ of complex numbers as follows: for any $H = (V_H, E_H, \mathcal{L}_H) \in \mathcal{G}$ and any $\ell \in \mathbb{N}^+$,*

$$\zeta_{H,\ell} = \lambda_{H,\ell} - \sum_{s=1}^{\ell-1} \frac{s}{\ell} \sum_{\substack{S,T\subseteq V_H \\ S\cup T = V_H}} \zeta_{H[S],s}\lambda_{H[T],\ell-s}. \tag{9}$$

It holds that $\zeta_{H,\ell} \neq 0$ only if H is connected and $|V_H| \leq \alpha\ell$. Moreover, for any $G = (V,E,\mathcal{L}) \in \mathcal{G}$,

$$\log f_G(z) = \log f_G(0) + \sum_{\ell=1}^{+\infty} \left(\sum_{S\subseteq V} \zeta_{G[S],\ell}\right) z^\ell. \tag{10}$$

As in [29, 27], the following result of Borgs et al. [8] is used.

▶ **Fact 18** (Lemma 2.1 (c) in [8]). *Let $G = (V, E)$ be a graph with maximum degree Δ, $v \in V$ be a vertex and $\ell \in \mathbb{N}_{\geq 1}$. Then the number of connected subgraphs of size ℓ containing v is at most $\frac{(\mathrm{e}\Delta)^{\ell-1}}{2}$.*

With this fact, we can enumerate all connected induced subgraphs $G[S]$ of $|S| \leq \ell$ vertices efficiently.

▶ **Lemma 19.** *There exists a deterministic algorithm which takes a dependency graph $G = (V, E, \mathcal{L})$ on $n = |V|$ vertices with maximum degree Δ and a positive integer $\ell \in \mathbb{N}^+$ as input, and outputs*

$$\mathcal{C}_{\leq \ell} = \{S \subseteq V \mid |S| \leq \ell, G[S] \text{ is connected}\}, \tag{11}$$

in time $\widetilde{O}(n(\mathrm{e}\Delta)^{\ell})$, where $\widetilde{O}(\cdot)$ hides $\mathrm{poly}(\Delta, \ell, \log(n))$.

Proof. Let $\mathcal{C}^v_{=\ell}$ denote the collection of such $S \subseteq V$ containing $v \in V$ that $|S| = \ell$ and $G[S]$ is connected. Clearly $\mathcal{C}_{\leq \ell} = \bigcup_{\substack{v \in V \\ j \leq \ell}} \mathcal{C}^v_{=j}$. Now construct each $\mathcal{C}^v_{=\ell}$ inductively. When $\ell = 1$, $\mathcal{C}^v_{=1} = \{\{v\}\}$. For $\ell \geq 2$, we enumerate all $S \in \mathcal{C}^v_{=\ell-1}$ and $u \in V \setminus S$ such that $G[S \cup \{u\}]$ is connected, and include $S \cup \{u\}$ into $\mathcal{C}^v_{=\ell}$. It is easy to see that this correctly constructs $\mathcal{C}^v_{=\ell}$. By Fact 18, $|\mathcal{C}^v_{=\ell}| \leq (\mathrm{e}\Delta)^{\ell-1}/2$. Representing each set S as a string of vertices in S sorted in increasing order of vertices, the set $\mathcal{C}^v_{=\ell}$ can be stored by a standard dynamic data structure such as prefix trees, so that it takes $O(\Delta\ell(\mathrm{e}\Delta)^{\ell-1})$ time to iterate over all $(S, u) \in \mathcal{C}^v_{=\ell-1} \times V$ such that $G[S \cup \{u\}]$ may be connected, and for each such (S, u) pair it takes $\mathrm{poly}(\Delta, \ell, \log n)$ time to check weather $G[S \cup \{u\}]$ is connected or $S \cup \{u\}$ is already in $\mathcal{C}^v_{=\ell}$, and insert S into $\mathcal{C}^v_{=\ell}$ if necessary. Overall, it takes $\widetilde{O}(n(\mathrm{e}\Delta)^{\ell})$ time to construct $\mathcal{C}_{\leq \ell}$. ◀

Combining the above algorithm with (9), we can compute coefficients $\zeta_{H,\ell}$ for $\log f_G$ efficiently.

▶ **Lemma 20.** *Let $\mathcal{G}$ be a downward-closed family of dependency graphs, and f. be (α, β)-bounded on $\mathcal{G}$ for $\alpha, \beta \geq 1$. There exists a deterministic algorithm which takes a dependency graph $G = (V, E, \mathcal{L}) \in \mathcal{G}$ on $n = |V|$ vertices with maximum degree Δ and a positive integer $\ell \in \mathbb{N}^+$ as input, and outputs $(\zeta_{G[S],\ell})_{S \in \mathcal{C}_{\leq \alpha\ell}}$ within $\widetilde{O}\left(n(8\mathrm{e}\beta\Delta)^{\alpha\ell}\right)$ time, where $\mathcal{C}_{\leq \alpha\ell}$ is defined in Eq.* (11)*.*

The lemma follows by first enumerating all $S \in \mathcal{C}_{\leq \alpha\ell}$, which takes $\widetilde{O}\left(n(\mathrm{e}\Delta)^{\alpha\ell}\right)$ time according to Lemma 19, and second for every $S \in \mathcal{C}_{\leq \alpha\ell}$, taking $H = G[S]$ and computing $\zeta_{H,\ell}$ using a dynamic programming given by Eq. (9), which takes $\widetilde{O}(8^{\alpha\ell}\beta^{\ell})$ time.

Let $\log f_G(z) = \log f_G(0) + \sum_{\ell=1}^{+\infty} g_{G,\ell} z^{\ell}$. Due to Lemma 17, $\zeta_{G[S],\ell} = 0$ if $G[S]$ is disconnected or $|S| > \alpha\ell$, thus due to Eq. (9), the ℓ-th Taylor coefficient of $\log f_G$ is given by

$$g_{G,\ell} = \sum_{S \subseteq V} \zeta_{G[S],\ell} = \sum_{S \in \mathcal{C}_{\leq \alpha\ell}} \zeta_{G[S],\ell}.$$

Therefore, Theorem 11 is proved. It only remains to prove Lemma 17.

Proof of Lemma 17. Fix an arbitrary $G \in \mathcal{G}$, and consider f_G. Let $\log f_G = \log f_G(0) + \sum_{\ell=1}^{+\infty} g_{G,\ell} z^{\ell}$ denote the Taylor's expansion of $\log f_G$ at the origin, and $f_G(z) = f_G(0) + \sum_{\ell=1}^{+\infty} f_{G,\ell} z^{\ell}$ denote the Taylor's expansion of f_G at the origin. We prove by induction on $\ell \geq 1$ that

$$g_{G,\ell} = \sum_{S \subseteq V} \zeta_{G[S],\ell}, \tag{12}$$

which implies (10).

For the induction basis, when $\ell = 1$, by Lemma 7 we have $g_{G,1} = f_{G,1}$. by the definition of bounded graph function in Definition 10, $f_{G,1} = \sum_{S\subseteq V} \lambda_{G[S],1}$; and it follows from (9) that $\lambda_{G[S],1} = \zeta_{G[S],1}$. Altogether, we have

$$g_{G,1} = f_{G,1} = \sum_{S\subseteq V} \lambda_{G[S],1} = \sum_{S\subseteq V} \zeta_{G[S],1}.$$

Now suppose that the induction hypothesis (12) holds for all $\ell' < \ell$. We have

$$\begin{aligned}
\sum_{S\subseteq V} \zeta_{G[S],\ell} &= \sum_{S\subseteq V} \left(\lambda_{G[S],\ell} - \sum_{s=1}^{\ell-1} \frac{s}{\ell} \sum_{\substack{L,R\subseteq V \\ L\cup R = S}} \zeta_{G[L],s} \cdot \lambda_{G[R],\ell-s} \right) \\
&= \sum_{S\subseteq V} \lambda_{G[S],\ell} - \sum_{s=1}^{\ell-1} \frac{s}{\ell} \left(\sum_{L\subseteq V} \zeta_{G[L],s} \right) \left(\sum_{R\subseteq V} \lambda_{G[R],\ell-s} \right) \\
&= f_{G,\ell} - \sum_{s=1}^{\ell-1} \frac{s}{\ell} g_{G,s} f_{G,\ell-s} && \text{(I.H.)} \\
&= g_{G,\ell}. && \text{(Lemma 7)}
\end{aligned}$$

This finishes the inductive proof of (12).

Next, we prove that $\zeta_{H,\ell} = 0$ if $H = (V_H, E_H, \mathcal{L}_H) \in \mathcal{G}$ is disconnected or $|V_H| > \alpha\ell$. Recall that $f.$ is (α, β)-bounded, we have $\lambda_{H,\ell} = 0$ for $|V_H| > \alpha\ell$. Then the fact that $\zeta_{H,\ell} = 0$ for $|V_H| > \alpha\ell$ can be verified by induction on $\ell \geq 1$. Specifically, by Eq. (9),

$$\zeta_{H,\ell} = \lambda_{H,\ell} - \sum_{s=1}^{\ell-1} \frac{s}{\ell} \sum_{\substack{S,T\subseteq V_H \\ S\cup T = V_H}} \zeta_{H[S],s} \lambda_{H[T],\ell-s}.$$

For the basis, $\zeta_{H,1} = \lambda_{H,1} = 0$ when $|V_H| > \alpha$. In general, observe that assuming $|V_H| > \alpha\ell$, for any $S \cup T = V_H$, it must hold that either $|S| > \alpha s$ or $|T| > \alpha(\ell - s)$. Therefore, assuming $|V_H| > \alpha\ell$, $\zeta_{H,\ell} = 0$ follows from the induction hypothesis.

Finally, it remains to verify that $\zeta_{H,\ell} = 0$ if H is disconnected, which follows from the multiplicative property of $f.$. By contradiction, assume that $\zeta_{H,\ell} \neq 0$ for some disconnected $H \in \mathcal{G}$. Let $S^* \subseteq V_H$ be a minimal subset of V such that $H[S^*]$ is disconnected and $\zeta_{H[S^*],\ell} \neq 0$. Since $H[S^*]$ is disconnected, there exist nonempty $L, R \subseteq S^*$ such that $L \cup R = S^*$ and L, R are disconnected in $H[S^*]$. Due to the multiplicative property of $f.$, we have $f_{G[S^*]} = f_{G[L]} \cdot f_{G[R]}$. Therefore,

$$g_{G[S^*],\ell} = g_{G[L],\ell} + g_{G[R],\ell} = \sum_{S\subseteq L} \zeta_{G[S],\ell} + \sum_{S\subseteq R} \zeta_{G[S],\ell}, \tag{13}$$

where the first equation can be formally verified for any disjoint dependency graphs $G_1, G_2 \in \mathcal{G}$ and any z in the neighborhood of the origin, such that for an arbitrary path P in Ω connecting z and the origin,

$$\begin{aligned}
\log f_{G_1\cup G_2}(z) &= \log f_{G_1\cup G_2}(0) + \int_P \frac{f'_{G_1\cup G_2}(z)}{f_{G_1\cup G_2}(z)}\, dz \\
&= \log f_{G_1}(0) + \log f_{G_2}(0) + \int_P \left(\frac{f'_{G_1}(z)}{f_{G_1}(z)} + \frac{f'_{G_2}(z)}{f_{G_2}(z)} \right) dz \\
&= \log f_{G_1}(z) + \log f_{G_2}(z).
\end{aligned}$$

On the other hand,

$$g_{G[S^*],\ell} = \sum_{S \subseteq S^*} \zeta_{G[S],\ell} = \zeta_{G[S^*],\ell} + \sum_{S \subset S^*} \zeta_{G[S],\ell} = \zeta_{G[S^*],\ell} + \sum_{S \subseteq L} \zeta_{G[S],\ell} + \sum_{S \subseteq R} \zeta_{G[S],\ell}, \quad (14)$$

where the last equation is due to the minimality of S^*. Combining (13) and (14), we have $\zeta_{G[S^*],\ell} = 0$, a contradiction. ◀

5 Applications

In this section, we prove that any zero-free partition function of local Hamiltonians with bounded maximum degree is polynomial-time approximable if the inverse temperature is close enough to 0. This is formally stated by the following theorem. Recall the definition of the partition function $Z_{H,\mathcal{O}}$ induced by Hamiltonian H under measurement $\mathcal{O}$ in (5).

▶ **Theorem 21.** *Let $k, d \in \mathbb{N}^+$, $h > 0$, $\delta \in (0,1)$ and $\beta_0 = \frac{1}{5ekdh}$. There is a deterministic algorithm such that given any (k,d)-Hamiltonian $H = \sum_{j=1}^m H_j$ on n sites satisfying $\|H_j\| \le h$ and tensorized measurement $\mathcal{O}$, for any temperature $\beta \in \mathbb{D}_{(1-\delta)\beta_0}$ and error bound $\varepsilon \in (0,1)$, the algorithm outputs an estimation of $Z_{H,\mathcal{O}}(\beta)$ within ε-multiplicative error in $\widetilde{O}\left(\left(\frac{n}{\delta\varepsilon}\right)^C\right)$ time with $C = \frac{1}{\delta}\left(\ln 8ed + 3k \ln q\right) + 1$.*

It was established in [18] that any partition function $Z_H(\beta)$ exhibits zero-freeness property in a high-temperature regime (when the inverse temperature β is close to 0). A similar lemma holds for partition functions $Z_{H,\mathcal{O}}(\beta)$ under tensorized measurement $\mathcal{O}$.

▶ **Lemma 22** (high temperature zero-freeness)**.** *Let $h \in \mathbb{R}^+$, $H = \sum_{j=1}^m H_j$ be a (k,d)-Hamiltonian on n sites, and $\mathcal{O}$ be a tensorized measurement. If $\|H_j\| \le h$ for all $1 \le j \le m$, then for any $\beta \in \mathbb{D}_{\beta_0}$ where $\beta_0 = \frac{1}{5edkh}$, it holds that $\left|\log \frac{Z_{H,\mathcal{O}}(\beta)}{\mathbf{Tr}[\mathcal{O}]}\right| \le n$.*

Theorem 21 follows directly from Lemma 22 and Theorem 15.

The proof of Lemma 22 extends the zero-freeness analysis in [18] to the case where a tensorized measurement $\mathcal{O}$ is present. We will see that the same inductive proof based on cluster expansion works for this more general case.

Define H_X the Hamiltonian H restricted on X by

$$H_X = \sum_{\substack{i \in [m] \\ \mathrm{supp}(H_i) \subseteq X}} H_i,$$

and define the partition function $Z_{H,\mathcal{O}}$ restricted on X by $Z^X_{H,\mathcal{O}}(\beta) = \mathbf{Tr}_X[\exp(-\beta H_X)\mathcal{O}_X]$, where $O_X = \bigotimes_{v \in X} \mathcal{O}_v$, and $Z^{\varnothing}_{H,\mathcal{O}} = 1$ by convention. Here the subscript X in $\mathbf{Tr}_X$ indicates that the operators act on the sites in X.

Moreover, recall the dependency graph $G_H = (U, E, \mathcal{L})$ defined in the proof of Theorem 15:

1. $U = [m]$;
2. $E = \{(x,y) \in U \times U \mid x \ne y, \mathrm{supp}(H_x) \cap \mathrm{supp}(H_y) \ne \varnothing\}$;
3. $L_x = H_x$ for any $x \in U$.

We are now ready to introduce the cluster expansions of partition functions. The following lemma was an extension of [18, Lemma 26] with tensorized measurement $\mathcal{O}$.

▶ **Lemma 23** (high temperature expansion [18, Lemma 26]). *Let $h \in \mathbb{R}^+$, $H = \sum_{i=1}^m H_i$ be a (k,d)-Hamiltonian, $\mathcal{O}$ be a tensorized measurement and $\beta_0 = \frac{1}{\mathrm{e}(\mathrm{e}-1)dh}$. If $\|H_i\| \le h$, then the following holds for all $\Lambda \subseteq V$, $x \in \Lambda$ and $\beta \in \mathbb{D}_{\beta_0}$.*

$$Z^{\Lambda}_{H,\mathcal{O}}(\beta) = \mathbf{Tr}[\mathcal{O}_x] Z^{\Lambda\setminus\{x\}}_{H,\mathcal{O}}(\beta) + \sum_{\substack{S\subseteq[m]\\ \exists j\in S, x\in \mathrm{supp}(H_j)\\ G_H[S] \text{ is connected}}} W_S(\beta) Z^{\Lambda\setminus R_S}_{H,\mathcal{O}}(\beta),$$

where $W_S(\beta) = \sum_{l=|S|}^{+\infty} \frac{(-\beta)^l}{l!} \sum_{\substack{(i_1,\ldots,i_l)\in S^l\\ \cup_{j=1}^l \{i_j\}=S}} \mathbf{Tr}_{R_S}\left[\prod_{j=1}^l H_{i_j} \mathcal{O}_{R_S}\right]$, *and* $R_S = \bigcup_{j\in S} \mathrm{supp}(H_j)$.

We also need the following technical lemma from [18].

▶ **Lemma 24** (Harrow, Mehraban and Soleimanifar [18, Lemma 27]). *Let $H = \sum_{i=1}^m H_i$ be a (k,d)-Hamiltonian, and $\beta_0 = \frac{1}{5\mathrm{e}dkh}$, then for $|\beta| < \beta_0$*

$$\sum_{\substack{S\subseteq[m]\\ \exists j\in S, x\in \mathrm{supp}(H_j)\\ G_H[S] \text{ is connected}}} \left(\mathrm{e}^{|\beta|h} - 1\right)^{|S|} \exp(dh\mathrm{e}^2 |\beta| |R_S|) \le \mathrm{e}(\mathrm{e}-1)dh|\beta|.$$

Proof Sketch of Lemma 22. Fix an arbitrary $\Lambda \subseteq V$. As observed in [18], it suffices to prove that removal of any single site $x \in \Lambda$ can only produce a bounded additive overhead to $\log Z^{\Lambda}_{H,\mathcal{O}}(\beta)$. Formally, we are going to prove that when $|\beta| < \beta_0$, for any $x \in \Lambda$,

$$\left|\log\left|\frac{1}{\mathbf{Tr}[\mathcal{O}_x]} \frac{Z^{\Lambda}_{H,\mathcal{O}}(\beta)}{Z^{\Lambda\setminus\{x\}}_{H,\mathcal{O}}(\beta)}\right|\right| \le \mathrm{e}^2 dh|\beta|. \tag{15}$$

The proof is by induction on $|\Lambda|$. The induction basis with $|\Lambda| = 1$ is easy to establish. Now suppose that (15) holds for all smaller Λ. By Lemma 23 and Lemma 24, we have

$$\begin{aligned}
\left|\log\left|\frac{1}{\mathbf{Tr}[\mathcal{O}_x]} \frac{Z^{\Lambda}_{H,\mathcal{O}}(\beta)}{Z^{\Lambda\setminus\{x\}}_{H,\mathcal{O}}(\beta)}\right|\right| &= \left|\log\left|1 + \sum_{\substack{S\subseteq[m]\\ \exists j\in S, x\in \mathrm{supp}(H_j)\\ G_H[S] \text{ is connected}}} W_S(\beta) \left(\frac{1}{\mathbf{Tr}[\mathcal{O}_x]} \frac{Z^{\Lambda\setminus R_S}_{H,\mathcal{O}}(\beta)}{Z^{\Lambda\setminus\{x\}}_{H,\mathcal{O}}(\beta)}\right)\right|\right| \\
&\overset{(*)}{\le} -\log\left(1 - \sum_{\substack{S\subseteq[m]\\ \exists j\in S, x\in \mathrm{supp}(H_j)\\ G_H[S] \text{ is connected}}} |W_S(\beta)| \left(\frac{\exp\left(-\mathrm{e}^2 dh|\beta|\right)}{\mathbf{Tr}[\mathcal{O}_{R_S}]}\right)\right) \\
&\le -\log\left(1 - \sum_{\substack{S\subseteq[m]\\ \exists j\in S, x\in \mathrm{supp}(H_j)\\ G_H[S] \text{ is connected}}} \frac{(\exp(|\beta|h) - 1)^{|S|}}{\exp(\mathrm{e}^2 dh|\beta|)}\right) \\
&\le -\log(1 - \mathrm{e}(\mathrm{e}-1)dh|\beta|) \le \mathrm{e}^2 dh|\beta|,
\end{aligned}$$

where $(*)$ follows from the induction hypothesis and the last inequality follows from the fact that $-\log\left(1 - \frac{\mathrm{e}-1}{\mathrm{e}} y\right) \le y$ for all $y \in [0,1]$. ◀

Besides estimation of partition function, another related important computational problem is to sample according to the Gibbs state.

The quantum Gibbs state specified by Hamiltonian $H \in \mathcal{V}$ and inverse temperature $\beta \in \mathbb{R}^+$ is:

$$\rho_H(\beta) = \frac{\exp(-\beta H)}{Z_H(\beta)}.$$

The classical distribution $\mu_{H,\beta}$ over $[q]^V$ is the quantum Gibbs state $\rho_H(\beta)$ after measurement in the computational basis, i.e.

$$\forall \sigma \in [q]^V, \quad \mu_{H,\beta}(\sigma) = \frac{Z_{H,\mathcal{O}_\sigma}(\beta)}{Z_H(\beta)},$$

where $\mathcal{O}_\sigma = |\sigma\rangle\langle\sigma|$. Note that $\mu_{H,\beta}$ is a well-defined distribution over $[q]^V$. To see this, first note that $\sum_\sigma \mathcal{O}_\sigma = I$ is the identity matrix in $\mathcal{V}$, and hence $\sum_\sigma Z_{H,\mathcal{O}_\sigma}(\beta) = Z_H(\beta)$; and second, both $\mathcal{O}_\sigma$ and $\exp(\beta H)$ are positive semidefinite since H is Hermitian and $\beta \in \mathbb{R}^+$, and hence $Z_{H,\mathcal{O}_\sigma}(\beta) = \mathbf{Tr}\left[\exp(\beta H)\mathcal{O}_\sigma\right] \geq 0$.

In the same regime as Theorem 21, we have a polynomial-time approximate sampler from $\mu_{H,\beta}$, the classical distribution obtained after measurement of the quantum Gibbs state in the computational basis.

▶ **Theorem 25.** *Let $k, d \in \mathbb{N}^+$, $h > 0$, $\delta \in (0,1)$ and $\beta_0 = \frac{1}{5ekdh}$. There is a randomized algorithm such that given any (k,d)-Hamiltonian $H = \sum_{j=1}^m H_j$ on n sites satisfying $\|H_j\| \leq h$, for any temperature $\beta \in \mathbb{D}_{(1-\delta)\beta_0}$ and error bound $\varepsilon \in (0,1)$, the algorithm outputs an approximate sample $\sigma \in [q]^V$ within ε total variation distance from the distribution $\mu_{H,\beta}$, in $\widetilde{O}\left(\left(\frac{n}{\delta\varepsilon}\right)^C\right)$ time with $C = \frac{1}{\delta}\left(2\ln 8ed + 6k\ln q\right) + 3$.*

Proof. We leverage the algorithm in Theorem 21 as a subroutine, and give the following classical algorithm for approximate sampling from $\mu_{H,\beta}$.

Without loss of generality, we may assume that $V = [n]$. Let $\mathcal{M}_j = |j\rangle\langle j|$ for $1 \leq j \leq q$, and $\mathcal{M}_{v,j} = \left(\bigotimes_{\ell=1}^{v-1} I\right) \otimes \mathcal{M}_j \otimes \left(\bigotimes_{\ell=v+1}^{n} I\right)$. Our procedure for sampling $\sigma \in [q]^V$ is as follows.

1. Initialize $\mathcal{O}$ with the identity operator on Hilbert space $\mathcal{H}$;
2. Iterate v from 1 to n;
3. For each j from 1 to n, estimate $z_{v,j} = Z_{H,\mathcal{O}_{v-1}\mathcal{M}_{v,j}}(\beta)$ within $\varepsilon_0 = \frac{\varepsilon}{10n}$-multiplicative error.
4. samples $j \in [q]$ proportional to $\tilde{z}_{v,j}$, the estimation of $z_{v,j}$, updates $\mathcal{O}$ with $\mathcal{O}\mathcal{M}_{v,j}$, and assigns $\sigma(v)$ with j.

Note that $\mathcal{O}$ is a tensorized measurement during the process. Hence, Theorem 21 guarantees an estimation of $z_{v,j}$ within ε_0-multiplicative error in $\widetilde{O}(\left(\frac{n}{\varepsilon\delta}\right)^C)$ time with $C = \frac{1}{\delta}\left(2\ln 8ed + 6k\ln q\right) + 2$. Furthermore, note that for each configuration $\sigma \in [q]^V$,

$$\frac{\mathbf{Pr}[\sigma \text{ is generated}]}{\mu_{H,\mathcal{O}}(\sigma)} = \prod_{v=1}^{n} \frac{z_{v,\sigma(v)}}{\sum_{j\in[q]} z_{v,j}} \frac{\sum_{j=1}^q \tilde{z}_{v,j}}{\tilde{z}_{v,\sigma(v)}},$$

and for each $v \in V$ and $j \in [q]$,

$$1 - \varepsilon_0 \leq \frac{\tilde{z}_{v,j}}{z_{v,j}} \leq 1 + \varepsilon_0.$$

Hence,

$$1-\varepsilon \overset{(*)}{<} \left(\frac{1-\varepsilon}{1+\varepsilon}\right)^n \leq \frac{\mathbf{Pr}[\sigma \text{ is generated}]}{\mu_{H,\mathcal{O}}(\sigma)} \leq \left(\frac{1+\varepsilon_0}{1-\varepsilon_0}\right)^n \overset{(\star)}{<} 1+\varepsilon,$$

where $(\star)$ follows from $\left(\frac{1+\varepsilon_0}{1-\varepsilon_0}\right) \leq (1+3\varepsilon_0)^n < \exp(\frac{3}{10}\varepsilon) < 1+\varepsilon$, and $(*)$ follows from $(\star)$ and $\frac{1}{1+\varepsilon} > 1-\varepsilon$. Therefore, the total varaince distance between $\mu_{H,\mathcal{O}}$ and the output from our sampler will differ at most ε.

We conclude the proof by observing that our algorithm calls the subrountine $O(nq)$ times with parameter $\varepsilon_0 = \frac{\varepsilon}{10n}$, which takes $\widetilde{O}\left(\left(\frac{n}{\varepsilon}\right)^C\right)$ time with $C = \frac{1}{\delta}\left(2\ln 8ed + 6k\ln q\right) + 3$ in total. ◀

References

1. Alexander Barvinok. Computing the partition function for cliques in a graph. *Theory of Computing*, 11(13):339–355, 2015.
2. Alexander Barvinok. Computing the permanent of (some) complex matrices. *Foundations of Computational Mathematics*, 16(2):329–342, 2016.
3. Alexander Barvinok. Approximating permanents and hafnians. *Discrete Analysis*, page 1244, 2017.
4. Alexander Barvinok. *Combinatorics and Complexity of Partition Functions*. Springer Publishing Company, Incorporated, 1st edition, 2017.
5. Alexander Barvinok. Computing the partition function of a polynomial on the boolean cube. In *A Journey Through Discrete Mathematics*, pages 135–164. Springer, 2017.
6. Ivona Bezáková, Andreas Galanis, Leslie Ann Goldberg, and Daniel Štefankovič. Inapproximability of the independent set polynomial in the complex plane. *SIAM J. Comput.*, 49(5), 2020.
7. Ivona Bezáková, Andreas Galanis, Leslie Ann Goldberg, and Daniel Štefankovič. The complexity of approximating the matching polynomial in the complex plane. *ACM Trans. Comput. Theory*, 13(2):13:1–13:37, 2021.
8. Christian Borgs, Jennifer Chayes, Jeff Kahn, and László Lovász. Left and right convergence of graphs with bounded degree. *Random Struct. Algorithms*, 42(1):1–28, January 2013.
9. Pjotr Buys, Andreas Galanis, Viresh Patel, and Guus Regts. Lee-yang zeros and the complexity of the ferromagnetic ising model on bounded-degree graphs. In *SODA*, pages 1508–1519. SIAM, 2021.
10. Zongchen Chen, Kuikui Liu, and Eric Vigoda. Optimal mixing of glauber dynamics: Entropy factorization via high-dimensional expansion. In *STOC*, pages 1537–1550, 2021.
11. Zongchen Chen, Kuikui Liu, and Eric Vigoda. Spectral independence via stability and applications to holant-type problems. *arXiv preprint*, 2021. `arXiv:2106.03366`.
12. Matthew Coulson, Ewan Davies, Alexandra Kolla, Viresh Patel, and Guus Regts. Statistical physics approaches to unique games. In *35th Computational Complexity Conference (CCC)*, volume 169 of *LIPIcs*, pages 13:1–13:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
13. Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability for antiferromagnetic spin systems in the tree nonuniqueness region. *J. ACM*, 62(6):Art. 50, 60, 2015.
14. Sevag Gharibian, Yichen Huang, Zeph Landau, and Seung Woo Shin. Quantum hamiltonian complexity. *Found. Trends Theor. Comput. Sci.*, 10(3):159–282, 2015.
15. Leslie Ann Goldberg, Mark Jerrum, and Mike Paterson. The computational complexity of two-state spin systems. *Random Structures Algorithms*, 23(2):133–154, 2003.
16. Heng Guo, Chao Liao, Pinyan Lu, and Chihao Zhang. Zeros of holant problems: locations and algorithms. *ACM Transactions on Algorithms (TALG)*, 17(1):1–25, 2020.

17 Heng Guo, Jingcheng Liu, and Pinyan Lu. Zeros of ferromagnetic 2-spin systems. In *SODA*, pages 181–192. SIAM, 2020.

18 Aram W Harrow, Saeed Mehraban, and Mehdi Soleimanifar. Classical algorithms, correlation decay, and complex zeros of partition functions of quantum many-body systems. In *STOC*, pages 378–386, 2020.

19 Tyler Helmuth, Will Perkins, and Guus Regts. Algorithmic pirogov–sinai theory. *Probability Theory and Related Fields*, 176(3):851–895, 2020.

20 Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22(5):1087–1116, 1993.

21 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM (JACM)*, 51(4):671–697, 2004.

22 Alexei Yu Kitaev, Alexander Shen, Mikhail N Vyalyi, and Mikhail N Vyalyi. *Classical and quantum computation*. American Mathematical Soc., 2002.

23 Tomotaka Kuwahara, Kohtaro Kato, and Fernando GSL Brandão. Clustering of conditional mutual information for quantum gibbs states above a threshold temperature. *Physical review letters*, 124(22):220601, 2020.

24 Tsung-Dao Lee and Chen-Ning Yang. Statistical theory of equations of state and phase transitions. ii. lattice gas and ising model. *Physical Review*, 87(3):410, 1952.

25 Liang Li, Pinyan Lu, and Yitong Yin. Correlation decay up to uniqueness in spin systems. In *SODA*, pages 67–84. SIAM, 2013.

26 Jingcheng Liu, Alistair Sinclair, and Piyush Srivastava. Correlation decay and partition function zeros: Algorithms and phase transitions. *arXiv preprint*, 2019. `arXiv:1906.01228`.

27 Jingcheng Liu, Alistair Sinclair, and Piyush Srivastava. The ising partition function: Zeros and deterministic approximation. *Journal of Statistical Physics*, 174:287–315, 2019.

28 Ryan L Mann and Tyler Helmuth. Efficient algorithms for approximating quantum partition functions. *Journal of Mathematical Physics*, 62(2):022201, 2021.

29 Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM Journal on Computing*, 46(6):1893–1919, January 2017.

30 Han Peters and Guus Regts. On a conjecture of sokal concerning roots of the independence polynomial. *Michigan Math. J.*, 68(1):33–35, 2019.

31 Guus Regts. Zero-free regions of partition functions with applications to algorithms and graph limits. *Combinatorica*, 38(4):987–1015, 2018.

32 Jesús Salas and Alan D Sokal. Absence of phase transition for antiferromagnetic Potts models via the Dobrushin uniqueness theorem. *J. Stat. Phys.*, 86(3):551–579, 1997.

33 Shuai Shao and Yuxin Sun. Contraction: A unified perspective of correlation decay and zero-freeness of 2-spin systems. In *ICALP*, volume 168 of *LIPIcs*, pages 96:1–96:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

34 Alistair Sinclair, Piyush Srivastava, and Marc Thurley. Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. *Journal of Statistical Physics*, 155(4):666–686, 2014. (conference version in *SODA*'12).

35 Allan Sly. Computational transition at the uniqueness threshold. In *FOCS*, pages 287–296, 2010.

36 Allan Sly and Nike Sun. The computational hardness of counting in two-spin models on d-regular graphs. In *FOCS*, pages 361–369, 2012.

37 Daniel Štefankovič, Santosh S. Vempala, and Eric Vigoda. Adaptive simulated annealing: A near-optimal connection between sampling and counting. *J. ACM*, 56(3):18:1–18:36, 2009.

38 E.M. Stein and R. Shakarchi. *Complex Analysis*. Princeton lectures in analysis. Princeton University Press, 2010.

39 Dror Weitz. Counting independent sets up to the tree threshold. In *STOC*, pages 140–149, 2006.

Faster Cut-Equivalent Trees in Simple Graphs

Tianyi Zhang
Tel Aviv University, Israel

Abstract

Let $G = (V, E)$ be an undirected connected simple graph on n vertices. A cut-equivalent tree of G is an edge-weighted tree on the same vertex set V, such that for any pair of vertices $s, t \in V$, the minimum (s, t)-cut in the tree is also a minimum (s, t)-cut in G, and these two cuts have the same cut value. In a recent paper [Abboud, Krauthgamer and Trabelsi, STOC 2021], the authors propose the first subcubic time algorithm for constructing a cut-equivalent tree. More specifically, their algorithm has [1]$\widetilde{O}(n^{2.5})$ running time. Later on, this running time was significantly improved to $n^{2+o(1)}$ by two independent works [Abboud, Krauthgamer and Trabelsi, FOCS 2021] and [Li, Panigrahi, Saranurak, FOCS 2021], and then to $(m + n^{1.9})^{1+o(1)}$ by [Abboud, Krauthgamer and Trabelsi, SODA 2022].

In this paper, we improve the running time to $\widetilde{O}(n^2)$ graphs if near-linear time max-flow algorithms exist, or $\widetilde{O}(n^{17/8})$ using the currently fastest max-flow algorithm. Although our algorithm is slower than previous works, the runtime bound becomes better by a sub-polynomial factor in dense simple graphs when assuming near-linear time max-flow algorithms.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases graph algorithms, minimum cuts, max-flow

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.109

Category Track A: Algorithms, Complexity and Games

Related Version *arXiv Version*: `https://arxiv.org/abs/2106.03305`

Funding This publication has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV).

Acknowledgements The author would like to thank helpful discussions with Prof. Ran Duan and Dr. Amir Abboud.

1 Introduction

It is well known from Gomory and Hu [10] that any undirected graph can be compressed into a single tree while all pairwise minimum cuts are preserved exactly. More specifically, given any undirected graph $G = (V, E)$ on n vertices and m edges, there exists an edge weighted tree T on the same set of vertices V, such that: for any pair of vertices $s, t \in V$, the minimum (s, t)-cut in T is also a minimum (s, t)-cut in G, and their cut values are equal. Such trees are called Gomory-Hu trees or cut-equivalent trees. In the original paper [10], Gomory and Hu showed an algorithm that reduces the task of building a cut-equivalent tree to $n - 1$ max-flow instances. Gusfield [13] modified the original algorithm Gomory and Hu so that no graph contractions are needed when applying max-flow subroutines. So far, in weighted graphs, faster algorithms for building cut-equivalent trees were only byproducts of faster max-flow algorithms. In the recent decade, there has been a sequence of improvements on max-flows using the interior point method [16, 20, 19, 15, 6], and the current best running time is $\widetilde{O}(m + n^{1.5})$ by [6], so computing a cut-equivalent tree takes time $\widetilde{O}(mn + n^{2.5})$.

[1] $\widetilde{O}$ hides poly-logarithmic factors.

When G is a simple graph, several improvements have been made over the years. Bhalgat et al. [14] designed an $\widetilde{O}(mn)$ time algorithm for cut-equivalent trees using a tree packing approach based on [9, 7]. Recent advances include an upper bound of $O(m^{3/2}n^{1/6})$ by Abboud, Krauthgamer and Trabelsi [2], and in a subsequent work [3] by the same set of authors, they proposed the first subcubic time algorithm that constructs cut-equivalent trees in simple graphs, and their running time is $n^{2.5+o(1)}$. Recently, by two independent works [4, 18], this running time was improved to $n^{2+o(1)}$ which is almost-optimal in dense graphs, and further to a subquadratic time $(m+n^{1.9})^{1+o(1)}$ by [5].

All of these upper bounds rely on the current fastest max-flow algorithm with runtime $\widetilde{O}(m+n^{1.5})$. However, even if we assume the existence of a $\widetilde{O}(m)$-time max-flow algorithm, the above algorithms still have $n^{2+o(1)}$ running time in dense graphs which contains an extra sub-polynomial factor.

1.1 Our results

Let $\mathsf{MF}(m_0, n_0)$ be the running time complexity of max-flow computation in unweighted multi-graphs with m_0 edges and n_0 vertices, and let $\mathsf{MF}(m_0) = \mathsf{MF}(m_0, m_0)$ for convenience.

The main result of this paper is a near-quadratic time algorithm assuming existence of quasi-linear time max-flow algorithms. For a detailed comparison with recent published works, please refer to the table below where conditional runtime refers to the assumption of near-linear time max-flow algorithms.

▶ **Hypothesis 1.** $\mathsf{MF}(m_0, n_0) = \widetilde{O}(m_0 + n_0)$.

▶ **Theorem 2.** *Let $G = (V, E)$ be a simple on n vertices. Under Hypothesis 1, there is a randomized algorithm that constructs a cut-equivalent tree of G in $\widetilde{O}(n^2)$ time with high probability. Using the current fastest max-flow algorithm [6], the running time becomes $\widetilde{O}(n^{17/8})$.*

reference	conditional runtime	unconditional runtime
[3]	$\widetilde{O}(n^{2.5})$	$n^{2.5+o(1)}$
[4]	$n^{2+o(1)}$	$n^{2+o(1)}$
[18]	$n^{2+o(1)}$	$n^{2+o(1)}$
[5]	$(m+n^{1.75})^{1+o(1)}$	$(m+n^{1.9})^{1+o(1)}$
new	$\widetilde{O}(n^2)$	$\widetilde{O}(n^{17/8})$

Comparison with subsequent works. In a very recent but unpublished online preprint [1] (see also a note by [21]), significant progress has been made where an unconditional $\widetilde{O}(n^2)$ runtime has been achieved for cut-equivalent trees in general weighted graphs, which completely subsumes our result.

1.2 Technical overview

Our algorithm is largely based on the framework of [3]. In this subsection, we will discuss the running time bottlenecks of [3] and how to bypass them. For simplification, consider the following task. Let $\mathcal{T}$ be a partition tree which is an intermediate tree of the Gomory-Hu algorithm. Take an arbitrary node $N \subseteq V$ of $\mathcal{T}$ which represents a vertex subset of V. Let $G_{\mathcal{T}}[N] = (V_{\mathcal{T}}[N], E_{\mathcal{T}}[N])$ be the auxiliary graph obtained by contracting each component of $\mathcal{T} \setminus \{N\}$ into a single vertex in the original graph G.

Fix a pivot vertex $p \in N$, we want to find a sequence of vertices $v_1, v_2, \cdots, v_l \in N$, and compute a sequence of latest minimum cuts $(L_i, V_{\mathcal{T}}[N] \setminus L_i), 1 \leq i \leq l$ in $G_{\mathcal{T}}[N]$ for $(v_i, p), 1 \leq i \leq l$, where $v_i \in K_i, p \notin K_i$, such that:

(1) $l \geq \Omega(|N|)$.

(2) For each $1 \leq i \leq l$, $|L_i| \leq |N|/2$.

If we cut all sides $L_i \cap N$ off of N and form tree nodes, then by the above two properties all tree nodes are vertex subsets of N of size at most $|N|/2$. So, if we can recursively repeat this procedure on smaller subsets, then it would produce a cut-equivalent tree in logarithmically many rounds.

For this task, the basic idea of [3] is to apply expander decompositions. Suppose the original graph G is decomposed into disjoint clusters $V = C_1 \cup C_2 \cup \cdots$ such that each $G[C_i]$ is a ϕ-expander, and the total number of inter-cluster edges is bounded by $\widetilde{O}(\phi n^2)$. For simplicity let us assume G is a roughly regular graph and each vertex $v \in V$ has degree $\deg_G(v) \in [n/2, n)$. For each v, suppose $(L_v, V \setminus L_v)$ is the latest min-cut for (v, p), and let C_v be the ϕ-expander of the expander decomposition that contains v. Vertices of N are divided into three types.

1. Vertices in clusters whose size is less than $n/4$, namely $|C_v| < n/4$.
2. Vertices in clusters whose size is at least $n/4$, namely $|C_v| \geq n/4$, plus that $|C_v \cap L_v| \leq 10/\phi$. Note that there are only a constant number of such clusters.
3. $|C_v| \geq n/4$, plus that $|C_v \setminus L_v| \leq 10/\phi$.

The first bottleneck

To compute L_v for type-1 vertices, we simply go over all such v's, and compute the max-flow from v to p in $G_{\mathcal{T}}[N]$. Since each type-1 vertex must contribute $n/2 - n/4 = n/4$ inter-cluster edges as the input graph G is simple, the total number of type-1 vertices does not exceed $\widetilde{O}(\phi n)$, summing over all tree nodes N of $\mathcal{T}$.

For type-2 vertices, using the isolating cut lemma devised in [3, 17], we can compute all the sides L_v in $\widetilde{O}(\mathsf{MF}(\mathsf{vol}_G(N))/\phi)$ time, which sum to $\widetilde{O}(\mathsf{MF}(n^2)/\phi)$ over all nodes N of $\mathcal{T}$. So, under Hypothesis 1, the total time cost of type-1 and type-2 vertices is $\widetilde{O}(\phi n^3 + n^2/\phi)$, which is always larger than $n^{2.5}$. So in their algorithm [3], parameter ϕ is equal to $1/\sqrt{n}$.

Our observation is that applying max-flow for each type-1 vertex is too costly. To overcome this bottleneck, we simply avoid computing cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ for both type-1 and type-2 vertices. If the total number of type-2&3 vertices is larger than the total number of type-1 vertices, then we can skip all type-1 vertices. However, if the number of type-1 vertices dominates in N, then the number of type-2&3 vertices is at most $\widetilde{O}(\phi n)$ over all such kind of N. In this case, the total degree $\mathsf{vol}_G(N)$ is at most $\widetilde{O}(\phi n^2)$, and therefore, when summing over all nodes N of $\mathcal{T}$, computing all type-1 vertices takes time at most $\widetilde{O}(\phi^2 n^3)$, instead of $\widetilde{O}(\phi n^3)$, and so the new balance would be $\widetilde{O}(\phi^2 n^3 + n^2/\phi)$. Therefore, if we choose $\phi = n^{-1/3}$, it becomes $n^{7/3}$ which is already better than $n^{2.5}$. In the final algorithm, we will classify expander sizes using $\log n$ many different thresholds, instead of just one threshold (which is n/4 here), and so in the end we can set $\phi = 1/\log^{O(1)} n$. In general cases where graph G has various vertex degrees, we need to apply boundary-linked expander decomposition from a recent work [11]; especially we need to make use of property (3) in Definition 4.2 of [11].

The second bottleneck

In the work [3], in order to compute latest min-cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ for type-3 vertices, they consider the laminar family formed by all sides L_v. If the laminar family has tree depth at most k, then their algorithm can compute cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ by applying $k + 10/\phi$ max-flows in $G_{\mathcal{T}}[N]$. To ensure that the depth is bounded by k, they need a first randomly refine node N into $|N|/k$ sub-nodes which takes $|N|/k$ Gomory-Hu steps. Hence, in total, it requires at least $k + |N|/k > \sqrt{|N|}$ max-flow invocations, which leads to a $n^{2.5}$ running time under Hypothesis 1.

To bypass this barrier, the observation is that the depth of the laminar family in each ϕ-expander is already small, so actually we do not need the help from the refinement step. More precisely, instead of looking at the entire laminar family formed by sets $\{L_v\}_{v\in N}$, we only look at the laminar family formed by sets $\{C_v \cap L_v\}_{v\in N}$ for each cluster C. It can be proved that the depth of this smaller laminar family is always bounded by $O(1/\phi)$. In the end, to compute latest min-cuts $(L_v, V_{\mathcal{T}}[N] \setminus L_v)$ for all type-2&3 vertices, we will only use $O(1/\phi)$ max-flow instances in total.

2 Preliminaries

Let $G = (V, E)$ be an arbitrary simple graph on n vertices and m edges with unit-capacities. For any $v \in V$, let $\deg_G(v)$ be the number of its neighbors in V. For any subset $S \subseteq V$, define $\mathsf{vol}_G(S) = \sum_{v\in S} \deg_G(v)$, and let $\mathsf{out}_G(S)$ count the number of edges in $E \cap (S \times (V \setminus S))$, and define $G[S]$ to be the induced subgraph of S on G.

Introduced in [8], the latest minimum (s, t)-cut is a minimum (s, t)-cut such that the side containing s has minimum size as well. It is proved that latest minimum cuts are unique, and can be computed by any max-flow algorithm for (s, t).

Here are some basic facts about min-cuts.

▶ **Lemma 3** (Lemma 2.8 in [3]). *For any vertices $a, b, p \in V$, assume $(A, V\setminus A)$ and $(B, V\setminus B)$ are min-cuts for $(a, p), (b, p)$ respectively. If $b \in A$, then $(A \cup B, V \setminus (A \cup B))$ is a min-cut for (a, p) as well.*

▶ **Lemma 4** (Lemma 2.9 in [3]). *For any vertices $a, b, p \in V$, assume $(A, V\setminus A)$ and $(B, V\setminus B)$ are min-cuts for $(a, p), (b, p)$ respectively. If $a \notin B$ and $b \notin A$, then $(A \setminus B, V \setminus (A \setminus B))$ is a min-cut for (a, p).*

2.1 Cut-equivalent trees

A cut-equivalent tree is a tree $\mathcal{T}$ on V with weighted edges, such that for any pair $s, t \in V$, there is a minimum cut $(S, V \setminus S)$ in G such that it is also a minimum cut in $\mathcal{T}$ with the same cut value. Now let us turn to define some terminologies for cut-equivalent trees.

Partition trees

A partition tree $\mathcal{T}$ of graph G is a tree whose nodes $U_1, U_2, \cdots, U_l$ represent disjoint subsets of V such that $V = U_1 \cup U_2 \cup \cdots \cup U_l$. For each node U of $\mathcal{T}$, the auxiliary graph $G_{\mathcal{T}}[U] = (V_{\mathcal{T}}[U], E_{\mathcal{T}}[U])$ of U is built by contracting each component of $\mathcal{T} \setminus \{U\}$ into a single vertex in the original graph G.

Gomory-Hu algorithm

Gomory-Hu algorithm provides a flexible framework for constructing a cut-equivalent tree. The algorithm begins with a partition tree $\mathcal{T}$ which is the single node that subsumes the entire vertex set V, and creates more nodes iteratively by refining its nodes. In each iteration, the algorithm picks an arbitrary node that represent a non-singleton subset $U \subseteq V$, and selects two arbitrary vertex $s, t \in U$. Then, compute the minimum cut $(S, V_{\mathcal{T}}[U] \setminus S)$ between s, t in the auxiliary graph $G_{\mathcal{T}}[U]$. Finally, split node U into two nodes that correspond to subsets $S \cap U$ and $(V_{\mathcal{T}}[U] \setminus S) \cap U$ respectively, connected by an edge with weight equal to the value of the cut $(S, V_{\mathcal{T}}[U] \setminus S)$ in $G_{\mathcal{T}}[U]$. For each node W that was U's neighbor on $\mathcal{T}$, reconnect W to node $S \cap U$ if S contains the contracted node that subsumes W; otherwise reconnect W to node $(V_{\mathcal{T}}[U] \setminus S) \cap U$.

A tree is called **GH-equivalent**, if it is a partition tree that can be constructed during Gomory-Hu algorithm by certain choice of nodes U to split and pairs of vertices $s, t \in U$.

Refinement with respect to subsets

Consider a partition tree $\mathcal{T}$ which is GH-equivalent. Let U be one of $\mathcal{T}$'s node and let $R \subseteq U$ be a subset. A refinement of $\mathcal{T}$ with respect to R is to repeatedly execute a sequence of Gomory-Hu iterations by always picking two different vertices from $s, t \in R$ that are currently in the same node of $\mathcal{T}$ and refine $\mathcal{T}$ using a minimum (s, t)-cut. So after the refinement of $\mathcal{T}$ with respect to R, $\mathcal{T}$ is still GH-equivalent.

▶ **Lemma 5** ([12]). *For any node U of $\mathcal{T}$ and any subset $R \subseteq U$, a refinement of $\mathcal{T}$ with respect to R can be computed in time $\widetilde{O}(|R| \cdot \mathsf{MF}(\mathsf{vol}_G(U)))$.*

▶ **Lemma 6** (see definition of partial trees in [3]). *After the refinement on a GH-equivalent tree $\mathcal{T}$ with respect to R, for any $a, b \in R$, let $N_a \ni a$ and $N_b \ni b$ be nodes of $\mathcal{T}$. Then, the min-cut of (N_a, N_b) in $\mathcal{T}$ is a min-cut in G for (a, b) as well.*

k-partial trees

A k-partial tree $\mathcal{T}$ is a GH-equivalent partition tree such that all vertices $u \in V$ such that $\deg_G(u) \leq k$ are singletons of $\mathcal{T}$. The following lemma states that k-partial trees always exist and can be computed efficiently for small k's.

▶ **Lemma 7** ([14]). *There is an algorithm that, given an undirected graph with unit edge capacities and parameter k on n vertices, computes a k-partial tree in time $\min\{\widetilde{O}(nk^2), \widetilde{O}(mk)\}$.*

2.2 Expander decomposition

For any pair of disjoint sets $S, T \subseteq V$, let $E_G(S, T)$ be the set of edges between S, T in G. The conductance of a cut $(S, V \setminus S)$ is $\Phi_G(S) = |E_G(S, V \setminus S)| / \min\{\mathsf{vol}_G(S), \mathsf{vol}_G(V \setminus S)\}$, and the conductance of a graph G is defined as $\Phi_G = \min_S \Phi_G(S)$. A graph G is a ϕ-expander if $\Phi_G \geq \phi$.

For any vertex subset $S \subseteq V$ and positive value $x > 0$, let $G[S]^x$ be the subgraph induced on S where we add $\lceil x \rceil$ self-loops to each vertex $v \in S$ for every boundary edge $(v, w), w \notin S$. As an example, in graph $G[S]^1$ the degrees of all vertices are the same as in the original graph G.

We need a strong expander decomposition algorithm from a recent work [11].

▶ **Definition 8** (boundary-linked expander decomposition [11]). *Let $G = (V, E)$ be a graph on n vertices and m edges, and $\alpha, \phi \in (0, 1)$ be parameters. An (α, ϕ)-expander decomposition of V consists of a partition $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ of V such that the following holds.*
(1) $\sum_{i=1}^{k} \mathsf{out}_G(C_i) \leq \log^4 n \cdot \phi m$.
(2) *For any i, $G[U_i]^{\alpha/\phi}$ is a ϕ-expander.*
(3) *For any i, $\mathsf{out}_G(C_i) \leq \log^7 n \cdot \phi \mathsf{vol}_G(C_i)$.*
In the original paper [11], the upper bounds are stated with $O(\cdot)$ notations that hide constant factors; here we simply raise the exponent of log-factors to simplify the notations.

▶ **Lemma 9** ([11]). *Given any unweighted graph $G = (V, E)$ on n vertices and m edges, for any $\alpha, \phi \in (0, 1)$ such that $\alpha \leq 1/\log^{\mathsf{c}} m$ where c is a certain constant, a (α, ϕ)-expander decomposition can be computed in $\widetilde{O}(m/\phi)$ with high probability.*

2.3 Isolating cuts

▶ **Lemma 10** (isolating cuts [3]). *Given an undirected edge-weighted graph $H = (X, F, \omega)$, a pivot vertex $p \in X$, and a set of terminal vertices $T \subseteq X$. For each $u \in T$, let $(K_u, X \setminus K_u)$ be the latest minimum (u, p)-cut for each $u \in T$. Then, in time $\widetilde{O}(\mathsf{MF}(|F|, |X|))$ we can compute $|T|$ disjoint sets $\{K'_u\}_{u \in T}$ such that for each $u \in T$, if $K_u \cap T = \{u\}$ then $K'_u = K_u$.*

3 Quadratic time Gomory-Hu tree under Hypothesis 1

3.1 The main algorithm

In this section we try to prove the first half of Theorem 2. Let $G = (V, E)$ denote the simple graph as our input data. Define some parameters: $\phi = \frac{1}{10 \log^{\mathsf{c}+10} n}$ is a global conductance parameter that is used to construct expander decompositions, and $r = 10 \log^5 n$ is a sampling parameter which is needed when choosing pivots; here c is the same constant as in Definition 8. Without loss of generality, assume $\sqrt{n}$ is an integral power of 2. Define a degree set $\mathcal{D} = \{\sqrt{n}, 2\sqrt{n}, 2^2\sqrt{n}, \cdots, n\}$.

Preparation

Throughout the algorithm, $\mathcal{T}$ will be the cut-equivalent tree under construction, where each of $\mathcal{T}$'s node will represent a subset of vertices of V. As a preparation step, compute a (ϕ, ϕ)-expander decomposition on G and obtain a partitioning $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ of V. Categorize clusters in $\mathcal{C}$ according to their sizes: for each 2^i, define $\mathcal{C}_i$ to be the set of clusters whose sizes are within interval $[2^i, 2^{i+1})$.

At the beginning, initialize $\mathcal{T}$ to be a $\sqrt{n}$-partial tree by applying the algorithm from [14] that takes running time $\widetilde{O}(n^2)$.

Iteration

In each round, we will divide simultaneously all nodes of $\mathcal{T}$ which contains at least $20r$ vertices in V. In the end, the total number of rounds will be bounded by $\widetilde{O}(1)$. To describe our algorithm, let us focus on any single node $N \subseteq V$ of $\mathcal{T}$ whose size $|N|$ is at least $20r$. The first step is to refine the partition of N by a set of random pivots. More specifically, sample a pivot subset $R \subseteq N$ of size $10r$ by picking each vertex with probability proportional to its degree in G; more precisely, repeatedly sample for $10r$ times a vertex from N where each vertex $v \in N$ is selected with probability $\deg_G(v)/\mathsf{vol}_G(N)$.

Then, refine the node N of $\mathcal{T}$ by computing a partial tree with respect to R using Lemma 5, which further divides N into several subsets each containing a distinct vertex from R. After applying this pivot-sampling & refining step to each of the original node of $\mathcal{T}$, $\mathcal{T}$ has undergone one pass of partition, and now each node U of $\mathcal{T}$ is associated with a unique pivot vertex $p \in U \cap R$.

For the rest, let us focus on each node U of the current $\mathcal{T}$ as well as its pivot p, such that $U \subseteq N$ is a subdivision of the previous node N but $\mathsf{vol}_G(U) \geq 0.5\mathsf{vol}_G(N)$. For each $k \in \mathcal{D}$, define $U_k = \{u \in U \mid \deg_G(u) \in [k, 2k)\}$, so $U = \bigcup_{k\in\mathcal{D}} U_k$. Take a parameter $d \in \mathcal{D}$ such that $d|U_d|$ is maximized, namely $d \in \arg\max_{k\in\mathcal{D}} k|U_k|$. Therefore, $2d|U_d| \geq \mathsf{vol}_G(U)/\log n$. Next, for each index i, define $\mathsf{cnt}[i]$ to be the number of vertices from U_d that lie within clusters from $\mathcal{C}_i$. Take $s = 2^{i_U}$ such that $\mathsf{cnt}[i_U]$ is maximized.

For each cluster $C \in \mathcal{C}_{i_U}$ such that $C \cap U_d$ is nonempty, conduct the expander search routine described in the next subsection (Algorithm 2) to compute cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ in the auxiliary graph $G_{\mathcal{T}}[U]$ for a set W_C of vertices $u \in W_C \subseteq C \cap U_d$ with respect to pivot p; so $u \in K_u, p \in V_{\mathcal{T}}[U] \setminus K_u$. It will be guaranteed that for each $u \in W_C$, we have $\mathsf{vol}_G(K_u \cap U) \leq 0.5\mathsf{vol}_G(N)$. In the end, define $W = \bigcup_{C\in\mathcal{C}_{i_U}} W_C$.

We will prove that all $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ are latest min-cuts for (u, p). Since all latest minimum cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ are with respect to p, they should form a laminar family. Then, for each $u \in W$ such that K_u is maximal in the laminar family, split $K_u \cap U$ off the node U and create a new node for vertex set $K_u \cap U$. Since we always take maximal K_u's, all of these sets are disjoint in $V_{\mathcal{T}}[U]$, so the creation of new nodes on $\mathcal{T}$ is well-defined. Pseudo-code CondGomoryHu summarizes our algorithm.

Algorithm 1 CondGomoryHu$(G = (V, E))$.

initialize a partition tree $\mathcal{T}$, as well as parameters ϕ, r;
while $\exists N \subseteq V$, U *a node of* $\mathcal{T}$, $|N| \geq 20r$ **do**
  for *node* N *of* $\mathcal{T}$ *with* $|N| \geq 20r$ **do**
    repeat for $10r$ times: each time we sample a vertex $u \in N$ with probability $\frac{\deg_G(u)}{\mathsf{vol}_G(N)}$, and let the sampled set be R;
    call Lemma 5 on node N with respect to R;
    for *node* $U \subseteq N$ *of* $\mathcal{T}$ *such that* $\mathsf{vol}_G(U) > 0.5\mathsf{vol}_G(N)$ **do**
      take $d \in \mathcal{D}$ such that $d|U_d|$ is maximized;
      take $s = 2^{i_U}$ such that $\mathsf{cnt}[i_U]$ is maximized;
      for *each* $C \in \mathcal{C}_{i_U}$ **do**
        run expander search on C within node U to compute a subset $W_C \subseteq C \cap U_d$, and the latest min-cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ for each $u \in W_C$;
      define $W = \bigcup_{C\in\mathcal{C}_{i_U}} W_C$;
      for each $u \in W$ such that K_u is maximal, split $K_u \cap U$ off of U and create a new node on $\mathcal{T}$;
for *node* N *of* $\mathcal{T}$ *such that* $|N| < 20r$ **do**
  repeatedly refine N using the generic Gomory-Hu steps until all nodes are singletons;
return $\mathcal{T}$ as a cut-equivalent tree;

3.2 Finding latest min-cuts in expanders

Our algorithm is similar to the one from [3]. The input to this procedure is a node $U \subseteq V$ of the current partition tree $\mathcal{T}$ under construction, together with parameters s, d defined previously, as well as an expander $C \in \mathcal{C}_{i_U}$ that intersects U_d. The output of this procedure will be a subset $W_C \subseteq C \cap U_d$, and their cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ for all $u \in W_C$ in the auxiliary graph $G_{\mathcal{T}}[U]$, with the extra property that $\mathsf{vol}_G(K_u \cap U) \leq 0.5\mathsf{vol}_G(N)$. In the end, we will show that with high probability, all these cuts are latest min-cuts in $G_{\mathcal{T}}[U]$.

To describe our algorithm, consider all vertices $u \in C \cap U_d$ and their latest minimum cuts $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ with respect to pivot $p \in U$. Let λ_u be the cut value of $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ in $G_{\mathcal{T}}[U]$. All of the sets $L_u \cap C \cap U_d$ should form a laminar family, which corresponds to a tree structure $\mathcal{T}_p^d[C]$, where each tree node M of $\mathcal{T}_p^d[C]$ packs a subset of $C \cap U_d$, such that for all $u \in M$ the set $L_u \cap C \cap U_d$ are the same. More specifically, $\mathcal{T}_p^d[C]$ is constructed as follows: first arrange the laminar family $\{L_u \cap C \cap U_d\}_{u \in C \cap U_d}$ as a tree, and then for each node $L_u \cap C \cap U_d$ on this tree, associate with this node the set $M = \{v \in C \cap U_d \mid L_v \cap C \cap U_d = L_u \cap C \cap U_d\} \subseteq L_u \cap C \cap U_d$.

We need to emphasize that our algorithm does not know $\mathcal{T}_p^d[C]$ at the beginning, but it will gradually explore part of $\mathcal{T}_p^d[C]$ during the process.

Preparation

Initialize $W_C = \emptyset$. Assume $|C \cap U_d| \geq 10/\phi^2$; otherwise we could simply reset $W_C = C \cap U_d$ and run $|C \cap U_d|$ instances of max-flow to compute all latest cuts. As a preparatory step, the algorithm repeatedly takes a random subset $T \subseteq C \cap U_d$ by selecting each vertex independently with probability ϕ. Then apply Lemma 10 on graph $G_{\mathcal{T}}[U]$ to compute isolating cuts of terminal vertices from T with respect to pivot p. This procedure goes on for $10 \log n/\phi$ iterations, and for each $u \in C \cap U_d$, let $(A_u^i, V_{\mathcal{T}}[U] \setminus A_u^i)$ be the isolating cut computed for u in the i-th iteration; if u was not selected by T in the i-th iteration, simply set $A_u^i = \{u\}$. Finally, let A_u be the set among $\{A_u^i\}_{1 \leq i \leq 10 \log n/\phi}$ such that the cut value of $(A_u, V_{\mathcal{T}}[U] \setminus A_u)$ in the auxiliary graph $G_{\mathcal{T}}[U]$ is minimized; to break ties, we select A_u^i that minimizes $|A_u^i|$. Let κ_u be the cut value of $(A_u, V_{\mathcal{T}}[U] \setminus A_u)$ for $u \in C \cap U_d$.

If one of $|A_u \cap C \cap U_d| > 2/\phi$, then the algorithm fails and aborts; we will prove that the failure probability is small.

Exploring $\mathcal{T}_p^d[C]$

A node M of $\mathcal{T}_p^d[C]$ is called **large** if $|C \cap U_d \setminus L_u| \leq 2/\phi$ for any $u \in M$, and if $|L_u \cap C \cap U_d| \leq 2/\phi$ it is called **small**. Similarly, a vertex $u \in C \cap U_d$ is called large, if $|C \cap U_d \setminus L_u| \leq 2/\phi$; otherwise if $|L_u \cap C \cap U_d| \leq 2/\phi$, it is called small.

▶ **Lemma 11.** *All large nodes on $\mathcal{T}_p^d[C]$ should lie on a single path ended at root.*

Proof. Suppose otherwise there exists two different large nodes M_1, M_2 of $\mathcal{T}_p^d[C]$ such that $M_1 \cap M_2 = \emptyset$. Take any $u_1 \in M_1, u_2 \in M_2$. Since M_1, M_2 are large nodes, we know $|C \cap U_d \setminus L_{u_1}| \leq 2/\phi$, $|C \cap U_d \setminus L_{u_2}| \leq 2/\phi$. As $C \cap U_d \cap L_{u_1}$ and $C \cap U_d \cap L_{u_2}$ are disjoint, we have $|C \cap U_d| \leq |C \cap U_d \setminus L_{u_1}| + |C \cap U_d \setminus L_{u_2}| \leq 4/\phi$, which contradicts that $|C \cap U_d| \geq 10/\phi^2$. ◀

The main idea of our algorithm is to find the lowest large node on $\mathcal{T}_p^d[C]$; to clarify a bit more, here "lowest" means farthest from root. Initialize a set $S \leftarrow C \cap U_d$ and maintain an ordering of vertices in S according to the cut value of κ_u, also initialize variable $Q \leftarrow \emptyset$.

Repeat the following procedure: take $u \in S \setminus Q$ such that κ_u is maximized. Apply max-flow in graph $G_{\mathcal{T}}[U]$ to compute the latest min-cut L_u for (u, p). Consider two possibilities.

- L_u is small. Then assign $S \leftarrow S \setminus (Q \cup L_u)$, and $Q \leftarrow \emptyset$.
- L_u is large. If $L_u \cap C \cap U_d = S$, then add u to Q; otherwise if $L_u \cap C \cap U_d \neq S$, reset $S \leftarrow L_u \cap C \cap U_d$ and $Q \leftarrow \{u\}$.

The repetition terminates if either (1) $|C \cap U_d \setminus S| > 2/\phi$ or (2) $|Q| > 2/\phi$. In the first case, assign $W_C \leftarrow \{u \in S \mid \mathsf{vol}_G(A_u \cap U) \leq 0.5\mathsf{vol}_G(N)\}, K_u \leftarrow A_u$ and terminate. Note that this notation $\mathsf{vol}_G(A_u \cap U)$ is well-defined, since all vertices in U are also vertices in V, not contracted vertices in $V_{\mathcal{T}}[U]$.

Now suppose we are in the second case. Let $L = L_v$ for an arbitrary $v \in Q$. We will prove afterwards that $(L, V_{\mathcal{T}}[U] \setminus L)$ is the latest min-cut corresponding to the lowest large node. Let κ be the cut value of $(L, V_{\mathcal{T}}[U] \setminus L)$, and define $B = \{u \in S \mid \kappa_u > \kappa\}$. Assign $W_C \leftarrow \{u \in S \setminus B \mid \mathsf{vol}_G(A_u \cap U) \leq 0.5\mathsf{vol}_G(N)\}, K_u \leftarrow A_u$ and for each $u \in W_C$.

After that, take an arbitrary $v \in Q$. If it satisfies that $\mathsf{vol}_G(L \cap U) \leq 0.5\mathsf{vol}_G(N)$, then update $W_C = W_C \cup \{v\}$ and $K_v \leftarrow L$. The whole exploration procedure is summarized as pseudo-code ExploreTree.

Algorithm 2 ExploreTree(U, p, d, C).

```
prepare A_u and κ_u for all u ∈ C ∩ U_d;
initialize S ← C ∩ U_d, Q ← ∅;
while max{|C ∩ U_d \ S|, |Q|} ≤ 2/φ do
    take u ∈ arg max_{v∈S\Q}{κ_v};
    apply max-flow to compute L_u;
    if u is small then
        S ← S \ (Q ∪ L_u), and Q ← ∅;
    else
        if L_u ∩ C ∩ U_d = S then
            Q ← Q ∪ {u};
        else
            S ← L_u ∩ C ∩ U_d, Q ← {u};
if |C ∩ U_d \ S| > 2/φ then
    return W_C ← {u ∈ S | vol_G(A_u ∩ U) ≤ 0.5vol_G(N)}, K_u ← A_u, ∀u ∈ W_C;
else
    define B = {u ∈ S | κ_u > κ} where κ is the cut value of (S, V_T[U] \ S);
    W_C ← {u ∈ S \ B | vol_G(A_u ∩ U) ≤ 0.5vol_G(N)}, K_u ← A_u, ∀u ∈ W_C;
    draw an arbitrary vertex v ∈ Q and set L = L_v;
    if vol_G(L ∩ U) ≤ 0.5vol_G(N) then
        assign W_C ← W_C ∪ {v} and K_v ← L;
    return W_C;
```

3.3 Proof of correctness

First we prove a basic property of isolating cuts, which is also used in [3].

▶ **Lemma 12** ([3]). *For each $u \in C \cap U_d$, either $|L_u \cap C \cap U_d| \leq 2/\phi$ or $|C \cap U_d \setminus L_u| \leq 2/\phi$; namely each vertex is either large or small. Furthermore, with high probability, when u is small, $A_u = L_u$.*

Proof. Since $\deg_G(u) < 2d$, the cut value of $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is smaller than $2d$. Unpack all contracted vertices of $V_{\mathcal{T}}[U]$, and let $L'_u \subseteq V$ be the set of vertices belonging to L_u or contracted in L_u. Therefore, since $G_{\mathcal{T}}[U]$ is a contracted graph of G, the cut value of $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is equal to the cut value of $(L'_u, V \setminus L'_u)$.

Suppose otherwise that $|L'_u \cap C \cap U_d| > 2/\phi$ and $|C \cap U_d \setminus L'_u| > 2/\phi$. Then, by property (2) of the (ϕ, ϕ)-expander decomposition, $G[C]^1$ is a ϕ-expander, and so the number of edges between $L'_u \cap C$ and $C \setminus L'_u$ is at least $\phi \cdot \min\{\mathsf{vol}_G(L'_u \cap C), \mathsf{vol}_G(C \setminus L'_u)\} > 2d$, which is a contradiction.

Let us turn to the second half of the statement. Suppose u is small, and so $|L_u \cap C \cap U_d| \leq 2/\phi$. Then, since T selects each vertex in $C \cap U_d$ with probability ϕ, with probability $\phi \cdot (1-\phi)^l > \phi/8$, $T \cap L_u \cap C = \{u\}$ is a singleton. In this case, by Lemma 10, $A^i_u = L_u$. As T is sampled for $10 \log n/\phi$ times, with high probability $A_u = L_u$. ◀

Here is a basic fact regarding large vertices.

▶ **Lemma 13.** *For any large vertex u, $\lambda_u < \kappa_u$.*

Proof. If $\lambda_u = \kappa_u$, then the latest min-cut should be contained in A_u, which contains at most $2/\phi$ vertices from $C \cap U_d$, and so u cannot be large. ◀

Next we analyze the behavior of the while-loop in ExploreTree.

▶ **Lemma 14.** *If $Q \neq \emptyset$, then for each $u \in Q$, $L_u \cap C \cap U_d = S$.*

Proof. Each time S is updated, either Q adds a vertex u on line-10 such that $L_u \cap C \cap U_d = S$, or Q is updated to $\{u\}$ on line-12. So the equality always holds. ◀

▶ **Lemma 15.** *At the beginning of any iteration of the while-loop, $\forall v \in S$, if v is large, then we have $L_v \cap C \cap U_d \subseteq S$.*

Proof. We prove this statement by induction on the number of iterations. Initially, this holds as $S = C \cap U_d$. For any intermediate iteration, consider two cases.

- u is small. We claim that before updating S, for all large vertices $v \in S \setminus (Q \cup L_u)$, L_v and $L_u \cup Q$ are disjoint; if this can be proved, then we conclude $L_v \cap C \cap U_d \subseteq S \setminus (Q \cup L_u)$, as $L_v \cap C \cap U_d \subseteq S$ holds before.
  Suppose that $L_v \cap L_u \neq \emptyset$. Then as all latest minimum cuts form a laminar family and that $v \notin L_u$, it must be $L_u \subseteq L_v$. As v is large, $L_v \cap C \cap U_d$ contains more vertices than $A_v \cap C \cap U_d$, and so by Lemma 13, we have $\lambda_v < \kappa_v$. Now, by line-4, since κ_u is the largest among all vertices in $S \setminus Q$, $\kappa_v \leq \kappa_u$. Finally, using Lemma 12, we know $\kappa_u = \lambda_u$ as u is small. Concatenating all the inequalities we have:

  $$\lambda_v < \kappa_v \leq \kappa_u = \lambda_u$$

  which contradicts the fact that $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is a min-cut for (u, p) as $L_u \subseteq L_v$.
  Now suppose that $L_v \cap Q \neq \emptyset$, say $w \in L_v \cap Q$. Then by Lemma 14, $v \in S \subseteq L_w$, and so both w, v are in $L_v \cap L_w$, which means $L_v = L_w$, and so $L_v \cap L_u = L_w \cap L_u = L_u \neq \emptyset$, which is a contradiction as discussed just before.

- u is large. In this case, the algorithm would reassign $S \leftarrow L_u \cap C \cap U_d$. Then, for all $v \in S$, as $(L_v, V_{\mathcal{T}}[U] \setminus L_v)$ is the latest minimum cut, it must be $L_v \subseteq L_u$, irrespective of whether v is large or not. ◀

Next we prove that when the while-loop ends, either all vertices in S are small, or S corresponds to the cut of the lowest large node on $\mathcal{T}_p^d[C]$.

▶ **Lemma 16.** *If $|C \cap U_d \setminus S| > 2/\phi$, then all vertices in S are small.*

Proof. Consider any vertex $u \in S$. If u is large, then by Lemma 15, $L_u \cap C \cap U_d \subseteq S$, and so $|C \cap U_d \setminus L_u| \geq |C \cap U_d \setminus S| > 2/\phi$, which contradicts the definition of being large. ◀

▶ **Lemma 17.** *After the while-loop ends, if $|C \cap U_d \setminus S| \leq 2/\phi$ and $|Q| > 2/\phi$, then $(L, V_{\mathcal{T}}[U] \setminus L)$ is the latest min-cut of the lowest large node on $\mathcal{T}_p^d[C]$. Moreover, $B = \{u \in S \mid \kappa_u > \kappa\}$ is the set of all vertices u such that $L_u \cap C \cap U_d = S$, and consequently all $L_u, \forall u \in B$ are equal.*

Proof. As the while-loop ends with $|Q| > 2/\phi$, the last iteration must have ended on line-10. Therefore, $(L, V_{\mathcal{T}}[U] \setminus L)$ is the latest min-cut of some $u \in Q$. Suppose otherwise $(L, V_{\mathcal{T}}[U] \setminus L)$ is not the latest min-cut of the lowest large node on the imaginary tree $\mathcal{T}_p^d[C]$. Then, there exists a large vertex $v \in L \cap C \cap U_d$ such that $L_v \cap C \cap U_d \subsetneq S$ but $|C \cap U_d \setminus L_v| \leq 2/\phi$. As $|Q| > 2/\phi$, there must exist $w \in L_v \cap Q$. By Lemma 14, $v \in L \cap C \cap U_d = S = L_w \cap C \cap U_d$, so both v, w are in $L_v \cap L_w$, and consequently $L_v = L_w$, $L_v \cap C \cap U_d = S$, contradiction.

Now let us turn to the second half of the statement. Consider any $u \in B$. $(A_u, V_{\mathcal{T}}[U] \setminus A_u)$ cannot be a min-cut as $\kappa_u > \kappa$. By Lemma 12, u must be a large vertex. On the one hand, by Lemma 15, $L_u \cap C \cap U_d \subseteq S$, and on the other hand, $L_u \cap C \cap U_d$ cannot be strictly smaller than S as S is the lowest already. Hence $L_u \cap C \cap U_d = S$.

For any $u \notin B$, by definition $\kappa_u \leq \kappa$. If u is large, then $\lambda_u < \kappa_u \leq \kappa$, so $L_u \cap C \cap U_d \subsetneq S$, which also contradicts that S corresponds to the lowest large node on $\mathcal{T}_p^d[C]$. ◀

Finally, we prove that all cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ output by the algorithm are latest min-cuts with high probability.

▶ **Lemma 18.** *All cuts $(K_u, V_{\mathcal{T}}[U] \setminus K_u)$ output by the algorithm are latest min-cuts with high probability.*

Proof. If the algorithm terminates on line-14, then by Lemma 16, all vertices in W_C are small. So by Lemma 12, $L_u = A_u = K_u, \forall u \in W_C$. Otherwise, if the algorithm terminates on line-21, then by Lemma 17, all vertices in W_C are small. Hence, by Lemma 12, $L_u = A_u = K_u, \forall u \in W_C \setminus B$; also, for any $u \in W_C \cap B$, we have $L_u = L = K_u$. ◀

3.4 Running time analysis

First we analyze the running time of each call of expander search.

▶ **Lemma 19.** *The total running time of the expander search in graph $G_{\mathcal{T}}[U]$ is bounded by $\widetilde{O}(\mathsf{MF}(\mathsf{vol}_G(U), |V_{\mathcal{T}}[U]|)/\phi)$.*

Proof. During the preparation step, each invocation of Lemma 10 induces a set of max-flow instances whose total size is bounded by $\widetilde{O}(|E_{\mathcal{T}}[U]|) = \widetilde{O}(\mathsf{vol}_G(U))$. Since it is repeated for $O(\log n/\phi)$ times, the total time is at most $\widetilde{O}(\mathsf{MF}(\mathsf{vol}_G(U), |V_{\mathcal{T}}[U]|)/\phi)$.

Next, let us analyze the cost of ExploreTree.

▷ Claim 20. After each iteration of the while-loop, the value of $|C \cap U_d \setminus S| + |Q|$ always increases by at least 1, so the total number of max-flow instances during the loop is bounded by $O(1/\phi)$.

Proof of claim. If an iteration ends on line-10, Q increases by one while S does not change. If an iteration of the while-loop ends on line-7, then on the one hand, by Lemma 14 we have $Q \subseteq S$; on the other hand, by the pseudo-code, $u \notin Q$ before updating S, Q. Hence, after line-7, $|C \cap U_d \setminus S| + |Q|$ increases by at least 1.

If an iteration ends on line-12, we claim that before updating S, Q, we have $L_u \cap Q = \emptyset$. In fact, by Lemma 14, for any $w \in Q$, $L_w \cap C \cap U_d = S$. By Lemma 15, as $L_u \cap C \cap U_d \neq S$, it must be $L_u \cap C \cap U_d \subsetneq S = L_w \cap C \cap U_d$. Hence, $w \notin L_u$. As w is arbitrary, we know $Q \cap L_u = \emptyset$. Therefore, after updating $S \leftarrow L_u \cap C \cap U_d$, $|C \cap U_d \setminus S|$ has increased by $|Q|$. Notice that after updating Q, $|Q| = 1$. So $|C \cap U_d \setminus S| + |Q|$ has increased by one. ◁

Since each while-loop conducts one max-flow in graph $G_{\mathcal{T}}[U]$, by the above claim, the total cost of the while-loop involves max-flow instances of total size $\widetilde{O}(\mathsf{vol}_G(U)/\phi)$, and the reduction time is dominated by the same amount. After the while-loop, the running time is linear in the size of output, so it is not the bottleneck. ◀

Next we analyze the running time during refinement of U.

▶ **Lemma 21.** *The total running time of cutting vertices (the for-loop on line-6 of CondGomoryHu) from U takes total time of $\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(2d \cdot \mathsf{cnt}[i_U] \log^2 n, |V_{\mathcal{T}}[U]|))$.*

Proof. On the one hand, the number of clusters in $\mathcal{C}_{i_U}$ is at most n/s since each cluster has size at least s. So, by Lemma 19, the total time of expander search is $\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(\mathsf{vol}_G(U)))$. By maximality of $d|U_d|$ and $\mathsf{cnt}[i_U]$, we have that:

$$\mathsf{vol}_G(U) \leq 2d|U_d| \log n \leq 2d \cdot \mathsf{cnt}[i_U] \log^2 n$$

Since the number of edges in $G_{\mathcal{T}}[U]$ is $\mathsf{vol}_G(U)$, the overall time complexity would be $\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(2d \cdot \mathsf{cnt}[i_U] \log^2 n, |V_{\mathcal{T}}[U]|))$. ◀

To bound the total time across all different nodes of $\mathcal{T}$ that correspond to the same choice of (s, d), we need the following lemma.

▶ **Lemma 22.** *In any single iteration of the while-loop on line-2 of CondGomoryHu, over all different nodes U of $\mathcal{T}$ that correspond to the same choice of (s, d), we have $\sum_U \mathsf{cnt}[i_U] \leq 4sn/d$.*

Proof. If $s \geq d/4$, then since all such nodes U are packing disjoint subsets of vertices of V, $\sum_U \mathsf{cnt}[i_U] \leq n \leq 4sn/d$. So next we only consider the case where $s < d/4$.

When $s < d/4$, we can upper bound the total number of vertices in clusters in $\mathcal{C}_{i_U}$ whose degrees in G are within the interval $[d, 2d)$. Take any cluster $C \in \mathcal{C}_{i_U}$ and any vertex $u \in C \cap U_d$. Since $|C| < 2s = d/2$ and G is a simple graph, at least $d/2$ of u's neighbors in G are outside of C. So the crossing edges contributed by u is at least $d/2$. By property (3) of the Definition 8, the total number of crossing edges should be bounded as:

$$\begin{aligned}\sum_{C \in \mathcal{C}_{i_U}} \mathsf{out}_G(C) &\leq \log^7 n \cdot \phi \cdot \sum_{C \in \mathcal{C}_{i_U}} \mathsf{vol}_G(C) \leq \log^7 n \cdot \phi \cdot \sum_{C \in \mathcal{C}_{i_U}} (4s^2 + \mathsf{out}_G(C)) \\ &\leq 4 \log^7 n \cdot \phi sn + \log^7 n \cdot \phi \sum_{C \in \mathcal{C}_{i_U}} \mathsf{out}_G(C)\end{aligned}$$

As $\phi = \frac{1}{10\log^{c+10} n}$, we have $\sum_{C\in\mathcal{C}_{i_U}} \mathsf{out}_G(C) \le 8\log^7 n \cdot \phi sn < sn$. As each vertex in $C \cap U_d$ contributes $d/2$ to the above summation, the total number of vertices from U_d in $\mathcal{C}_{i_U}$ is bounded by $2sn/d$. ◀

Combining the above two lemmas gives the following corollary.

▶ **Corollary 23.** *Under Hypothesis 1, the time of dividing all nodes of $\mathcal{T}$ for a single iteration of the while-loop on line-2 in* CondGomoryHu *is bounded by $\widetilde{O}(n^2)$.*

The next thing would be analyzing the total number of rounds of the while-loop. Similar to [3], we first need to prove that with high probability, the number of $u \in C \cup U_d$ such that $\mathsf{vol}_G(K_u) > 0.5\mathsf{vol}_G(U)$ is roughly at most $|U_d|/r$.

▶ **Lemma 24.** *With high probability over the choice of $R \subseteq N$, the total number of $u \in U_d$ such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$ is at most $\frac{4|U_d|\log^2 n}{r}$.*

Proof. The proof is similar to the one in [3]. To avoid confusion, let $\mathcal{T}^{\text{old}}$ be the version of $\mathcal{T}$ before refining with respect to R, and let $\mathcal{T}$ refer to the tree after refinement. For each pair of vertices in the super node $x, q \in N$, define $(\Gamma^q_x, V_{\mathcal{T}^{\text{old}}}[N] \setminus \Gamma^q_x)$ to be the latest minimum cut of (x, q) in $G_{\mathcal{T}^{\text{old}}}[N]$. Define M^q_x to be the set of all vertices $y \in N$ such that $x \in \Gamma^q_y$. Basic concentration inequalities show that for any q, x, if $\mathsf{vol}_G(M^q_x) \ge \frac{\log n}{r}\mathsf{vol}_G(N)$, then with high probability, $M^q_x \cap R \neq \emptyset$.

The following claim is a crucial relationship between Γ^p_u and L_u.

▷ Claim 25 (Observation 4.3 in [3]). $\Gamma^p_u \cap U = L_u \cap U$.

Proof of claim. As $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is the latest minimum cut in $G_{\mathcal{T}}[U]$ which is a contracted graph of $G_{\mathcal{T}^{\text{old}}}[N]$ following standard Gomory-Hu steps, $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is a min-cut for (u, p) in $G_{\mathcal{T}^{\text{old}}}[N]$. Since $(\Gamma^p_u, V_{\mathcal{T}^{\text{old}}}[N] \setminus \Gamma^p_u)$ is the latest minimum cut in $G_{\mathcal{T}^{\text{old}}}[N]$, we have $\Gamma^p_u \cap U \subseteq L_u \cap U$. Next we only focus on the other direction.

Let $W_1, W_2, \cdots, W_l \subseteq V_{\mathcal{T}^{\text{old}}}[N]$ be all contracted vertices of $V_{\mathcal{T}}[U]$ which are crossed by Γ^p_u; in other words, $\Gamma^p_u \cap W_i \neq \emptyset$ and $W_i \setminus \Gamma^p_u \neq \emptyset$ for all $1 \le i \le l$. Since N is refined using pivots from R, according to Lemma 6, we know that for each i there exists a pivot $q_i \in R \cap W_i$ such that $(W_i, V_{\mathcal{T}^{\text{old}}}[N] \setminus W_i)$ is a minimum cut for (q_i, p); in fact, $q_i \in R \cap W_i$ are in the neighboring nodes of U in $\mathcal{T}$.

We claim that $q_i \in \Gamma^p_u$; otherwise if $q_i \notin \Gamma^p_u$, as $u \notin W_i$, by Lemma 4, the cut $(X, V_{\mathcal{T}^{\text{old}}}[N] \setminus X)$ where $X = \Gamma^p_u \setminus W_i$ is also a minimum cut for (u, p), which contradicts that $(\Gamma^p_u, V_{\mathcal{T}^{\text{old}}}[N] \setminus \Gamma^p_u)$ is the latest min-cut.

Construct a new cut $(Y, V_{\mathcal{T}^{\text{old}}}[N] \setminus Y)$ where $Y = \Gamma^p_u \cup \bigcup_{i=1}^{l} W_i$. On the one hand, as $q_i \in \Gamma^p_u, \forall i$, by repeatedly applying Lemma 3 we know $(Y, V_{\mathcal{T}^{\text{old}}}[N] \setminus Y)$ is a minimum cut for (u, p) as well; On the other hand, Y does not cross any contracted nodes in $G_{\mathcal{T}}[U]$, so $(Y, V_{\mathcal{T}}[U] \setminus Y)$ is a valid cut in $G_{\mathcal{T}}[U]$ as well. As $(L_u, V_{\mathcal{T}}[U] \setminus L_u)$ is the latest cut in $G_{\mathcal{T}}[U]$, we know $L_u \cap U \subseteq Y \cap U = \Gamma^p_u \cap U$. This concludes our proof. ◁

Consider the set of all $u \in U_d$ such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$; let them be $u_1, u_2, \cdots, u_l$. By the above claim, it must be $\mathsf{vol}_G(\Gamma^p_{u_i} \cap N) \ge \mathsf{vol}_G(\Gamma^p_{u_i} \cap U) = \mathsf{vol}_G(L_{u_i} \cap U) > 0.5\mathsf{vol}_G(N)$ as well. Therefore, any two sets $\Gamma^p_{u_i}, \Gamma^p_{u_j}$ must intersect. Since $(\Gamma^p_{u_i}, V_{\mathcal{T}^{\text{old}}}[N] \setminus \Gamma^p_{u_i})$ are latest cuts with respect to the same pivot p, they should form a total order, say $\Gamma^p_{u_1} \subseteq \Gamma^p_{u_2} \subseteq \cdots \Gamma^p_{u_l}$, and so by definition $u_2, u_3, \cdots, u_l \in M^p_{u_1}$.

▷ Claim 26. $M^p_{u_1} \cap R = \emptyset$.

Proof of claim. If $\exists w \in M^p_{u_1} \cap R$, then by definition, $u \in \Gamma^p_w$. As $(\Gamma^p_w, V_{\mathcal{T}^{\text{old}}}[N] \setminus \Gamma^p_w)$ is the latest min-cut for (w,p) in $G_{\mathcal{T}^{\text{old}}}[N]$, any min-cut for (w,p) in $G_{\mathcal{T}^{\text{old}}}[N]$ should contain u on the same side as w. By Lemma 6, u should belong to the part which contains w after the refinement with respect to R, which makes a contradiction as u stays with p in the same part. ◁

By the above lemma, we know $\mathsf{vol}_G(M^p_{u_1}) < \frac{\log n}{r}\mathsf{vol}_G(N)$, and hence we have:

$$d(l-1) \le \mathsf{vol}_G(M^p_{u_1}) < \frac{\log n}{r}\mathsf{vol}_G(N) \le \frac{2\log n}{r}\mathsf{vol}_G(U) \le \frac{4\log^2 n}{r}d|U_d|$$

So $l \le \frac{4|U_d|\log^2 n}{r}$. ◀

Finally we need to bound the total number of rounds in the while-loop. Call a cluster $C \in \mathcal{C}_{i_U}$ **bad**, if the total number of vertices $u \in C \cap U_d$ such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$ is more than $0.1|C \cap U_d|$; otherwise it is called **good**.

▶ **Lemma 27.** *Consider any invocation of ExploreTree with input parameters U, p, d, C. Suppose cluster C is good, then $|W_C| > 0.8|C \cap U_d|$.*

Proof. First consider the case where ExploreTree terminated on line-14. The while-loop must have terminated on line-7. Then as u is small, $|S| \ge |C \cap U_d| - 2/\phi - |Q| - |C \cap L_u \cap U_d| \ge |C \cap U_d| - 6/\phi$. Therefore, $|W_C| \ge |C \cap U_d| - 6/\phi - 0.1|C \cap U_d| > 0.8|C \cap U_d|$.

Now suppose ExploreTree terminated on line-21. In this case, $C \cap U_d \setminus W_C$ only includes vertices in $C \cap U_d \setminus S$, plus vertices $v \in C \cap U_d$ such that $\mathsf{vol}_G(L_v \cap U) > 0.5\mathsf{vol}_G(N)$. Since C is good, we know $|W_C| \ge |C \cap U_d| - 2/\phi - 0.1|C \cap U_d| > 0.8|C \cap U_d|$. ◀

▶ **Lemma 28.** $\sum_{C \in \mathcal{C}_{i_U} \text{ is bad}} |C \cap U_d| \le \frac{40|U_d|\log^2 n}{r}$.

Proof. By Lemma 24, the total number of vertices u such that $\mathsf{vol}_G(L_u \cap U) > 0.5\mathsf{vol}_G(N)$ is at most $\frac{4|U_d|\log^2 n}{r}$. By definition of badness, we have $\sum_{C \in \mathcal{C}_{i_U} \text{ is bad}} |C \cap U_d| \le \frac{40|U_d|\log^2 n}{r}$. ◀

▶ **Lemma 29.** *For each node U such that $\mathsf{vol}_G(U) > 0.5\mathsf{vol}_G(N)$, and for each set $K_u = L_u$ which is cut off by our algorithm, we have $\mathsf{vol}_G(K_u \cap U) \le 0.5\mathsf{vol}_G(N)$. Furthermore, let P be the rest of U after cutting all K_u's. Then $\mathsf{vol}_G(P) \le (1 - \frac{1}{2\log^2 n})\mathsf{vol}_G(U)$.*

Proof. The first half of the claim is automatically guaranteed by the algorithm. Let us only consider the second half.

By Lemma 27, the total volume that has been cut off from U is at least

$$\begin{aligned}\sum_{C \in \mathcal{C}_{i_U} \text{ is good}} d|W_C| &\ge \sum_{C \in C_{i_U} \text{ is good}} 0.8d|C \cap U_d| \ge \frac{0.8d}{\log n}|U_d| - 0.8d \sum_{C \in \mathcal{C}_{i_U} \text{ is bad}} |C \cap U_d| \\ &\ge \frac{0.8d}{\log n}|U_d| - \frac{32d\log^2 n}{r}|U_d| \ge \frac{0.8}{\log^2 n}\mathsf{vol}_G(U) - \frac{32}{\log^3 n}\mathsf{vol}_G(U) \\ &\ge \frac{1}{2\log^2 n}\mathsf{vol}_G(U)\end{aligned}$$

Hence, the volume of $\mathsf{vol}_G(P)$ is reduced by a factor of at most $1 - \frac{1}{2\log^2 n}$. ◀

By Lemma 29, after each round of the while-loop, for each node $U \subseteq N$, either we already have $\mathsf{vol}_G(U) \le 0.5\mathsf{vol}_G(N)$ after the refinement with respect to random set R, or U is further divided into sub-nodes whose volume are at most $\max\{0.5\mathsf{vol}_G(N), (1 - \frac{1}{2\log^2 n})\mathsf{vol}_G(U)\}$. Therefore the number of rounds is at most $\log^3 n$. So the total running time should be $\widetilde{O}(n^2)$ as well under Hypothesis 1.

4 Unconditional cut-equivalent trees

4.1 The main algorithm

In this section we will prove the second half of Theorem 2 using existing max-flow algorithms. The algorithm is mostly the same as the previous algorithm conditioning on Hypothesis 1, and the extra work is to deal with the additive $n^{1.5}$ term that appears in the running time of max-flow algorithm from [6]. Similar to the previous algorithm, we will also use the same set of parameters ϕ, r, and use degree set $\mathcal{D} = \{\sqrt{n}, 2\sqrt{n}, 2^2\sqrt{n}, \cdots, n\}$.

Preparation

Throughout the algorithm, $\mathcal{T}$ will be the cut-equivalent tree under construction, where each of $\mathcal{T}$'s node will represent a subset of vertices of V. As a preparation step, compute a (ϕ, ϕ)-expander decomposition on G and obtain a partitioning $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ of V. Categorize clusters in $\mathcal{C}$ according to their sizes: for each 2^i, define $\mathcal{C}_i$ to be the set of clusters whose sizes are within interval $[2^i, 2^{i+1})$.

Iteration

In each round, the algorithm tries to simultaneously subdivide all nodes of $\mathcal{T}$ which contains at least $20r$ vertices in V. Following the same procedure as in the previous algorithm, for each node N of $\mathcal{T}$, further refine N into a set of smaller sub-nodes. Then, for each such sub-node U, define variables d, U_d and $s = 2^{i_U}$ accordingly. If (1) $d \geq n^{3/4}$ or (2) $s > n^{3/4}/\sqrt{d}$, we would continue to the do same as in algorithm CondGomoryHu which invokes the expander search procedure.

The unconditional algorithm diverges from the conditional algorithm in Theorem 2 from here if $d < n^{3/4}$ and $s \leq n^{3/4}/\sqrt{d}$. Intuitively, when s is relatively small, the number of expanders whose sizes are roughly s would be large, and so expander searches would be costly because of the additive term $n^{1.5}$ in the running time of computing max-flow. What we would do is to directly apply Lemma 7 on graph $G_{\mathcal{T}}[U]$ to isolate all vertices in U_d on the tree $\mathcal{T}$ once and for all. The pseudo-code is summarized as GomoryHu.

4.2 Running time analysis

▶ **Lemma 30.** *Each round of the while-loop in GomoryHu takes time* $\widetilde{O}(n^{17/8})$.

Proof. Let us study an arbitrary iteration. Suppose the condition on line-6 holds, namely $d \geq n^{3/4}$ or $s > n^{3/4}/\sqrt{d}$. Then in this case we would do exactly the same as in the conditional algorithm, and the only difference we are invoking the max-flow algorithm from [6]. According to Lemma 21, we could upper bound the running time as

$$\widetilde{O}(\frac{n}{s} \cdot \mathsf{MF}(2d \cdot \mathsf{cnt}[i_U] \log^2 n, |V_{\mathcal{T}}[U]|)) = \widetilde{O}(\frac{nd}{s}\mathsf{cnt}[i_U] + \frac{n}{s} \cdot |V_{\mathcal{T}}[U]|^{1.5})$$

for each node U. Since all tree nodes U are disjoint vertex subsets of V, $\sum_{U \in \mathcal{T}} |V_{\mathcal{T}}[U]|^{1.5} \leq n^{1.5}$. Therefore, by Lemma 22, this sums to $\widetilde{O}(n^2 + \frac{n^{2.5}}{s})$.

Algorithm 3 GomoryHu($G = (V, E)$).

```
1  initialize a partition tree 𝒯, as well as parameters φ, r;
2  while ∃N ⊆ V, U a node of 𝒯, |N| ≥ 20r do
3      for node N of 𝒯 with |N| ≥ 20r do
4          repeat for 10r times: each time we sample a vertex u ∈ N with probability
             deg_G(u) / vol_G(N), and let the sampled set be R;
5          call Lemma 5 on node N with respect to R;
6          for node U ⊆ N of 𝒯 such that vol_G(U) > 0.5 vol_G(N) do
7              take d such that d|U_d| is maximized;
8              take s = 2^{i_U} such that cnt[i_U] is maximized;
9              if d ≥ n^{3/4} or s > n^{3/4}/√d then
10                 for each C ∈ 𝒞_{i_U} do
11                     run expander search on C within node U to compute a subset
                         W_C ⊆ C ∩ U_d, and the latest min-cuts (K_u, V_𝒯[U] \ K_u) for each
                         u ∈ W_C;
12                 define W = ⋃_{C∈𝒞_{i_U}} W_C;
13                 for each u ∈ W such that K_u is maximal, split K_u ∩ U off of U and
                     create a new node on 𝒯;
14             else
15                 apply Lemma 7 on the auxiliary graph G_𝒯[U] with input parameter
                     k = 2d, so that all vertices in U_d become singletons in 𝒯;
16 for node U of 𝒯 such that |U| < 20r do
17     repeatedly refine U using the generic Gomory-Hu steps until all nodes are
         singletons;
18 return 𝒯 as a cut-equivalent tree;
```

We first claim that $s \geq \sqrt{2d}$. In fact, by maximality of $\mathsf{cnt}[i_U]$, there exists at least one cluster $C \in \mathcal{C}_{i_U}$ that intersects U_d. Take any $u \in C \cap U_d$. Then since G is a simple graph, more than $d - s$ neighbors of u are outside of C, thus $\mathsf{out}_G(C) > d - s$. By property (3) of Definition 8, we have:

$$\mathsf{out}_G(C) \leq \log^7 n \cdot \phi \mathsf{vol}_G(C) \leq \log^7 n \cdot \phi(4s^2 + \mathsf{out}_G(C))$$

As $\phi = \frac{1}{10 \log^{c+10}}$, we have $\mathsf{out}_G(C) \leq 0.4s^2 + 0.1\mathsf{out}_G(C)$, and so $\mathsf{out}_G(C) < 0.5s^2$. As $\mathsf{out}_G(C) > d - s$, we have $s > \sqrt{2d}$.

When $d \geq n^{3/4}$, as $s \geq \sqrt{2d} > n^{3/8}$ we have $\widetilde{O}(n^2 + \frac{n^{2.5}}{s}) = \widetilde{O}(n^{17/8})$. If $d < n^{3/4}$ and $s > n^{3/4}/\sqrt{d}$, then we also bound the total running time as $\widetilde{O}(n^2 + \frac{n}{s\phi} \cdot n^{1.5}) = \widetilde{O}(n^{17/8})$.

Now suppose the condition on line-6 does not hold, then $d < n^{3/4}$ and $s \leq n^{3/4}/\sqrt{d}$. In this case, similar to Lemma 22, we can prove that the total volume $\mathsf{vol}_G(U)$ over all different U's is bounded by $\widetilde{O}(ns)$. So applying Lemma 7 in this round takes time at most $\widetilde{O}(nsd) = \widetilde{O}(n^{17/8})$. ◀

▶ **Lemma 31.** *The total number of rounds of the while-loop in GomoryHu is bounded by* $O(\log^3 n)$.

Proof. If each round of the while-loop, if $d \geq n^{3/4}$ or $s > n^{3/4}/\sqrt{d}$ for node U, then according to the proof of Lemma 29, the volume of each subdivision is bounded by $\max\{0.5\mathsf{vol}_G(N), (1-\frac{1}{2\log^2 n})\mathsf{vol}_G(U)\}$. If $d < n^{3/4}$ and $s \leq n^{3/4}/\sqrt{d}$, then all vertices in U_d become singletons on $\mathcal{T}$; also, and for the same reason, all subdivision of U should be at most $(1-\frac{1}{2\log^2 n})\mathsf{vol}_G(U)$. Therefore, the number of while-loop iterations is at most $O(\log^3 n)$. ◀

References

1. Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Gomory-hu tree in subcubic time. *arXiv preprint*, 2021. `arXiv:2111.04958`.
2. Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 48–61. SIAM, 2020.
3. Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory-hu tree in unweighted graphs. *arXiv preprint*, 2020. `arXiv:2012.10281`.
4. Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Apmf< apsp? gomory-hu tree for unweighted graphs in almost-quadratic time. *arXiv preprint*, 2021. `arXiv:2106.02981`.
5. Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Friendly cut sparsifiers and faster gomory-hu trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3630–3649. SIAM, 2022.
6. Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. *arXiv preprint*, 2021. `arXiv:2101.05719`.
7. Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 11–26. Springer, 2003.
8. Harold N Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 812–821. IEEE Computer Society, 1991.
9. Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.
10. Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.
11. Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228. SIAM, 2021.
12. Frieda Granot and Refael Hassin. Multi-terminal maximum flows in node-capacitated networks. *Discrete applied mathematics*, 13(2-3):157–163, 1986.
13. Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.
14. Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An $o(mn)$ gomory-hu tree construction algorithm for unweighted graphs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 605–614, 2007.
15. Tarun Kathuria, Yang P Liu, and Aaron Sidford. Unit capacity maxflow in almost $m^{4/3}$ time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130. IEEE, 2020.
16. Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\widetilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.
17. Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *Annual Symposium on Foundations of Computer Science*, 2020.
18. Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. *arXiv preprint*, 2021. `arXiv:2106.02233`.

19 Yang P Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 803–814, 2020.

20 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

21 Tianyi Zhang. Gomory-hu trees in quadratic time. *arXiv preprint*, 2021. `arXiv:2112.01042`.

Universal Complexity Bounds Based on Value Iteration and Application to Entropy Games

Xavier Allamigeon
INRIA, Palaiseau, France
CMAP, École polytechnique, IP Paris, CNRS, Palaiseau, France

Stéphane Gaubert
INRIA, Palaiseau, France
CMAP, École polytechnique, IP Paris, CNRS, Palaiseau, France

Ricardo D. Katz
CIFASIS-CONICET, Rosario, Argentina

Mateusz Skomra
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Abstract

We develop value iteration-based algorithms to solve in a unified manner different classes of combinatorial zero-sum games with mean-payoff type rewards. These algorithms rely on an oracle, evaluating the dynamic programming operator up to a given precision. We show that the number of calls to the oracle needed to determine exact optimal (positional) strategies is, up to a factor polynomial in the dimension, of order R/sep, where the "separation" sep is defined as the minimal difference between distinct values arising from strategies, and R is a metric estimate, involving the norm of approximate sub and super-eigenvectors of the dynamic programming operator. We illustrate this method by two applications. The first one is a new proof, leading to improved complexity estimates, of a theorem of Boros, Elbassioni, Gurvich and Makino, showing that turn-based mean payoff games with a fixed number of random positions can be solved in pseudo-polynomial time. The second one concerns entropy games, a model introduced by Asarin, Cervelle, Degorre, Dima, Horn and Kozyakin. The *rank* of an entropy game is defined as the maximal rank among all the ambiguity matrices determined by strategies of the two players. We show that entropy games with a fixed rank, in their original formulation, can be solved in polynomial time, and that an extension of entropy games incorporating weights can be solved in pseudo-polynomial time under the same fixed rank condition.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Mean-payoff games, entropy games, value iteration, Perron root, separation bounds, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.110

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2206.09044`

Acknowledgements We thank the reviewers for detailed and helpful comments.

1 Introduction

1.1 Motivation

Deterministic and turn-based stochastic Mean Payoff games are fundamental classes of games with an unsettled complexity. They belong to the complexity class NP ∩ coNP [21, 47] but they are not known to be polynomial-time solvable. Various algorithms have been developed and analyzed. The pumping algorithm is a pseudo-polynomial iterative scheme introduced by

Gurvich, Karzanov and Khachyan [25] to solve the optimality equation of deterministic mean payoff games. Zwick and Paterson [47] derived peudo-polynomial bounds for the same games by analyzing value iteration. Friedmann showed that policy iteration, originally introduced by Hoffman and Karp in the setting of zero-sum games [26], and albeit being experimentally fast on typical instances, is generally exponential [22]. We refer to the survey of [9] for more information and additional references.

Entropy games have been introduced by Asarin et al. [10]. They are combinatorial games, in which one player, called Tribune, wants to maximize a topological entropy, whereas its opponent, called Despot, wishes to minimize it. This topological entropy quantifies the freedom of a half-player, called People. Although the formalization of entropy game is recent, specific classes or variants of entropy games appeared earlier in several fields, including the control of branching processes, population dynamics and growth maximization [42, 38, 37, 46], risk sensitive control [27, 8], mathematical finance [5], or matrix multiplication games [10]. Asarin et al. showed that entropy games also belong to the class NP $\cap$ coNP. Akian et al. showed in [1] that entropy games reduce to ordinary stochastic mean payoff games with infinite action spaces (actions consist of probability measures and the payments are given by relative entropies), and deduced that the subclass of entropy games in which Despot has a fixed number of significant positions (positions with a non-trivial choice) can be solved in polynomial time. The complexity of entropy games without restrictions on the number of (significant) Despot positions is an open problem.

1.2 Main Results

We develop value iteration-based algorithms to solve in a unified manner different classes of combinatorial zero-sum games with mean-payoff type rewards. These algorithms rely on an oracle, evaluating approximately the dynamic programming operator of the game. Our main results include universal estimates, providing explicit bounds for the error of approximation of the value, as a function of two characteristic quantities, of a metric nature. The first one is the *separation* sep, defined as the minimal difference between distinct values induced by (positional) strategies. The second one, R, is defined in terms the norm of approximate sub and super-optimality certificates. These certificates are vectors, defined as sub or super-solutions of non-linear eigenproblems. For games such that the mean payoff is independent of the initial state, we show that (exact) optimal strategies can be found in a number of calls to the oracle bounded by the ratio R/sep, up to a factor polynomial in the number of states, see Theorems 10 and 18. We also obtain a similar complexity bound for games in which the mean payoff does depend on the initial state, under additional assumptions.

We provide two applications of this method.

The first application is a new proof of an essential part of the theorem of Boros, Elbassioni, Gurvich and Makino [16], showing that turn-based stochastic mean payoff games with a fixed number of random positions can be solved in pseudo-polynomial time. The original proof relies on a deep analysis of a generalization to the stochastic case of the "pumping algorithm" of [25]. Our analysis of value iteration leads to improved complexity estimates. Indeed, we bound the characteristic numbers R and sep in a tight way, by exploiting bit-complexity estimates for the solutions of Fokker–Planck and Poisson-type equations of discrete Markov chains.

The second application concerns entropy games. Let us recall that in such a game, the value of a pair of (positional) strategies of the two players is given by the Perron root of a certain principal submatrix of a nonnegative matrix, which we call the *ambiguity matrix*,

as it measures the number of nondeterministic choices of People. We show that entropy games with a fixed rank, and in particular, entropy games with a fixed number of People's states, can be solved in pseudo-polynomial time; see Corollary 32. These results concern the extended model of entropy games introduced in [1], taking into account weights. Then, entropy games in the sense of [10] (implying a unary encoding of weights) that have a fixed rank can be solved in polynomial time. These results rely on separation bounds for algebraic numbers arising as the eigenvalues of integer matrices with a fixed rank.

1.3 Related Work

The idea of applying value iteration to analyze the complexity of deterministic mean-payoff games goes back to the classical work of Zwick and Paterson [47]. In some sense, the present approach extends this idea to more general classes of games. When specialized to stochastic mean payoff games with perfect information, our bounds should be compared with the ones of Boros, Elbassioni, Gurvich, and Makino [16, 15]. The authors of [16] generalize the "pumping" algorithm, developed for deterministic games by Gurvich, Karzanov, and Khachiyan [25], to the case of stochastic games. The resulting algorithm is also pseudopolynomial if the number of random positions is fixed, see Remark 26 for a detailed comparison. The algorithm of Ibsen-Jensen and Miltersen [28] yields a stronger bound in the case of simple stochastic games, still assuming that the number of random positions is fixed. A different approach, based on an analysis of strategy iteration, was developed by Gimbert and Horn [24] and more recently by Auger, Badin de Montjoye and Strozecki [11]. The value iteration algorithm for concurrent mean payoff games, under an ergodicity condition, has been studied by Chatterjee and Ibsen-Jensen [19]. Theorem 18 there bounds the number of iterations needed to get an ϵ-approximation of the mean-payoff. When specialized to this case, Theorem 13 below improves this bound by a factor of $|\log \epsilon|$.

We build on the operator approach for zero-sum games, see [13, 34, 36]. Our study of entropy games is inspired by the works of Asarin et al. [10] and Akian et al. [1]. We rely on the existence of optimal positional strategies for entropy games, established in [1] by an o-minimal geometry approach [14] and also, on results of non-linear Perron–Frobenius theory, especially the Collatz–Wielandt variational formulation of the escape rate of an order preserving and additively homogeneous mapping [35, 23, 2, 4].

The present work, providing complexity bounds based on value iteration, grew out from an effort to understand the surprising speed of value iteration on random stochastic games examples arising from tropical geometry [7], by investigating suitable notions of condition numbers [6]. An initial version of some of the present results (concerning turn based stochastic games) appeared in the PhD thesis of one of the authors [40].

1.4 Organization of the Paper

In Section 2 we recall the definitions and basic properties of turn-based stochastic mean payoff games and entropy games, and also key notions in the "operator approach" of zero-sum games, including the Collatz–Wielandt optimality certificates.

The universal complexity bounds based on value iteration are presented in Section 3. First, we deal with games whose value is independent of the initial state, and then, we extend these results to determine the set of initial states with a maximal value.

The applications to turn-based stochastic mean payoff games and to entropy games are provided in Section 4 and Section 5. The detailed proofs can be found in the extended version of the present paper.

2 Preliminaries on Dynamic Programming Operators and Games

2.1 Introducing Shapley Operators: The Example of Stochastic Turn-Based Zero-Sum Games

Shapley operators are the two-player version of the Bellman operators (a.k.a. dynamic programming or one-day operators) which are classically used to study Markov decision processes. In this section, we introduce the simplest example of Shapley operator, arising from stochastic turn-based zero-sum games.

A *stochastic turn-based zero-sum game* is a game played on a digraph $(\mathscr{V}, \mathscr{E})$ in which the set of vertices $\mathscr{V}$ has a non-trivial partition $\mathscr{V} = \mathscr{V}_{\mathrm{Min}} \uplus \mathscr{V}_{\mathrm{Max}} \uplus \mathscr{V}_{\mathrm{Nat}}$. There are two players, called *Min* and *Max*, and a half-player, *Nature*. The sets $\mathscr{V}_{\mathrm{Min}}$, $\mathscr{V}_{\mathrm{Max}}$ and $\mathscr{V}_{\mathrm{Nat}}$ represent the sets of states at which Min, Max, and Nature respectively play. The set of edges $\mathscr{E}$ represents the allowed moves. We assume $\mathscr{E} \subset \mathscr{V}_{\mathrm{Min}} \times \mathscr{V}_{\mathrm{Max}} \cup \mathscr{V}_{\mathrm{Max}} \times \mathscr{V}_{\mathrm{Nat}} \cup \mathscr{V}_{\mathrm{Nat}} \times \mathscr{V}_{\mathrm{Min}}$, meaning that Min, Max, and Nature alternate their moves. More precisely, a turn consists of three successive moves: when the current state is $j \in \mathscr{V}_{\mathrm{Min}}$, Min selects and edge (j, i) in $\mathscr{E}$ and the next state is $i \in \mathscr{V}_{\mathrm{Max}}$. Then, Max selects an edge (i, k) in $\mathscr{E}$ and the next state is $k \in \mathscr{V}_{\mathrm{Nat}}$. Next, Nature chooses an edge $(k, j') \in \mathscr{E}$ and the next state is $j' \in \mathscr{V}_{\mathrm{Min}}$. This process can be repeated, alternating moves of Min, Max, and Nature.

We make the following assumption.

▶ **Assumption 1.** *Each player has at least one available action in each state in which he has to play, i.e., for all $j \in \mathscr{V}_{\mathrm{Min}}, i \in \mathscr{V}_{\mathrm{Max}}$, and $k \in \mathscr{V}_{\mathrm{Nat}}$, the sets $\{i' \colon (j, i') \in \mathscr{E}\}$, $\{k' \colon (i, k') \in \mathscr{E}\}$ and $\{j' \colon (k, j') \in \mathscr{E}\}$ are non-empty.*

Furthermore, every state $k \in \mathscr{V}_{\mathrm{Nat}}$ controlled by Nature is equipped with a probability distribution on its outgoing edges, i.e., we are given a vector $(P_{kj})_{j \in \mathscr{V}_{\mathrm{Min}}}$ with rational entries such that $P_{kj} \geqslant 0$ for all i and $\sum_{(k,j) \in \mathscr{E}} P_{kj} = 1$. We suppose that Nature makes its decisions according to this probability distribution, i.e., it chooses an edge (k, j) with probability P_{kj}. Moreover, we are given two integer matrices $A, B \in \mathbb{Z}^{\mathscr{V}_{\mathrm{Max}} \times \mathscr{V}_{\mathrm{Min}}}$. These matrices encode the payoffs of the game in the following way: if the current state of the game is $j \in \mathscr{V}_{\mathrm{Min}}$ and Min selects an edge (j, i), then Player Min pays to Max the amount $-A_{ij}$. Similarly, if the current state of the game is $i \in \mathscr{V}_{\mathrm{Max}}$ and Max selects an edge (i, k), then Max receives from Min the payment B_{ik}.

We first consider the *game in horizon N*, in which each of the two players Min and Max makes N moves, starting from a known initial state, which by convention we require to be controlled by Min. In this setting, a *history* of the game consists of the sequence of states visited up to a given stage. A *strategy* of a player is a function which assigns to a history of the game a decision of this player. A pair of strategies (σ, τ) of players Min and Max induces a probability measure on the set of finite sequences of states. Then, the expected reward of Max, starting from the initial position j_0, is defined by

$$R_{j_0}(\sigma, \tau) \coloneqq \mathbb{E}_{\sigma\tau}\Big(\sum_{p=0}^{N-1} (-A_{i_p j_p} + B_{i_p k_p})\Big)\,,$$

in which the expectation $\mathbb{E}_{\sigma,\tau}$ refers to the probability measure induced by (σ, τ), and $j_0, i_0, k_0, j_1, i_1, k_1, \dots$ is the random sequence of states visited when applying this pair of strategies. The objective of Max is to maximize this reward, while Min wants to minimize it. The game in horizon N starting from state j is known to have a *value* v_j^N and optimal strategies σ^* and τ^*, meaning that

$$R_j(\sigma^*, \tau) \leqslant v_j^N \coloneqq R_j(\sigma^*, \tau^*) \leqslant R_j(\sigma, \tau^*)\,,$$

for all strategies σ of Min and τ of Max. The *value vector* $v^N := (v^N_j)_{j\in\mathscr{V}_{\mathrm{Min}}}$ keeps track of the values of all initial states. A classical dynamic programming argument, see e.g. [33, Th. IV.3.2], shows that

$$v^0 = 0, \qquad v^N = F(v^{N-1}),$$

where the *Shapley operator* F is the map from $\mathbb{R}^{\mathscr{V}_{\mathrm{Min}}}$ to $\mathbb{R}^{\mathscr{V}_{\mathrm{Min}}}$ defined by

$$F_j(x) := \min_{(j,i)\in\mathscr{E}} \Big(-A_{ij} + \max_{(i,k)\in\mathscr{E}} \big(B_{ik} + \sum_{(k,l)\in\mathscr{E}} P_{kl}x_k\big)\Big), \text{ for all } j \in \mathscr{V}_{\mathrm{Min}}. \tag{1}$$

Assumption 1 guarantees that F is well defined. One can also consider the *mean-payoff* stochastic game, in which the payment $g_{j_0}(\sigma,\tau)$ received by Player Max becomes the limiting average of the sum of instantaneous payments, i.e.,

$$g_{j_0}(\sigma,\tau) := \liminf_{N\to+\infty} \mathbb{E}_{\sigma\tau}\Big(\frac{1}{N}\sum_{p=0}^{N-1}(-A_{i_pj_p} + B_{i_pk_p})\Big). \tag{2}$$

We say that a strategy is *positional* if the decision of the player depends only of the current state. A result of Liggett and Lippman [31] entails that a mean payoff game has a value χ_j and that there exists a pair of optimal positional strategies (σ^*,τ^*), meaning that

$$g_j(\sigma^*,\tau) \leqslant \chi_j := g_j(\sigma^*,\tau^*) \leqslant g_j(\sigma,\tau^*),$$

for every initial state $j \in \mathscr{V}_{\mathrm{Min}}$ and pair of non-necessarily positional strategies (σ,τ) of players Min and Max. A result of Mertens and Neyman [32] entails in particular that the value of the mean-payoff game coincides with the limit of the normalized value of the games in horizon N, i.e.,

$$\chi = \lim_{N\to\infty} \frac{v^N}{N} = \lim_{N\to\infty} \frac{F^N(0)}{N},$$

where $F^N = F\circ\cdots\circ F$ denotes the Nth iterate of F and 0 the vector that has all entries equal to 0.

▶ Remark 1. In our model, players Min, Max, and Nature play successively, so that a turn decomposes in three stages, resulting in a Shapley operator of the form (1). Alternative models, like the one of [16], in which a turn consists of a single move, reduce to our model by adding linearly many dummy states, and rescaling the mean payoff by a factor 3.

2.2 The Operator Approach to Zero-Sum Games

We shall develop a general approach, which applies to various classes of zero-sum games with a mean-payoff type payment. To do so, it is convenient to introduce an abstract version of Shapley operators, following the "operator approach" of stochastic games [36, 34]. This will allow us to apply notions from nonlinear Perron–Frobenius theory, especially sub and super eigenvectors, and Collatz-Wielandt numbers, which play a key role in our analysis.

Recall that the sup-norm is defined by $\|x\|_\infty := \max_{i\in[n]} |x_i|$. We also use the *Hilbert's seminorm* [23], which is defined by $\|x\|_{\mathrm{H}} := \mathbf{t}(x) - \mathbf{b}(x)$, where $\mathbf{t}(x) := \max_{i\in[n]} x_i$ (read "top") and $\mathbf{b}(x) := \min_{i\in[n]} x_i$ (read "bottom"). We endow $\mathbb{R}$ with the standard order $\leqslant$, which is extended to vectors entrywise.

A self-map F of $\mathbb{R}^n$ is said to be *order-preserving* when

$$x \leqslant y \implies F(x) \leqslant F(y) \text{ for all } x,y \in \mathbb{R}^n, \tag{3}$$

and *additively homogeneous* when

$$F(\lambda + x) = \lambda + F(x) \text{ for all } \lambda \in \mathbb{R} \text{ and } x \in \mathbb{R}^n\,, \tag{4}$$

where, for any $z \in \mathbb{R}^n$, $\lambda + z$ stands for the vector with entries $\lambda + z_i$.

▶ **Definition 2.** *A self-map F of $\mathbb{R}^n$ is an* (abstract) Shapley operator *if it is order-preserving and additively homogeneous.*

A basic example is provided by the Shapley operator of a turn-based stochastic mean-payoff game (1). Here, the additive homogeneity axiom captures the absence of discount. We shall see in the next section a different example, arising from entropy games.

We point out that any order-preserving and additively homogeneous self-map F of $\mathbb{R}^n$ is nonexpansive in the sup-norm, meaning that

$$\|F(x) - F(y)\|_\infty \leqslant \|x - y\|_\infty \text{ for all } x, y \in \mathbb{R}^n\,.$$

Using the nonexpansiveness property, we get that the existence and the value of the limit $\lim_{N\to\infty}(F^N(x)/N)$ are independent of the choice of $x \in \mathbb{R}^N$. We call this limit the *escape rate* of F, and denote it by $\chi(F)$. When F is the Shapley operator of a turn-based stochastic mean-payoff game, fixing $x = 0$, we see that $F^N(x)$ coincides with the value vector in horizon N, and so $\chi_j(F)$ yields the mean-payoff when the initial state is j, consistently with our notation χ_j in Section 2.1.

The escape rate is known to exist under some "rigidity" assumptions. The case of semialgebraic maps is treated in [34], whereas the generalization to o-minimal structures (see [43] for background), which is needed in the application to entropy games, is established in [14].

▶ **Theorem 3** ([34] and [14])**.** *Suppose that the function $F\colon \mathbb{R}^n \to \mathbb{R}^n$ is nonexpansive in any norm and that it is semialgebraic, or, more generally, defined in an o-minimal structure. Then, the escape rate $\chi(F)$ does exist.*

This applies in particular to Shapley operators of turn-based mean-payoff games, since in this case the operator F, given by (1), is piecewise affine, and a fortiori semialgebraic. In the case of entropy games, we shall see in the next section that the relevant Shapley operator is defined by a finite expression involving the maps log, exp, as well as the arithmetic operations, and so that it is definable in a richer stucture, which is still o-minimal. We emphasize that no knowledge of o-minimal techniques is needed to follow the present paper, it suffices to admit that the escape rate does exist for all the classes of maps considered here, and this follows from Theorem 3.

When the map F is piecewise-affine, a result finer than Theorem 3 holds:

▶ **Theorem 4** ([30])**.** *A piecewise affine self-map F of $\mathbb{R}^n$ that is nonexpansive in any norm admits an* invariant half-line*, meaning that there exist $z, w \in \mathbb{R}^n$ such that*

$$F(z + \beta w) = z + (\beta + 1)w$$

for any $\beta \in \mathbb{R}$ large enough. In particular, the escape rate $\chi(F)$ exists, and is given by the vector w.

This entails that $F^k(z + \beta w) = z + (\beta + k)w$, and so, by nonexpansiveness of F, for all $x \in \mathbb{R}^n$, $F^k(x) = k\chi(F) + O(1)$ as $k \to \infty$. This expansion is more precise than Theorem 3, which only states that $F^k(x) = k\chi(F) + o(k)$.

For a general order-preserving and additively homogeneous self-map of $\mathbb{R}^n$, and in particular, for the Shapley operators of the entropy games considered below, an invariant half-line may not exist. However, we can still recover information about the sequences $(F^k(x)/k)_k$ through non-linear spectral theory methods. Assuming that F is an order-preserving and additively homogeneous self-map of $\mathbb{R}^n$, the *upper Collatz–Wielandt number* of F is defined by:

$$\overline{\mathrm{cw}}(F) \coloneqq \inf\{\mu \in \mathbb{R} \colon \exists z \in \mathbb{R}^n, F(z) \leqslant \mu + z\}\,, \tag{5}$$

and the *lower Collatz–Wielandt number* of F by:

$$\underline{\mathrm{cw}}(F) \coloneqq \sup\{\mu \in \mathbb{R} \colon \exists z \in \mathbb{R}^n, F(z) \geqslant \mu + z\}\,. \tag{6}$$

It follows from Fekete's subadditive lemma that the two limits $\lim_{k\to\infty} \mathbf{t}(F^k(0)/k)$ and $\lim_{k\to\infty} \mathbf{b}(F^k(0)/k)$, which may be thought of as upper and lower regularizations of the escape rate, always exist, see [23]. In the examples of interest to us, the escape rate $\chi(F)$ does exist, it represents the mean-payoff vector, and then $\lim_{k\to\infty} \mathbf{t}(F^k(0)/k) = \mathbf{t}(\chi(F)) = \max_j \chi_j(F)$ is the maximum of the mean payoff among all the initial states. Similarly, $\lim_{k\to\infty} \mathbf{b}(F^k(0)/k) = \mathbf{b}(\chi(F))$ is the minimum of these mean payoffs.

The interest of the vectors z arising in the definition of Collatz-Wielandt numbers is to provide *approximate optimality certificates*, allowing us to bound mean payoffs from above and from below. Indeed, if $F(z) \leqslant \mu + z$, using the order-preserving property and additively homogeneity of F, we get that $F^k(z) \leqslant k\mu + z$ for all $k \in \mathbb{N}$, and, by nonexpansiveness of F, $\lim_{k\to\infty} \mathbf{t}(F^k(0)/k) = \lim_{k\to\infty} \mathbf{t}(F^k(z)/k) \leqslant \mu$. Similarly, if $F(z) \geqslant \mu + z$, we deduce that $\lim_{k\to\infty} \mathbf{b}(F^k(0)/k) \geqslant \mu$. The following result of [23], which can also be obtained as a corollary of a minimax result of Nussbaum [35], see [2], shows that these bounds are optimal.

▶ **Theorem 5** ([23, Prop. 2.1], [2, Lemma 2.8 and Rk. 2.10])**.** *Let F be an order-preserving and additively homogeneous self-map of $\mathbb{R}^n$. Then, $\lim_{k\to\infty} \mathbf{t}(F^k(x)/k) = \overline{\mathrm{cw}}(F)$ and $\lim_{k\to\infty} \mathbf{b}(F^k(x)/k) = \underline{\mathrm{cw}}(F)$ for any $x \in \mathbb{R}^n$.*

Thus, when F is the Shapley operator of a game, the quantities $\overline{\mathrm{cw}}(F)$ and $\underline{\mathrm{cw}}(F)$ respectively correspond to the greatest and smallest mean payoff among all the initial states.

A simpler situation arises when there is a vector $v \in \mathbb{R}^n$ and a scalar $\lambda \in \mathbb{R}$ such that

$$F(v) = \lambda + v\,. \tag{7}$$

The scalar λ, which is unique, is known as the *ergodic constant*, and (7) is referred to as the *ergodic equation*. Then, $\underline{\mathrm{cw}}(F) = \overline{\mathrm{cw}}(F) = \lambda$. The vector v is known as a *bias* or *potential*. It will be convenient to have a specific notation for the ergodic constant λ when the ergodic equation is solvable, then, we set $\mathrm{erg}(F) := \lambda$. The existence of a solution (λ, v) of (7) is guaranteed by certain "ergodicity" assumptions [3]. When the Shapley operator F is piecewise affine, it follows form Kohlberg's theorem (Theorem 4) that the ergodic equation (7) is solvable if and only if the mean payoff is independent of the initial state.

Denote $\bar{\mathbb{R}} \coloneqq \mathbb{R} \cup \{-\infty\}$. Properties (3) and (4) also make sense for self-maps of $\bar{\mathbb{R}}^n$, by requiring them to hold for all $x, y \in \bar{\mathbb{R}}^n$ and $\lambda \in \bar{\mathbb{R}}$. Any order-preserving and additively homogeneous self-map F of $\mathbb{R}^n$ admits a unique continuous extension $\bar{F}$ to $\bar{\mathbb{R}}^n$, obtained by setting, for $x \in \bar{\mathbb{R}}^n$,

$$\bar{F}(x) \coloneqq \inf\{F(y) \colon y \in \mathbb{R}^n, y \geqslant x\}\ . \tag{8}$$

Moreover, $\bar{F}$ is still order-preserving and additively homogeneous, see [18] for details. Hence, in the sequel, we assume that any order-preserving and additively homogeneous self-map F of $\mathbb{R}^n$ is canonically extended to $\bar{\mathbb{R}}^n$, and we will not distinguish between F and $\bar{F}$.

2.3 Entropy Games

Entropy games were introduced in [10]. We follow the presentation of [1] since it extends the original model, see Remark 9 for a comparison.

Similarly to stochastic turn-based zero-sum games, an *entropy game* is played on a digraph $(\mathscr{V}, \mathscr{E})$ in which the set of vertices $(\mathscr{V}, \mathscr{E})$ has a non-trivial partition $\mathscr{V} = \mathscr{V}_{\mathrm{Min}} \uplus \mathscr{V}_{\mathrm{Max}} \uplus \mathscr{V}_{\mathrm{Nat}}$. As in the case of stochastic turn-based games, players Min, Max, and Nature control the states $\mathscr{V}_{\mathrm{Min}}$, $\mathscr{V}_{\mathrm{Max}}$, $\mathscr{V}_{\mathrm{Nat}}$ respectively and they alternate their moves, i.e., $\mathscr{E} \subset \mathscr{V}_{\mathrm{Min}} \times \mathscr{V}_{\mathrm{Max}} \cup \mathscr{V}_{\mathrm{Max}} \times \mathscr{V}_{\mathrm{Nat}} \cup \mathscr{V}_{\mathrm{Nat}} \times \mathscr{V}_{\mathrm{Min}}$. We also suppose that the underlying graph satisfies Assumption 1. In the context of entropy games, player Min is called *Despot*, player Max is called *Tribune*, and Nature is called *People*. For this reason, we denote $\mathscr{V}_D \coloneqq \mathscr{V}_{\mathrm{Min}}$, $\mathscr{V}_T \coloneqq \mathscr{V}_{\mathrm{Min}}$, and $\mathscr{V}_P \coloneqq \mathscr{V}_{\mathrm{Nat}}$. The name "Tribune" coined in [10], refers to the magistrates interceding on behalf of the plebeians in ancient Rome.

The first difference between stochastic turn-based games and entropy games lies in the behavior of Nature: while in stochastic games Nature makes its decisions according to some fixed probability distribution, in entropy games People is a *nondeterministic* player, i.e., nothing is assumed about the behavior of People. The second difference lies in the definition of the payoffs received by Tribune. We suppose that every edge $(p, d) \in \mathscr{E}$ with $p \in \mathscr{V}_P$ and $d \in \mathscr{V}_D$ is equipped with a *multiplicity* m_{pd} which is a (positive) natural number. The *weight* of a path is defined to be the product of the weights of the arcs arising on this path. For instance, the path $(d_0, t_0, p_0, d_1, t_1, p_1, d_2, t_2)$ where $d_i \in \mathscr{V}_D$, $t_i \in \mathscr{V}_T$ and $p_i \in \mathscr{V}_P$, makes 2 and 1/3 turn, and its weight is $m_{p_0 d_1} m_{p_1 d_2}$. A *game in horizon* N is then defined as follows: if (σ, τ) is a pair of strategies of Despot and Tribune, then we denote by $R_d^N(\sigma, \tau)$ the sum of the weights of paths with initial state d that make N turns and that are consistent with the choice of (σ, τ). Tribune wants to maximize this quantity, while Despot wants to minimize it. As for stochastic turn-based games, a dynamic programming argument given in [1] shows that the value $V^N \in \mathbb{R}_{>0}^{\mathscr{V}_D}$ of this game does exist, and that it satisfies the recurrence

$$V^0 = \mathbf{1}, \qquad V^N = T(V^{N-1}),$$

where $\mathbf{1}$ is the vector whose entries are identically one and the operator $T \colon \mathbb{R}_{>0}^{\mathscr{V}_D} \to \mathbb{R}_{>0}^{\mathscr{V}_D}$ is defined by

$$T_d(x) \coloneqq \min_{(d,t)\in\mathscr{E}} \max_{(t,p)\in\mathscr{E}} \sum_{(p,l)\in\mathscr{E}} m_{pl} x_l, \text{ for all } d \in \mathscr{V}_D . \tag{9}$$

To define a game that lasts for an infinite number of turns, we consider the limit

$$V_d^\infty(\sigma, \tau) \coloneqq \limsup_{N\to+\infty} (R_d^N(\sigma, \tau))^{1/N},$$

which may be thought of as a measure of the freedom of People. The logarithm of this limit is known as a *topological entropy* in symbolic dynamics. The following result shows that the value of the entropy game V_d^∞ does exist and that it coincides with the limit of the renormalized value $(V_d^N)^{1/N} = [T^N(\mathbf{1})]_d^{1/N}$ of the finite horizon entropy game, so that the situation is similar to the case of stochastic turn-based games, albeit the renormalization now involves a Nth geometric mean owing to the multiplicative nature of the payment.

▶ **Theorem 6** ([1]). *The entropy game with initial state d has a value V_d^∞. Moreover, there are (positional) strategies σ^* and τ^* of Despot and Tribune, such that, for all $d \in \mathscr{V}_D$,*

$$V_d^\infty(\sigma^*, \tau) \leqslant V_d^\infty = V_d^\infty(\sigma^*, \tau^*) \leqslant V_d^\infty(\sigma, \tau^*),$$

for all strategies σ and τ of the two players. In addition, the value vector $V^\infty := (V_d^\infty)_{d\in\mathscr{V}_D}$ coincides with the vector

$$\lim_{N\to\infty} \big(T^N(\mathbf{1})\big)^{1/N} \in \mathbb{R}^{\mathscr{V}_D}_{>0} \ ,$$

in which the operation $\cdot^{1/N}$ is understood entrywise.

Entropy games can be cast in the general operator setting of Section 2.2, by introducing the conjugate operator $F\colon \mathbb{R}^{\mathscr{V}_D} \to \mathbb{R}^{\mathscr{V}_D}$,

$$F := \log \circ T \circ \exp \tag{10}$$

in which $\exp\colon \mathbb{R}^{\mathscr{V}_D} \mapsto \mathbb{R}^{\mathscr{V}_D}_{>0}$ is the map which applies the exponential entrywise, and $\log := \exp^{-1}$. Since the maps log and exp are order preserving, and since the weights m_{pl} appearing in the expression of $T(x)$ in (9) are nonnegative, the operator F is order preserving. Moreover, using the morphism property of the map log and exp with respect to multiplication and addition, we see that F is also additively homogeneous, hence, it is an abstract Shapley operator in the sense of Definition 2. Moreover, it is definable in the real exponential field, which was shown to be an o-minimal structure by Wilkie [45], and this is precisely how Theorem 6 is derived in [1] from Theorem 3. Actually, entropy games are studied in [1] in a more general setting, allowing history dependent strategies and showing that positional strategies are optimal. It is also shown there that the game has a uniform value in the sense of Mertens and Neyman [32].

When the (positional) strategies σ, τ are fixed, the value can be characterized by a classical result of Perron–Frobenius theory.

▶ **Definition 7.** *Given a pair of strategies (σ,τ) of Despot and Tribune, we define the* ambiguity matrix $M^{\sigma,\tau} \in \mathbb{R}^{\mathscr{V}_D\times\mathscr{V}_D}_{\geqslant 0}$*, with entries $(M^{\sigma,\tau})_{k,l} = m_{\tau(\sigma(k)),l}$ if $\big(\tau(\sigma(k)),l\big) \in \mathscr{E}$ and $(M^{\sigma,\tau})_{k,l} = 0$ otherwise, i.e., this is the weighted transition matrix of the subgraph $\mathscr{G}^{\sigma,\tau}$ obtained by keeping only the arcs $\mathscr{V}_D \to \mathscr{V}_T$ and $\mathscr{V}_T \to \mathscr{V}_P$ determined by the two strategies.*

The digraph $\mathscr{G}^{\sigma,\tau}$ can generally be decomposed in strongly connected components $\mathscr{C}_1,\dots,\mathscr{C}_s$, and each of these components, $\mathscr{C}_i$, determines a principal submatrix of $M^{\sigma,\tau}$, denoted by $M^{\sigma,\tau}[\mathscr{C}_i]$, obtained by keeping only the rows and columns in $\mathscr{C}_i \cap \mathscr{V}_D$. We denote by $\rho(\cdot)$ the spectral radius of a matrix, which is also known as the *Perron root* when the matrix is nonnegative and irreducible, see [12] for background.

▶ **Proposition 8** ([38], [46, Th. 5.1])**.** *The value of the subgame with initial state d, induced by a pair of strategies σ,τ, coincides with*

$$\max\{\rho(M^{\sigma,\tau}[\mathscr{C}_i])\colon \textit{there is a dipath } d\to\mathscr{C}_i \textit{ in } \mathscr{G}^{\sigma,\tau}\} \ .$$

▶ Remark 9. In the original model of Asarin et al. [10], an entropy game is specified by finite sets of states of Depot and Tribune, D and T, respectively, by a finite alphabet Σ representing actions, and by a transition relation $\Delta \subset T\times\Sigma\times D \cup D\times\Sigma\times T$. A turn consists of four successive moves by Despot, People, Tribune, and People: in state $d\in D$, Despot selects an action $a\in\Sigma$, then, People moves to one state $t\in P$ such that $(d,a,t)\in\Delta$. Then, Tribune selects an action $b\in\Sigma$, and People moves to one state $d'\in D$ such that $(t,b,d')\in\Delta$. This reduces to the model of [1] by introducing dummy states, identifying a turn in the game of [10] to a succession of two turns in the game of [1]. Another difference is that the payment, in [10], corresponds to $\max_{d\in D} \limsup_{N\to\infty}(R_d^N)^{1/N}$, and this is equivalent

Algorithm 1 Basic value iteration algorithm.

```
1: procedure ValueIteration(F)
2:   ▷ F a Shapley operator from ℝ^n to ℝ^n
3:   u := 0 ∈ ℝ^n
4:   repeat u := F(u)   ▷ At iteration ℓ, u = F^ℓ(0) is the value vector of the game in finite horizon ℓ
5:   until t(u) ⩽ 0 or b(u) ⩾ 0
6:   if t(u) ⩽ 0 then return "cw̄(F) ⩽ 0"   ▷ Player Min wins for all initial states
7:   else return "cw(F) ⩾ 0"   ▷ Player Max wins for all initial states
8:   end
9: end
```

to letting Tribune choose the initial state before playing the game. Then, the value of the game in [10] coincides with the maximum of the values of the initial states, $\max_d V_d^\infty$, see [1, Prop. 11]. Finally in [10], the arcs have multiplicity one, whereas we allow integer multiplicies (coded in binary), as in [1].

3 Bounding the Complexity of Value Iteration

In this section, F is an (abstract) Shapley operator, i.e., an order-preserving and additively homogeneous self-map of $\mathbb{R}^n$.

3.1 A Universal Complexity Bound for Value Iteration

The most straightforward idea to solve a mean-payoff game is probably value iteration: we infer whether or not the mean-payoff game is winning by solving the finite horizon game, for a large enough horizon. This is formalized in Algorithm 1.

When the non-linear eigenproblem $F(w) = \operatorname{erg}(F) + w$ is solvable, we shall use the following metric estimate, which represents the minimal Hilbert's seminorm of a bias vector

$$R(F) := \inf \{ \|w\|_{\mathrm{H}} \colon w \in \mathbb{R}^n,\ F(w) = \operatorname{erg}(F) + w \} \ .$$

In general, however, this non-linear eigenproblem may not be solvable. Then, we consider, for $\lambda \in \mathbb{R}$,

$$S_\lambda(F) = \{ v \in \mathbb{R}^n \colon \lambda + v \leqslant F(v) \}, \qquad S^\lambda(F) = \{ v \in \mathbb{R}^n \colon \lambda + v \geqslant F(v) \} \ .$$

▶ **Theorem 10.** *Procedure* ValueIteration *(Algorithm 1) is correct as soon as* $\underline{\mathrm{cw}}(F) > 0$ *or* $\overline{\mathrm{cw}}(F) < 0$*, and it terminates in a number of iterations* N_{vi} *bounded by*

$$\inf \left\{ \frac{\|v\|_{\mathrm{H}}}{\lambda} \colon \lambda > 0\,,\ v \in S_\lambda(F) \cup S^{-\lambda}(F) \right\} \ . \tag{11}$$

In particular, if F *has a bias vector and* $\operatorname{erg}(F) \neq 0$*, we have* $N_{vi} \leqslant \frac{R(F)}{|\operatorname{erg}(F)|}$.

We prove this theorem by using the Collatz-Wielandt variational characterization of the limits $\lim_{k\to\infty} \mathbf{t}(F^k(x)/k)$ and $\lim_{k\to\infty} \mathbf{b}(F^k(x)/k)$, see Theorem 5. A special case of Theorem 10 in which the existence of a bias vector is assumed appeared in [6] (without proof).

▶ Remark 11. The infimum in (11) is generally not attained. Consider for instance $F : \mathbb{R}^2 \to \mathbb{R}^2$ given by $F(x) = (\log(\exp(x_1) + \exp(x_2)), x_2) - \alpha$, where $\alpha > 0$. Then, since $F_2(x) = x_2 - \alpha < x_2$, we have $S_\lambda(F) = \emptyset$ for $\lambda > 0$. Besides, since $x - \lambda \geqslant F(x)$ if and only if $x_1 - \lambda \geqslant \log(\exp(x_1) + \exp(x_2)) - \alpha$ and $x_2 - \lambda \geqslant x_2 - \alpha$, it follows that $S^{-\lambda}(F) \neq \emptyset$

Algorithm 2 Value iteration in finite precision arithmetics.

procedure FPValueIteration($\tilde{F}$)
  $u \coloneqq 0 \in \mathbb{R}^n$, $\ell \coloneqq 0 \in \mathbb{N}$, $\epsilon \in \mathbb{R}_{>0}$
  repeat $u \coloneqq \tilde{F}(u)$; $\ell \coloneqq \ell + 1$ ▷ *We suppose that the operator F is evaluated in approximate arithmetics, so that $\tilde{F}(u)$ is at most at distance ϵ in the sup-norm from its true value $F(u)$.*
  until $\ell\epsilon + \mathbf{t}(u) \leqslant 0$ or $-\ell\epsilon + \mathbf{b}(u) \geqslant 0$
  if $\ell\epsilon + \mathbf{t}(u) \leqslant 0$ **then return** "$\overline{\mathrm{cw}}(F) \leqslant 0$" ▷ *Player Min wins for all initial states*
  end
  if $-\ell\epsilon + \mathbf{b}(u) \geqslant 0$ **then return** "$\underline{\mathrm{cw}}(F) \geqslant 0$" ▷ *Player Max wins for all initial states*
  end
end

Algorithm 3 Approximating the value of a mean-payoff game when it is independent of the initial state, and computing approximate optimality certificates, working in finite precision arithmetic.

procedure ApproximateConstantMeanPayoff(F)
  $u, x, y \coloneqq 0 \in \mathbb{R}^n$, $\ell \coloneqq 0 \in \mathbb{N}$, $\delta \in \mathbb{R}_{>0}$ ▷ *The number δ is the desired precision of approximation.*
  repeat $u \coloneqq \tilde{F}(u)$; $\ell \coloneqq \ell + 1$ ▷ *The operator F is evaluated in approximate arithmetic, so that $\tilde{F}(u)$ is at most at distance $\epsilon \coloneqq \delta/8$ in the sup-norm from its true value $F(u)$.*
  until $\mathbf{t}(u) - \mathbf{b}(u) \leqslant (3/4)\delta\ell$
$\kappa \coloneqq \mathbf{b}(u)/\ell$; $\lambda \coloneqq \mathbf{t}(u)/\ell$
$u \coloneqq 0$
  for $i = 1, 2, \dots, \ell - 1$ **do** $u \coloneqq \tilde{F}(u)$; $x \coloneqq \max\{x, -i\kappa + u\}$; $y \coloneqq \min\{y, -i\lambda + u\}$
  done
return "$[\underline{\mathrm{cw}}(F), \overline{\mathrm{cw}}(F)]$ is included in the interval $[\kappa - \delta/8, \lambda + \delta/8]$, which is of width at most δ. Furthermore, we have $\kappa - \delta/8 + x \leqslant F(x)$ and $\lambda + \delta/8 + y \geqslant F(y)$." ▷ *All initial states have a value in $[\kappa - \delta/8, \lambda + \delta/8]$.*
end

if and only if $\lambda < \alpha$. Now let $v \in S^{-\lambda}(F)$ for some $\lambda < \alpha$. Without loss of generality, we may assume $\mathbf{b}(v) = 0$. Then, we have $v_1 - \lambda \geqslant \log(\exp(v_1) + \exp(v_2)) - \alpha \geqslant \log 2 - \alpha$ and so $\frac{\|v\|_{\mathrm{H}}}{\lambda} \geqslant 1 + \frac{\log 2 - \alpha}{\lambda}$. We conclude that the infimum in (11) is equal to $\frac{\log 2}{\alpha}$ but it is not attained.

3.2 Value Iteration in Finite Precision Arithmetics

Algorithm 1 can be adapted to work in finite precision arithmetic. Consider the variant given in Algorithm 2. We assume that each evaluation of the Shapley operator F is performed with an error of at most $\epsilon > 0$ in the sup-norm. In this section, we denote by $\tilde{F} \colon \mathbb{R}^n \to \mathbb{R}^n$ the operator which approximates F, as in Procedure FPValueIteration, so it satisfies

$$\|\tilde{F}(x) - F(x)\|_\infty \leqslant \epsilon \text{ for all } x \in \mathbb{R}^n. \tag{12}$$

The following result is established by exploiting nonexpansiveness properties of Shapley operators.

▶ **Theorem 12.** *Procedure FPValueIteration (Algorithm 2) is correct as soon as $\underline{\mathrm{cw}}(F) > 2\epsilon$ or $\overline{\mathrm{cw}}(F) < -2\epsilon$, and it terminates in a number of iterations N^ϵ_{vi} bounded by*

$$\inf \left\{ \frac{\|v\|_{\mathrm{H}}}{\lambda - 2\epsilon} \colon \lambda > 2\epsilon\,,\ v \in S_\lambda(F) \cup S^{-\lambda}(F) \right\}\,. \tag{13}$$

In particular, if F has a bias vector and $|\operatorname{erg}(F)| > 2\epsilon$, we have $N^\epsilon_{vi} \leqslant \frac{R(F)}{|\operatorname{erg}(F)| - 2\epsilon}$.

Procedure ApproximateConstantMeanPayoff returns sub and super-eigenvectors x and y, satisfying $\kappa - \delta/8 + x \leqslant F(x)$ and $\lambda + \delta/8 + y \geqslant F(y)$, which, by Theorem 5, entails

that $[\underline{\mathrm{cw}}(F), \overline{\mathrm{cw}}(F)]$ is included in the interval $[\kappa - \delta/8, \lambda + \delta/8]$. The construction of these sub and sup-eigenvectors, by taking infima and suprema of normalized orbits of F, is inspired by [23, Proof of Lemma 2].

▶ **Theorem 13.** *Suppose that $\overline{\mathrm{cw}}(F) = \underline{\mathrm{cw}}(F)$, and let ρ denote this common value. Then, Procedure* ApproximateConstantMeanPayoff *(Algorithm 3) halts and is correct for any given desired precision of approximation $\delta \in \mathbb{R}_{>0}$. Furthermore, if $R \coloneqq \max\{\|v\|_{\mathrm{H}}, \|w\|_{\mathrm{H}}\}$, where $v, w \in \mathbb{R}^n$ are any two vectors that satisfy $\rho - \delta/8 + v \leqslant F(v)$ and $\rho + \delta/8 + w \geqslant F(w)$, then this procedure stops after at most $\lceil 8R/\delta \rceil$ iterations of the first loop.*

3.3 Finding the States of Maximal Value

In this section, we will show how the value iteration algorithm can be adapted to decide whether or not a given game has constant value, and to find the set of states that have the maximal value. Our analysis is based on an abstract notion of dominion. As previously, we suppose that $F\colon \mathbb{R}^n \to \mathbb{R}^n$ is an order-preserving and additively homogeneous operator. Recall that thanks to (8), F is canonically extended to define a self-map of $\bar{\mathbb{R}}^n$. Furthermore, given a nonempty set $\mathscr{S} \subset [n]$, we define the operator $F^{\mathscr{S}}\colon \bar{\mathbb{R}}^{\mathscr{S}} \to \bar{\mathbb{R}}^{\mathscr{S}}$ as $F^{\mathscr{S}} \coloneqq \mathbf{p}^{\mathscr{S}} \circ F \circ \mathbf{i}^{\mathscr{S}}$, where $\mathbf{p}^{\mathscr{S}}\colon \bar{\mathbb{R}}^n \to \bar{\mathbb{R}}^{\mathscr{S}}$ is the projection on the coordinates in $\mathscr{S}$ which is defined as usual by $\mathbf{p}^{\mathscr{S}}_j(x) = x_j$ for $j \in \mathscr{S}$, and $\mathbf{i}^{\mathscr{S}}\colon \bar{\mathbb{R}}^{\mathscr{S}} \to \bar{\mathbb{R}}^n$ is defined by $\mathbf{i}^{\mathscr{S}}_j(x) = x_j$ if $j \in \mathscr{S}$ and $\mathbf{i}^{\mathscr{S}}_j(x) = -\infty$ otherwise.

▶ **Definition 14.** *A* dominion (of Player Max) *is a nonempty set $\mathscr{D} \subset [n]$ such that $F^{\mathscr{D}}$ preserves $\mathbb{R}^{\mathscr{D}}$, i.e., such that $F^{\mathscr{D}}(x) \in \mathbb{R}^{\mathscr{D}}$ for all $x \in \mathbb{R}^{\mathscr{D}}$.*

As discussed in [7, 3], for stochastic mean-payoff games (with finite action spaces), a dominion of a player can be interpreted as a set of states such that the player can force the game to stay in this set if the initial state belongs to it. This terminology differs from the one of [29], in which a dominion is required in addition to consist only of initial states that are winning for this player. The algorithms that we discuss in this section require an additional assumption on the structure of the Shapley operator F.

▶ **Assumption 2.** *We assume that the limit $\chi^{\mathscr{D}} \coloneqq \lim_{\ell\to\infty} \frac{(F^{\mathscr{D}})^\ell(0)}{\ell} \in \mathbb{R}^{\mathscr{D}}$ exists for every dominion $\mathscr{D} \subset [n]$. Furthermore, we assume that the set $\mathscr{D}_{\max} \coloneqq \{j \in [n]\colon \chi^{[n]}_j = \overline{\mathrm{cw}}(F)\}$ is a dominion and that it satisfies $\underline{\mathrm{cw}}(F^{\mathscr{D}_{\max}}) = \overline{\mathrm{cw}}(F^{\mathscr{D}_{\max}}) = \overline{\mathrm{cw}}(F)$.*

▶ Remark 15. We note that the first part of Assumption 2 holds automatically when the Shapley operator $F\colon \mathbb{R}^n \to \mathbb{R}^n$ is definable in an o-minimal structure. Indeed, in this case the relation (8) implies that $F^{\mathscr{D}}$ is definable in the same structure for every dominion $\mathscr{D}$, so $\chi^{\mathscr{D}}$ exists by Theorem 3. We will see that the second part of the assumption applies to the games considered in this paper.

▶ Remark 16. Assumption 2 will allow us to make an induction on the number states, by a reduction to a simpler game with a reduced state space $\mathscr{D}$. In particular, the assumption that the limit $\chi^{\mathscr{D}} = \lim_{\ell\to\infty} \frac{(F^{\mathscr{D}})^\ell(0)}{\ell}$ exists will allow us to apply value iteration to the Shapley operator of the reduced game, $F^{\mathscr{D}}$.

From now on, we denote $\chi \coloneqq \chi^{[n]}$ and $\mathscr{D}_{\max} \coloneqq \{j \in [n]\colon \chi_j = \overline{\mathrm{cw}}(F)\}$. The following theorem applies to Shapley operators for which an a priori separation bound is known: if $\overline{\mathrm{cw}}(F) > \underline{\mathrm{cw}}(F)$, it requires an apriori bound $\delta > 0$ such that $\overline{\mathrm{cw}}(F) - \underline{\mathrm{cw}}(F) > \delta$. We note that the existence of the approximate sub and super-eigenvectors v and w used in this theorem follows from Theorem 5 and from Assumption 2.

Algorithm 4 Deciding if the value is constant.

```
1: procedure DecideConstantValue(F, δ, R)
2:     u := 0 ∈ R^n, ℓ := 0 ∈ N.
3:     F̃ := any map such that F̃(u) is at most at distance ϵ := δ/8 in the sup-norm from F(u).
4:     repeat u := F̃(u); ℓ := ℓ + 1
5:     until t(u) − b(u) ⩽ (3/4)δℓ or ℓ = 1 + ⌈8R/δ⌉
6:     if ℓ = 1 + ⌈8R/δ⌉ then
7:         S := {i : u_i = b(u)}
8:         return S    ▷ The value of the game depends on the initial state. We have χ_i < cw̄(F) for all i ∈ S.
9:     else
10:        return ∅    ▷ The value of the game is independent of the initial state.
11:    end
12: end
```

▶ **Theorem 17.** *Suppose that F is such that either $\overline{\mathrm{cw}}(F) = \underline{\mathrm{cw}}(F)$ or $\overline{\mathrm{cw}}(F) - \underline{\mathrm{cw}}(F) > \delta$ for some $\delta > 0$. Let $\mathscr{D}_{\max}$ be the set of states of maximal value and $R := \max\{\|v\|_{\mathrm{H}}, \|w\|_{\mathrm{H}}\}$, where $v, w \in \mathbb{R}^{\mathscr{D}_{\max}}$ are any two vectors that satisfy $\overline{\mathrm{cw}}(F) - \delta/8 + v \leqslant F^{\mathscr{D}_{\max}}(v)$ and $\overline{\mathrm{cw}}(F) + \delta/8 + w \geqslant F^{\mathscr{D}_{\max}}(w)$. Then, Procedure DecideConstantValue (Algorithm 4) is correct.*

Let

$$\mathrm{sep}(F) := \inf_{\mathscr{D}} \left(\overline{\mathrm{cw}}(F^{\mathscr{D}}) - \underline{\mathrm{cw}}(F^{\mathscr{D}})\right)$$

where the infimum is taken over all the dominions $\mathscr{D}$ of F which contain all the states of maximal value and satisfy $\overline{\mathrm{cw}}(F^{\mathscr{D}}) - \underline{\mathrm{cw}}(F^{\mathscr{D}}) > 0$.

To state the final result of this section, we will suppose that we have an access to an oracle that approximates F to a given precision $\epsilon > 0$. More precisely, given a point $x \in \bar{\mathbb{R}}^n$, the oracle is supposed to output a point $y \in \bar{\mathbb{R}}^n$ that satisfies $y_j = -\infty$ for all $j \in [n]$ such that $F_j(x) = -\infty$ and $|F_j(x) - y_j| \leqslant \epsilon$ for all j such that $F_j(x) \neq -\infty$. We have

▶ **Theorem 18.** *Let $\delta > 0$ be such that $\delta < \mathrm{sep}(F)$, $\mathscr{D}_{\max}$ be the set of states of maximal value and $R := \max\{\|v\|_{\mathrm{H}}, \|w\|_{\mathrm{H}}\}$, where $v, w \in \mathbb{R}^{\mathscr{D}_{\max}}$ are any two vectors that satisfy $\overline{\mathrm{cw}}(F) - \delta/8 + v \leqslant F^{\mathscr{D}_{\max}}(v)$ and $\overline{\mathrm{cw}}(F) + \delta/8 + w \geqslant F^{\mathscr{D}_{\max}}(w)$. Then, the set of initial states of maximal value can be found by making at most $n^2 + n\lceil 8R/\delta \rceil$ calls to oracle that approximates F to precision $\epsilon := \delta/8$.*

4 Application to Stochastic Mean-Payoff Games

In this section, we apply our results to stochastic mean-payoff games. We start by bounding the separation sep and the metric estimate $R(F)$, when F is the Shapley operator of a stochastic turn-based zero-sum game as in (1). We recall that the entries of A and B are integers. This is not more special than assuming that the entries of A and B are rational numbers (we may always rescale rational payments so that they become integers). We set

$$W := \max\{|A_{ij} - B_{ik}| : i \in \mathscr{V}_{\mathrm{Max}}, j \in \mathscr{V}_{\mathrm{Min}}, k \in \mathscr{V}_{\mathrm{Nat}}\} . \tag{14}$$

We also assume that the probabilities P_{kj} are rational, and that they have a common denominator $M \in \mathbb{N}_{>0}$, $P_{kj} = Q_{kj}/M$, where $Q_{kj} \in [M]$ for all $k \in \mathscr{V}_{\mathrm{Nat}}$ and $j \in \mathscr{V}_{\mathrm{Min}}$. We say that a state $k \in \mathscr{V}_{\mathrm{Nat}}$ is a *significant random state* if there are at least two indices $j, j' \in \mathscr{V}_{\mathrm{Min}}$ such that $P_{kj} > 0$ and $P_{kj'} > 0$. We denote by s the number of significant random states and by $n := |\mathscr{V}_{\mathrm{Min}}|$ the number of states controlled by Min. The following estimates follow from optimal bit-complexity results for Markov chains, established in [41]. These improve an estimate in [16].

▶ **Lemma 19.** *We have* $\operatorname{sep}(F) > 1/(nM^{\min\{s,n-1\}})^2$.

▶ **Lemma 20.** *Suppose that* $\underline{\mathrm{cw}}(F) = \overline{\mathrm{cw}}(F)$. *Then, there exists a vector* $u \in \mathbb{R}^{\mathcal{V}_{\mathrm{Min}}}$ *such that* $F(u) = \overline{\mathrm{cw}}(F) + u$ *and*

$$R(F) \leqslant \|u\|_{\mathrm{H}} \leqslant 8nWM^{\min\{s,n-1\}} \ .$$

The existence of the bias vector follows from Kohlberg's theorem (Theorem 4). The bias is generally not unique (even up to an additive constant) and the main difficulty then is to find a "short" bias. The one which is constructed in the proof of this lemma relies on the notion of Blackwell optimality. This notion requires to consider the *discounted* version of the game, in which the payment (2) is replaced by $\mathbb{E}_{\sigma\tau}\sum_{p=0}^{\infty}(1-\alpha)^p(-A_{i_pj_p}+B_{i_pk_p})$, where $0<\alpha<1$ and $1-\alpha$ is the discount factor. The discounted game with initial state i has a value, $x_i(\alpha)$, and the value vector, $x(\alpha)=(x_i(\alpha))\in\mathbb{R}^n$ is the unique solution of the fixed point problem $x(\alpha)=F((1-\alpha)x(\alpha))$. Then, a strategy of a player is *Blackwell optimal* if it is optimal in all the discounted games with a discount factor sufficiently close to 1. It can be obtained by selecting minimizing or maximizing actions when evaluating the expression $F((1-\alpha)x(\alpha))$, for $\alpha>0$ close enough to 0. Moreover, it follows from Kohlberg's proof that $x(\alpha)$ admits an expansion $x(\alpha)=\chi(F)/\alpha+u+o(\alpha)$ when $\alpha\to 0^+$, where u is a bias vector. If Q is the stochastic matrix determined by a pair of Blackwell optimal strategies of the two players, and if r is the associated one-stage payment vector, then, the bias vector u satisfies a Poisson-type equation $\chi(F)+u=r+Qu$. Moreover, this special bias vector has the remarkable property of having a zero expectation with respect to all invariant measures of Q, and together with the bit-complexity estimate of [41, Th. 1.5] for the solution of Poisson-type equations, this leads to the proof of Lemma 20.

Thanks to these estimates, we arrive at the following corollaries.

▶ **Corollary 21.** *Let* F *be a Shapley operator as above, supposing that* F *has a bias vector and that* $\operatorname{erg}(F)$ *is nonzero. Then, procedure* ValueIteration *stops after*

$$N_{vi} \leqslant 8n^2WM^{2\min\{s,n-1\}} \tag{15}$$

iterations and correctly decides which of the two players is winning.

▶ Remark 22. When specialized to deterministic mean-payoff games, i.e., when $s=0$, Corollary 21 yields $N_{\mathrm{vi}}=O(n^2W)$ which is precisely the bound that follows from the analysis of value iteration by Zwick and Paterson [47].

▶ **Corollary 23.** *Suppose that* F *has a bias vector and let* $\mu \coloneqq nM^{\min\{s,n-1\}}$. *Then, Procedure* ApproximateConstantMeanPayoff, *applied to* F *and to* $\delta \coloneqq \mu^{-2}$, *terminates in at most*

$$128n^3WM^{3\min\{s,n-1\}}$$

calls to the oracle. Moreover the interval returned by this procedure contains a unique rational number of denominator at most μ, *which coincides with the value, and optimal policies can be obtained from the approximate optimality certificates generated by the procedure.*

Let us explain how the optimal strategies are obtained from the output of Procedure ApproximateConstantMeanPayoff. This procedure returns sub and super-eigenvectors x and y that satisfy $\kappa-\delta/8+x\leqslant F(x)$ and $\lambda+\delta/8+y\geqslant F(y)$ where $\lambda=\operatorname{erg}(F)$. By selecting, for each state $j\in\mathcal{V}_{\mathrm{Min}}$, a minimizing action in the expression

$$F_j(y)=\min_{(j,i)\in\mathcal{E}}\Big(-A_{ij}+\max_{(i,k)\in\mathcal{E}}\big(B_{ik}+\sum_{(k,l)\in\mathcal{E}}P_{kl}y_k\big)\Big),$$

one gets a positional strategy which guarantees to Min a value at most $\lambda + \delta/8$. A similar method is used to construct a positional strategy of Max. Then, since $\delta = \mu^{-2}$ is smaller than the separation bound between values of different strategies, we deduce these policies guarantee a value of λ to each of the players, and so, they are optimal.

▶ **Corollary 24.** *Let $\mu \coloneqq nM^{\min\{s,n-1\}}$. Then, we can find the set of states with maximal value of a stochastic mean-payoff game by performing at most $65n^4WM^{3\min\{s,n-1\}}$ calls to the oracle approximating F with precision $\delta \coloneqq 1/\mu^2$.*

▶ Remark 25. If we are given the matrices A, B, P explicitly, then the operator F can be evaluated exactly in $O(E)$ complexity, where E is the number of edges of the graph representing a stochastic mean-payoff game. In particular, there is no need to construct an approximation oracle in order to apply the results of this section. Nevertheless, even in this case it may be beneficial to use an approximation oracle. Indeed, if we evaluate F exactly, then each value iteration $u \coloneqq F(u)$ increases the number of bits needed to encode u. As a result, value iteration would require exponential memory. In order to avoid this problem, one can replace F with an approximation oracle $\tilde{F}$ obtained as follows. Let $\mu \coloneqq nM^{\min\{s,n-1\}}$ and $\epsilon \coloneqq 1/(8\mu^2)$. Given $x \in \bar{\mathbb{R}}^{\mathcal{V}_{\mathrm{Min}}}$, we first compute $y \coloneqq F(x)$ exactly and then round the finite coordinates of y in such a way that the rounded vector $\tilde{y}$ satisfies $|y_j - \tilde{y}_j| \leqslant \epsilon$ whenever $y_j \neq -\infty$ and $\tilde{y}_j$ is a rational number with denominator at most $8\mu^2$. One can check that if we use $\tilde{F}$ obtained in this way as an approximation oracle, then all algorithms presented in this section require $O\big(nE\log(nMW)\big)$ memory, which is polynomial in the size of the input.

▶ Remark 26. Since a single call to the oracle approximating F can be done in $O(E)$ arithmetic operations, by combining Corollary 24 with Corollary 23 we see that the set of states with maximal value, and a pair of optimal strategies within this set can be found in $O(n^4EWM^{3\min\{s,n-1\}})$ complexity. This should be compared with the algorithm BWR-FindTop from [16] which achieves the same aim using a pumping algorithm instead of value iteration. If we combine the estimate from [41] with the complexity bound presented in [16] for the pumping algorithm, then we get that BWR-FindTop has $O(V^6EWs2^sM^{4s} + V^3EW\log W)$ complexity, where V is the number of vertices of the graph. In particular, our result gives a better complexity bound. Furthermore, the authors of [16] show that, given an oracle access to BWR-FindTop and to another oracle that solves deterministic mean-payoff games, one can completely solve stochastic mean-payoff games with pseudopolynomial number of calls to these oracles, provided that s is fixed. Hence, we can speed-up this algorithm by replacing the oracle BWR-FindTop with our algorithms.

5 Solving Entropy Games With Bounded Rank

Recall that the dynamic programming operator T of an entropy game, as well as its conjugate F, which we call the Shapley operator of an entropy game, were defined in (9),(10). As in the last section, we denote $n \coloneqq |\mathcal{V}_D|$ and we put $W \coloneqq \max_{(p,k)\in\mathcal{E}} m_{pk}$.

We define the *rank* of the entropy game to be the maximum of the ranks of the ambiguity matrices, see Definition 7. The following result is established by combining a separation bound of Rump [39] for algebraic numbers, with bounds on determinants of nonnegative matrices with entries in an interval, building on the study of Hadamard's maximal determinant problem for matrices with entries in $\{0,1\}$ [20].

▶ **Theorem 27.** *Suppose two pairs of strategies yield distinct values in an entropy game of rank r, with n Despot's states. Then, these values differ at least by $\nu_{n,r}^{-1}$ where*

$$\nu_{n,r} \coloneqq 2^r(r+1)^{8r}r^{-2r^2+r+1}(ne)^{4r^2}\big(1 \vee \frac{W}{2}\big)^{4r^2} \quad .$$

We show that the value of an entropy game is always in the interval $[1, nW]$. Then, a separation bound for values of different pairs of strategies entails a separation bound for their logarithm, differing only by a nW factor:

▶ **Corollary 28.** *Suppose two pairs of strategies yield distinct values in an entropy game of rank r, with n Despot's states. Then, the logarithms of these values differ at least by $\hat{\nu}_{n,r}^{-1}$ where*

$$\hat{\nu}_{n,r} \coloneqq nW\nu_{n,r} \ .$$

▶ **Proposition 29.** *Let $0 < \delta < 1$. Then, there exist vectors $w, z \in \mathbb{R}_{>0}^{\mathscr{V}_D}$ such that $e^{-\delta}\mathbf{b}(V^\infty)w \leqslant T(w)$, $e^{\delta}\mathbf{t}(V^\infty)z \geqslant T(z)$, and $\max\{\|\log w\|_{\mathrm{H}}, \|\log z\|_{\mathrm{H}}\} \leqslant 1200(n^3 \log W + n^2 \log \delta^{-1})$.*

This proposition is established by observing that for a given value of δ, w and z are defined by semi-linear constraints, and by using bitlength estimates on the generators and vertices of polyhedra defined by inequalities.

By applying Theorem 18 to the Shapley operator $F = \log \circ T \circ \exp$, and by using the bounds Corollary 28 and Proposition 29, we get:

▶ **Theorem 30.** *In an entropy game of rank at most r, we can find the set of initial states with maximal value by performing $O(nR_{n,r}\hat{\nu}_{n,r})$ calls to an oracle approximating F with precision $\delta/8$ where $\delta = (\hat{\nu}_{n,r})^{-1}$.*

The following decomposition property for entropy games extends a classical property of deterministic mean payoff games. Once the set of Despot's states with maximal value is known, it allows one to determine the value of the other states by reduction to an entropy game induced by the other states of Despot. To state this property formally, we denote by $\mathscr{D}_{\max} \subset \mathscr{V}_D$ the states of Despot of maximal value, and we put ${\mathscr{V}_P}^{\mathscr{D}_{\max}} \coloneqq \{p \in \mathscr{V}_P \colon \exists k \in \mathscr{D}_{\max}, (p,k) \in \mathscr{E}\}$ and ${\mathscr{V}_T}^{\mathscr{D}_{\max}} \coloneqq \{t \in \mathscr{V}_T \colon \exists p \in {\mathscr{V}_P}^{\mathscr{D}_{\max}}, (t,p) \in \mathscr{E}\}$.

▶ **Lemma 31** (Decomposition property)**.** *Let $\mathscr{S}_1 \coloneqq \mathscr{D}_{\max} \uplus {\mathscr{V}_T}^{\mathscr{D}_{\max}} \uplus {\mathscr{V}_P}^{\mathscr{D}_{\max}}$ and $\mathscr{S}_2 \coloneqq \mathscr{V} \setminus \mathscr{S}_1$. Furthermore, suppose that $\mathscr{S}_2$ is nonempty. Consider the induced digraphs $\mathscr{G}[\mathscr{S}_1]$ and $\mathscr{G}[\mathscr{S}_2]$ of the original graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$. Then, the entropy games arising by restricting the graph to $\mathscr{G}[\mathscr{S}_1]$ and $\mathscr{G}[\mathscr{S}_2]$ satisfy Assumption 1. Furthermore, if (σ_1, τ_1) are optimal strategies of Despot an Tribune in the induced entropy game on $\mathscr{G}[\mathscr{S}_1]$ and (σ_2, τ_2) are optimal strategies of Despot and Tribune in the induced entropy game on $\mathscr{G}[\mathscr{S}_2]$, then the joint strategies*

$$\forall k \in \mathscr{V}_D,\ \sigma(k) = \begin{cases} \sigma_1(k) & \text{if } k \in \mathscr{D}_{\max}, \\ \sigma_2(k) & \text{otherwise,} \end{cases} \quad \forall t \in \mathscr{V}_T,\ \tau(t) = \begin{cases} \tau_1(t) & \text{if } t \in {\mathscr{V}_T}^{\mathscr{D}_{\max}}, \\ \tau_2(t) & \text{otherwise.} \end{cases} \tag{16}$$

are optimal in the original game.

We also note that the Shapley operator of an entropy game can be approximated in polynomial time, this follows by using a result of Borwein and Borwein [17] on the approximation of the log and exp maps, together with a scaling argument, see [1, Lemma 27]. Then, by combining Theorem 30 and Lemma 31, we get:

▶ **Corollary 32.** *A pair of optimal policies of an entropy game of rank r can be found in $O(n^2 R_{n,r}\hat{\nu}_{n,r})$ calls to an oracle that return F with a precision of $1/(16\hat{\nu}_{n,r})$. Then, entropy games in the original model of Asarin et al. [10] and with a fixed rank are polynomial-time solvable, whereas entropy games with weights, in the model of Akian et al. [1], and with a fixed rank, are pseudo-polynomial time solvable.*

▶ **Corollary 33.** *Entropy games with weights and with a fixed number of People's positions are pseudo-polynomial time solvable.*

We say that a state of a game is *significant* if there are several options in this state, in particular, a state p of People is significant if there are at least two distinct arcs (p,k) and (p,l) in $\mathscr{E}$. We may ask whether the statement of Corollary 33 carries over to entropy games with a fixed number of significant People's states. The following result shows that this can not be derived from the universal value iteration bounds, since value iteration needs $\Omega(W^{n-1})$ iterations to recognize the optimal strategy.

▶ **Theorem 34.** *There is a family of $G_n(W)$ of Despot-free entropy games, and a constant $C > 0$, with the following properties:*

1. *$G_n(W)$ has arc weights $\leqslant W$, only one significant Tribune's position, with two actions, and $2n+1$ People's positions among which there are only 4 significant positions;*
2. *The action of Tribune that is optimal in the mean-payoff entropy game is never played, if Tribune plays optimally in the entropy game of finite horizon k, for all $k \leqslant CW^{n-1}$.*

Let us explain how the game $G_n(W)$ is constructed. We start by estimating the positive root of a special polynomial p_n.

▶ **Proposition 35.** *Consider the polynomial $p_n(x) = x^n - W(x^{n-1} + \cdots + 1)$, where $W > 0$. Then, p_n has a unique positive root, $x_n(W)$, which satisfies*

$$x_n(W) = W + 1 - 1/W^{n-1} + o(1/W^{n-1}) \ , \qquad \text{as } W \to \infty \ .$$

This is established by applying the Newton-Puiseux algorithm [44] to the equation $p_n(x) = 0$ parameterized by W. Then, we define the companion matrix of p_n, $A_n = A_n(W)$, together with the sequence $(z(k))_{k\in\mathbb{N}}$,

$$A_n(W) := \left(\begin{array}{ccc|c} W & \cdots & W & W \\ \hline & I_{n-1} & & 0 \end{array}\right) \ , \ z(k) = \max(\mathbf{1}_n^\top A_n^k \mathbf{1}_n, \alpha \mathbf{1}_{n-1}^\top A_{n-1}^k \mathbf{1}_{n-1}) \ , \tag{17}$$

where for all $k \geqslant 1$, $\mathbf{1}_k$ is the vector of dimension k with unit entries, I_k the identity matrix of dimension k, and $\alpha > 1$. We note that A_n is an irreducible nonnegative matrix, so, the unique positive root of p_n is actually the Perron root of A_n. Using Proposition 35, as well as explicit bounds for the left Perron eigenvector of A_n, we can show that the maximum in (17) is achieved by the rightmost term for $k \leqslant k^*$ and by the leftmost term for $k > k^*$ where $k^* = (\log 2)W^{n-1} + o(W^{n-1})$. Moreover, $z(k)$ can be interpreted as the value in horizon $k+1$ of an entropy game satisfying the conditions of the theorem. Indeed, the leftmost term $\mathbf{1}_n^\top A_n^k \mathbf{1}_n$ can be interpreted as the value in horizon $k+1$ of a Despot-free and Tribune-free entropy game, with $n+1$ People's states, among which there are only two significant states: one encoding the first row of A_n, and another one encoding the row vector $\mathbf{1}_n^\top$. The term $\alpha \mathbf{1}_{n-1}^\top A_{n-1}^k \mathbf{1}_{n-1}$ also admits an interpretation as the value of a similar game. We finally construct the entropy game $G_n(W)$ by allowing Tribune to choose whichever of these elementary games is played. This can be implemented by taking the disjoint union of the graphs of the two elementary games, and adding one significant state of Tribune, with only two actions. One action gives rise to the left term of the maximum in (17), whereas the other action gives rise to the right term, so that the value of the corresponding entropy game in horizon k is precisely $z(k-1)$. Since $\lambda_n > \lambda_{n-1}$, in the mean-payoff entropy game, the optimal action for Tribune is to choose the term with the highest geometric growth, i.e., to play "left", which guarantees a geometric growth of λ_n. However, for $k \leqslant k^* + 1$, the optimal action of Tribune in the game of horizon k is always to play "right". This shows Theorem 34.

6 Concluding Remarks

We developed generic value iteration algorithms, which apply to various classes of zero-sum games with mean payoffs. These algorithms admit universal complexity bounds, in an approximate oracle model – we only need an oracle evaluating approximately the Shapley operator. These bounds involve three fundamental ingredients: the number of states, a separation bound between the values induced by different strategies, and a bound on the norms of Collatz-Wielandt vectors. We showed that entropy games with a fixed rank (and in particular, entropy games with a fixed number of People's states) are pseudo-polynomial time solvable. This should be compared with the result of [1], showing that entropy games with a fixed number of Despot positions are polynomial-time solvable. Since fixing the number of states of Despot or People leads to improved complexity bounds, one may ask whether entropy games with a fixed number of significant Tribune states are polynomial or at least pseudo-polynomial, this is still an open question.

References

1 M. Akian, S. Gaubert, J. Grand-Clément, and J. Guillaud. The operator approach to entropy games. *Theor. Comp. Sys.*, 63(5):1089–1130, July 2019. doi:10.1007/s00224-019-09925-z.

2 M. Akian, S. Gaubert, and A. Guterman. Tropical polyhedra are equivalent to mean payoff games. *Int. J. Algebra Comput.*, 22(1):125001 (43 pages), 2012. doi:10.1142/S0218196711006674.

3 M. Akian, S. Gaubert, and A. Hochart. A game theory approach to the existence and uniqueness of nonlinear Perron-Frobenius eigenvectors. *Discrete & Continuous Dynamical Systems - A*, 40:207–231, 2020. doi:10.3934/dcds.2020009.

4 M. Akian, S. Gaubert, and R. Nussbaum. A Collatz-Wielandt characterization of the spectral radius of order-preserving homogeneous maps on cones. arXiv:1112.5968, 2011.

5 M. Akian, A. Sulem, and M. I. Taksar. Dynamic optimization of long-term growth rate for a portfolio with transaction costs and logarithmic utility. *Mathematical Finance*, 11(2):153–188, April 2001. doi:10.1111/1467-9965.00111.

6 X. Allamigeon, S. Gaubert, R. D. Katz, and M. Skomra. Condition numbers of stochastic mean payoff games and what they say about nonarchimedean semidefinite programming. In *Proceedings of the 23rd International Symposium on Mathematical Theory of Networks and Systems (MTNS)*, pages 160–167, 2018. URL: http://mtns2018.ust.hk/media/files/0213.pdf.

7 X. Allamigeon, S. Gaubert, and M. Skomra. Solving generic nonarchimedean semidefinite programs using stochastic game algorithms. *J. Symbolic Comput.*, 85:25–54, 2018. doi:10.1016/j.jsc.2017.07.002.

8 V. Anantharam and V. S. Borkar. A variational formula for risk-sensitive reward. *SIAM J. Contro Optim.*, 55(2):961–988, 2017. arXiv:1501.00676.

9 D. Andersson and P. B. Miltersen. The complexity of solving stochastic games on graphs. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, volume 5878 of *Lecture Notes in Comput. Sci.*, pages 112–121. Springer, 2009. doi:10.1007/978-3-642-10631-6_13.

10 E. Asarin, J. Cervelle, A. Degorre, C. Dima, F. Horn, and V. Kozyakin. Entropy games and matrix multiplication games. In *Proceedings of the 33rd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 11:1–11:14, Wadern, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2016.11.

11 D. Auger, X. Badin de Montjoye, and Y. Strozecki. A generic strategy improvement method for simple stochastic games. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27,*

2021, Tallinn, Estonia, volume 202 of *LIPIcs*, pages 12:1–12:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.12.

12 A. Berman and R.J. Plemmons. *Nonnegative matrices in the mathematical sciences*. SIAM, 1994.

13 T. Bewley and E. Kohlberg. The asymptotic theory of stochastic games. *Math. Oper. Res.*, 1(3):197–208, 1976. doi:10.1287/moor.1.3.197.

14 J. Bolte, S. Gaubert, and G. Vigeral. Definable zero-sum stochastic games. *Mathematics of Operations Research*, 40(1):171–191, 2014. doi:10.1287/moor.2014.0666.

15 E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. A convex programming-based algorithm for mean payoff stochastic games with perfect information. *Optim. Lett.*, 11(8):1499–1512, 2017. doi:10.1007/s11590-017-1140-y.

16 E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. A pseudo-polynomial algorithm for mean payoff stochastic games with perfect information and few random positions. *Inform. and Comput.*, 267:74–95, 2019. doi:10.1016/j.ic.2019.03.005.

17 J. M. Borwein and P. B. Borwein. On the complexity of familiar functions and numbers. *SIAM Review*, 30(4):589–601, 1988.

18 A. D. Burbanks, R. D. Nussbaum, and C. T. Sparrow. Extension of order-preserving maps on a cone. *Proc. Roy. Soc. Edinburgh Sect. A*, 133(1):35–59, 2003. doi:10.1017/S0308210500002274.

19 K. Chatterjee and R. Ibsen-Jensen. The complexity of ergodic mean-payoff games. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 8573 of *Lecture Notes in Comput. Sci.*, pages 122–133. Springer, 2014. doi:10.1007/978-3-662-43951-7_11.

20 John H. E. Cohn. On the value of determinants. *Proceedings of the American Mathematical Society*, 14(4):581–588, 1963.

21 A. Condon. The complexity of stochastic games. *Inform. and Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.

22 O. Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Logical Methods in Computer Science*, 7(3:19):1–42, 2011.

23 S. Gaubert and J. Gunawardena. The Perron-Frobenius theorem for homogeneous, monotone functions. *Trans. Amer. Math. Soc.*, 356(12):4931–4950, 2004. doi:10.1090/S0002-9947-04-03470-1.

24 H. Gimbert and F. Horn. Simple stochastic games with few random vertices are easy to solve. In *Proceedings of the 11th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS)*, volume 4962 of *Lecture Notes in Comput. Sci.*, pages 5–19. Springer, 2008. doi:10.1007/978-3-540-78499-9_2.

25 V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and finding minimax mean cycles in digraphs. *Zh. Vychisl. Mat. Mat. Fiz.*, 28(9):1406–1417, 1988. doi:10.1016/0041-5553(88)90012-2.

26 A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Manag. Sci.*, 12(5):359–370, 1966. doi:10.1287/mnsc.12.5.359.

27 R. A. Howard and J. E. Matheson. Risk-sensitive markov decision processes. *Management Science*, 18(7):356–369, 1972. doi:10.1287/mnsc.18.7.356.

28 R. Ibsen-Jensen and P. B. Miltersen. Solving simple stochastic games with few coin toss positions. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, volume 7501 of *Lecture Notes in Comput. Sci.*, pages 636–647. Springer, 2012. doi:10.1007/978-3-642-33090-2_55.

29 M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008. doi:10.1137/070686652.

30 E. Kohlberg. Invariant half-lines of nonexpansive piecewise-linear transformations. *Math. Oper. Res.*, 5(3):366–372, 1980. doi:10.1287/moor.5.3.366.

31 T. M. Liggett and S. A. Lippman. Stochastic games with perfect information and time average payoff. *SIAM Rev.*, 11(4):604–607, 1969. doi:10.1137/1011093.

32 J.-F. Mertens and A. Neyman. Stochastic games. *Internat. J. Game Theory*, 10(2):53–66, 1981. doi:10.1007/BF01769259.

33 J.-F. Mertens, S. Sorin, and S. Zamir. *Repeated games*, volume 55 of *Econom. Soc. Monogr.* Cambridge University Press, Cambridge, 2015. doi:10.1017/CBO9781139343275.

34 A. Neyman. Stochastic games and nonexpansive maps. In A. Neyman and S. Sorin, editors, *Stochastic Games and Applications*, volume 570 of *NATO Science Series C*, pages 397–415. Kluwer Academic Publishers, 2003. doi:10.1007/978-94-010-0189-2_26.

35 R. D. Nussbaum. Convexity and log convexity for the spectral radius. *Linear Algebra Appl.*, 73:59–122, 1986. doi:10.1016/0024-3795(86)90233-8.

36 D. Rosenberg and S. Sorin. An operator approach to zero-sum repeated games. *Israel J. Math.*, 121(1):221–246, 2001. doi:10.1007/BF02802505.

37 U. G. Rothblum. Multiplicative markov decision chains. *Mathematics of Operations Research*, 9(1):6–24, 1984.

38 U. G. Rothblum and P. Whittle. Growth optimality for branching markov decision chains. *Mathematics of Operations Research*, 7(4):582–601, 1982.

39 S. M. Rump. Polynomial minimum root separation. *Mathematics of Computation*, 145(33):327–336, 1979.

40 M. Skomra. *Tropical spectrahedra: Application to semidefinite programming and mean payoff games*. PhD thesis, Université Paris-Saclay, 2018. URL: https://pastel.archives-ouvertes.fr/tel-01958741.

41 M. Skomra. Optimal bounds for bit-sizes of stationary distributions in finite Markov chains. arXiv:2109.04976, 2021.

42 K. Sladký. *On dynamic programming recursions for multiplicative Markov decision chains*, pages 216–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 1976. doi:10.1007/BFb0120753.

43 L. van den Dries. *Tame topology and o-minimal structures*, volume 248 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1998. doi:10.1017/CBO9780511525919.

44 R. J. Walker. *Algebraic Curves*. Springer, New York, 1978.

45 A. J. Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted Pfaffian functions and the exponential function. *J. Amer. Math. Soc.*, 9(4):1051–1094, 1996.

46 W. H. M. Zijm. Asymptotic expansions for dynamic programming recursions with general nonnegative matrices. *J. Optim. Theory Appl.*, 54(1):157–191, 1987. doi:10.1007/BF00940410.

47 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoret. Comput. Sci.*, 158(1–2):343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

Computability of Finite Simplicial Complexes

Djamel Eddine Amir
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Mathieu Hoyrup
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

The topological properties of a set have a strong impact on its computability properties. A striking illustration of this idea is given by spheres and closed manifolds: if a set X is homeomorphic to a sphere or a closed manifold, then any algorithm that semicomputes X in some sense can be converted into an algorithm that fully computes X. In other words, the topological properties of X enable one to derive full information about X from partial information about X. In that case, we say that X has computable type. Those results have been obtained by Miller, Iljazović, Sušić and others in the recent years. A similar notion of computable type was also defined for pairs (X, A) in order to cover more spaces, such as compact manifolds with boundary and finite graphs with endpoints.

We investigate the higher dimensional analog of graphs, namely the pairs (X, A) where X is a finite simplicial complex and A is a subcomplex of X. We give two topological characterizations of the pairs having computable type. The first one uses a global property of the pair, that we call the ϵ-surjection property. The second one uses a local property of neighborhoods of vertices, called the surjection property. We give a further characterization for 2-dimensional simplicial complexes, by identifying which local neighborhoods have the surjection property.

Using these characterizations, we give non-trivial applications to two famous sets: we prove that the dunce hat does not have computable type whereas Bing's house does. Important concepts from topology, such as absolute neighborhood retracts and topological cones, play a key role in our proofs.

2012 ACM Subject Classification Theory of computation → Computability; Mathematics of computing → Point-set topology; Mathematics of computing → Algebraic topology

Keywords and phrases Computable Type, Simplicial Complex, Surjection Property, Topological Cone, Absolute Neighborhood Retract, Dunce Hat, Bing's House

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.111

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://hal.inria.fr/hal-03564904` [14]

Acknowledgements We want to thank Guilhem Gamard and Emmanuel Jeandel for interesting discussions and comments about this work.

1 Introduction

Computable analysis is a theory formalizing computations on real numbers using finite but arbitrary precision, and allowing to investigate the theoretical possibility of solving problems on real numbers. The computable aspects of topology are an important research topic in computable analysis. Computability of homology groups was investigated in [10], computability of planar continua in [18], computability of the Brouwer fixed-point theorem was studied in [20] and [3], and computability of Polish spaces is addressed in [11].

A particularly rich topic is the computability of subsets of the plane and of Euclidean spaces. For instance, the computability of Julia sets has thoroughly been studied [4], the computability of the Mandelbrot set is still an open problem [13] and the computability of the set of solutions of a computable equation is generally a non-trivial problem [21].


These studies reveal that many natural definitions of sets induce a *semi-algorithm*, and finding a proper *algorithm* computing the set can be challenging. Informally, a compact subset of the plane is *semicomputable* if there is an algorithm that for each pixel, semidecides whether the pixel is disjoint from the set, i.e. halts exactly in that case. A compact subset of the plane is *computable* if there is an algorithm that decides, for each pixel, whether it intersects the set. This idea can be generalized to higher dimensions, and to subsets of many mathematical spaces.

Although semicomputability of compact sets is strictly weaker than computability in general, it turns out that they are equivalent for many natural sets, and that this phenomenon comes from the topological properties of these sets. For instance, it was prove by Miller [19] that semicomputability and computability are equivalent for spheres, and for every set that is homeomorphic to a sphere. This result leads to the following definition: say that a compact space X has **computable type** if any semicomputable set Y that is homeomorphic to X is actually computable. This property has been intensively studied by Miller [19] and more recently by Iljazović and his co-authors [5, 17, 9, 15, 6, 7] in the recent years. A striking aspect of this property is that it builds a bridge between computability theory and topology. The following results were obtained:

- The n-dimensional sphere $\mathbb{S}_n$ (which is the higher dimensional analog of the circle) has computable type [19],
- Every closed n-manifold (these are compact spaces which are locally homeomorphic to $\mathbb{R}^n$, e.g. the n-dimensional sphere and the n-dimensional torus) has computable type [17].

A line segment or a disk fails to have this property: it is not difficult to build a semicomputable disk which is not computable. However, a similar result can be proved if one requires in addition that the boundary of the set is semicomputable. It leads to the following generalization from compact spaces X to pairs (X, A) where X and $A \subseteq X$ are compact: a pair (X, A) has **computable type** if for any semicomputable pair (Y, B) that is homeomorphic to (X, A), Y is computable. The following results have been obtained for pairs:

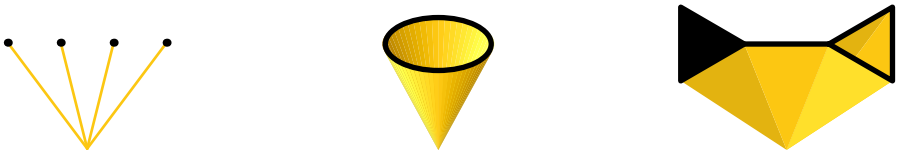
- The n-dimensional ball (which is the higher dimensional analog of the disk) with its bounding sphere $(\mathbb{B}_n, \mathbb{S}_{n-1})$ has computable type [19],
- Every compact manifold with boundary $(M, \partial M)$ has computable type [17],
- Every finite (topological) graph (G, V_1), where V_1 is the set of vertices of degree 1, has computable type [15].

Our goal in this paper is to study the property of having computable type for a broader class of spaces, to characterize the pairs having computable type and to develop a unifying argument for the known examples. Our first observation is that graphs and manifolds have the common property that they are locally topological cones as follows (see Figure 1 for an illustration of this idea):

- A finite graph is locally a cone of a finite set,
- A 2-dimensional manifold is locally a disk, which is the cone of a circle, and more generally an n-dimensional manifold is locally an n-ball, which is the cone of an $(n-1)$-sphere.

In this article, we study the class of finite simplicial complexes which is a large class of spaces that are also locally topological cones, as illustrated in Figure 1c.

Finite simplicial complexes are the higher dimensional analogs of finite graphs. They are made of simplices that are attached together along their faces. This class of compact topological spaces is large enough to include many examples (e.g., most common compact manifolds, geometrical models from computer graphics) and can be easily described using finite combinatorial information, so we can hope to obtain a full characterization of computable type for them. We do not consider infinite simplicial complexes because the usual topologies make them non-compact.



(a) A local cone in a graph.

(b) A local cone in a 2-manifold.

(c) A local cone in a simplicial complex.

Figure 1 Examples of local cones in 3 types of spaces.

Let (X, A) be a pair consisting of a finite simplicial complex X and a subcomplex A. We call such a pair a **simplicial pair**. Our main problem is to understand which simplicial pairs (X, A) have computable type. We give a thorough answer, by giving two topological characterizations of the simplicial pairs (X, A) having computable type. One of them is global whereas the other one is local. The local characterization makes it very easy to check whether a simplicial pair (X, A) has computable type, by inspecting the neighborhoods of each vertex separately. Those neighborhoods are called **local cones**, because they are topological cones with the vertex as the tip (precise definitions will be given in the article). We then use the local characterization to prove or disprove that specific sets, such as **Bing's house** and the **dunce hat**, have computable type. The previous techniques developed in the literature were too specific to be applied to these sets. Our techniques not only make it possible to treat any simplicial complex, but also provide a simple and visual way to settle the question for many sets.

The proofs are non-trivial but the statements are elegant and easy to apply. For instance it is very easy to apply our results to show that the dunce hat does not have computable type. However, the internals of the proofs of the theorems are rather involved and we are not aware of any simpler, more direct argument. Therefore our results provide significant progress in the understanding of the computable type property. Moreover, our approach in this article is new in the sense that the proofs are very different from the arguments developed in the literature on the computable type property.

It turns out that the computability property we are studying is intimately related to topology, so we need to use topology in our investigation. However, we only assume familiarity with basic topology (e.g., continuity and compactness). When we use more advanced topological notions, we give the necessary background (e.g. cones, simplicial complexes).

The results. Let us summarize the main results of this paper. We will be working with pairs (X, A) consisting of a compact metric space X and a compact subset A, to be informally thought as the boundary of X. A typical example is given by the pair $(\mathbb{B}_{n+1}, \mathbb{S}_n)$ consisting of the $(n+1)$-dimensional ball and the n-dimensional sphere:

$$\mathbb{B}_{n+1} = \{x \in \mathbb{R}^{n+1} : \|x\| \leq 1\},$$
$$\mathbb{S}_n = \{x \in \mathbb{R}^{n+1} : \|x\| = 1\},$$

where $\|\cdot\|$ is the Euclidean norm or any equivalent norm. We introduce two important properties of pairs, given in Definition 3.1 and restated here.

▶ **Definition.** A pair (X, A) has the **surjection property** if every continuous function $f : X \to X$ satisfying $f|_A = \mathrm{id}_A$ is surjective.

Let $\epsilon > 0$. A pair (X, A) has the **ϵ-surjection property** if every continuous function $f : X \to X$ satisfying $f|_A = \mathrm{id}_A$ and $d(f, \mathrm{id}_X) < \epsilon$ is surjective.

For instance, a consequence of Brouwer's fixed-point theorem is that the pair $(\mathbb{B}_{n+1}, \mathbb{S}_n)$ has the surjection property.

The main result of the paper is Theorem 3.4, which relates computable type with these two properties. We restate it here. We recall that a **simplicial pair** (X, A) consists of a finite simplicial complex X and a subcomplex $A \subseteq X$.

▶ **Theorem.** *Let (X, A) be a simplicial pair such that A has empty interior in X. The following conditions are equivalent:*

1. *(X, A) has computable type,*
2. *There exists $\epsilon > 0$ such that (X, A) has the ϵ-surjection property,*
3. *Every local cone pair of (X, A) has the surjection property.*

Condition 2. is the global property mentioned above and condition 3. is the local one. This theorem reduces a computability-theoretic property to purely topological ones. We develop further techniques to determine whether a pair has computable type, by applying this theorem or by analyzing when the topological properties are satisfied. The first one is stability under finite unions (Theorem 4.1 and Corollary 4.2).

▶ **Theorem** (Finite union)**.** *Let (X, A) be a simplicial pair and $(X_i, A_i)_{i \leq n}$ be pairs of subcomplexes such that $X = \bigcup_{i \leq n} X_i$ and $A = \bigcup_{i \leq n} A_i$. If each (X_i, A_i) has computable type, then (X, A) has computable type.*

The second one is a further characterization of the 2-dimensional simplicial pairs having computable type, by reducing the surjection property for local cone pairs to a simple property of graphs (Theorem 4.4). We demonstrate the strength of that result by giving non-trivial applications to two famous sets: the dunce hat (Figure 4a) and Bing's house (Figure 5).

In order to make the paper understandable to a larger audience, we give informal proofs of the main results. The detailed proofs can be found in the full version [14].

The paper is organized as follows. In Section 2, we give the needed background on computability of sets, simplicial complexes and cone spaces. In Section 3, we define the surjection property and the ϵ-surjection property, state and prove our main result. In Section 4, we present techniques to prove or disprove the (ϵ-)surjection property. As an application, we prove that the dunce hat does not have computable type whereas the Bing's house does, by studying the local cones of each of the two sets. In Section 5, we briefly discuss the possible notions of boundary ∂X of a simplicial complex X that make the pair $(X, \partial X)$ have computable type. We finally formulate open questions and discuss a generalization of our results in Section 6.

2 Preliminaries

We give here some necessary preliminaries in computability theory and topology. We start with this central definition.

▶ **Definition 2.1.** *A **pair** (X, A) consists of a compact metrizable space X and a compact subset $A \subseteq X$. A **copy** of a pair (X, A) in a topological space Z is a pair (Y, B) such that $Y \subseteq Z$ is homeomorphic to X and A is sent to B by the homeomorphism.*

2.1 Computability of sets

We recall definitions and results about the Hilbert cube and computable type. We will mainly use the following notion from computability theory: a set $A \subseteq \mathbb{N}$ is **computably enumerable (c.e.)** if there exists a Turing machine that, on input $n \in \mathbb{N}$, halts if and only if $n \in A$. This notion immediately extends to subsets of countable sets, whose elements can be encoded by natural numbers.

Computability in the Hilbert cube. We work in the Hilbert cube because it is universal among the separable metrizable spaces, in particular every compact metrizable space embeds in the Hilbert cube.

▶ **Definition 2.2.** *The **Hilbert cube** is the space $Q = [0,1]^{\mathbb{N}}$ endowed with the metric $d(x,y) = \sum_i 2^{-i}|x_i - y_i|$. We let $(B_i)_{i\in\mathbb{N}}$ be a computable enumeration of the open balls $B(x,r)$ where $x \in Q$ has finitely many non-zero rational coordinates and $r > 0$ is rational; these B_is are called **rational balls**.*

▶ Notation 1. If $X \subseteq Q$ and $f, g : X \to Q$, then let $d_X(f,g) = \sup_{x\in X} d(f(x), g(x))$.

We recall definitions of computability of compact subsets of the Hilbert cube. The reader can find more details about computability of sets in [2, 16].

▶ **Definition 2.3** (Computability of sets). *A compact set $X \subseteq Q$ is:*
- ***Semicomputable** if there exists a c.e. set $E \subseteq \mathbb{N}$ such that $Q \setminus X = \bigcup_{i\in E} B_i$,*
- ***Computable** if it is semicomputable and $\{i \in \mathbb{N} : X \cap B_i \neq \emptyset\}$ is c.e.*

*A pair (X, A) in Q is **semicomputable** if both X and A are semicomputable.*

Intuitively, X is semicomputable if there is an algorithm that takes any rational cube as input (a voxel) and semidecides whether that cube is disjoint from X, i.e. halts exactly in this case. X is computable if there is an algorithm that *decides* whether a cube intersects the set.

For instance, the Mandelbrot set is semicomputable because its definition gives an algorithm that can eventually detect that a point is outside this set; whether it is computable is an open problem, see [13].

▶ **Example 2.4.** The line segment $I = [0,1]$, embedded in the simplest way as $[0,1] \times \{q\} \subseteq Q$ where $q = (0, 0, \ldots)$, is computable. However, if $A \subseteq \mathbb{N}$ is the halting set (a non-computable c.e. set) and $x_A = \sum_{n\in A} 2^{-n}$, then $[x_A, 1] \times \{q\}$ is a copy of I which is semicomputable but not computable.

The Hilbert cube itself is a computable subset of itself. A compact set $X \subseteq Q$ is semicomputable if and only if the set $\{(i_1, \ldots, i_n) \in \mathbb{N}^* : X \subseteq B_{i_1} \cup \ldots \cup B_{i_n}\}$ is c.e., and it is computable if and only if in addition it contains a dense computable sequence. A function $f : Q \to Q$ is **computable** if there exists a c.e. set $E \subseteq \mathbb{N}^2$ such that $f^{-1}(B_i) = \bigcup_{(i,j)\in E} B_j$. The image of a (semi)computable set under a computable function is a (semi)computable set. Semicomputable sets have very useful properties: if $X \subseteq Q$ is semicomputable and $f, g : X \to Q$ are computable, then $\{q \in \mathbb{Q} : d_X(f,g) < q\}$ is c.e.

Computable type. The next definition is the main notion of this article (see [17]).

▶ **Definition 2.5.** *A pair (X, A) has **computable type** if for every semicomputable copy (Y, B) of the pair in the Hilbert cube, Y is computable.*

A compact space X has computable type if the pair $(X, \emptyset)$ has.

▶ Remark 2.6. In fact, in [17] computable type was defined separately for copies in computable metric spaces and computably Haudorff spaces. In a forthcoming article, we show that taking the copies in computably Hausdorff spaces, computable metric spaces or the Hilbert cube are all equivalent using the fact that computable metric spaces embed effectively in the Hilbert cube, as well as Schröder's computable metrization theorem [22].

2.2 Topology

We recall some notions which will be used, like simplicial complexes and cone spaces. We will work with compact metrizable spaces only, and may omit this assumption in the statements.

▶ **Definition 2.7.** *Let (X, A) be a pair. A **retraction** $r : X \to A$ is a continuous function such that $r|_A = \mathrm{id}_A$. If a retraction exists, then we say that A is a **retract** of X.*

Simplicial complex. Let $V = \{0, \dots, n\}$ and $P_+(V)$ be the set of non-empty subsets of V. An **abstract finite simplicial complex** is a set $S \subseteq P_+(V)$ such that if $\sigma \in S$ and $\emptyset \neq \sigma' \subset \sigma$, then $\sigma' \in S$. Its elements $\sigma \in S$ are called the **simplices** of S. If $\sigma \in S$ has $n+1$ elements, then σ is an ***n*-simplex**. The **vertices** of S are the singletons $\{i\} \in S$. $\sigma \in S$ is **free** if there exists exactly one $\sigma' \in S$ with $\sigma \subsetneq \sigma'$. A **subcomplex** of S is an abstract simplicial complex contained in S.

The **support** of a vector $x = (x_0, \dots, x_n) \in [0,1]^{n+1}$ is $\mathrm{supp}(x) = \{i : x_i \neq 0\}$. The **standard realization** of an abstract simplicial complex S is the set

$$|S| = \Big\{x = (x_0, \dots, x_n) \in [0,1]^{n+1} : \sum_i x_i = 1, \mathrm{supp}(x) \in S\Big\}.$$

Any space homeomorphic to the standard realization of an abstract finite simplicial complex is called a **finite simplicial complex**. We often identify an abstract simplicial complex and its standard realization.

A **simplicial pair** (X, A) consists of a finite simplicial complex X and a subcomplex A.

▶ Remark 2.8. For technical reasons, we will implicitly assume that A contains all the free vertices of X (those are the points x having a neighborhood homeomorphic to $[0, 1)$ with a homeomorphism sending x to 0).

In a simplicial complex, each vertex has a neighborhood which is usually called a star and is topologically a cone. Our main result will relate the computable type property with a property of these local cones. Because we are dealing with pairs, we need to define local cone pairs, as follows.

▶ **Definition 2.9.** *Let (X, A) be the standard realization of a simplicial pair and $v_i = (0, \dots, 0, 1, 0, \dots, 0)$ be a vertex. The **local cone pair** at v_i is (K_i, M_i) defined by:*

$$K_i = \{x \in X : x_i \geq 1/2\},$$
$$M_i = \{x \in X : x_i = 1/2\} \cup (K_i \cap A).$$

Note that the coefficient $1/2$ is arbitrary and could be replaced by any number in $(0, 1)$.

▶ Remark 2.10. We call (K_i, M_i) a cone pair because K is a topological cone: let $L_i = \{x \in X : x_i = 1/2\}$, K_i is a copy of the cone of L_i, obtained from $L_i \times [0, 1]$ by identifying all the points $(l, 0)$ together. The point obtained by this identification is the tip of the cone and corresponds to the vertex v_i. If $N_i = \{x \in A : x_i = 1/2\}$, then M_i is the union of L_i and of the cone of N_i.

In the language of simplicial complexes, K_i corresponds to the *star* of v_i and L_i to the *link* of v_i. K_i is homeomorphic to the union of simplices containing v_i. Each such simplex has a face that does not contain v_i, and L_i is the union of these faces.

3 The (ϵ-)surjection property and computable type for simplicial pairs

We now present the main result of this paper, that identifies which simplicial pairs have computable type, using the following topological properties.

▶ **Definition 3.1.** *A pair (X, A) has the **surjection property** if every continuous function $f : X \to X$ satisfying $f|_A = \mathrm{id}_A$ is surjective.*

*A pair (X, A) in Q has the **ϵ-surjection property** for some $\epsilon > 0$, if every continuous function $f : X \to X$ satisfying $f|_A = \mathrm{id}_A$ and $d_X(f, \mathrm{id}_X) < \epsilon$ is surjective.*

▶ **Example 3.2.**

- For every $n \in \mathbb{N}$, the $(n+1)$-dimensional ball and its bounding n-dimensional sphere form a pair $(\mathbb{B}_{n+1}, \mathbb{S}_n)$ that has the surjection property. It is a consequence of an equivalent formulation of Brouwer's fixed-point theorem that $\mathbb{S}_n$ is not a retract of $\mathbb{B}_{n+1}$ (Corollary 2.15 in [12]).
- The pair $(\mathbb{S}_n, \emptyset)$ does not have the surjection property (take a constant function $f : \mathbb{S}_n \to \mathbb{S}_n$), but has the ϵ-surjection property if ϵ is sufficiently small. It can be proved using classical results in topology, or as a consequence of Theorem 3.4 below.
- If $A \subsetneq X$ is a retract of X, then the pair (X, A) does not have the surjection property, as witnessed by the retraction.

Although the ϵ-surjection property depends on the particular copy of a pair (X, A), quantifying over ϵ yields a topological invariant, i.e. a property of the pair that is satisfied either by all copies or by none of them.

▶ **Proposition 3.3.** *Whether there exists $\epsilon > 0$ such that (X, A) has the ϵ-surjection property does not depend on the copy of (X, A) in Q.*

Proof. If (Y, B) is a copy of (X, A), then let $\phi : X \to Y$ be a homeomorphism such that $\phi(A) = B$. By compactness of X, ϕ is uniformly continuous so given $\epsilon > 0$, there exists $\delta > 0$ such that if $d(x, x') < \delta$ then $d(\phi(x), \phi(x')) < \epsilon$. If (Y, B) has the ϵ-surjection property, then we show that (X, A) has the δ-surjection property. Let $f : X \to X$ be continuous, satisfying $f|_A = \mathrm{id}_A$ and $d_X(f, \mathrm{id}_X) < \delta$. Define $g = \phi \circ f \circ \phi^{-1} : Y \to Y$: one has $g|_B = \mathrm{id}_B$ and $d_Y(g, \mathrm{id}_Y) < \epsilon$ by choice of δ so g is surjective, hence f is surjective. ◀

We now state the main result of this paper.

▶ **Theorem 3.4** (The main theorem)**.** *Let (X, A) be a simplicial pair such that A has empty interior in X. The following statements are equivalent:*

1. *(X, A) has computable type,*
2. *(X, A) has the ϵ-surjection property for some $\epsilon > 0$,*
3. *All the local cone pairs (K_i, M_i) have the surjection property.*

We separate the proof into several independent parts.

▶ Remark 3.5. A single topological space X has many different simplicial decompositions, i.e. many abstract simplicial complexes whose realizations are homeomorphic to X. For instance, a triangle can be decomposed into many smaller triangles. At first sight, the third condition in Theorem 3.4 depends on the choice of the decomposition, because the local

cone pairs are taken at the vertices of the decomposition. However, the theorem implies that the choice of the simplicial decomposition is irrelevant, because conditions 1. and 2. do not depend on the decomposition: if all the cone pairs in a simplicial decomposition have the surjection property, then it is still true for all other simplicial decompositions of the space.

For a simplicial pair that is itself homeomorphic to a cone pair, we obtain a further equivalence, which is a consequence of Theorem 3.4.

Let (L, N) be a simplicial pair. The **simplicial cone pair** induced by (L, N) is the pair (K, M) defined by $K = \mathrm{Cone}(L)$ and $N = L \cup \mathrm{Cone}(N)$, as in Remark 2.10.

▶ **Corollary 3.6.** *Let (K, M) be a simplicial cone pair such that M has empty interior in K. The following statements are equivalent:*

1. *(K, M) has computable type,*
2. *(K, M) has the ϵ-surjection property for some $\epsilon > 0$,*
3. *(K, M) has the surjection property.*

Proof. The surjection property implies the ϵ-surjection property for any pair. Conversely, if the pair (K, M) has the ϵ-surjection property then each local cone pair has the surjection property, but (K, M) is itself homeomorphic to one of its local cone pairs. ◀

The rest of this section is devoted to the proof of this result. We will give several applications in the next section.

3.1 The ϵ-surjection property implies computable type

In this section we give an informal idea of the proof of 2. $\Rightarrow$ 1. in Theorem 3.4. The idea of the proof is that if A has empty interior in X and (X, A) has the ϵ-surjection property, then for an open set U the following conditions are equivalent:

- U intersects X,
- There exists a continuous non-surjective function $g : (X \setminus U) \cup A \to X$ such that $g|_A = \mathrm{id}_A$ and $d_X(g, \mathrm{id}_X) < \epsilon$.

This equivalence is straightforward. If U intersects X, then let g be the inclusion map. Conversely, if such a g exists then $(X \setminus U) \cup A$ must differ from X by the ϵ-surjection property for (X, A), so U intersects X.

The finite simplicial complex X has good topological properties because it is a compact Absolute Neighborhood Retract (ANR), which means that any copy of X in Q is a retract of some neighborhood of that copy. The detailed proof consists in showing how to use these properties to prove that the existence of such a function g can be detected by an algorithm if (X, A) is semicomputable. The main idea is that one does not need to search for an arbitrary continuous function g, but for a computable one. Therefore, one can test whether an open set U intersects X, which makes X computable.

3.2 The ϵ-surjection property is equivalent to the local surjection property

In this section we give an informal proof of the equivalence 2. $\Leftrightarrow$ 3. in Theorem 3.4

The ϵ-surjection property implies the local surjection property. It is easy to see that if a local cone pair does not have the surjection property, then for any $\epsilon > 0$, the pair (X, A) does not have the ϵ-surjection property. It relies on the particular property of a cone that it contains arbitrarily small copies of itself, obtained by scaling it down: for any $\lambda \in (0, 1)$,

the set $K_i(\lambda) = \{x \in X : x_i \geq \lambda\}$ is a copy of K_i and it has arbitrarily small diameter as λ approaches 1. Given $\epsilon > 0$, consider λ such that $K_i(\lambda)$ has diameter less than ϵ. Take a non-surjective function f from $K_i(\lambda)$ to itself which is the identity on the corresponding set $M_i(\lambda)$, and extend it to a non-surjective function $g : X \to X$ by simply defining $g(x) = x$ for x outside $K_i(\lambda)$. One has $d(g, \mathrm{id}_X) < \epsilon$, showing that (X, A) does not have the ϵ-surjection property.

The local surjection property implies the ϵ-surjection property. Now, assume that for every $\epsilon > 0$, (X, A) does not have the ϵ-surjection property. We show that some local cone pair does not have the surjection property. The idea is to start from a sufficiently small $\epsilon > 0$, to be defined later, and a non-surjective function $h : X \to X$ such that $h|_A = \mathrm{id}_A$ and $d(h, \mathrm{id}_X) < \epsilon$ and consider its restriction h_0 to a local cone K which is not contained in the image of h. This function h_0 does not immediately disprove the surjection property for the local cone pair (K, M) because $h_0(K)$ may not be contained in K and h_0 may not be the identity on M. However, h_0 almost satisfies these properties: $h_0(K)$ is at distance ϵ from K and h_0 is ϵ-close to the identity on M. Again, using the fact that K is a compact Absolute Neighborhood Retract (ANR) and the properties derived from that, if one takes ϵ sufficiently small, then one can transform h_0 into a continuous function G that sends K to itself, is the identity on M and is still non-surjective. Therefore, (K, M) does not have the surjection property.

3.3 Computable type implies the ϵ-surjection property

We prove 1. $\Rightarrow$ 2. in Theorem 3.4. We show that if a simplicial pair (X, A) does not have the ϵ-surjection property for any $\epsilon > 0$, then it has a semicomputable copy in Q that is not computable. In order to build that semicomputable copy, we show that the pair fails in a computable way to have the ϵ-surjection property, which is expressed by Definition 3.7.

For two non-empty compact sets $A, B \subseteq Q$, their Hausdorff distance is

$$d_H(A, B) = \max(\max_{a \in A} d(a, B), \max_{b \in B} d(b, A)).$$

▶ **Definition 3.7.** *Let $\epsilon > 0$ and $(X, A) \subseteq Q$ fail to have the ϵ-surjection property. Say that $\delta > 0$ is an **ϵ-witness** if there exists a continuous function $f : X \to X$ such that $f|_A = \mathrm{id}_A$, $d_X(f, \mathrm{id}_X) < \epsilon$ and $d_H(f(X), X) > \delta$.*

*Say that (X, A) has **computable witnesses** if there is a computable function $\epsilon \mapsto \delta(\epsilon)$ such that for every $\epsilon > 0$, $\delta(\epsilon)$ is an ϵ-witness.*

For a compact pair (X, A) (not necessarily simplicial), having computable witnesses is sufficient to build a semicomputable copy which is not computable.

▶ **Theorem 3.8.** *Let $(X, A) \subseteq Q$ be a computable pair having computable witnesses. (X, A) does not have computable type.*

Informal proof. In order to give some intuition, let us show precisely another but related result: if we only assume that (X, A) does not have the surjection property, then one can encode the halting problem for *one* program p in a copy of (X, A), in the following sense. Given p, one can produce an algorithm that semicomputes a copy (X_p, A_p) of (X, A); any algorithm computing X_p could be used to decide whether p halts.

Let $(X_0, A_0) \subseteq Q$ be a semicomputable copy of (X, A) and $\delta > 0$ be such that there exists a non-surjective continuous function $f : X_0 \to X_0$ such that $f|_{A_0} = \mathrm{id}_{A_0}$ and $d_H(X_0, f(X_0)) > \delta$.

Given a program p, we define a copy (X_p, A_p). If p does not halt, then $(X_p, A_p) = (X_0, A_0)$. If p halts, then (X_p, A_p) is another copy (X_1, A_1) defined by the following algorithm.

Start enumerating the complements of X_0 and A_0. If p eventually halts then consider a copy (X_1, A_1) of (X_0, A_0) with the following properties:

- (X_1, A_1) is compatible with (i.e. disjoint from) the current enumeration of the complements of X_0 and A_0,
- $d_H(X_1, X_0) > \delta$.

The existence of f implies the existence of (X_1, A_1), which can be effectively found. We then continue enumerating the complements of X_1 and A_1.

We have just given an algorithm that semicomputes a copy (X_p, A_p) of (X, A), be it (X_0, A_0) or (X_1, A_1). Any algorithm that computes X_p could be used to know whether p halts: p halts if and only if $d_H(X_p, X_0) > \delta$, which can be decided from the computable information about X_p.

Now, assuming that (X, A) does not have the ϵ-surjection property for any ϵ, and using the assumption that a witness $\delta(\epsilon)$ can be computed from any ϵ, we apply this strategy against all the programs in parallel and at infinitely many scales. The idea is simple but the details are rather technical. ◀

Note that the standard realization of a simplicial pair is computable. We now show that if it has witnesses, then it always have *computable* witnesses, which together with Theorem 3.8 concludes the proof of 1. ⇒ 2. in Theorem 3.4.

▶ **Proposition 3.9.** *If a simplicial pair (X, A) does not have the ϵ-surjection property for any $\epsilon > 0$, then its standard realization has computable witnesses.*

Proof. By 3. ⇒ 2. in Theorem 3.4, there exists a local cone pair (K_i, M_i) which does not have the surjection property, so there exists a non-surjective function $f_0 : K_i \to K_i$ such that $f_0|_{M_i} = \mathrm{id}_{M_i}$. One can assume w.l.o.g. that $d_X(f_0, \mathrm{id}_X) < 1$. Let $\delta_0 > 0$ be such that $d_H(f_0(X), X) > \delta_0$. Given $\epsilon > 0$, the number $\delta = \delta_0\epsilon$ can be computed from ϵ and is an ϵ-witness. Indeed, the function f obtained by applying f_0 to a version of K_i scaled by a factor ϵ and extended as the identity elsewhere satisfies all the conditions. ◀

4 Techniques for the (ϵ-)surjection property

Theorem 3.4 enables one to reduce the computable type property to topological properties, namely the ϵ-surjection property and the surjection property for local cone pairs. Proving or disproving these properties may not be straightforward, so we develop a few techniques that help in many cases.

4.1 Finite union

The first result is a way to prove that a simplicial pair has the ϵ-surjection property by decomposing it as a finite union of pairs that all have the ϵ-surjection property.

▶ **Theorem 4.1** (Finite union). *Let (X, A) be a finite simplicial pair and $(X_i, A_i)_{i \leq n}$ be pairs of subcomplexes such that $X = \bigcup_{i \leq n} X_i$ and $A = \bigcup_{i \leq n} A_i$. If every pair (X_i, A_i) has the ϵ-surjection property for some $\epsilon > 0$, then (X, A) has the δ-surjection property for some $\delta > 0$.*

Informal proof. We are using good topological properties of finite simplicial complexes. For each i, there exists a neighborhood U_i of X_i and a retraction $r_i : U_i \to X_i$ with a special property: if x belongs to the topological interior of X_i, then the only preimage of x by r_i is x.

Let δ be sufficiently small and assume that (X, A) does not have the δ-surjection property. Let $f : X \to X$ be continuous, non-surjective and satisfy $f|_A = \mathrm{id}_A$ and $d_X(f, \mathrm{id}_X) < \delta$. There must be $i \leq n$ and x in the interior of X_i that is not in the image of f. We can then create a function $f_i : X_i \to X_i$ as follows: f_i is the restriction of $r_i \circ f$ to X_i (it is possible if δ is sufficiently small, so that $f(X_i) \subseteq U_i$).

The special property of r_i implies that x is not in the image of f_i. Moreover, f_i is continuous, is the identity on A_i and is ϵ-close to id_{X_i} if δ is sufficiently small. ◀

▶ **Corollary 4.2.** *Let (X, A) be a simplicial pair and $(X_i, A_i)_{i\leq n}$ be pairs of subcomplexes such that $X = \bigcup_{i\leq n} X_i$ and $A = \bigcup_{i\leq n} A_i$. If every pair (X_i, A_i) has computable type, then (X, A) has computable type.*

For instance, if a finite simplicial complex X is a finite union of subcomplexes that are homeomorphic to spheres, then X has computable type. More generally, if a finite simplicial pair (X, A) is a finite union of pairs of subcomplexes (X_i, A_i) that are homeomorphic to pairs $(\mathbb{S}_n, \emptyset)$ or $(\mathbb{B}_{n+1}, \mathbb{S}_n)$, then (X, A) has computable type.

4.2 Cone of a graph

In a 2-dimensional simplicial pair, the local cones are cones of graphs. We obtain a characterization of the surjection property for such cones. In order to state the result, we need to define the cone pair induced by a pair, already informally discussed in Remark 2.10. Let (L, N) be a pair. We define the cone pair $(K, M) := \mathrm{Cone}(L, N)$ as follows:

- $K = \mathrm{Cone}(L)$ is the quotient of $L \times [0, 1]$ by the equivalence relation $(x, 0) \sim (y, 0)$,
- $M = L \cup \mathrm{Cone}(N)$, where L is embedded in K as $L \times \{1\}$.

The space L is called the **base** of the cone $K = \mathrm{Cone}(L)$, and the equivalence class $L \times \{0\}$ is called the **tip** of K.

▶ **Example 4.3.** Let us illustrate this notion on the usual example of balls and spheres:

- $\mathrm{Cone}(\mathbb{S}_n, \emptyset) = (\mathbb{B}_{n+1}, \mathbb{S}_n)$ with the tip at the center of $\mathbb{B}_{n+1}$,
- $\mathrm{Cone}(\mathbb{B}_n, \mathbb{S}_{n-1}) = (\mathbb{B}_{n+1}, \mathbb{S}_n)$ with the tip in $\mathbb{S}_n$.

Here is the main result of this section.

▶ **Theorem 4.4.** *Let (L, N) be a pair such that L is a finite graph and N is a subset of its vertices. The following statements are equivalent:*

1. $\mathrm{Cone}(L, N)$ *has the surjection property,*
2. *Every edge is in a cycle or a path starting and ending in N.*

We follow the usual convention that in a graph, a path and a cycle do not visit a vertex twice, i.e. they are topologically a line segment and a circle respectively. In particular, a path connects two different points.

▶ **Example 4.5** (Star pair)**.** Fix some $n \geq 1$ and let X be the star with n branches and A be the n endpoints of these branches (see Figure 2), with a special case for $n = 1$: $\mathrm{Cone}(\{v\}, \emptyset) = (\mathbb{B}_1, \mathbb{S}_0)$. The pair (X, A) is precisely $\mathrm{Cone}(A, \emptyset)$. As A has no edge, it satisfies the conditions of Theorem 4.4, therefore (X, A) has the surjection property. One can then obtain Iljazović's result that every finite graph has computable type [15], because the local cones of a finite graph are stars, which have the surjection property.

(a) Star with 5 branches. **(b)** Star with 1 branch.

Figure 2 The star pairs (X, A) have the surjection property (Example 4.5) (X in yellow, A in black).

▶ **Example 4.6** (n squares)**.** Fix some $n \geq 2$ and let X be the union of n squares which all meet in one common edge and A be the union of all the other edges (see Figure 3). The pair (X, A) has the surjection property. Indeed, $(X, A) = \mathrm{Cone}(A, \emptyset)$ and A is a graph which is a union of circles (each circle is the boundary of the union of two squares). Therefore, $\mathrm{Cone}(A, \emptyset)$ has the surjection property by Theorem 4.4. Finally, (X, A) has computable type by Corollary 3.6.

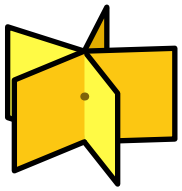


Figure 3 A union of 5 squares is the cone of a graph; the tip is at the center, the graph is in black (Example 4.6).

We expect a generalization of Theorem 4.4 to cones of arbitrary simplicial complexes, by using the notions of n-cycles and relative n-cycles from homology, generalizing cycles and paths respectively [12].

In the next section we apply Theorem 4.4, giving an example of a cone pair of a graph which does not have the surjection property.

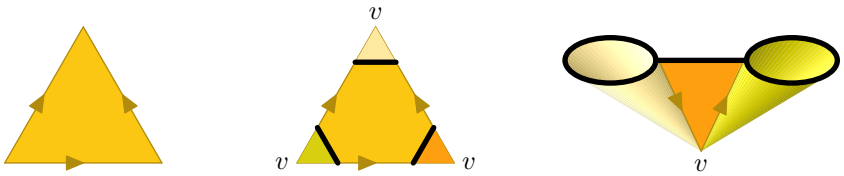
4.3 The dunce hat

The **dunce hat** D is the space obtained from a solid triangle by gluing its three sides together, with the orientation of one side reversed (see Figure 4a). It is a classical example, introduced by Zeeman [23], of a space that is contractible but not intuitively so. It is a 2-dimensional simplicial complex with no free edge, i.e. no edge that belongs to one triangle only.

▶ **Theorem 4.7.** *The dunce hat does not have computable type.*

Proof. First, it is possible to turn the dunce hat into a simplicial complex, so we can apply our results. The vertices of the triangle are identified to a point v, and we show that the local cone pair at that point does not have the surjection property. Indeed, in Figure 4c one can see that the local cone pair at v is $\mathrm{Cone}(L, \emptyset) = (\mathrm{Cone}(L), L)$ where L is the graph consisting of two circles joined by a line segment.

We apply Theorem 4.4: L is a finite graph containing an edge which is neither in a cycle nor in a path from N to N (N is empty), therefore $\mathrm{Cone}(L, N)$ does not have the surjection property. Theorem 3.4 then implies that the dunce hat does not have computable type. ◀



(a) Dunce hat. **(b)** Local cone pair. **(c)** Local cone pair.

Figure 4 (a) The dunce hat is obtained by gluing the edges with the indicated orientations; (b) and (c) a local cone pair $(\mathrm{Cone}(L), L) = \mathrm{Cone}(L, \emptyset)$ with tip at v, with L in black.

As far as we know, there is no simple and visual way of building a semicomputable copy of the dunce hat that is not computable, i.e. the involved construction carried out in the proof of Theorem 3.4 cannot be avoided. The same remark applies to the pair $(\mathrm{Cone}(L), L)$ depicted in Figure 4c.

If A is the identified edges of the triangle, then it can be proved, by analyzing its local cone pairs, that the pair (D, A) has computable type. In particular, the local cone pair at v is $\mathrm{Cone}(L, N)$ where N consists of the two endpoints of the middle interval, so L is the union of two circles and a line segment between two points of N, hence $\mathrm{Cone}(L, N)$ has the surjection property by Theorem 4.4.

▶ Remark 4.8 (Quotient vs pair). It is proved in [8] that for any compact pair (X, A) where A has empty interior, if the quotient space X/A has computable type then the pair (X, A) has computable type. It is also proved that the converse implication fails, the counter-example is given by the circle X and a subset A consisting of a converging sequence together with its limit. The pair (X, A) has computable type, simply because X itself has computable type. However, X/A is homeomorphic to the Hawaiian earring which does not have computable type. This quotient is not a finite simplicial complex.

We give another counter-example of a quotient space which is a finite simplicial complex. Let $L = C_1 \vee I \vee C_2$, X be the cylinder of L and A the two bases of the cylinder. Inspecting the local cones one can show that (X, A) has computable type but X/A does not.

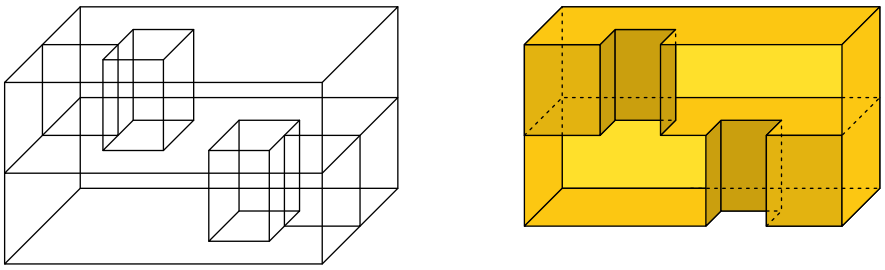
4.4 Bing's house, or the house with two rooms

All the known examples of sets having computable type are non-contractible (note that we are not considering pairs, but single sets), and one might conjecture that no contractible set has computable type. We give a counter-example, which is a famous space that was defined as a counter-example for other properties. It was invented by Bing [1] and is now called **Bing's house**, or the house with two rooms. The set is depicted in Figure 5, together with a half-cut to help visualizing it. It is an example of a space which is contractible but not intuitively so. It can be endowed with a simplicial complex structure (by triangulating each flat surface). It is then a 2-dimensional simplicial complex with no free edge, which means that every edge belongs to at least two triangles.

Using our results we easily show that this set has computable type as a single set, i.e. without adjoining a boundary to it.

▶ **Theorem 4.9.** *Bing's house has computable type.*

It is worth noticing that thanks to our results, it can be proved by looking at pictures only, although the argument can be formalized.



(a) Bing's house. **(b)** Half-cut.

Figure 5 Bing's house with two rooms and a half-cut of it (the full house is obtained by adding the symmetric reflection of the half-cut through the front vertical plane). It consists of two rooms, each of which can be accessed from outside through a tunnel crossing the other room. Each tunnel is linked by an internal wall to a side wall.

Proof. Using Theorem 3.4, it is sufficient to inspect the possible local cones. One easily sees that there are three types of possible cones, depicted in Figure 6. The basis of each cone is a graph which is a union of 1, 2 or 3 cycles, so by Theorem 4.4 each cone pair has the surjection property, therefore Bing's house has computable type by Theorem 3.4.

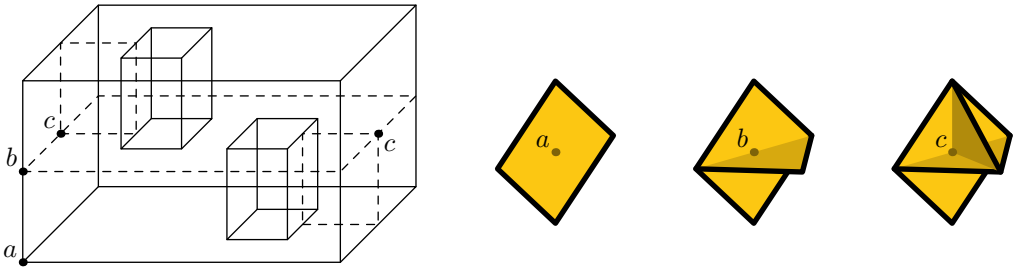


Figure 6 The local cones in Bing's house: their bases (in black) are graphs that are unions of cycles. Each point of Bing's house is the tip of one of these three cones: two points are tips of the third cone, all the other points on the dashed lines are tips of the second cone, all the other points are tips of the first cone. ◀

5 Boundary

Given a simplicial complex X, a natural problem is to understand whether there is a minimal notion of boundary ∂X such that the pair $(X, \partial X)$ has computable type. We make a few observations about three possible candidates. Let

- $\partial_1 X$ be the union of simplices that are contained in *exactly* one simplex of the next dimension, i.e. $\partial_1 X$ is the union of the free simplices of X,
- $\partial_+ X$ be the union of simplices that are contained in *at least* one simplex of the next dimension,
- $\partial_{\text{odd}} X$ be the union of simplices that are contained in *an odd number* of simplices of the next dimension.

In the proofs of the next results, we say that a simplex M in X is **maximal** if it is not contained in a higher-dimensional simplex of X.

▶ **Proposition 5.1.** *Every simplicial pair $(X, \partial_+ X)$ has computable type.*

Proof. Let $(M_i)_{i\leq n}$ be an enumeration of the maximal simplices of X. M_i is a ball, let ∂M_i be its bounding sphere, which is a subcomplex of M_i. One has $X = \bigcup_{i\leq n} M_i$ and $\partial_+ X = \bigcup_{i\leq n} \partial M_i$. Each pair $(M_i, \partial M_i)$ has the surjection property (Example 3.2), so $(X, \partial_+ X)$ has the ϵ-surjection property for some ϵ by Theorem 4.1. As a result, $(X, \partial_+ X)$ has computable type by Theorem 3.4. ◀

▶ **Proposition 5.2.** *Let X be a finite simplicial complex and A a subcomplex. If (X, A) has computable type, then A contains $\partial_1 X$.*

Proof. Assume that some simplex Δ belongs to $\partial_1 X$ but not to A. We show that for every $\epsilon > 0$, (X, A) does not have the ϵ-surjection property, implying that (X, A) does not have computable type by Theorem 3.4. Let $\epsilon > 0$. Let Δ' be the unique maximal simplex having Δ as a face (Δ' has one more vertex than Δ). There is a non-surjective function $f : \Delta' \to \Delta'$ which is ϵ-close to the identity and is the identity on the other faces of Δ': f slightly pushes points of Δ' away from Δ. We extend f as the identity on the rest of X, which gives a continuous function because Δ is free. As Δ is not in A, f is the identity on A. ◀

The following observations can be made:

- Although $(X, \partial_1 X)$ has computable type when X is a 1-dimensional complex (i.e., a graph), it is no more true for 2-dimensional complexes. For the dunce hat D, one has $\partial_1 D = \emptyset$ but we saw in Theorem 4.7 that $(D, \emptyset)$ does not have computable type.
- While $(X, \partial_+ X)$ always has computable type by Proposition 5.1, $\partial_+ X$ is far from optimal. For instance, it is always non-empty (unless X is a single point), but for any sphere $\mathbb{S}_n$, the pair $(\mathbb{S}_n, \emptyset)$ already has computable type.
- In a subsequent paper we prove that $(X, \partial_{\text{odd}} X)$ always has computable type, using homology. Observe that $\partial_{\text{odd}} X$ is in general not optimal, as the example of graphs shows: $(X, \partial_1 X)$ has computable type and $\partial_1 X$ is usually smaller than $\partial_{\text{odd}} X$, which contains all the vertices of odd degrees.

6 Open questions and generalization

We leave two open questions.

▶ **Question 1.** *Is there a canonical notion of boundary ∂X for a simplicial complex X, such that $(X, \partial X)$ always has computable type, and ∂X is minimal in some sense?*

▶ **Question 2.** *For simplicial pairs (L, N), is it possible to characterize the surjection property for $\text{Cone}(L, N)$ in terms of the homology of (L, N)?*

We finally mention that the proof of the main result actually applies to more general spaces. For instance one can prove that if $(M, \partial M)$ is a compact manifold with boundary, then $\text{Cone}(M, \partial M)$ has computable type because it satisfies the surjection property, although it is not always a simplicial complex. These results will appear in a forthcoming article.

References

1 R.H. Bing. Some aspects of the topology of 3-manifolds related to the Poincaré conjecture. *Lectures on Modern Mathematics*, II:93–128, 1964.

2 Vasco Brattka and Gero Presser. Computability on subsets of metric spaces. *Theoretical Computer Science*, 305(1):43–76, 2003. Topology in Computer Science. doi:10.1016/S0304-3975(02)00693-X.

3 Vasco Brattka, Stéphane Le Roux, Joseph S. Miller, and Arno Pauly. Connected choice and the Brouwer fixed point theorem. *J. Math. Log.*, 19(1):1950004:1–1950004:46, 2019. doi:10.1142/S0219061319500041.

4 Mark Braverman and Michael Yampolsky. *Computability of Julia Sets*. Springer, 2008.

5 Konrad Burnik and Zvonko Iljazović. Computability of 1-manifolds. *Log. Methods Comput. Sci.*, 10(2), 2014.

6 Matea Čelar and Zvonko Iljazović. Computability of glued manifolds. *Journal of Logic and Computation*, 32(1):65–97, October 2021. doi:10.1093/logcom/exab063.

7 Matea Čelar and Zvonko Iljazović. Computability of products of chainable continua. *Theory Comput. Syst.*, 65(2):410–427, 2021.

8 Matea Čelar and Zvonko Iljazović. Computable type of an unglued space, 2021. Talk at Computability and Complexity in Analysis (CCA).

9 Eugen Čičković, Zvonko Iljazović, and Lucija Validžić. Chainable and circularly chainable semicomputable sets in computable topological spaces. *Arch. Math. Log.*, 58(7-8):885–897, 2019.

10 Pieter Collins. Computability of homology for compact absolute neighbourhood retracts. In Andrej Bauer, Peter Hertling, and Ker-I Ko, editors, *Sixth International Conference on Computability and Complexity in Analysis, CCA 2009, August 18-22, 2009, Ljubljana, Slovenia*, volume 11 of *OASICS*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

11 Matthew Harrison-Trainor, Alexander G. Melnikov, and Keng Meng Ng. Computability of Polish spaces up to homeomorphism. *J. Symb. Log.*, 85(4):1664–1686, 2020. doi:10.1017/jsl.2020.67.

12 Allen Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002.

13 Peter Hertling. Is the Mandelbrot set computable? *Math. Log. Q.*, 51(1):5–18, 2005.

14 Mathieu Hoyrup and Djamel Eddine Amir. Computability of finite simplicial complexes. Preprint, February 2022. URL: https://hal.inria.fr/hal-03564904.

15 Zvonko Iljazović. Computability of graphs. *Mathematical Logic Quarterly*, 66(1):51–64, 2020.

16 Zvonko Iljazović and Takayuki Kihara. Computability of subsets of metric spaces. In Vasco Brattka and Peter Hertling, editors, *Handbook of Computability and Complexity in Analysis*. Springer, 2020.

17 Zvonko Iljazović and Igor Sušić. Semicomputable manifolds in computable topological spaces. *Journal of Complexity*, 45:83–114, 2018.

18 Takayuki Kihara. Incomputability of simply connected planar continua. *Comput.*, 1(2):131–152, 2012. doi:10.3233/COM-12012.

19 Joseph S. Miller. Effectiveness for embedded spheres and balls. *Electronic Notes in Theoretical Computer Science*, 66(1):127–138, 2002. CCA 2002, Computability and Complexity in Analysis.

20 Eike Neumann. *Universal envelopes of discontinuous functions*. PhD thesis, Aston University, 2018.

21 Stéphane Le Roux and Arno Pauly. Finite choice, convex choice and finding roots. *Logical Methods in Computer Science*, 11, 2015.

22 Matthias Schröder. Effective metrization of regular spaces. In Ker-I Ko, Anil Nerode, Marian B. Pour-El, Klaus Weihrauch, and Jiří Wiedermann, editors, *Computability and Complexity in Analysis*, volume 235, pages 63–80. Informatik Berichte, FernUniversität Hagen, 1998.

23 E.C. Zeeman. On the dunce hat. *Topology*, 2(4):341–358, 1963. doi:10.1016/0040-9383(63)90014-4.

Unboundedness for Recursion Schemes: A Simpler Type System

David Barozzini
Institute of Informatics, University of Warsaw, Poland

Paweł Parys
Institute of Informatics, University of Warsaw, Poland

Jan Wróblewski
Institute of Informatics, University of Warsaw, Poland

Abstract

Decidability of the problems of unboundedness and simultaneous unboundedness (aka. the diagonal problem) for higher-order recursion schemes was established by Clemente, Parys, Salvati, and Walukiewicz (2016). Then a procedure of optimal complexity was presented by Parys (2017); this procedure used a complicated type system, involving multiple flags and markers. We present here a simpler and much more intuitive type system serving the same purpose. We prove that this type system allows to solve the unboundedness problem for a widely considered subclass of recursion schemes, called safe schemes. For unsafe recursion schemes we only have soundness of the type system: if one can establish a type derivation claiming that a recursion scheme is unbounded then it is indeed unbounded. Completeness of the type system for unsafe recursion schemes is left as an open question. Going further, we discuss an extension of the type system that allows to handle the simultaneous unboundedness problem.

We also design and implement an algorithm that fully automatically checks unboundedness of a given recursion scheme, completing in a short time for a wide variety of inputs.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Rewrite systems; Mathematics of computing → Lambda calculus

Keywords and phrases Higher-order recursion schemes, boundedness, intersection types, safe schemes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.112

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version Missing proof details are available in the full version of this paper:
Full Version: `https://arxiv.org/abs/2204.11023`

Supplementary Material Implementation of the presented algorithm is available at:
Software (Source Code): `https://github.com/xi-business/infsat`
archived at `swh:1:dir:075c57b341bf44dc0abeb89fe48ed8aec1691828`

Funding Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

1 Introduction

Higher-order recursion schemes (*recursion schemes* for short) proved to be useful in model-checking programs using higher-order functions, see e.g. Kobayashi [13] (recursion schemes are algorithmically manageable abstractions of such programs, faithfully representing the control flow). Higher-order functions are widely used in functional programming languages, like Haskell, OCaml, and Lisp; additionally, higher-order features are now present in most mainstream languages like Java, JavaScript, Python, or C++.


The formalism of recursion schemes is equivalent via direct translations to simply-typed λY-calculus [27] and to higher-order OI grammars [9, 16]. Collapsible pushdown systems [11] and ordered tree-pushdown systems [6] are other equivalent formalisms. Recursion schemes cover some other models such as indexed grammars [1] and ordered multi-pushdown automata [3].

The usefulness of recursion schemes follows from the fact that trees generated by them have decidable MSO theory [20]. When the property to be verified is given by a parity automaton (a formalism equivalent to the MSO logic), and when the recursion scheme is of order m, the model-checking problem is m-EXPTIME-complete; already for reachability properties the problem is $(m-1)$-EXPTIME-complete [14]. Although this high complexity may be threatening, there exist algorithms that behave well in practice. They make use of appropriate systems of intersection types. Namely, a Japanese group created model-checkers TRecS [13] and HorSat [4], and prototype verification tools, MoCHi [15] and EHMTT verifier [28], on top of them. A group hosted in Oxford created model-checkers HORSC [19] and TravMC2 [18]. Necessarily these model-checkers are very slow on some worst-case examples, but on schemes generated from some real life higher-order programs they usually work in a reasonable time (while in the worst-case a huge type derivation may be required, it is often the case that there exists a small type derivation that can be found quickly).

In recent years, interest has arisen in model checking recursion schemes against properties that are not regular (i.e., not expressible in the MSO logic). This primarily concerns the *unboundedness problem* for word languages recognized by recursion schemes [22], and its generalization – the *simultaneous unboundedness problem* (aka. the diagonal problem) [10, 7, 26]. Decidability of the latter problem implied computability of downward closures (with respect to the subsequence relation) for these languages [29], and decidability of their separability by piecewise testable languages [8]. It was also possible to establish decidability of the WMSO+U logic (an extension of MSO with a quantifier U, talking about unboundedness) over trees generated by recursion schemes [25].

Moreover, there is also a link to asynchronous shared-memory programs, being a common way to manage concurrent requests in a system. In asynchronous programming, each asynchronous function is a sequential program. When run, it can change the global shared state of the program and run other asynchronous functions. A scheduler repeatedly and non-deterministically executes pending asynchronous functions. Majumdar, Thinniyam and Zetzsche [17] have proven that when asynchronous functions are modeled as recursion schemes, the question whether there is an a priori upper bound on the number of pending executions of asynchronous functions can be reduced to the problem we consider here, namely the unboundedness problem for a single recursion scheme.

In this paper we revisit the (simultaneous) unboundedness problem for word languages recognized by recursion schemes. This problem asks, for a set of letters Σ_{a} and a language of words L (recognized by a recursion scheme), whether for every $n \in \mathbb{N}$ there is a word in L where every letter from Σ_{a} occurs at least n times. Equivalently, one can consider a recursion scheme generating an infinite tree t and ask whether for every $n \in \mathbb{N}$ there is a finite branch in t where every letter from Σ_{a} occurs at least n times. The problem is already interesting when $|\Sigma_{\mathsf{a}}| = 1$; then we talk about the unboundedness problem. Decidability of the simultaneous unboundedness problem was first established by Hague, Kochems, and Ong [10], for a well-recognized subclass of recursion schemes, called *safe* schemes [9, 12, 2, 5, 27]. The solution was then generalized to all recursion schemes by Clemente, Parys, Salvati, and Walukiewicz [7]. These two algorithms are useless in practice, not only because their complexity is much higher than the optimal one, but mainly because they perform some transformations of recursion

schemes that are extremely costly in every case, not only in the worst case. The complexity of the problem for recursion schemes (namely, $(m-1)$-EXPTIME-completeness for schemes of order m) was settled by Parys [26]; moreover, his solution uses intersection types, and thus it is potentially suitable for implementation (by analogy to the regular model-checking case, where algorithms using type systems led to reasonable implementations). In the single-letter case ($|\Sigma_{\mathsf{a}}| = 1$) the problem is still $(m-1)$-EXPTIME-complete; a slightly simpler type system for this case was presented by Parys [22]. Unfortunately, the type systems of Parys [22, 26] have two drawbacks. First, they are quite complicated: type judgments can be labeled by different kinds of flags and markers, which influence type derivations in a convoluted way. Second (and related to first), it seems that the number of choices for these flags and markers is quite large, and thus finding type derivations even for quite simple recursions schemes may be very costly. Having these drawbacks in mind, we leave experimental evaluation of these algorithms for future work.

In this paper we rather consider a much simpler type system, proposed by Parys in his survey [24], based on an earlier work concerning lambda-terms representing functions on numerals [21]. This type system is much easier than the previous one: every type is labeled only by a single "productivity flag" having an intuitive meaning. Namely, the flag says whether the lambda-term under consideration is responsible for creating occurrences of the letter from Σ_{a}. It was only conjectured that this type system may be used to solve the unboundedness problem.

Our contributions are as follows:

- We prove that the type system allows to solve the unboundedness problem for all safe recursion schemes.
- We show that the algorithm using the type system solves the unboundedness problem for safe recursion schemes of order m in $(m-1)$-EXPTIME (being optimal).
- We prove soundness of the type system for all (i.e., also unsafe) recursion schemes, saying that if one has found a type derivation claiming that a recursion schemes is unbounded then it is indeed unbounded. Completeness of the type system for unsafe recursion schemes is left as an open problem.
- We implement an algorithm solving the unboundedness problem by means of the proposed type system, and we present results of our experiments. The outcome of the algorithm is always correct if the recursion scheme given on input is safe. When the recursion scheme is not safe, proofs of its unboundedness found by the algorithm are still guaranteed to be correct. However, our theorems do not guarantee that the algorithm will find a proof of unboundedness in the unsafe case, so if the algorithm fails to find such a proof, it does not mean that the unsafe recursive scheme is necessarily bounded.
- We then generalize the type system to the simultaneous unboundedness problem (i.e., the multiletter case), obtaining the same properties: algorithm in $(m-1)$-EXPTIME for safe recursion schemes of order m, and soundness for all recursion schemes.
- We implement an optimized version of our algorithm, InfSat, which is fast for a wide variety of inputs. We build upon the implementation of HorSat [4]. We modify HorSat benchmarks to conform to InfSat input format, and present their results to show that InfSat is able to handle inputs described by Broadbent and Kobayashi [4] as "practical" as well as additional benchmarks crafted to measure the speed of InfSat.

2 Preliminaries

The set of *sorts* is constructed from a unique basic sort o using a binary operation $\to$. Thus o is a sort and if α, β are sorts, so is $(\alpha \to \beta)$. The order of a sort is defined by: $ord(\mathsf{o}) = 0$, and $ord(\alpha \to \beta) = \max(1 + ord(\alpha), ord(\beta))$.

A *signature* Σ is a set of typed constants, that is, symbols with associated sorts. We assume that sorts of all constants in the signature are of order at most 1, that is, of the form $\underbrace{\mathsf{o} \to \cdots \to \mathsf{o} \to}_{r} \mathsf{o}$; for a constant of such a sort, r is called its *arity*. We fix a distinguished constant $\omega \in \Sigma$ of arity 0 (it will be used in places where a computation diverges).

The set of *infinitary simply-typed lambda-terms* is defined coinductively as follows. A constant $a \in \Sigma$ of sort α is a lambda-term of sort α. For each sort α there is a countable set of variables $x^\alpha, y^\alpha, \ldots$ that are also lambda-terms of sort α. If M is a lambda-term of sort β and x^α a variable of sort α then $\lambda x^\alpha.M$ is a lambda-term of sort $\alpha \to \beta$. Finally, if M is a lambda-term of sort $\alpha \to \beta$ and N is a lambda-term of sort α then $M\,N$ is a lambda-term of sort β. As usual, we identify lambda-terms up to alpha-conversion. We often omit the sort annotation of variables, but formally every variable has a sort. We use the standard notions of free variables, substitution, and beta-reduction (of course during substitution and beta-reduction we rename bound variables to avoid name conflicts). A lambda-term is called *closed* when it does not have free variables. For a lambda-term M of sort α, the order of M, denoted $ord(M)$, is defined as $ord(\alpha)$.

The *complexity* of a lambda-term M is the maximum of orders of those subterms of M that are not of the form $a\,M_1\,\ldots\,M_k$, where a is a constant and $k \geq 0$ (or 0 if there are no such subterms). Note that for most lambda-terms M the complexity is just the maximum of orders of all subterms of M; the difference is only at complexity 0 and 1: we want lambda-terms built entirely from constants to have complexity 0, not 1.

A closed lambda-term of sort o and complexity 0 is called a *tree*. Equivalently, a lambda-term is a tree if it is of the form $a\,M_1\,\ldots\,M_r$, where $M_1, \ldots, M_r$ are trees, and r is the arity of a. While talking about a *branch* of a tree, we mean a finite branch that ends in a leaf not being ω-labeled. Formally, a *(finite) branch* of a tree $T = a\,T_1\,\ldots\,T_r$ is a sequence of constants $a_1, a_2, \ldots, a_k$ such that $a_1 = a \neq \omega$, and either $a_2, \ldots, a_k$ (with $k \geq 2$) is a finite branch of some T_i, or $r = 0$ and $k = 1$.

We consider Böhm trees only for closed lambda-terms of sort o. For such a lambda-term M, its *Böhm tree* is constructed by coinduction, as follows: if there is a sequence of beta-reductions from M to a lambda-term of the form $a\,M_1\,\ldots\,M_r$, and $T_1, \ldots, T_r$ are Böhm trees of $M_1, \ldots, M_r$, respectively, then $a\,T_1\,\ldots\,T_r$ is a Böhm tree of M; if there is no such a sequence of beta-reductions from M, then the constant ω is a Böhm tree of M. It is folklore that every closed lambda-term of sort o has exactly one Böhm tree (the order in which beta-reductions are performed does not matter); this tree is denoted by $BT(M)$. Notice that a Böhm tree is indeed a tree, and that if M is finite then its Böhm tree equals the beta-normal form of M.

A *recursion scheme* $\mathcal{G}$ is a finite representation of a closed lambda-term $\Lambda(\mathcal{G})$ that is of sort o and regular, that is, has finitely many different subterms. We postpone the definition of a recursion scheme until Section 6. As the *order* of $\mathcal{G}$ we understand the complexity of $\Lambda(\mathcal{G})$. We do not claim that every regular lambda-term of sort o can be directly represented by a recursion scheme, however we remark that every regular lambda-term of sort o is equivalent (in the sense of having the same Böhm tree) to some lambda-term represented by a recursion scheme.

Simultaneous unboundedness problem. Fix a set of *important constants* Σ_{a}. We say that a closed lambda-term M of sort o is *unbounded* (with respect to Σ_{a}) if for every $n \in \mathbb{N}$ there exists a finite branch of $BT(M)$ with at least n occurrences of every constant from Σ_{a}. The *simultaneous unboundedness problem* (SUP) is to decide, given a recursion scheme $\mathcal{G}$, whether $\Lambda(\mathcal{G})$ is unbounded.

In the special case of $|\Sigma_{\mathsf{a}}| = 1$ we talk about the *unboundedness problem*.

3 Type system

In this section we present a type system that allows us to solve the (single-letter) unboundedness problem. The type system was first proposed in Parys' survey [24], without any correctness proofs.

In the remaining part of the paper, except for Section 7, we assume that the signature Σ consists of four constants: a of arity 1, b of arity 2, and c and ω of arity 0, where only a is important, that is $\Sigma_{\mathsf{a}} = \{\mathsf{a}\}$. This is without loss of generality, since we can replace a constant of arbitrary arity by a combination of these four constants, without changing the answer to the unboundedness problem.

Before defining the type system (and safety), let us state a theorem describing its desired properties:

▶ **Theorem 3.1.** *The following two statements are equivalent for every safe closed lambda-term M of sort o:*

(1) *M is unbounded (i.e., for every $n \in \mathbb{N}$ there exists a finite branch of $BT(M)$ with at least n occurrences of the constant a);*

(2) *for every $n \in \mathbb{N}$ there exists $v \geq n$ such that one can derive $\emptyset \vdash M : (v, \mathsf{r})$.*

In this theorem, type judgments contain a natural number v, called a *productivity value*. The goal of this value is to approximately count the number of occurrences of the important constant a on a selected branch of $BT(M)$.

Types in the type system differ from sorts in that on the left side of $\to$, instead of a single type, we have a set of so-called *type pairs* (f, τ), where τ is a *type*, and $f \in \{\mathsf{pr}, \mathsf{np}\}$ is a *productivity flag* (where pr stands for productive, and np for nonproductive). The unique atomic type is denoted r. More precisely, for each sort α we define the set $\mathcal{T}^\alpha$ of types of sort α as follows:

$$\mathcal{T}^{\mathsf{o}} = \{\mathsf{r}\}, \qquad \mathcal{T}^{\alpha\to\beta} = \mathcal{P}(\{\mathsf{pr}, \mathsf{np}\} \times \mathcal{T}^\alpha) \times \mathcal{T}^\beta,$$

where $\mathcal{P}$ denotes the powerset. A type $(T, \tau) \in \mathcal{T}^{\alpha\to\beta}$ is denoted as $\bigwedge T \to \tau$, or $\bigwedge_{i\in I}(f_i, \tau_i) \to \tau$ when $T = \{(f_i, \tau_i) \mid i \in I\}$. In this notation we implicitly assume that all the pairs (f_i, τ_i) are different. The empty intersection is denoted by $\top$. Moreover, to our terms we will not only assign a type τ, but also a productivity flag $f \in \{\mathsf{pr}, \mathsf{np}\}$ (which together form a pair (f, τ)). Let us emphasize that for every sort α the set $\mathcal{T}^\alpha$ is finite.

Intuitively, a lambda-term has type $\bigwedge T \to \tau$ when it can return τ, while taking an argument for which we can derive all type pairs from T; simultaneously, while having such a type, the lambda-term is obligated to use its arguments in all ways described by type pairs from T. For example, the lambda-term $\lambda x.\mathsf{c}$ does not use its argument, and hence it is necessarily of type $\top \to \mathsf{r}$ (i.e., $\bigwedge T \to \tau$ with $T = \emptyset$).

To determine the productivity flag f assigned to a lambda-term M, we should imagine that M is a subterm of a closed term K of sort o, and we should select some finite branch in $BT(K)$ (with the intuition that different choices of the branch correspond to different type derivations). Then, we assign to M the flag pr (productive) when the subterm M is responsible for increasing the number of occurrences of the constant a on the selected branch. To be more precise, a lambda-term is responsible for producing occurrences of a constant a in two cases. First, when it explicitly contains the constant a – assuming that this a will be placed on the selected branch. Second, when it takes a productive argument (i.e., an argument responsible for producing a) and uses it at least twice. The first possibility occurs for example in the lambda-term $M_1 = \lambda x.\mathsf{a}\,x$; the constant a is explicitly produced. In order to see

the second possibility, consider the lambda-term $M_2 = \lambda y.\lambda x.y\,(y\,x)$, and suppose that the argument received for y is the aforementioned productive lambda-term $\lambda x.\mathsf{a}\,x$, which outputs the constant a. Then, the lambda-term M_2 is itself responsible for increasing the number of occurrences of a in the resulting tree, compared to the number of occurrences produced by the argument. Notice that the same lambda-term M_2 has also another type: the argument received for y may be nonproductive (say $\lambda x.x$), and then M_2 becomes nonproductive as well. Next, let us compare M_2 with the lambda-term $M_2' = \lambda y.\lambda x.y\,x$, when used with the argument $\lambda x.\mathsf{a}\,x$. Although the argument is used, and one a is output, the lambda-term M_2' has nothing to do with increasing the number of occurrences of a; it is nonproductive.

A *type judgment* is of the form $\Gamma \vdash M : (v, \tau)$, where we require that the type τ and the lambda-term M are of the same sort. The *type environment* Γ is a set of bindings of variables of the form $x^\alpha : (g, \tau)$, where $g \in \{\mathsf{pr}, \mathsf{np}\}$ and $\tau \in \mathcal{T}^\alpha$. In Γ we may have multiple bindings for the same variable. By $dom(\Gamma)$ we denote the set of variables x which are bound by Γ, and by $\Gamma{\restriction_{\mathsf{pr}}}$ we denote the set of only those bindings $x : (g, \tau)$ from Γ in which $g = \mathsf{pr}$. We are not only interested in whether some type can be derived, but we also want to assign a value to every derivation; to this end, a type judgment contains a number $v \in \mathbb{N}$, which is called a *productivity value*. Having $v = 0$ corresponds to the np flag, while positive values correspond to the pr flag. Thus, while a productivity flag says only whether any occurrence of the important constant a is produced, the productivity value approximates (is a lower bound on) the number of produced occurrences of this constant. Note that in type judgments we store the productivity value, coming from the infinite set $\mathbb{N}$, while in types we abstract this value to the productivity flag, which allows us to have finitely many types.

We now present rules of the type system, starting from rules for constants:

$$\emptyset \vdash \mathsf{a} : (1, (f, \mathsf{r}) \to \mathsf{r}) \qquad \emptyset \vdash \mathsf{c} : (0, \mathsf{r})$$

$$\emptyset \vdash \mathsf{b} : (0, (f, \mathsf{r}) \to \top \to \mathsf{r}) \qquad \emptyset \vdash \mathsf{b} : (0, \top \to (f, \mathsf{r}) \to \mathsf{r})$$

Notice that in the rule for b we have a type (f, r) (with an arbitrary flag f) only for one of the two arguments; this corresponds to the fact that we are interested in a single branch of the Böhm tree, so we want to descend only to a single child. Moreover, we do not have a rule for the constant ω, because by definition a branch cannot contain occurrences of this constant.

While typing a variable x, we take its type from the type environment, and we use 0 as the productivity value. The lambda-term x itself is not responsible for producing any constants, no matter whether the lambda-term substituted for x will produce any constants or not. A lambda-term becomes productive when a productive variable x is used twice; we account for that in the application typing rule (@).

$$x : (f, \tau) \vdash x : (0, \tau)$$

When we pass through a lambda-binder, we simply move some type pairs between the argument and the type environment:

$$\frac{\Gamma \cup \{x : (f_i, \tau_i) \mid i \in I\} \vdash K : (v, \tau) \qquad x \notin dom(\Gamma)}{\Gamma \vdash \lambda x.K : (v, \bigwedge_{i \in I} (f_i, \tau_i) \to \tau)}\ (\lambda)$$

Before giving the last rule, we need one more definition. Given a family of type environments $(\Gamma_i)_{i \in J}$, a *duplication factor*, denoted $dupl((\Gamma_i)_{i \in J})$, equals $\sum_{i \in J} \left|\Gamma_i{\restriction_{\mathsf{pr}}}\right| - \left|\bigcup_{i \in J} \Gamma_i{\restriction_{\mathsf{pr}}}\right|$. It counts the number of repetitions ("duplications") of productive type bindings in the type environments: a productive type binding belonging to one type environment does not add anything, a productive type binding belonging to two type environments adds 1, and so on.

$$\frac{0 \notin I \qquad \forall i \in I.\ (f_i = \mathsf{pr}) \Leftrightarrow (v_i > 0 \vee \Gamma_i{\restriction_{\mathsf{pr}}} \neq \emptyset) \qquad \Gamma_0 \vdash K : (v_0, \bigwedge_{i\in I}(f_i,\tau_i) \to \tau) \qquad \Gamma_i \vdash L : (v_i,\tau_i) \text{ for each } i \in I}{\bigcup_{i\in\{0\}\cup I} \Gamma_i \vdash K\, L : (\mathit{dupl}((\Gamma_i)_{i\in\{0\}\cup I}) + \sum_{i\in\{0\}\cup I} v_i, \tau)}\ (@)$$

Here, by using the notation $\bigwedge_{i\in I}(f_i,\tau_i)$, we assume that the pairs (f_i,τ_i) are all different.

In the rule above, the condition $(f_i = \mathsf{pr}) \Leftrightarrow (v_i > 0 \vee \Gamma_i{\restriction_{\mathsf{pr}}} \neq \emptyset)$ means that when K requires a "productive" argument, then either we can apply an argument L that is itself productive, or we can apply a nonproductive L that uses a productive variable (the argument obtained after substituting something for this variable will become productive).

Using the *dupl* function we realize the intuition that when a variable responsible for creating occurrences of a (i.e., productive) is used at least twice, then the lambda-term is itself responsible for increasing the number of occurrences of a; thus we add *dupl* to the productivity value.

Because we are interested in counting duplications of type bindings, it was necessary to require that every type binding from the type environment is actually used somewhere (in particular $\Gamma \vdash M : (f,\tau)$ does not necessarily imply $\Gamma, x : (g,\sigma) \vdash M : (f,\tau)$). On the other hand, a type environment is a set: a repeated usage of a type binding is counted in the productivity value, but is not reflected in the type environment.

Let us underline that although we consider infinite lambda-terms, all type derivations are required to be finite. This may look suspicious, but note that when a lambda-term has type $\top \to \tau$ (like, e.g., the constant b), then we need no derivation for its argument.

▶ **Example 3.2.** Let us give an example of a derivation for a lambda-term $\lambda y.\lambda z.y\,(y\,(\mathsf{a}\,z))$ of sort $(\mathsf{o}\to\mathsf{o})\to\mathsf{o}\to\mathsf{o}$. In this particular derivation, we will assume that y and z are both productive.

$$\frac{\vdash \mathsf{a} : (1,(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \qquad z : (\mathsf{pr},\mathsf{r}) \vdash z : (0,\mathsf{r})}{z : (\mathsf{pr},\mathsf{r}) \vdash \mathsf{a}\,z : (1,\mathsf{r})}\ (@)$$

In the innermost application, we have an important constant a and a productive variable z. The important constant has value 1. The value of z is 0 because, even though it is productive, any important constants it produces will be just moved from the argument substituted for z. No important constant will be lost or produced during this process. There are no duplicates of variables in the application $\mathsf{a}\,z$, so the value of the application is equal to sum of values, that is, 1.

$$\frac{y : (\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \vdash y : (0,(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \qquad z : (\mathsf{pr},\mathsf{r}) \vdash \mathsf{a}\,z : (1,\mathsf{r})}{z : (\mathsf{pr},\mathsf{r}),\ y : (\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \vdash y\,(\mathsf{a}\,z) : (1,\mathsf{r})}\ (@)$$

We apply y to $\mathsf{a}\,z$. Variable y is productive and takes a productive argument, which means that it incorporates its argument into the tree it produces and somehow increases the number of important constants in the process. However, this increase is computed in the lambda-term that is substituted for y, not here, hence it has value 0.

$$\frac{y : (\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \vdash y : (0,(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \qquad z : (\mathsf{pr},\mathsf{r}),\ y : (\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \vdash y\,(\mathsf{a}\,z) : (1,\mathsf{r})}{z : (\mathsf{pr},\mathsf{r}),\ y : (\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \vdash y\,(y\,(\mathsf{a}\,z)) : (2,\mathsf{r})}\ (@)$$

Both sides have y in the environment here, so the duplication factor in $y\,(y\,(\mathsf{a}\,z))$ is 1. We add it to the sum of values from both sides of the application, obtaining 2.

$$\dfrac{\dfrac{z : (\mathsf{pr},\mathsf{r}),\ y : (\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \vdash y\,(y\,(\mathsf{a}\,z)) : (2,\mathsf{r})}{y : (\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r}) \vdash \lambda z.y\,(y\,(\mathsf{a}\,z)) : (2,(\mathsf{pr},\mathsf{r})\to\mathsf{r})}\,(\lambda)}{\vdash \lambda y.\lambda z.y\,(y\,(\mathsf{a}\,z)) : (2,(\mathsf{pr},(\mathsf{pr},\mathsf{r})\to\mathsf{r})\to(\mathsf{pr},\mathsf{r})\to\mathsf{r})}\,(\lambda)$$

The lambda-binders move z and y from the environment into the type without changing the value. The final value for this closed lambda-term is 2, as it always adds one important constant to the Böhm tree and, when something that increases the number of important constants is substituted for y, it applies that twice, causing at least one extra important constant to be added compared to what just one instance of y would do.

Note that it is possible to derive other type judgments for the lambda-term $y\,(y\,(\mathsf{a}\,z))$. For example, the value would be equal to one if y was not productive, even if z was productive:

$$y : (\mathsf{np},(\mathsf{pr},\mathsf{r})\to\mathsf{r}),\ z : (\mathsf{pr},\mathsf{r}) \vdash y\,(y\,(\mathsf{a}\,z)) : (1,\mathsf{r}).$$

The rationale is that while y takes productive arguments, it uses them exactly once and does not add an important constant to the Böhm tree either, so applying it any number of times will not change the number of important constants.

4 Soundness

In this section we prove the soundness of the type system, as described by following lemma, from which the (2) ⇒ (1) implication of Theorem 3.1 follows immediately:

▶ **Lemma 4.1.** *If we can derive $\emptyset \vdash M : (v,\mathsf{r})$, where M is a closed lambda-term of sort o, then $BT(M)$ has a finite branch containing at least v occurrences of a.*

We prove Lemma 4.1 as follows. First, as in work of Parys [26, 24] we observe that instead of working directly with infinite lambda-terms M, we can "cut off" parts of M not involved in the finite derivation of $\emptyset \vdash M : (v,\mathsf{r})$, (i.e., subterms used as arguments for which no type pair is required). Formally, by cutting off we mean replacing by lambda-terms of the form $\lambda x_1.\cdots.\lambda x_k.\omega$, where the variables $x_1,\ldots,x_k$ are chosen so that the sort of the lambda-term is appropriate. In consequence, it is enough to prove Lemma 4.1 for finite lambda-terms.

Second, we repeatedly use Lemma 4.2 below, reducing M to its beta-normal form N, and never decreasing the productivity value. Finally, we observe that the beta-normal form N is simply a tree; a derivation of $\emptyset \vdash N : (v',\mathsf{r})$, using only the rules for constants and application, describes some branch of this tree containing exactly v' occurrences of the important constant a. It remains to justify the following lemma, which describes a single beta-reduction:

▶ **Lemma 4.2.** *If we can derive $\emptyset \vdash M : (v,\mathsf{r})$, where M is a finite closed lambda-term of sort o, and M is not in the beta-normal form, then we can derive $\emptyset \vdash N : (v',\mathsf{r})$ for a lambda-term N such that $M \to_\beta N$, and for some v' satisfying $v \le v'$.*

While proving this lemma, we consider the leftmost outermost redex, $(\lambda x.K)\,L$. Thanks to this choice, L is necessarily a closed subterm. This simplifies the situation: it is enough to consider empty type environments (we remark, however, that Lemma 4.2 can be shown in a similar way for every reduction, not only for the leftmost outermost reduction). We want to replace every subderivation D for a type judgment $\emptyset \vdash (\lambda x.K)\,L : (w,\tau)$ concerning

this redex with a derivation D' for $\Gamma \vdash K[L/x] : (w', \tau)$. We obtain D' by appropriately reorganizing subderivations of D: we take the subderivation of D concerning K, we replace every leaf deriving a type σ for x by the subderivation of D deriving this type σ for L, and we update type environments and productivity values appropriately.

Notice that every subderivation concerning L is moved to at least one leaf concerning x (nothing can disappear). The only reason why the value of the derivation can decrease is that potentially a productive type binding $x : (\mathsf{pr}, \sigma)$ was duplicated (say, n times) in the derivation concerning K. In D' this binding is no longer present (in $K[L/x]$ there is no x) so the value decreases by n, but in this situation the subderivation deriving σ for L becomes inserted in $n+1$ leaves. This subderivation is productive, so by creating n additional copies of this subderivation we increase the value at least by n, compensating the loss caused by elimination of x. This implies that $w \leq w'$, and consequently $v \leq v'$. Check Appendix A of the full version for more details.

5 Completeness for safe lambda-terms

In this section we prove completeness for a subclass of lambda-terms called safe lambda-terms. The question whether completeness holds for general lambda-terms is left open.

A lambda-term M is *superficially safe* when for every free variable x of M it holds that $ord(M) \leq ord(x)$. A lambda-term M is *safe* if it is superficially safe, and if for its every subterm of the form $K\, L_1 \ldots L_k$, where K is not an application and $k \geq 1$, all subterms $K, L_1, \ldots, L_k$ are superficially safe. This definition of safety coincides with the definitions from Salvati and Walukiewicz [27], and from Blum and Ong [2].

Completeness for safe lambda-terms is given by the following lemma.

▶ **Lemma 5.1.** *For every $m \in \mathbb{N}$ there exists a function $H_m \colon \mathbb{N} \to \mathbb{N}$ such that if M is a closed safe lambda-term of sort o and complexity at most m, and in $BT(M)$ there is a finite branch having at least n occurrences of a, then we can derive $\emptyset \vdash M : (v, \mathsf{r})$ for some v such that $n \leq H_m(v)$.*

While proving this lemma, it is convenient to split the productivity value into two parts. Given some fixed $\ell \in \mathbb{N}$, instead of type judgments of the form $\Gamma \vdash N : (v, \tau)$ we consider extended type judgments of the form $\Gamma \vdash N : (u \oplus_\ell w, \tau)$, where $u + w = v$. On u we accumulate only duplication factors concerning variables of order at least ℓ, and on w the remaining part of the value (i.e., duplication factors concerning variables of order smaller than ℓ, plus the number of rules for the constant a). Having this definition, we can state a counterpart of Lemma 4.2:

▶ **Lemma 5.2.** *Suppose that we can derive $\Gamma \vdash K[L/x] : (u \oplus_\ell w, \tau)$, where L is closed, has order ℓ, and does not use any variables of order at least ℓ. Then we can derive $\Gamma \vdash (\lambda x.K)\, L : (u' \oplus_\ell w', \tau)$ for some u', w' such that $2^u \cdot w \leq 2^{u'} \cdot w'$ and $u + w = 0 \Rightarrow u' + w' = 0$.*

Similarly to Lemma 4.2, the derivation concerning $(\lambda x.K)\, L$ in the above lemma is obtained by appropriately reorganizing subderivations of the derivation concerning $K[L/x]$. In the type derivation concerning $K[L/x]$, there are some (zero or more) subderivations concerning L. A difficulty is caused by the fact that the same type pair (f, σ) may be derived for multiple copies of L in $K[L/x]$, using different subderivations. For every such (f, σ) derived for L we should choose just one subderivation, so that we can use it for the only copy of L in $(\lambda x.K)\, L$. We choose the subderivation that provides the largest second component of the value; the other subderivations are removed. The value of the new derivation decreases, because some subderivations are removed, and increases, because we have a new variable x, that may cause some duplications.

It remains to see that the value $u' \oplus_\ell w'$ of the new derivation satisfies the inequality $2^u \cdot w \leq 2^{u'} \cdot w'$. Consider some type pair (f, σ) derived for L. If $f = \mathsf{np}$, the value of the removed subderivations is 0, and the duplications of $x : (f, \sigma)$ in the type environment are not counted, so such a type pair does not cause any change of the value. Suppose that $f = \mathsf{pr}$, and that we have removed n subderivations proving the type pair (f, σ) for L; this may decrease the second component of the value at most $n + 1$ times (the n removed subderivations have values not greater than the one that remains). Simultaneously, in K we use $n + 1$ times the variable x with this type pair (f, σ), which increases the value (the total duplication factor) by n. The key point is that L does not use any variables of order at least ℓ, and thus the first component (concerning variables of order at least ℓ) of the value of subderivations for L is 0. Thus, we decrease the second component of the value at most $n + 1$ times, and we increase the first component of the value by at least n. Since $2^n \geq n + 1$, we obtain the required inequality between values.

It is also important that L is closed so that after removing some subderivations concerning L, the type environment of the whole derivation remains unchanged. This finishes the proof of Lemma 5.2 (a more formal proof, containing all details, can be found in Appendix B.1 of the full version).

We now come back to the proof of Lemma 5.1. First, as in the previous section, we can assume that M is finite by "cutting off" (i.e., replacing with $\lambda x_1.\cdots\lambda x_k.\omega$) its parts not needed for producing the selected finite branch of $BT(M)$.

Second, it is convenient to assume here that M is homogeneous. A sort $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}$ is *homogeneous* if $ord(\alpha_1) \geq \cdots \geq ord(\alpha_k)$ and all $\alpha_1, \ldots, \alpha_k$ are homogeneous. A lambda-term is homogeneous if its every subterm has a homogeneous sort. It is known that every finite closed safe lambda-term M of sort o can be converted into a lambda-term M' that is additionally homogeneous [23], but has the same beta-normal form. Analyzing how M is transformed into M' (in [23]), it is tedious but straightforward to check that we can derive $\emptyset \vdash M : (v, \mathsf{r})$ if and only if we can derive $\emptyset \vdash M' : (v, \mathsf{r})$; this is done in Appendix B.2 of the full version.

It is thus enough to prove Lemma 5.1 assuming that M is finite and homogeneous. To this end, we consider a particular sequence of reductions leading from M to its beta-normal form $BT(M)$. Namely, whenever a lambda-term N reached so far (i.e., after some number of reductions) from M is of complexity $\ell + 1$, then we reduce in N a redex $(\lambda x.K)\, L$ such that $ord(\lambda x.K) = \ell + 1$, and no variables of order at least ℓ occur in L, both as free variables and as bound variables; we call such a redex *ℓ-good*. Such a redex always exists: among redexes with $ord(\lambda x.K) = \ell + 1$ it is enough to choose the rightmost one. The complexity of a lambda-term cannot increase during a beta-reduction, thus in the sequence of reductions we can find lambda-terms $M_m, M_{m-1}, \ldots, M_0$, where $M_m = M$, and $M_0 = BT(M)$, and the complexity of every $M_{\ell+1}$ is at most $\ell + 1$, and M_ℓ can be reached from $M_{\ell+1}$ by a sequence of ℓ-good reductions.

Homogeneity is preserved during beta-reductions. Moreover, ℓ-good beta-reductions preserve safety (while this is not true for arbitrary beta-reductions). Indeed, homogeneity for an ℓ-good redex $(\lambda x.K)\, L$ implies that $ord(L) = \ell$ (the first argument is of the highest order, i.e., of order ℓ), thus safety implies that all free variables of L are of order at least ℓ. But, by the definition of an ℓ-good redex, no variables of order at least ℓ occur in L. It follows that L is closed. While substituting a closed lambda-term L for x, all superficially safe subterms of K remain superficially safe (no new free variables appear). Thus the lambda-term after the ℓ-good beta-reduction remains safe.

Recall the sequence of lambda-terms $M_m, M_{m-1}, \ldots, M_0$ defined above. Assuming that in $BT(M) = M_0$ there is a finite branch having exactly n occurrences of a, we can derive $\emptyset \vdash M_0 : (n, \mathsf{r})$ using only the rules for application and constants. Suppose now that we can derive $\emptyset \vdash M_\ell : (v_\ell, \mathsf{r})$ for some $\ell \in \{0, \ldots, m-1\}$. We want to find a derivation of $\emptyset \vdash M_{\ell+1} : (v_{\ell+1}, \mathsf{r})$, where for arbitrarily large values v_ℓ, the values $v_{\ell+1}$ are also arbitrarily large (more precisely, we obtain the inequality $v_\ell \leq 2^{v_{\ell+1}}$). Notice that between $M_{\ell+1}$ and M_ℓ there may be arbitrarily many (ℓ-good) reductions. Consider thus some ℓ-good redex $(\lambda x.K)\, L$ that is reduced at some moment between $M_{\ell+1}$ and M_ℓ. By definition, L does not use any variables of order at least ℓ and, as already observed, L is closed; thus, Lemma 5.2 can be applied. We start with $\emptyset \vdash M_\ell : (0 \oplus_\ell v_\ell, \mathsf{r})$ (notice that in M_ℓ there are no variables of order ℓ or higher, so the first component of the value is 0). Then, we use Lemma 5.2 for every ℓ-good beta-reduction between $M_{\ell+1}$ and M_ℓ. This leads to $\emptyset \vdash M_{\ell+1} : (u_{\ell+1} \oplus_\ell w_{\ell+1}, \mathsf{r})$, where $v_\ell \leq 2^{u_{\ell+1}} \cdot w_{\ell+1} \leq 2^{v_{\ell+1}}$ (taking $v_{\ell+1} = u_{\ell+1} + w_{\ell+1}$), as required.

The function H_m, appearing in the statement of Lemma 5.1, can be defined as a tower of powers of 2 of height m: $H_0(v) = v$ and $H_{\ell+1}(v) = H_\ell(2^v)$. Then from $n \leq H_\ell(v_\ell)$ it follows that $n \leq H_{\ell+1}(v_{\ell+1})$; since $n = H_0(v_0)$, we thus obtain the desired inequality $n \leq H_m(v_m)$.

6 The algorithm

Both design and implementation of our algorithm is based on the HorSat algorithm, a saturation-based algorithm for model checking recursion schemes against alternating automata by Broadbent and Kobayashi [4].

Our type system was described in previous sections to work with any infinitary lambda-term. The algorithm, in turn, inputs infinitary lambda-terms represented in a finite way, in the form of a recursion scheme. To be concrete, we make this representation explicit now: A recursion scheme $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, X_{\mathsf{st}})$ consists of

- a signature Σ (i.e., a set of constants with assigned arities),
- a set $\mathcal{N}$ of nonterminals with assigned sorts (formally, nonterminals are just distinguished variables),
- a mapping $\mathcal{R}$ from nonterminals in $\mathcal{N}$ to finite lambda-terms such that $\mathcal{R}(X)$ is of the same sort as X, has no free variables other than nonterminals from $\mathcal{N}$, and is of the form $\lambda x_1.\cdots.\lambda x_n.K$, where the subterm K does not contain any lambda-binders, and
- a starting nonterminal $X_{\mathsf{st}} \in \mathcal{N}$ of sort o.

We assume that elements of $\mathcal{N}$ are not used as bound variables, and that $\mathcal{R}(X)$ is not a nonterminal. Furthermore, as in previous sections, we assume for simplicity that $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, where the constants $\mathsf{a}, \mathsf{b}, \mathsf{c}$ have arity $1, 2, 0$, respectively (the implementation, however, accepts arbitrary signatures).

Given a recursion scheme $\mathcal{G}$, and a lambda-term M (possibly containing some nonterminals from $\mathcal{N}$), let $\Lambda_{\mathcal{G}}(M)$ be the lambda-term obtained as a limit of applying repeatedly the following operation to M: take an occurrence of some nonterminal X, and replace it by $\mathcal{R}(X)$ (the nonterminals should be chosen so that every nonterminal is eventually replaced). We remark that while substituting $\mathcal{R}(X)$ for a nonterminal X, there is no need for any renaming of variables (capture-avoiding substitution), since $\mathcal{R}(X)$ does not have free variables other than nonterminals. The infinitary lambda-term *represented by* $\mathcal{G}$ is defined as $\Lambda_{\mathcal{G}}(X_{\mathsf{st}})$, and denoted $\Lambda(\mathcal{G})$. Observe that $\Lambda(\mathcal{G})$ is a closed lambda-term of sort o. We say that $\mathcal{G}$ is *safe* if $\mathcal{R}(X)$ is safe for every $X \in \mathcal{N}$; then $\Lambda(\mathcal{G})$ is safe as well.

The goal of the algorithm is to determine whether $\Lambda(\mathcal{G})$ for a given recursion scheme $\mathcal{G}$ is unbounded or not. In the light of Theorem 3.1, this boils down to checking whether one can derive $\emptyset \vdash \Lambda(\mathcal{G}) : (v, \mathsf{r})$ for arbitrarily large values of v. The idea is to find a single derivation with a productive "loop"; by repeating this loop, one can increase the productivity value arbitrarily. We make this more precise now.

First, we determine type pairs that can be derived for non-terminals. Namely, for $v \in \mathbb{N}$ let $\mathfrak{p}(v) = \mathsf{pr}$ if $v > 0$ and $\mathsf{pr}(v) = \mathsf{np}$ otherwise. Let $\mathcal{T}_{\mathcal{G}}$ be the smallest set containing all bindings $X : (\mathfrak{p}(v), \tau)$ (with $X \in \mathcal{G}$) such that we can derive $\emptyset \vdash \mathcal{R}(X) : (v, \tau)$, potentially using $\emptyset \vdash Y : (w, \sigma)$ for $(Y : (\mathfrak{p}(w), \sigma)) \in \mathcal{T}_{\mathcal{G}}$ as assumptions (i.e., as additional typing rules).

Note that productivity values in derivations may be shifted (i.e., increased/decreased by a constant): we can derive $\emptyset \vdash \mathcal{R}(X) : (v, \tau)$ out of an assumption $\emptyset \vdash Y : (w, \sigma)$ if and only if we can derive $\emptyset \vdash \mathcal{R}(X) : (v + (w' - w), \tau)$ out of an assumption $\emptyset \vdash Y : (w', \sigma)$ with $\mathfrak{p}(w) = \mathfrak{p}(w')$. It is thus enough to try assumptions $\emptyset \vdash Y : (w, \sigma)$ with $w = 0$ and $w = 1$ only.

We have only finitely many bindings that may be potentially added to $\mathcal{T}_{\mathcal{G}}$, as there are finitely many possible types per sort and each nonterminal of $\mathcal{G}$ has a fixed sort. Moreover, taking into account the above, there are only finitely many derivations that may be potentially created for lambda-terms $\mathcal{R}(X)$. It follows that $\mathcal{T}_{\mathcal{G}}$ may be computed using saturation (i.e., as the least fixed point). The following lemma is immediate:

▶ **Lemma 6.1.** *For any nonterminal $X \in \mathcal{N}$ and any type pair (f, τ) one can derive $\Lambda_{\mathcal{G}}(X) : (v, \tau)$ for some v with $\mathfrak{p}(v) = f$ if and only if $(X : (f, \tau)) \in \mathcal{T}_{\mathcal{G}}$.*

Next, knowing which type judgments may be derived, we want to detect a productive cycle. To this end, we create a graph with bindings from $\mathcal{T}_{\mathcal{G}}$ as nodes, called a *derivation graph*. We draw an edge from $X : (\mathfrak{p}(v), \tau)$ to $Y : (\mathfrak{p}(w), \sigma)$ if one can derive $\emptyset \vdash \mathcal{R}(X) : (v, \tau)$ using $\emptyset \vdash Y : (w, \sigma)$ as an assumption, and potentially using some other assumptions $\emptyset \vdash Z : (u, \rho)$ with $(Z : (\mathfrak{p}(u), \rho)) \in \mathcal{T}_{\mathcal{G}}$ (the assumption $\emptyset \vdash Y : (w, \sigma)$ necessarily has to be used, at least once). Such an edge is called *productive* if $v > w$. Note that this may happen for three reasons: 1) in the derivation there is some positive duplication factor or an important constant, 2) some other assumption $\emptyset \vdash Z : (u, \rho)$ with $u > 0$ is used, or 3) the assumption $\emptyset \vdash Y : (w, \sigma)$ is used more than once and $w > 0$. Note that edges of the derivation graphs can be incrementally computed at the time of computing $\mathcal{T}_{\mathcal{G}}$. We now obtain the main theorem, adding Point (3) equivalent to Point (2) from Theorem 3.1:

▶ **Theorem 6.2.** *The following two statements are equivalent for every recursion scheme $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, X_{\mathsf{st}})$:*

(2) *one can derive $\emptyset \vdash \Lambda(\mathcal{G}) : (v, \mathsf{r})$ for arbitrarily large values of v,*

(3) *$(X_{\mathsf{st}} : (\mathsf{pr}, \mathsf{r})) \in \mathcal{T}_{\mathcal{G}}$ and the derivation graph contains a cycle with a productive edge, reachable from $X_{\mathsf{st}} : (\mathsf{pr}, \mathsf{r})$.*

By Theorem 3.1, assuming additionally that $\mathcal{G}$ is safe, Point (2) (and hence Point (3) as well) is also equivalent to the property we want to check:

(1) $\Lambda(\mathcal{G})$ is unbounded.

Proof. Point (2) follows from Point (3) quite directly. Indeed, an edge from $X : (\mathsf{pr}, \tau)$ to $Y : (\mathsf{pr}, \sigma)$ means that any derivation of $\emptyset \vdash \Lambda_{\mathcal{G}}(Y) : (w, \sigma)$ can be extended to a derivation of $\emptyset \vdash \Lambda_{\mathcal{G}}(X) : (v, \tau)$ for some $v \geq w$ (in the sense that the latter contains the former as a subderivation), where $v > w$ if the edge is productive. Following edges on the cycle with a

productive edge we can thus extend a derivation of $\emptyset \vdash \Lambda_{\mathcal{G}}(X) : (v, \tau)$ into a larger derivation of $\emptyset \vdash \Lambda_{\mathcal{G}}(X) : (v', \tau)$, where $v' > v$; doing this repeatedly increases the productivity value arbitrarily. Finally, we can use the path from $X_{\mathsf{st}} : (\mathsf{pr}, \mathsf{r})$ to the cycle, making the created derivation a part of a derivation concerning $\Lambda(\mathcal{G}) = \Lambda_{\mathcal{G}}(X_{\mathsf{st}})$.

Let us move on to the implication from Point (2) to Point (3). First, consider all possible derivations of $\emptyset \vdash \mathcal{R}(X) : (v, \tau)$ (with arbitrary $X \in \mathcal{N}$, v, τ) that may use assumptions (additional typing rules) $\emptyset \vdash Y : (v', \tau')$ for nonterminals Y, as in the definition of $\mathcal{T}_{\mathcal{G}}$. Because $\mathcal{R}(X)$ for every $X \in \mathcal{N}$ is a finite lambda-term, and $|\mathcal{N}|$ is finite as well, there exists a bound B such that any derivation as above uses assumptions for nonterminals in at most B leaves, and simultaneously the growth of productivity value caused by duplication factor or important constants in the derivation is at most B.

Next, we prove (by induction on the size of a derivation) that if we can derive $\emptyset \vdash \Lambda_{\mathcal{G}}(X) : (v, \tau)$ with $v \geq (2B)^k$ for some $k \in \mathbb{N}$, then $(X : (\mathfrak{p}(v), \tau)) \in \mathcal{T}_{\mathcal{G}}$ and in the derivation graph there is a path from this node having at least k productive edges. Indeed, out of a derivation of $\emptyset \vdash \Lambda_{\mathcal{G}}(X) : (v, \tau)$ we can reconstruct a derivation of $\emptyset \vdash \mathcal{R}(X) : (v, \tau)$; whenever the former derivation contains a (strictly smaller) subderivation concerning $\Lambda_{\mathcal{G}}(Y)$ for some nonterminal Y, in the latter we use an assumption concerning Y, which is in $\mathcal{T}_{\mathcal{G}}$ by the induction hypothesis. This already shows that $(X : (\mathfrak{p}(v), \tau)) \in \mathcal{T}_{\mathcal{G}}$. If $k = 0$, we are done. Suppose that $k \geq 1$. By properties of the bound B, in the derivation of $\emptyset \vdash \mathcal{R}(X) : (v, \tau)$ there has to be an assumption $\emptyset \vdash Y : (v', \tau')$ with $v' \geq \frac{v-B}{B} \geq \frac{v}{2B}$ (from v we subtract B for duplication factors and important constants inside the derivation of $\emptyset \vdash \mathcal{R}(X) : (v, \tau)$, and we divide the remaining part of the productivity value into at most B assumptions), that is, we can derive $\emptyset \vdash \Lambda_{\mathcal{G}}(Y) : (v', \tau')$. If $v' = v \geq (2B)^k$, then we have an edge to a node from which there is a path having at least k productive edges, by the induction hypothesis. Otherwise $v > v' \geq (2B)^{k-1}$, so we have a productive edge to a node, from which there is a path having at least $k - 1$ productive edges, by the induction hypothesis.

Starting with $X = X_{\mathsf{st}}$ and $k > |\mathcal{T}_{\mathcal{G}}|$ we obtain a path from $(X_{\mathsf{st}} : (\mathsf{pr}, \mathsf{r}))$ containing more than $|\mathcal{T}_{\mathcal{G}}|$ productive edges; this path has to reach a cycle with a productive edge, as needed for Point (3). ◀

6.1 Implementation

Our implementation is based on the implementation of HorSat [4] (more precisely: of HorSat2, a revised version of the HorSat algorithm, due to Kobayashi and Terao). The main reason behind the efficiency of our (and HorSat's) implementation is that we do not compute all possible bindings in $\mathcal{T}_{\mathcal{G}}$. We derive only types which may be useful in a derivation of a type of the starting nonterminal X_{st}.

To this end, we first perform a 0-CFA analysis of the input to compute which lambda-terms may flow into particular nonterminal parameters. For example, if we have a nonterminal X with a parameter x, we find all applications in the form $M_X\, N$, where M_X is a lambda-term where the head is X or may eventually be substituted by X; then the lambda-term N is flowing into the parameter x. Note that 0-CFA analysis gives an overestimation. For example, for a nonterminal Y with two parameters x and y it is possible to write two applications, $Y\, M\, N$ and $Y\, M'\, N'$, transformed in such a way that, according to 0-CFA output, M may flow into x at the same time as N' into y. This behavior can exponentially (with respect to the number of arguments) increase the number of possible typings of Y that the algorithm has to check.

Thanks to 0-CFA analysis, we have a complete list of lambda-terms which may be substituted for parameters of nonterminals, which limits the set of computed types of nonterminals while still retaining all types required to type X_{st}. This is sufficient to start a

loop where new types are found. In this loop, nonterminals (more precisely: the lambda-terms $\mathcal{R}(X)$ for nonterminals X) are typed using information about types flowing into their parameters. New types of nonterminals enable finding new types of lambda-terms in which these nonterminals occur. These lambda-terms again flow into parameters of some nonterminals, so the new types enable finding additional typings of these nonterminals, returning to the beginning of the loop. This loop continues until a fixpoint, that is, until no new typings of nonterminals or lambda-terms flowing into their parameters can be computed. As there is a finite number of possible typings of nonterminals and lambda-terms in any given recursion scheme, this fixpoint will be reached after finitely many steps. During this loop, each time we type a nonterminal, we take note of typings of nonterminals that were used in the derivation (i.e., are subtrees in the final derivation tree), and incrementally construct the derivation graph described earlier in this section. This way we find all parts of the derivation graph that are reachable from $X_{\mathsf{st}} : (\mathsf{pr}, \mathsf{r})$; the optimizations made by 0-CFA only remove unreachable parts.

We perform typing of terms without lambda-bindings in a top-down manner, that is, we iterate over all possible types an application may have, find compatible types for its left-hand side, and then type-check its right-hand side under each possible environment with its desired, already known type. The top-down approach is particularly efficient when the constant b is present in the term, since its argument with type $\top$ does not have to be type-checked. This is also the case for nonterminals with arguments of type $\top$.

We also include three optimizations similar to the ones present in HorSat:

- After a new type of a nonterminal is found, we type all lambda-terms that contain it in a way where the new typing is used at least once. This optimization can be turned off with a flag `-nofntty`.
- After a new type of a lambda-term flowing into a parameter x of a nonterminal X is found, we search for new typings of $\mathcal{R}(X)$ (or its subterms that flow into parameters of other nonterminals) in a way where at least one instance of the parameter x has the new type. This optimization can be turned off with a flag `-noftty`.
- When no parameter of a nonterminal X is used as a left-hand side of an application in $\mathcal{R}(X)$, we infer types of parameters of X from left-hand sides of applications instead of using types of lambda-terms flowing into these parameters. Then $\mathcal{R}(X)$ does not have to be typed whenever a new type is computed for lambda-terms flowing into its parameters. This optimization can be turned off with a flag `-nohvo`.

6.2 Benchmarks

We prepared benchmarks for the implementation of our algorithm by modifying benchmarks presented in the HorSat paper [4]. HorSat is an algorithm that efficiently checks whether an alternating tree automaton (ATA) accepts the Böhm tree of the lambda-term defined by a recursion scheme. This problem is different from ours, however, it also performs an analysis on the same trees and is also m-EXPTIME-complete, where the difficulty depends on the order m of the lambda-term. Hence, we use benchmark results presented in the HorSat paper [4] as an indication that the terms used there are difficult to analyze.

The details on the original benchmarks and their origin can be found in the HorSat paper [4]. According to authors of the aforementioned paper, these benchmarks contain practical data. Indeed, some of them model analysis of an XHTML document or a short computer program. Analysis of many of them was not completed in a reasonable time by other model checking algorithms similar to HorSat, while HorSat only failed to analyze benchmark `fibstring` in a reasonable time.

Table 1 InfSat benchmark results in all combinations of optimization flags.

		Flags and run times in seconds							
Benchmark name	$ord(\mathcal{G})$	no flags	`-noftty`	`-nofntty`	`-nohvo`	`-noftty -nofntty`	`-noftty -nohvo`	`-nofntty -nohvo`	`-noftty -nofntty -nohvo`
`jwig-cal_main`	2	0.942	0.930	1.572	0.611	1.574	0.607	0.728	0.783
`spec_cps_coerce1-c`	3	66.28	3.921	3.285	102.2	0.647	3.517	38.105	4.557
`xhtmlf-div-2`	2	9.591	9.555	7.457	9.238	7.426	9.007	7.387	7.478
`xhtmlf-m-church`	2	9.689	9.655	7.481	9.235	7.470	9.172	7.459	7.562
`fold_fun_list`	7	0.425	0.025	OOM	0.402	OOM	0.024	OOM	OOM
`fold_right`	5	25.01	0.017	0.277	23.75	0.028	0.016	0.265	0.028
`search-e-church`	6	127.7	14.96	338.1	119.5	12.19	14.11	325.9	13.53
`zip`	4	2.618	153.4	64.79	27.30	201.5	150.0	68.63	98.81
`filepath`	2	TO	OOM	OOM	OOM	OOM	OOM	OOM	OOM
`filter-nonzero-1`	5	59.69	1.635	37.56	64.19	2.636	1.641	39.52	1.922
`filter-nonzero`	5	0.641	0.106	0.570	0.670	0.138	0.106	0.653	0.083
`map-plusone-2`	5	5.384	1.082	2.591	5.065	0.851	1.092	5.633	1.125
`cfa-life2`	14	TO	TO	OOM	TO	TO	TO	TO	TO
`cfa-matrix-1`	8	1.794	2.376	1.672	1.708	2.124	2.310	1.582	2.280
`cfa-psdes`	7	0.017	0.022	0.019	0.026	0.022	0.018	0.041	0.029
`tak_inf`	8	10.48	5.842	12.16	9.759	11.88	5.739	11.58	8.858
`dna`	2	0.118	0.106	0.050	0.115	0.074	0.109	0.055	0.074
`fibstring`	4	0.022	0.024	0.016	0.023	0.020	0.023	0.015	0.020
`g45`	4	41.12	45.73	7.819	16.46	56.61	46.54	1.629	46.06
`l`	3	0.012	0.023	0.016	0.017	0.017	0.022	0.015	0.017
`fib02_odd_fin`	4	0.007	0.014	0.009	0.013	0.013	0.013	0.014	0.013
`fib_even_inf`	4	0.545	2.854	0.697	0.505	2.555	2.637	0.697	2.604
`two_add_inf`	4	0.022	0.018	0.023	0.024	0.015	0.012	0.021	0.015
`two_succ_inf`	4	0.005	0.006	0.005	0.006	0.004	0.005	0.006	0.005

TO means timeout (10 minutes), OOM means out of memory error.

Let us describe how we modified the benchmarks to fit our analysis. We left the original recursion scheme intact and selected a few important constants that are present close to leaves of generated trees to increase the difficulty. The result was a decision problem whether the operations described by important constants in modelled programs could be executed unbounded number of times, assuming the program halted. Additionally, we added four benchmarks specific to InfSat that answer mathematical questions such as whether there exist arbitrarily large odd numbers defined as Church numerals.

We present results of our benchmarks in Table 1. They were performed on a laptop with Intel Core i5-7600K, 16GB RAM. We consider these results a success, as only two benchmarks on practical data failed to compute within ten minutes. As InfSat is the first efficient algorithm solving the simultaneous unboundedness problem, we do not have any data to which we could compare our benchmark results. At this stage, we can assess effectiveness of our optimizations. We can see that using all optimization flags produced good results consistently, however, each optimization happened to slow down some benchmarks. The reason is that optimizations turned off by `-noftty` and `-nofntty` add substantial polynomial overhead when generating list of possible environments and optimization turned off by `-nohvo` changes the way types of some terms are computed. However, all of them can exponentially reduce the computation time in many cases which is why they are turned on by default.

7 Multi-letter case

It is not difficult to modify the type system to handle simultaneous unboundedness. In this part, instead of a single important constant a of arity 1, we consider important constants $\mathsf{a}_1, \ldots, \mathsf{a}_s$, all of arity 1 ($\Sigma_{\mathsf{a}}$ is the set containing them all). Besides them, in the signature Σ we have a constant b of arity 2, and constants c and ω of arity 0.

In the type system instead of a single productivity flag in $\{\mathsf{pr}, \mathsf{np}\}$, we have a productivity set, being a subset of Σ_{a}, and saying which important letters are produced. Likewise, instead of a productivity value in $\mathbb{N}$, we have a productivity function $v\colon \Sigma_{\mathsf{a}} \to \mathbb{N}$, specifying a value separately for every letter. The rules of the type system are adopted in the expected way: $v(\mathsf{a}_i)$ increases when we use constant a_i, and when we duplicate a type binding for a type pair (A, σ) with $\mathsf{a}_i \in A$.

One more change is, however, necessary. Indeed, while proving Lemma 5.2 in the multi-letter case, we cannot choose a single subderivation concerning L that has the greatest value, since now a value is not a number, but rather a function. Instead, for every constant a_i we can choose a subderivation concerning L for which the i-th coordinate of the value is the greatest. We thus need to allow s (i.e., $|\Sigma_{\mathsf{a}}|$) subderivations for every type pair.

To this end, on the left of the arrow in a type, we do not have a set of types, but rather a multiset of types, where we allow to have at most s copies of every type. Likewise, in the type environment we have at most s copies of every type binding.

Formally, an *s-multiset* is a multiset that contains at most s copies of every element. The set of s-multisets of elements of X is denoted $\mathcal{P}_{\leq s}(X)$. Notice that 1-multisets are just sets, hence $\mathcal{P}_{\leq 1}(X) = \mathcal{P}(X)$. A union of s-multisets U, V, denoted $U \cup V$, is defined as follows: if U and V contain, respectively, n and m copies of an element x, then $U \cup V$ contains $\min(n+m, s)$ copies of this element. We use the notation $\{x_i \mid i \in I\}$ for $\bigcup_{i\in I}\{x_i\}$, where $\{x_i\}$ is the multiset containing x_i once.

For each sort α we define the set $\mathcal{T}_s^\alpha$ of types of sort α as follows:

$$\mathcal{T}_s^\circ = \{\mathsf{r}\}, \qquad \mathcal{T}_s^{\alpha\to\beta} = \mathcal{P}_{\leq s}(\mathcal{P}(\Sigma_{\mathsf{a}}) \times \mathcal{T}_s^\alpha) \times \mathcal{T}_s^\beta.$$

We again use the notation with $\to$ and $\bigwedge$, but this time while writing $\bigwedge_{i\in I}(f_i, \tau_i) \to \tau$, we assume that any pair occurs as (f_i, τ_i) at most s times.

A *type judgment* is of the form $\Gamma \vdash M : (v, \tau)$, where $v\colon \Sigma_{\mathsf{a}} \to \mathbb{N}$, and where we require that the type τ and the lambda-term M are of the same sort. The *type environment* Γ is an s-multiset of bindings of variables of the form $x^\alpha : (A, \tau)$, where $A \subseteq \Sigma_{\mathsf{a}}$ is a *productivity set* and $\tau \in \mathcal{T}_s^\alpha$ is a type. For $a \in \Sigma_{\mathsf{a}}$ by $\Gamma\restriction_a$ we denote the s-multiset of those binding from Γ that have a in their productivity set.

By $\mathbf{0}$ we denote the function from Σ_{a} to $\mathbb{N}$ that maps every a_i to 0, and by χ_i the function that maps a_i to 1 and all other a_j to 0. The type system consists of the following rules:

$$\emptyset \vdash \mathsf{a}_i : (\chi_i, (A, \mathsf{r}) \to \mathsf{r}) \qquad \emptyset \vdash \mathsf{c} : (\mathbf{0}, \mathsf{r})$$

$$\emptyset \vdash \mathsf{b} : (\mathbf{0}, (A, \mathsf{r}) \to \top \to \mathsf{r}) \qquad \emptyset \vdash \mathsf{b} : (\mathbf{0}, \top \to (A, \mathsf{r}) \to \mathsf{r}) \qquad x : (A, \tau) \vdash x : (\mathbf{0}, \tau)$$

$$\frac{\Gamma \cup \{x : (A_i, \tau_i) \mid i \in I\} \vdash K : (v, \tau) \qquad x \notin \mathit{dom}(\Gamma)}{\Gamma \vdash \lambda x.K : (v, \bigwedge_{i\in I}(A_i, \tau_i) \to \tau)}\ (\lambda)$$

We define the duplication factor $dupl((\Gamma_j)_{j\in J})$, as the function (from Σ_{a} to $\mathbb{N}$) that maps every important constant $a \in \Sigma_{\mathsf{a}}$ to $\sum_{j\in J} |\Gamma_j\restriction_a| - \left|\bigcup_{j\in J} \Gamma_j\restriction_a\right|$.

$$\frac{\begin{array}{cc} 0 \notin I & \forall i \in I.\ A_i = \{a \in \Sigma_{\mathsf{a}} \mid v_i(a) > 0 \vee \Gamma_i\restriction_a \neq \emptyset\} \\ \Gamma_0 \vdash K : (v_0, \bigwedge_{i\in I}(A_i, \tau_i) \to \tau) & \Gamma_i \vdash L : (v_i, \tau_i) \text{ for each } i \in I \end{array}}{\bigcup_{i\in\{0\}\cup I} \Gamma_i \vdash K\ L : (dupl((\Gamma_i)_{i\in\{0\}\cup I}) + \sum_{i\in\{0\}\cup I} v_i, \tau)}\ (@)$$

Recall that this time any pair may occur as (f_i, τ_i) at most s times.

References

1. Alfred V. Aho. Indexed grammars—an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.
2. William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, 5(1), 2009. doi:10.2168/LMCS-5(1:3)2009.
3. Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.
4. Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.
5. Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.
6. Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.163.
7. Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.
8. Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017. doi:10.23638/DMTCS-19-4-1.
9. Werner Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.
10. Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.
11. Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. *ACM Trans. Comput. Log.*, 18(3):25:1–25:42, 2017. doi:10.1145/3091122.

12 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

13 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

14 Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011. doi:10.2168/LMCS-7(4:9)2011.

15 Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In Mary W. Hall and David A. Padua, editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 222–233. ACM, 2011. doi:10.1145/1993498.1993525.

16 Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015. doi:10.1016/j.ic.2014.12.015.

17 Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. General decidability results for asynchronous shared-memory programs: Higher-order and beyond. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 449–467. Springer, 2021. doi:10.1007/978-3-030-72016-2_24.

18 Robin P. Neatherway and C.-H. Luke Ong. TravMC2: higher-order model checking for alternating parity tree automata. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 129–132. ACM, 2014. doi:10.1145/2632362.2632381.

19 Robin P. Neatherway, Steven J. Ramsay, and C.-H. Luke Ong. A traversal-based algorithm for higher-order model checking. In Peter Thiemann and Robby Bruce Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 353–364. ACM, 2012. doi:10.1145/2364527.2364578.

20 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

21 Paweł Parys. A characterization of lambda-terms transforming numerals. *J. Funct. Program.*, 26:e12, 2016. doi:10.1017/S0956796816000113.

22 Paweł Parys. Intersection types and counting. In Naoki Kobayashi, editor, *Proceedings Eighth Workshop on Intersection Types and Related Systems, ITRS 2016, Porto, Portugal, 26th June 2016.*, volume 242 of *EPTCS*, pages 48–63, 2016. doi:10.4204/EPTCS.242.6.

23 Paweł Parys. Homogeneity without loss of generality. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSCD.2018.27.

24 Paweł Parys. Intersection types for unboundedness problems. In Michele Pagani and Sandra Alves, editors, *Proceedings Twelfth Workshop on Developments in Computational Models and Ninth Workshop on Intersection Types and Related Systems, DCM/ITRS 2018, Oxford, UK, 8th July 2018*, volume 293 of *EPTCS*, pages 7–27, 2018. doi:10.4204/EPTCS.293.2.

25 Paweł Parys. Recursion schemes, the MSO logic, and the U quantifier. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:20)2020.

26 Paweł Parys. A type system describing unboundedness. *Discret. Math. Theor. Comput. Sci.*, 22(4), 2020. doi:10.23638/DMTCS-22-4-2.

27 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Mathematical Structures in Computer Science*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

28 Hiroshi Unno, Naoshi Tabuchi, and Naoki Kobayashi. Verification of tree-processing programs via higher-order mode checking. *Mathematical Structures in Computer Science*, 25(4):841–866, 2015. doi:10.1017/S0960129513000054.

29 Georg Zetzsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2015. doi:10.1007/978-3-662-47666-6_35.

Parameterized Safety Verification of Round-Based Shared-Memory Systems

Nathalie Bertrand
Univ Rennes, Inria, CNRS, IRISA, France

Nicolas Markey
Univ Rennes, Inria, CNRS, IRISA, France

Ocan Sankur
Univ Rennes, Inria, CNRS, IRISA, France

Nicolas Waldburger
Univ Rennes, Inria, CNRS, IRISA, France

Abstract

We consider the parameterized verification problem for distributed algorithms where the goal is to develop techniques to prove the correctness of a given algorithm regardless of the number of participating processes. Motivated by an asynchronous binary consensus algorithm [3], we consider round-based distributed algorithms communicating with shared memory. A particular challenge in these systems is that 1) the number of processes is unbounded, and, more importantly, 2) there is a fresh set of registers at each round. A verification algorithm thus needs to manage both sources of infinity. In this setting, we prove that the safety verification problem, which consists in deciding whether all possible executions avoid a given error state, is PSPACE-complete. For negative instances of the safety verification problem, we also provide exponential lower and upper bounds on the minimal number of processes needed for an error execution and on the minimal round on which the error state can be covered.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Distributed algorithms

Keywords and phrases Verification, Parameterized models, Distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.113

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2204.11670

1 Introduction

Distributed algorithms received in the last decade a lot of attention from the automated verification community. Parameterized verification emerged as a subfield that specifically addresses the verification of distributed algorithms. The main challenge is that distributed algorithms should be proven correct for any number or participating processes. Parameterized models are thus infinite by nature and parameterized verification is in general unfeasible [2]. However, one can recover decidability by considering specific classes of parameterized models, as in the seminal work by German and Sistla where identical finite state machines interact via rendezvous communications [14]. Since then, various models have been proposed to handle various communication means (see [11, 7] for surveys).

Shared memory is one possible communication means. This paper makes first steps towards the parameterized verification of *round-based* distributed algorithms in the shared-memory model; examples of such algorithms can be found in [4, 3, 16]. In particular, our approach covers Aspnes' consensus algorithm [3] which we take as a motivating example. Shared-memory models *without rounds* have been considered in the literature: the verification

of safety properties for systems with a leader and many anonymous contributors interacting via a single shared register is coNP-complete [12, 13]; and for Büchi properties, it is NP-complete [10]. Randomized schedulers have also been considered for shared-memory models without leaders; the verification of almost-sure coverability is in EXPSPACE, and is PSPACE-hard [9]. Finally, safety verification is PSPACE-complete for so-called distributed memory automata, that combine local and global memory [8].

Round-based algorithms make verification particularly challenging since they use fresh copies of the registers at each round, and an unbounded number of asynchronous processes means that verification must handle a system with an unbounded number of registers. This is why existing verification techniques fall short at analyzing such algorithms combining two sources of infinity: an unbounded number of processes, and an unbounded number of rounds (hence of registers).

Algorithm 1 Aspnes' consensus algorithm [3].

```
1  int k := 0, bool p ∈ {0,1}, (rg_b[r])_{b∈{b0,b1},r∈ℕ} all initialized to ⊥;
2  while true do
3      read from rg_{b0}[k] and rg_{b1}[k];
4      if rg_{b0}[k] = ⊤ and rg_{b1}[k] = ⊥ then p := 0;
5      else if rg_{b0}[k] = ⊥ and rg_{b1}[k] = ⊤ then p := 1;
6      write ⊤ to rg_{bp}[k];
7      if k > 0 then
8          read from rg_{b1−p}[k−1];
9          if rg_{b1−p}[k−1] = ⊥ then return p;
10     k := k+1;
```

Algorithm 1 gives the pseudocode of the binary consensus algorithm proposed by Aspnes [3], in which the processes communicate through shared registers. The algorithm proceeds in asynchronous rounds, which means that there is no *a priori* bound on the round difference between pairs of processes. Furthermore, reading from and writing to registers are separate operations, and a sequence of a read and a write cannot be performed atomically. Each round r has two shared registers $\mathsf{rg}_{b_i}[r]$ for $i \in \{0,1\}$; notation b_i is used in register indices to avoid confusion with other occurrences of digits 0 and 1. All registers are initialized to a default value $\bot$, and within an execution, their value may only be updated to $\top$. Intuitively, $\mathsf{rg}_{b_i}[r] = \top$ if i is the proposed consensus value at round r.

As usual in distributed consensus algorithms, each process starts with a preference value p. At each round, a process starts by reading the value of the shared registers of that round (**Line 3**). If exactly one of them is set to $\top$, the process updates its preference p to the corresponding value (**Lines 4** and **5**). In all cases, it writes $\top$ to the current-round register that corresponds to its preference p (**Line 6**). Then, it reads the register of the previous round corresponding to the opposite preference $1-p$ (**Line 8**), and if it is $\bot$, the process decides its preference p as return value for the consensus (**Line 9**). To be able to decide its current preference value, a process thus has to win a race against others, writing to a register of its current round k while no other process has written to the register of round $k-1$ for the opposite value. Note that a process can read from and write to the registers of its current round, whereas the registers of the previous rounds are read-only.

The expected properties of such a distributed consensus algorithm are *validity*, *agreement* and *termination*. Validity expresses that if all processes start with the same preference p, then no process can return a value different from p. Agreement expresses that no two processes

can return different values. Finally, termination expresses that eventually all processes should return a value. The termination of Aspnes' algorithm is only guaranteed under some fairness constraints on the adversary that schedules the moves of processes [3]. Its validity and agreement properties hold unconditionally. Our objective is to develop automated verification techniques for safety properties, which include validity and agreement.

For a single round –corresponding to one iteration of the while loop– safety properties can be proved applying techniques from [12, 13]. The additional difficulty here lies in the presence of unboundedly many rounds and thus of unboundedly many shared registers. Other settings of parameterized verification exist for round-based distributed algorithms, but none of them apply to asynchronous shared-memory distributed algorithms: they either concern fault-tolerant threshold-based algorithms [5, 6], or synchronous distributed algorithms [15, 1].

Contributions

In this paper, we introduce round-based register protocols, a formalism that models round-based algorithms in which processes communicate via shared memory. Figure 1 depicts a representation of Aspnes' algorithm in this formalism.

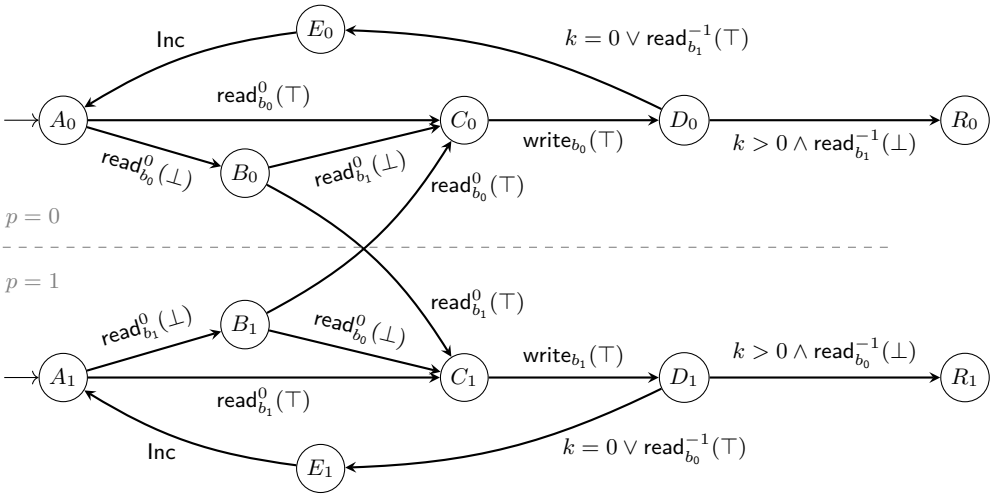


Figure 1 A round-based register protocol for Aspnes' noisy consensus algorithm. Since the first round ($k = 0$) slightly differs from the others, to avoid duplication of the state space, we allow for guards on round number k in the transition labels.

Round-based register protocols form a class of models inspired by register protocols [12, 9, 13], which were introduced to represent shared-memory distributed algorithms *without rounds*. In register protocols, states typically represent the control point of each process as well as the value of its private variables. For instance, the preference p of the process is encoded in the state space: in the top part, $p = 0$ and in the bottom part $p = 1$, as reflected by the states indices. To allow for multiple rounds and round increments, as in Line **10**, we extend register protocols with a new action Inc that labels the transitions from state E_p to state A_p, for each preference $p \in \{0, 1\}$. The processes may read from the registers of the current round but also from those of previous rounds, so reads must specify not only the register identifier but also the lookback distance to the current round: for a process in round k, $\mathsf{read}_{b_p}^{-d}(x)$ represents reading value x from register $\mathsf{rg}_{b_p}[k-d]$.

The validity and agreement properties translate as follows on the register protocols. For validity, one needs to check two properties, one for each common preference $p \in \{0, 1\}$. Namely, if all processes start in state A_0 (resp. A_1), then no processes can enter state R_0

(resp. R_1). Agreement requires that, independently from the initial state of each process in $\{A_0, A_1\}$, no executions reach a configuration with at least one process in R_0 and at least one process in R_1. Both validity and agreement are safety properties.

After introducing round-based register protocols, we study the parameterized verification of safety properties, with the objective of automatically checking whether a configuration involving an error state can be covered for arbitrarily many processes. Our main result is the PSPACE-completeness of this verification problem. We develop an algorithm exploiting the fact that the processes may only read the values of registers within a bounded window on rounds. However, a naive algorithm focusing on the v latest rounds only is hopeless: perhaps surprisingly, we show that the number of *active* rounds (i.e., rounds where a non-idle process is in) may need to be as large as exponential to find an execution covering an error state. The cutoff i.e., the minimal number of processes needed to cover an error state, may also be exponential. The design of our polynomial space algorithm addresses these difficulties by carefully tracking *first-write orders*, that is, the order in which registers are written to for the first time. One of the main technical difficulties of the algorithm is making sure that enough information is stored in this way, allowing the algorithm to solve the verification problem, while also staying in polynomial space.

The rest of the paper is structured as follows. To address the verification of safety properties for round-based register protocols, after introducing their syntax and semantics (Section 2.1), we first observe that they enjoy a monotonicity property (Section 2.2), which justifies the definition of a sound and complete abstract semantics (Section 2.3). We then highlight difficulties of coming up with a polynomial space decision procedure (Section 3.1). Namely, we provide exponential lower bounds on (1) the minimal round number, (2) the minimal number of processes, and (3) the minimal number of active rounds in error executions. We then introduce the central notion of *first-write orders* and its properties (Section 3.2). Section 3.3 details our polynomial-space algorithm, and Section 3.4 presents the complexity-matching lower bound.

2 Round-based shared-memory systems

2.1 Register protocols with rounds

▶ **Definition 1** (Round-based register protocols). *A* round-based register protocol *is a tuple* $\mathcal{P} = \langle Q, q_0, \mathsf{d}, \mathrm{D}, \mathsf{v}, \Delta \rangle$ *where*

- Q *is a finite set of states with a distinguished initial state* q_0*;*
- $\mathsf{d} \in \mathbb{N}$ *is the number of shared registers per round;*
- D *is a finite data alphabet containing* d_0 *the initial value and* $\mathrm{D} \setminus \{d_0\}$ *the values that can be written to the registers;*
- v *is the visibility range (a process on round* k *may read only from rounds in* $[k - \mathsf{v}, k]$*);*
- $\Delta \subseteq Q \times \mathcal{A} \times Q$ *is the set of transitions, where* $\mathcal{A} = \{\mathsf{Inc}\} \cup \{\mathsf{read}_\alpha^{-i}(x) \mid i \in [0, \mathsf{v}], \alpha \in [1, \mathsf{d}], x \in \mathrm{D}\} \cup \{\mathsf{write}_\alpha(x) \mid \alpha \in [1, \mathsf{d}], x \in \mathrm{D} \setminus \{d_0\}\}$ *is the set of actions.*

Intuitively, in a round-based register protocol, the behavior of a process is described by a finite-state machine with a local variable k representing its current round number; note that each process has its own round number, as processes are asynchronous and can be on different rounds. Moreover, there are d registers per round, and the transitions can read and modify these registers. Transitions in round-based register protocols can be labeled with three different types of actions: the Inc action simply increments the current round number

of the process; action $\mathsf{read}_\alpha^{-i}(x)$ can be performed by a process at round k when the value of register α of round $k-i$ is x; finally, with the action $\mathsf{write}_\alpha(x)$, a process at round k writes value x to the register α of round k. Note that all actions $\mathsf{read}_\alpha^{-i}(x)$ must satisfy $i \leq \mathsf{v}$; in other words, processes of round k can only read values of registers of rounds $k - \mathsf{v}$ to k.

For complexity purposes, we define the size of the protocol $\mathcal{P} = \langle Q, q_0, \mathsf{d}, \mathrm{D}, \mathsf{v}, \Delta\rangle$ as $|\mathcal{P}| = |Q| + |\mathrm{D}| + |\Delta| + \mathsf{v} + \mathsf{d}$ (thus implicitly assuming that v is given in unary).

Before defining the semantics of round-based register protocols, let us introduce some useful notations. For round number k, we write $\mathsf{rg}_\alpha[k]$ the register α of round k, we let $\mathsf{Reg}_k = \{\mathsf{rg}_\alpha[k] \mid \alpha \in [1, \mathsf{d}]\}$ denote the set of registers of round k, and $\mathsf{Reg} = \bigcup_{k\in\mathbb{N}} \mathsf{Reg}_k$ the set of all registers.

Round-based register protocols execute on several processes asynchronously. The processes communicate via the shared registers, and they progress in a fully asynchronous way through the rounds. A *location* $(q, k) \in Q \times \mathbb{N}$ describes the current state q and round number k of a process, and $\mathsf{Loc} = Q \times \mathbb{N}$ is the set of all locations. A configuration intuitively describes the location of each process, as well as the value of each register. Since processes are anonymous and indistinguishable, the locations of all processes can be represented by maps $\mathsf{Loc} \to \mathbb{N}$ describing how many processes populate each location. Formally, a *concrete configuration* is a pair $\gamma = (\mu, d) \in \mathbb{N}^{\mathsf{Loc}} \times \mathrm{D}^{\mathsf{Reg}}$ such that $\sum_{(q,k)\in\mathsf{Loc}} \mu(q, k) < \infty$. We write $\Gamma = \mathbb{N}^{\mathsf{Loc}} \times \mathrm{D}^{\mathsf{Reg}}$ for the set of all concrete configurations. For a concrete configuration $\gamma = (\mu, d)$, the location multiset μ is denoted $\mathsf{loc}(\gamma)$ and the value $d(k)(\alpha)$ of register α at round k in γ is written $\mathsf{data}_{\mathsf{rg}_\alpha[k]}(\gamma)$. The *size* of γ corresponds to the number of involved processes: $|\gamma| = \sum_{(q,k)\in\mathsf{Loc}} \mu(q, k)$. Configuration γ is *initial* if for every $(q, k) \neq (q_0, 0)$, $\mathsf{loc}(\gamma)(q, k) = 0$, and for every register ξ, $\mathsf{data}_\xi(\gamma) = d_0$. The set of initial concrete configurations therefore consists of all $\mathsf{init}_n = ((q_0, 0)^n, d_0^{\mathsf{Reg}})$. A register is *blank* when it still has initial value d_0. The *support* of the multiset $\mathsf{loc}(\gamma)$ is $\mathsf{supp}(\gamma) = \{(q, k) \mid \mathsf{loc}(\gamma)(q, k) > 0\}$. Finally, for $\gamma, \gamma' \in \Gamma$, we write $\mathsf{data}(\gamma) = \mathsf{data}(\gamma')$ whenever for all $\xi \in \mathsf{Reg}$, $\mathsf{data}_\xi(\gamma) = \mathsf{data}_\xi(\gamma')$.

The evolution from a concrete configuration to another reflects the effect of a process taking a transition in the register protocol. A *move* is thus an element $\theta = (\delta, k)$ consisting of a transition $\delta \in \Delta$ and a round number k; $\mathsf{Moves} = \Delta \times \mathbb{N}$ is the set of all moves. For two concrete configurations γ, γ', we say that γ' is a *successor* of γ if there is a move $((q, a, q'), k) \in \mathsf{Moves}$ satisfying one of the following conditions, depending on the action type:

- **(i)** $a = \mathsf{Inc}$, $\mathsf{loc}(\gamma)(q, k) > 0$, $\mathsf{loc}(\gamma') = \mathsf{loc}(\gamma) \ominus (q, k) \oplus (q', k{+}1)$, and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$;
- **(ii)** $a = \mathsf{read}_\alpha^{-i}(x)$ with $x \in \mathrm{D}$, $\mathsf{data}_{\mathsf{rg}_\alpha[k-i]}(\gamma) = x$, $\mathsf{loc}(\gamma)(q, k) > 0$, $\mathsf{loc}(\gamma') = \mathsf{loc}(\gamma) \ominus (q, k) \oplus (q', k)$ and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$;
- **(iii)** $a = \mathsf{write}_\alpha(x)$ with $x \in \mathrm{D} \setminus \{d_0\}$, $\mathsf{data}_{\mathsf{rg}_\alpha[k]}(\gamma') = x$, $\mathsf{loc}(\gamma)(q, k) > 0$, $\mathsf{loc}(\gamma') = \mathsf{loc}(\gamma) \ominus (q, k) \oplus (q', k)$ and for all $\xi \in \mathsf{Reg} \setminus \{\mathsf{rg}_\alpha[k]\}$, $\mathsf{data}_\xi(\gamma') = \mathsf{data}_\xi(\gamma)$.

Here, $\oplus$ and $\ominus$ are operations on multisets, respectively adding and removing elements. The first case represents round increment for a process and the register values are unchanged. The second case represents a read: it requires that the correct value is stored in the corresponding register, that the involved process moves, and that the register values are unchanged. By convention, here, if $k - i < 0$, i.e., for registers with negative round numbers, we let $\mathsf{data}_{\mathsf{rg}_\alpha[k-i]}(\gamma) = d_0$. Finally, the last case represents a write action; it only affects the corresponding register, and the state of the involved process. Note that in all cases, $|\gamma| = |\gamma'|$: the number of processes is constant. If γ' is a successor of γ by move θ, we write $\gamma \xrightarrow{\theta} \gamma'$. A *concrete execution* is an alternating sequence $\gamma_0, \theta_1, \gamma_1, \ldots, \gamma_{\ell-1}, \theta_\ell, \gamma_\ell$ of concrete configurations and moves such that for all i, $\gamma_i \xrightarrow{\theta_{i+1}} \gamma_{i+1}$. In such a case, we write $\gamma_0 \xrightarrow{*} \gamma_\ell$, and we say that γ_ℓ is reachable from γ_0. A location (q, k) is *coverable from* γ_0 when there exists $\gamma \in \mathsf{Reach}(\gamma_0)$ such that $(q, k) \in \mathsf{loc}(\gamma_0)$, and similarly a state q is *coverable from* γ_0 when there exist $k \in \mathbb{N}$ such that (q, k) is coverable from γ_0.

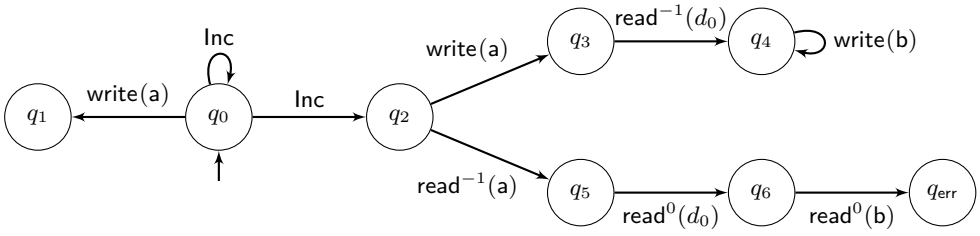


Figure 2 A simple round-based register protocol.

Given a concrete configuration $\gamma \in \Gamma$, $\mathsf{Reach}_{\mathsf{c}}(\gamma)$ denotes the set of all configurations that can be reached from γ: $\mathsf{Reach}_{\mathsf{c}}(\gamma) = \{\gamma' \mid \gamma \xrightarrow{*} \gamma'\}$.

We are now in a position to define our problem of interest:

SAFETY PROBLEM FOR ROUND-BASED REGISTER PROTOCOLS
Input: A round-based register protocol $\mathcal{P} = \langle Q, q_0, \mathsf{d}, \mathrm{D}, \mathsf{v}, \Delta\rangle$ and a state $q_{\mathsf{err}} \in Q$
Question: Is it the case that for every $n \in \mathbb{N}$, for every $\gamma \in \mathsf{Reach}_{\mathsf{c}}(\mathsf{init}_n)$ and for every round number k, $\mathsf{loc}(\gamma)(q_{\mathsf{err}}, k) = 0$?

The state q_{err} is referred to as an *error state* that all executions should avoid. An *error configuration* is a configuration in which the error state q_{err} appears, and an *error execution* is an execution containing an error configuration. Given a protocol $\mathcal{P}$ and a state q_{err}, in order to check whether $(\mathcal{P}, q_{\mathsf{err}})$ is a positive instance of the safety problem, we will look for an error execution, and therefore check the dual problem: whether there exist a size n and a configuration $\gamma \in \mathsf{Reach}_{\mathsf{c}}(\mathsf{init}_n)$ such that for some round number k, $\mathsf{loc}(\gamma)(q_{\mathsf{err}}, k) > 0$.

▶ **Example 2.** We illustrate round-based register protocols and their safety problem on the model depicted in Figure 2. This protocol has a single register per round ($\mathsf{d} = 1$, and the register identifier is thus omitted), and set of symbols $\mathrm{D} = \{d_0, a, b\}$. Let us give two examples of concrete executions. State q_4 is coverable from init_1 with the sequence of moves:

$$\pi_1 = \big(\langle(q_0,0)\rangle, \substack{\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=d_0}\big) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0} \big(\langle(q_2,1)\rangle, \substack{\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=d_0}\big) \xrightarrow{\langle q_2,\mathsf{write}(a),q_3\rangle,1} \big(\langle(q_3,1)\rangle, \substack{\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=a}\big)$$
$$\xrightarrow{\langle q_3,\mathsf{read}^{-1}(d_0),q_4\rangle,1} \big(\langle(q_4,1)\rangle, \substack{\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=a}\big).$$

State q_6 is coverable from init_2 as witnessed by the concrete execution:

$$\pi_2 = \big(\langle(q_0,0),(q_0,0)\rangle, \substack{\mathsf{rg}[0]=d_0\\ \mathsf{rg}[1]=d_0}\big) \xrightarrow{\langle q_0,\mathsf{write}(a),q_1\rangle,0} \big(\langle(q_0,0),(q_1,0)\rangle, \substack{\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0}\big) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0}$$
$$\big(\langle(q_2,1),(q_1,0)\rangle, \substack{\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0}\big) \xrightarrow{\langle q_2,\mathsf{read}^{-1}(a),q_5\rangle,1} \big(\langle(q_5,1),(q_1,0)\rangle, \substack{\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0}\big) \xrightarrow{\langle q_5,\mathsf{read}^{0}(d_0),q_6\rangle,1}$$
$$\big(\langle(q_6,1),(q_1,0)\rangle, \substack{\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=d_0}\big).$$

However, it can be observed that no concrete execution can cover both states *at the same round* whatever the number of processes, thus preventing from covering q_{err}. We justify this observation in Subsection 3.2. This example is a positive instance of the safety problem. ⌟

▶ **Example 3.** The validity of Aspnes' algorithm can be expressed as two safety properties, with A_0 (resp. A_1) as initial state, and R_1 (resp. R_0) as error state. Let us argue that the protocol of Figure 1 is safe for $q_0 = A_0$ and $q_{\mathsf{err}} = R_1$; the other case is symmetric. Towards a contradiction, suppose there exists an execution $\pi : \mathsf{init}_n \xrightarrow{*} \gamma_1 \xrightarrow{\theta} \gamma_2 \xrightarrow{*} \gamma$ where

γ_2 contains a process in the bottom part, and γ_2 is the first such configuration along π. Then $\theta = ((B_0, \mathsf{read}^0_{b_1}(\top), C_1), k)$ for some k, thus implying that $\mathsf{data}_{\mathsf{rg}_{b_1}[k]}(\gamma_1) = \top$. However, b_1 can only be written to $\mathsf{rg}_{b_1}[k]$ by a process already in the bottom part, which contradicts the minimality of γ_2.

To formally encode agreement of Aspnes' algorithm as a safety property, we make two slight modifications to the protocol from Figure 1. We add an extra initial state q_0 with silent outgoing transitions to A_0 and to A_1; we also add an error state q_{err} that can be covered only if R_0 and R_1 are covered in a same execution. To do so, one can mimick the gadget at q_4 and q_6 in Figure 2, using an extra letter $b \in \mathrm{D}$ and adding Inc loops on both R_0 and R_1, allowing processes to synchronize on the same round, before writing and reading b.

Checking validity and agreement automatically for Aspnes' algorithm requires the machinery that we develop in the rest of the paper. ⌟

2.2 Monotonicity

Similarly to other parameterized models, and specifically shared-memory systems [13, 9], round-based register protocols enjoy a monotonicity property called the copycat property. Intuitively, this property states that if a location can be populated with one process, then, increasing the size of the initial configuration, it can be populated by an arbitrary number of them without affecting the behaviour of the other processes. Formally:

▶ **Lemma 4** (Copycat property). *Let $q \in Q$, $k, n, N \in \mathbb{N}$ and $\gamma_{\mathsf{i}}, \gamma_{\mathsf{f}} \in \Gamma$ such that $\gamma_{\mathsf{f}} \in \mathsf{Reach}_{\mathsf{c}}(\gamma_{\mathsf{i}})$ and $(q, k) \in \mathsf{supp}(\gamma_{\mathsf{f}})$. Then there exist $\gamma'_{\mathsf{i}}, \gamma'_{\mathsf{f}} \in \Gamma$ such that $\gamma'_{\mathsf{f}} \in \mathsf{Reach}_{\mathsf{c}}(\gamma'_{\mathsf{i}})$ and:*

- $|\gamma'_{\mathsf{i}}| = |\gamma_{\mathsf{i}}| + N$, $\mathsf{supp}(\gamma'_{\mathsf{i}}) = \mathsf{supp}(\gamma_{\mathsf{i}})$, *and* $\mathsf{data}(\gamma'_{\mathsf{i}}) = \mathsf{data}(\gamma_{\mathsf{i}})$;
- $\mathsf{loc}(\gamma'_{\mathsf{f}}) = \mathsf{loc}(\gamma_{\mathsf{f}}) \oplus (q, k)^N$ *and* $\mathsf{data}(\gamma'_{\mathsf{f}}) = \mathsf{data}(\gamma_{\mathsf{f}})$.

The copycat property strongly relies on the fact that operations on the registers are non-atomic. In particular it is crucial that processes cannot atomically read and write to a given register, since that could prevent another process from copycating its behaviour.

By the copycat property, the existence of an execution covering the error state q_{err} implies the existence of similar executions for any larger number of processes, which motivates the notion of cutoff. Formally, given $(\mathcal{P}, q_{\mathsf{err}})$ a negative instance of the safety problem, the *cutoff* is the least $n_0 \in \mathbb{N}$ such that for every $n \geq n_0$ there exist $\gamma_n \in \mathsf{Reach}_{\mathsf{c}}(\mathsf{init}_n)$ and $k_n \in \mathbb{N}$ with $\mathsf{loc}(\gamma_n)(q_{\mathsf{err}}, k_n) > 0$.

Another consequence is that any value that has been written to a register can be rewritten, at the cost of increasing the number of involved processes.

▶ **Corollary 5.** *Let $n \in \mathbb{N}$, $\pi : \mathsf{init}_n \xrightarrow{*} \gamma_1 \xrightarrow{*} \gamma$ a concrete execution and $\xi \in \mathsf{Reg}$ a register such that $\mathsf{data}_\xi(\gamma_1) \neq d_0$. There exist $n' \geq n$ and a concrete execution $\pi' : \mathsf{init}_{n'} \xrightarrow{*} \gamma'$ such that $\mathsf{loc}(\gamma) \subseteq \mathsf{loc}(\gamma')$, $\mathsf{data}_\xi(\gamma') = \mathsf{data}_\xi(\gamma_1)$ and for all $\xi' \neq \xi$, $\mathsf{data}_{\xi'}(\gamma') = \mathsf{data}_{\xi'}(\gamma)$.*

2.3 Abstract semantics

The copycat property suggests that, for existential coverability properties, the precise number of processes populating a location is not relevant, only the support of the location multiset matters. As for registers, the only important information to remember is whether they still contain the initial value, or they have been written to (the support then suffices to deduce which values can be written and read). In this section, we therefore define an abstract semantics for round-based register protocols, and we prove it to be sound and complete for the safety problem.

Formally, an *abstract configuration*, or simply a *configuration*, is a pair $\sigma \in 2^{\mathsf{Loc}} \times 2^{\mathsf{Reg}}$, with location support $\mathsf{loc}(\sigma) \in 2^{\mathsf{Loc}}$ and set of written registers $\mathsf{FW}(\sigma) \in 2^{\mathsf{Reg}}$. We write Σ for the set $2^{\mathsf{Loc}} \times 2^{\mathsf{Reg}}$ of all configurations. The (unique) *initial* configuration is $\sigma_{\mathsf{init}} = (\{(q_0, 0)\}, \emptyset)$. Configuration σ' is a successor of configuration σ if there exists a move $\theta = ((q, a, q'), k) \in \mathsf{Moves}$ such that one of the following conditions holds:

- **(i)** $a = \mathsf{Inc}$, $(q, k) \in \mathsf{loc}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q', k{+}1)\}$, and $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma)$;
- **(ii)** $a = \mathsf{read}_\alpha^{-i}(x)$ with $x \neq d_0$, $(q, k) \in \mathsf{loc}(\sigma)$, $\mathsf{rg}_\alpha[k{-}i] \in \mathsf{FW}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q', k)\}$, $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma)$ and there is a transition $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$ with $(q_1, k{-}i), (q_2, k{-}i) \in \mathsf{loc}(\sigma)$;
- **(iii)** $a = \mathsf{read}_\alpha^{-i}(d_0)$, $(q, k) \in \mathsf{loc}(\sigma)$, $\mathsf{rg}_\alpha[k{-}i] \notin \mathsf{FW}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q', k)\}$ and $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma)$;
- **(iv)** $a = \mathsf{write}_\alpha(x)$ with $x \neq d_0$, $(q, k) \in \mathsf{loc}(\sigma)$, $\mathsf{loc}(\sigma') = \mathsf{loc}(\sigma) \cup \{(q', k)\}$ and $\mathsf{FW}(\sigma') = \mathsf{FW}(\sigma) \cup \{\mathsf{rg}_\alpha[k]\}$.

In this case, we write $\sigma \xrightarrow{\theta} \sigma'$. An (abstract) *execution* is an alternating sequence of configurations and moves $\rho = \sigma_0, \theta_1, \sigma_1, \ldots, \sigma_{\ell-1}, \theta_\ell, \sigma_\ell$ such that for all i, $\sigma_i \xrightarrow{\theta_{i+1}} \sigma_{i+1}$, and we write $\sigma \xrightarrow{*} \sigma_\ell$. Similarly to the concrete semantics, $\mathsf{Reach}(\sigma) = \{\sigma' \mid \sigma \xrightarrow{*} \sigma'\}$ denotes the set of *reachable configurations from* σ. Again, a location (q, k) is *coverable from* σ when there exists $\sigma' \in \mathsf{Reach}(\sigma)$ such that $(q, k) \in \mathsf{loc}(\sigma')$, and similarly a state q is *coverable from* σ when there exist $\sigma' \in \mathsf{Reach}(\sigma)$ and $k \in \mathbb{N}$ such that $(q, k) \in \mathsf{loc}(\sigma')$. We simply say that a configuration is *reachable* if it is reachable from the initial configuration σ_{init}, and that a location (resp. a state) is *coverable* if it is coverable from the initial configuration σ_{init}.

▶ **Example 6.** Consider again the protocol of Example 2. The (abstract) execution associated with the concrete execution π_1 in this example is

$$\rho_1 = \big(\{(q_0,0)\},\emptyset\big) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0} \big(\{(q_0,0),(q_2,1)\},\emptyset\big) \xrightarrow{\langle q_2,\mathsf{write}(a),q_3\rangle,1}$$
$$\big(\{(q_0,0),(q_2,1),(q_3,1)\},\{\mathsf{rg}[1]\}\big) \xrightarrow{\langle q_3,\mathsf{read}^{-1}(d_0),q_4\rangle,1} \big(\{(q_0,0),(q_2,1),(q_3,1),(q_4,1)\},\{\mathsf{rg}[1]\}\big).$$

Similarly, the execution associated with π_2 is

$$\rho_2 = \big(\{(q_0,0)\},\emptyset\big) \xrightarrow{\langle q_0,\mathsf{write}(a),q_1\rangle,0} \big(\{(q_0,0),(q_1,0)\},\{\mathsf{rg}[0]\}\big) \xrightarrow{\langle q_0,\mathsf{Inc},q_2\rangle,0}$$
$$\big(\{(q_0,0),(q_1,0),(q_2,1)\},\{\mathsf{rg}[0]\}\big) \xrightarrow{\langle q_2,\mathsf{read}^{-1}(a),q_5\rangle,1} \big(\langle(q_0,0),(q_1,0),(q_2,1),(q_5,1)\rangle,\{\mathsf{rg}[0]\}\big)$$
$$\xrightarrow{\langle q_5,\mathsf{read}^{0}(d_0),q_6\rangle,1} \big(\langle(q_0,0),(q_1,0),(q_2,1),(q_5,1),(q_6,1)\rangle,\{\mathsf{rg}[0]\}\big).$$ ⌟

Note that, in contrast to the concrete semantics, the location support of configurations cannot decrease along an abstract execution. One can easily be convinced that any concrete execution can be lifted to an abstract one, by possibly increasing the support, which is not a problem as long as one is interested in the verification of safety properties. Conversely, from an abstract execution, for a large enough number of processes, using the copycat property one can build a concrete execution with the same final location support. Altogether, the abstract semantics is therefore sound and complete to decide the safety problem on round-based register protocols.

▶ **Theorem 7.** *Let $\mathcal{P}$ be a round-based register protocol, q_{err} a state and $k \in \mathbb{N}$. Then:*

$$\exists n \in \mathbb{N}, \exists \gamma \in \mathsf{Reach}_{\mathsf{c}}(\mathsf{init}_n) : (q_{\mathsf{err}}, k) \in \mathsf{loc}(\gamma) \iff \exists \sigma \in \mathsf{Reach}(\sigma_{\mathsf{init}}) : (q_{\mathsf{err}}, k) \in \mathsf{loc}(\sigma).$$

Moreover, for negative instances of the safety problem, the proof of Theorem 7 yields an upper bound on the cutoff, which is linear in the round number at which q_{err} is covered.

▶ **Corollary 8.** *If there exists $k \in \mathbb{N}$ such that (q_{err}, k) is coverable, then, letting $N = 2|Q|(k{+}1){+}1$, there exists $\pi : \mathsf{init}_N \xrightarrow{*} \gamma$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\gamma)$.*

3 Decidability and complexity of the safety problem

3.1 Exponential lower bounds everywhere!

To highlight the challenges in coming up with a polynomial space algorithm, we first state three exponential lower bounds when considering safety verification of round-based register protocols. Namely, we prove that (1) the minimal round are which the error state is covered, (2) the minimal number of processes needed for an error execution, and (3) the minimal number of simultaneously active rounds within an error execution, all may need to be exponential in the size of the protocol.

Exponential minimal round

▶ **Proposition 9.** *There exists a family* $(\mathcal{BC}_m)_{m\geq 1}$ *of round-based register protocols with* q_{err} *an error state, visibility range* $\mathsf{v} = 0$ *and number of registers per round* $\mathsf{d} = 1$*, such that* $|\mathcal{BC}_m| = O(m)$ *and the minimum round at which* q_{err} *can be covered is in* $\Omega(2^m)$*.*

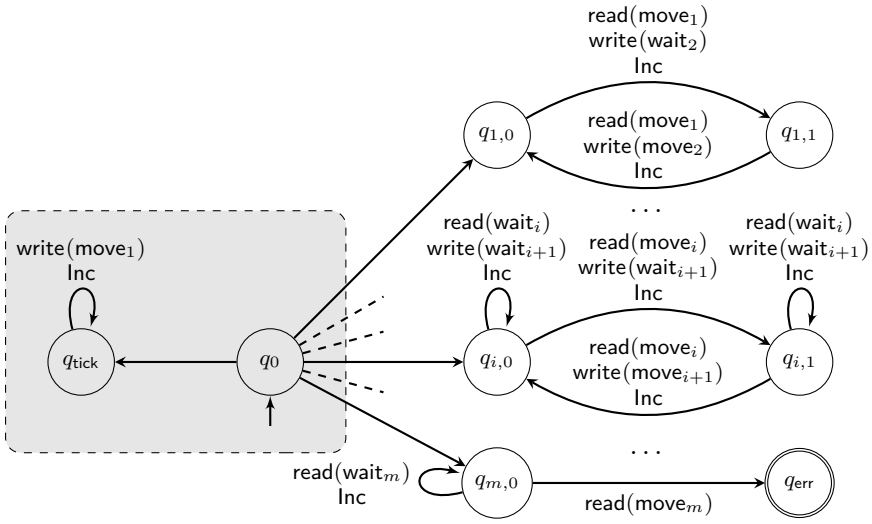


Figure 3 Protocol $\mathcal{BC}_m$ for which an exponential number of rounds is needed to cover q_{err}. For the sake of readability, transitions may be labelled by a sequence of actions: *e.g.*, the transition from $q_{i,0}$ to $q_{i,1}$ is labelled by $\mathsf{read}(\mathsf{move}_1)$, $\mathsf{write}(\mathsf{wait}_2)$, Inc. Such sequences of actions are not performed atomically: one should in principle add intermediate states to split the transition into several consecutive transitions, with one action each. We also use silent transitions (with no action label) that do not perform any action. The tick gadget in grey will be modified in subsequent figures.

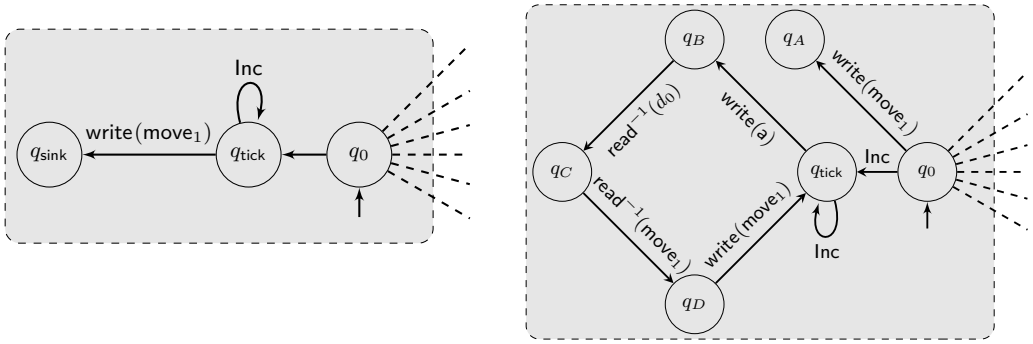
The protocol $\mathcal{BC}_m$, depicted in Figure 3, encodes a binary counter on m bits. The high-level idea of this protocol is that the counter value starts with 0 and is incremented at each round; setting the most significant bit to 1 puts a process in q_{err}. In order to cover q_{err}, any concrete execution needs at least $m+1$ processes: one in q_{tick} ticking every round, and one per bit, in states $\{q_{i,0}, q_{i,1}\}$ to represent the value of the counter's i-th bit. At round k, the value of the i-th least significant bit is 0 if at least one process is at $(q_{i,0}, k)$, and 1 if at least one process is at $(q_{i,1}, k)$. Finally, at round 2^{m-1}, setting the m-th least significant bit –of weight 2^{m-1}– to 1 corresponds to $(q_{\mathsf{err}}, 2^{m-1})$ being covered.

The following proposition is useful for the analysis of $\mathcal{BC}_m$. It states that, in register protocols where $\mathsf{v} = 0$ and $\mathsf{d} = 1$, coverable locations can be covered with a common execution.

▶ **Proposition 10.** *In a register protocol $\mathcal{P}$ with $\mathsf{v} = 0$ and $\mathsf{d} = 1$, for any finite set L of coverable locations, there exists $n \in \mathbb{N}$ and an execution $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that, for all $(q, k) \in L$, $(q, k) \in \mathsf{loc}(\sigma)$.*

Our protocol $\mathcal{BC}_m$ satisfies the following property, that entails Proposition 9.

▶ **Proposition 11.** *Let $k \in [0, 2^{m-1}]$. Location (q_{err}, k) is coverable in $\mathcal{BC}_m$ iff $k = 2^{m-1}$.*



(a) An exponential number of processes is needed to cover q_{err}.

(b) An exponential number of active rounds is needed to cover q_{err}.

Figure 4 Two modifications of the tick mechanism of $(\mathcal{BC}_m)_{m\geq 1}$ yielding protocols that need respectively an exponential number of processes and an exponential number of active rounds.

Exponential cutoff

▶ **Proposition 12.** *There exists a family $(\mathcal{P}_m)_{m\geq 1}$ of round-based register protocols with q_{err} an error state, $\mathsf{v} = 0$ and $\mathsf{d} = 1$, such that $|\mathcal{P}_m| = O(m)$ and the minimal number of processes to cover an error configuration is in $\Omega(2^m)$.*

The protocol $\mathcal{P}_m$ is easily obtained from $\mathcal{BC}_m$ by modifying the tick mechanism so that each tick must be performed by a different process, as illustrated in Figure 4a. Since exponentially many ticks are needed to cover q_{err}, the cutoff is also exponential.

Exponential number of simultaneously active rounds

We have seen that the minimal round at which the error state can be covered may be exponential. Perhaps more surprisingly, we now show that the processes may need to spread over exponentially many different rounds. We formalise this with the notion of active rounds. At a configuration along a given execution, round k is *active* when some process is at round k and not idle, i.e., it performs a move later in the execution. The *number of active rounds* of an execution is the maximum number of active rounds at each configuration along the execution.

Towards a polynomial space algorithm for the safety problem, a polynomial bound on the number of active rounds would allow one to guess on-the-fly an error execution by storing only non-idle processes for the current configuration. However, such a polynomial bound does not exist:

▶ **Proposition 13.** *There exists a family $(\mathcal{P}'_m)_{m\geq 1}$ of round-based register protocols with q_{err} an error state, $\mathsf{v} = 1$ and $\mathsf{d} = 1$, such that $|\mathcal{P}'_m| = O(m)$ and the minimal number of active rounds for any error execution is in $\Omega(2^m)$.*

The protocol $\mathcal{P}'_m$ is again obtained from $\mathcal{BC}_m$ by modifying the tick mechanism, as illustrated in Figure 4b. The transitions from q_{tick} to q_B and from q_B to q_C ensure that, for all $k \in [0, 2^{m-1}]$, a must be written to $\mathsf{rg}[k]$ before it is written to $\mathsf{rg}[k-1]$. The transitions from q_C to q_D and from q_D to q_{tick}, on the contrary, ensure that, for all $k \in [1, 2^{m-1}]$, move_1 must be written to $\mathsf{rg}[k-1]$ before it is written to $\mathsf{rg}[k]$. Hence, in an error execution, when move_1 is first written to $\mathsf{rg}[0]$, all rounds from 1 to 2^{m-1} must be active, and the number of active rounds is at least 2^{m-1}.

Note that Proposition 13 requires $\mathsf{v} > 0$. Generally for round-based register protocols with $\mathsf{v} = 0$, processes in different rounds do not interact and an error execution can be reordered: all moves on round 0 first, then all moves on round 1, and so on, so that the number of active rounds is at most 2. Therefore, when $\mathsf{v} = 0$, a naive polynomial-space algorithm for the safety problem consists in computing all coverable states round after round.

3.2 Compatibility and first-write orders

The *compatibility* of coverable locations expresses that they can be covered in a common execution. Formally, two locations (q_1, k_1) and (q_2, k_2) are *compatible* when there exists $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $(q_1, k_1), (q_2, k_2) \in \mathsf{loc}(\sigma)$. In contrast to several other classes of parameterized models (such as broadcast protocols for instance), for round-based register protocols, not all coverable locations are compatible, which makes the safety problem trickier.

▶ **Example 14.** The importance of compatibility can be illustrated on the protocol of Figure 2, whose safety relies on the fact that, for all $k \geq 1$, locations (q_4, k) and (q_6, k) –although both coverable– are *not* compatible. Intuitively, in order to cover (q_4, k), one must write a to $\mathsf{rg}[k]$ and then read d_0 from $\mathsf{rg}[k-1]$, while in order to cover (q_6, k), one must read a from $\mathsf{rg}[k-1]$ and then read d_0 from $\mathsf{rg}[k]$. Since d_0 cannot be written, covering (q_4, k) requires a write to $\mathsf{rg}[k]$ while $\mathsf{rg}[k-1]$ is still blank, and covering (q_6, k) requires the opposite. ⌟

More generally, the order in which registers are first written to appears to be crucial for compatibility. We thus define in the sequel the *first-write order* associated with an execution, and use it to give sufficient conditions for compatibility of locations, that we express as being able to combine executions covering these locations.

▶ **Definition 15.** *For $\rho = \sigma_0, \theta_1, \cdots \theta_\ell, \sigma_\ell$ an execution, move θ_i is a* first write *(to $\mathsf{rg}_\alpha[k]$) if $\theta_i = ((q, \mathsf{write}_\alpha(x), q'), k)$ and $\mathsf{rg}_\alpha[k] \notin \mathsf{FW}(\sigma_{i-1})$. The* first-write order *of ρ is the sequence of registers $\mathsf{fwo}(\rho) = \xi_1 : \ldots : \xi_m$ such that the j-th first write along ρ writes to ξ_j.*

Following Example 6, $\mathsf{fwo}(\rho_1) = \mathsf{rg}[1]$ and $\mathsf{fwo}(\rho_2) = \mathsf{rg}[0]$. Two executions with same first-write order can be combined into a "larger" one with same first-write order.

▶ **Lemma 16.** *Let $\rho_1 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ and $\rho_2 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$ be two executions such that $\mathsf{fwo}(\rho_1) = \mathsf{fwo}(\rho_2)$. Then, there exists $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $\mathsf{loc}(\sigma) = \mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2)$, $\mathsf{FW}(\sigma) = \mathsf{FW}(\sigma_1) = \mathsf{FW}(\sigma_2)$, and $\mathsf{fwo}(\rho) = \mathsf{fwo}(\rho_1) = \mathsf{fwo}(\rho_2)$.*

It follows that, for any fixed first-write order, there is a *maximal support* that can be covered by executions having that first-write order.

To extend the previous result, we exploit the fact that executions do not read registers arbitrarily far back. It is sufficient to require the first-write orders to have the same projections on all round windows of size v. Formally, for a first-write order f, and two round numbers $k, k' \in \mathbb{N}$ with $k \leq k'$, $\mathsf{proj}_{[k,k']}(\mathsf{f})$ denotes the restriction of f to registers from rounds k to k'.

▶ **Lemma 17.** *Let* $\rho_1 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ *and* $\rho_2 \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_2$ *be two executions of a register protocol with visibility range* v*, such that, for all* $k \in \mathbb{N}$*,* $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_1)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_2))$*. Then, there exists* $\rho \colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ *such that* $\mathsf{loc}(\sigma) = \mathsf{loc}(\sigma_1) \cup \mathsf{loc}(\sigma_2)$*,* $\mathsf{FW}(\sigma) = \mathsf{FW}(\sigma_1) = \mathsf{FW}(\sigma_2)$*, and, for all* $k \in \mathbb{N}$*,* $\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_1)) = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho_2))$*.*

▶ **Example 18.** Agreement of Aspnes' algorithm is closely related to the notion of location (in)compatibility. Intuitively, one requires that no pair of locations (R_0, k_0) and (R_1, k_1) are compatible. Their incompatibility is a consequence of a difference between the first-write orders of the executions that respectively cover them. First, for every $k \geq 1$ and every execution $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma \xrightarrow{*} \sigma'$, if $\mathsf{rg}_{b_i}[k] \in \mathsf{FW}(\sigma)$ and $\mathsf{rg}_{b_{1-i}}[k-1] \notin \mathsf{FW}(\sigma)$, then $\mathsf{rg}_{b_{1-i}}[k] \notin \mathsf{FW}(\sigma')$; indeed, since $\mathsf{rg}_{b_{1-i}}[k] \notin \mathsf{FW}(\sigma)$, all locations in $\mathsf{loc}(\sigma)$ whose states correspond to $p = 1 - i$ are either on round $\leq k-1$ or on round k not on state E_{1-i}, and $\perp$ can no longer be read from $\mathsf{rg}_{b_{1-i}}[k]$; by induction, for all $k' \geq k$, $\mathsf{rg}_{b_{1-i}}[k'] \notin \mathsf{FW}(\sigma')$. Let $\rho_0 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_0$ and $\rho_1 : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma_1$ such that, for all $i \in \{0,1\}$, $(R_i, k_i) \in \mathsf{loc}(\sigma_i)$. For all $i \in \{0,1\}$, moves $\theta_i := ((C_i, \mathsf{write}_{b_i}(\top), D_i), k_i)$ and $\theta'_i := ((D_i, \mathsf{read}^{-1}_{b_{1-i}}(\perp), R_i), k_i)$ are in ρ_i, and θ_i appears before θ'_i in ρ_i. Therefore, by letting i such that $k_i \leq k_{1-i}$, ρ_i requires that $\mathsf{rg}_{b_i}[k_i]$ is first-written while $\mathsf{rg}_{b_{1-i}}[k_i - 1]$ is still blank, and therefore that $\mathsf{rg}_{b_i}[k_{1-i}]$ is left blank, while ρ_{1-i} requires a first write on $\mathsf{rg}_{b_i}[k_{1-i}]$, which proves that (R_0, k_0) and (R_1, k_1) are incompatible. Note that $\mathsf{fwo}(\rho_0)$ and $\mathsf{fwo}(\rho_1)$ do not have the same projection on $[k_{1-i} - 1, k_{1-i}]$, which justifies that Lemma 17 does not apply. ⌟

3.3 Polynomial-space algorithm

We now present the main contribution of this paper.

▶ **Theorem 19.** *The safety problem for round-based register protocols is in PSPACE.*

To establish Theorem 19, because PSPACE is closed under complement and thanks to Savitch's theorem, it suffices to provide a nondeterministic procedure that finds an error execution (if one exists) within polynomial space. We do this in two steps: first, we give a nondeterministic procedure that iteratively guesses projections of a first-write order and computes the set of coverable locations under those projections, but does not terminate; second, we justify how to run this procedure in polynomial space and that it can be stopped after an exponential number of iterations (thus encodable by a polynomial space binary counter).

The high-level idea of the nondeterministic procedure is to iteratively guess a first-write order f, and to simultaneously compute the set of coverable locations under f. Thanks to Lemma 17, rather than considering a precise first-write order, the algorithm guesses its projections on windows of size v. Concretely, at iteration k, the algorithm guesses $F_k = \mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{f})$ and computes the set $S_k(F_k)$ of states that can be covered at round k under f. These sets are computed incrementally along the prefixes of F_k, called *progressions*, which are considered in increasing order. For each prefix, we check whether a first write to the last register is *feasible*, that is, whether some coverable location is the source of such a write; we reject the computation otherwise.

Algorithm 2 provides the skeleton of this procedure. In **Line 3** of Algorithm 2, the sequence of registers F_k is constructed from F_{k-1} by removing the registers at round $(k-\mathsf{v}-1)$ and non-deterministically inserting some registers at round k. By convention, in the special case where $k = 0$, F_0 is set to a sequence of registers of round 0. From **Line 4** on, one considers the successive progressions of F_k, i.e., prefixes of increasing length, **Line 5** setting f to the prefix of F_k of length i. At **Line 7**, the set of coverable states at round k for progression $f = g{:}\xi$ is inherited from the one for progression g.

Algorithm 2 Non-deterministic polynomial space algorithm to compute the set of coverable states round by round.

Variables computed : $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$, $(S_k(f))_{k\in\mathbb{N}, f\in\mathsf{Prefixes}(F_k)}$

1 **Initialisation**: $S_0(\varepsilon) := \{q_0\}$; $\forall (k,f) \neq (0,\varepsilon)$, $S_k(f) := \emptyset$; ;
2 **for** k *from* 0 *to* $+\infty$ **do**
3 non-deterministically choose F_k from F_{k-1} ;
4 **for** i *from* 0 *to* $\mathsf{length}(F_k)$ **do**
5 $f := \mathsf{prefix}_i(F_k)$;
6 **if** $f \neq \epsilon$ **then**
7 Let $f = g{:}\xi$, and set $S_k(f) := S_k(f) \cup S_k(g)$;
8 add to $S_k(f)$ the states that can be covered from round $k{-}1$ by Inc moves;
9 **if** *first write to* $\mathsf{last}(f)$ *is feasible* **then**
10 saturate $S_k(f)$ by read and write moves;
11 **else**
12 Reject;

The next line requires an extra definition. For every $k \in \mathbb{N}$ and every prefix f of F_k, the *synchronisation* $\phi^k_{k-1}(f)$ is the longest prefix of F_{k-1} that coincides with f on rounds $k{-}\mathsf{v}$ to $k{-}1$, i.e. such that $\mathsf{proj}_{[k-\mathsf{v},k-1]}(\phi^k_{k-1}(f)) = \mathsf{proj}_{[k-\mathsf{v},k-1]}(f)$. This is always well defined since F_k is obtained from F_{k-1} by removing registers of round $k{-}\mathsf{v}{-}1$, and inserting registers of round k. So $\phi^k_{k-1}(f)$ can be obtained from f by removing registers of round k, and inserting back those of round $k{-}\mathsf{v}{-}1$ that, in F_{k-1}, are before the first register of round in $[k-\mathsf{v}, k-1]$ that is not in f. Similarly, we define the prefixes of f corresponding to previous rounds. For every $r < k{-}1$ and every prefix f of F_k, the *synchronisation* $\phi^k_r(f)$ is defined inductively by $\phi^k_r(f) := \phi^{r+1}_r(\phi^k_{r+1}(f))$, so that $\phi^k_r(f) := \phi^{r+1}_r(\phi^{r+2}_{r+1}(\dots(\phi^{k-1}_{k-2}(\phi^k_{k-1}(f)))\dots))$. Last, by convention, $\phi^k_k(f) := f$.

▶ **Example 20.** We illustrate the notion of synchronisation function on a toy example. Consider the sequence of registers $F_1 = \alpha_1 : \beta_1 : \gamma_0 : \delta_0 : \epsilon_1 : \zeta_0$, where the subscripts denote the rounds, and assume that $\mathsf{v} = 1$. The sequence F_2 is obtained from F_1 by removing the round 0 registers $\gamma_0, \delta_0, \zeta_0$, and by inserting some registers of round 2. For instance, one nondeterministically construct $F_2 = \alpha_1 : \eta_2 : \beta_1 : \theta_2 : \epsilon_1$. In that case, for instance $\phi^2_1(\alpha_1 : \eta_2 : \beta_1) = \alpha_1 : \beta_1 : \gamma_0 : \delta_0$; in words, when we are at iteration 2 with progression $\alpha_1 : \eta_2 : \beta_1$, the corresponding progression at iteration 1 is $\alpha_1 : \beta_1 : \gamma_0 : \delta_0$. Also, $\phi^2_1(\alpha_1 : \eta_2) = \alpha_1$ and $\phi^2_1(\alpha_1 : \eta_2 : \beta_1 : \theta_2) = \alpha_1 : \beta_1 : \gamma_0 : \delta_0 : \epsilon_1 : \zeta_0$.

On iteration further, one could have $F_3 = \eta_2 : \kappa_3 : \theta_2$ and thus $\phi^3_1(\eta_2 : \kappa_3) = \phi^2_1(\phi^3_2(\eta_2 : \kappa_3)) = \phi^2_1(\alpha_1 : \eta_2 : \beta_1) = \alpha_1 : \beta_1 : \gamma_0 : \delta_0$. ⌟

Now, $S_k(f)$ is defined in two steps. First, **Line 8** adds to $S_k(f)$ the states that can be immediately obtained by an Inc move from states coverable at round $k{-}1$. Formally, $S_k(f) := S_k(f) \cup \{q' \in Q \mid \exists q \in S_{k-1}(\phi^k_{k-1}(f)), (q, \mathsf{Inc}, q') \in \Delta\}$. **Line 9** then checks that a first write to the last register in f is feasible; that is, if $f = g : \mathsf{rg}_\alpha[k]$, then, one checks whether there exists a write transition $(q, \mathsf{write}_\alpha(x), q') \in \Delta$ with $x \neq d_0$ and $q \in S_k(g)$. Second, in **Line 10**, we saturate $S_k(f)$ by all possible moves at round k. Formally, we add every state $q' \in Q \setminus S_k(f)$ such that there exist $q \in S_k(f)$ and $(q, a, q') \in \Delta$ where action a satisfies one of the following conditions:

- $a = \mathsf{read}^{-j}_\alpha(d_0)$ and $\mathsf{rg}_\alpha[k{-}j]$ does not appear in f;
- $a = \mathsf{read}^{-j}_\alpha(x)$ with $x \neq d_0$, $\mathsf{rg}_\alpha[k{-}j]$ appears in f and there exist $q_1, q_2 \in S_{k-j}(\phi^k_{k-j}(f))$ such that $(q_1, \mathsf{write}_\alpha(x), q_2) \in \Delta$;
- $a = \mathsf{write}_\alpha(x)$ and $\mathsf{rg}_\alpha[k]$ appears in f.

In **Line 12**, the computation is rejected since the guessed first-write order is not feasible.

Characterisation of the sets $S_k(F_k)$ computed in Algorithm 2

For a family of first-write order projections $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$ and a round k, we define $Q\mathsf{cover}(\mathcal{F}, k) = \{q \mid \exists \rho\colon \sigma_{\mathsf{init}} \xrightarrow{*} \sigma \text{ s.t. } (q,k) \in \mathsf{loc}(\sigma) \text{ and } \forall r \leq k, \mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)) = F_r\}$. In words, $Q\mathsf{cover}(\mathcal{F}, k)$ is the set of states that can be covered at round k by an execution whose first-write order projects to the family $\mathcal{F}$ on windows of size v.

Observe that the only non-deterministic choice in Algorithm 2 is the choice of the sequences F_k; hence, for a given $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$, there is at most one non-rejecting computation whose first-write order projections agrees with family $\mathcal{F}$. In that case, we say that the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting.

▶ **Theorem 21.** *For $\mathcal{F} = (F_k)_{k\in\mathbb{N}}$ a family of projections, if the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting, then the computed sets $(S_k(F_k))_{k\in\mathbb{N}}$ satisfy, for all $k \in \mathbb{N}$, $S_k(F_k) = Q\mathsf{cover}(\mathcal{F}, k)$. Also, for any execution ρ from σ_{init}, letting $\mathcal{F} = (\mathsf{proj}_{[k-\mathsf{v},k]}(\mathsf{fwo}(\rho)))_{k\geq 0}$, the $\mathcal{F}$-computation of Algorithm 2 is non-rejecting.*

Building on Algorithm 2, our objective it to design a polynomial space algorithm to decide the safety problem for round-based register protocols. Theorem 21 shows the correctness of the nondeterministic procedure in the following sense: a non-rejecting computation computes all coverable states for the guessed first-write order, and any possible first-write order admits a corresponding non-rejecting computation. To conclude however, the space complexity should be polynomial in the size of the protocol, and termination must be guaranteed by some stopping criterion.

Staying within space budget. As presented, Algorithm 2 needs unbounded space to execute since it stores all sequences of first-write orders F_k and all sets $S_k(f)$. To justify that polynomial space is sufficient, we first observe that some computed values can be ignored after each iteration. Precisely, iteration k only uses variables of iteration $k-1$ for increments and of iterations $k-\mathsf{v}$ to $k-1$ for read/write moves. Thus, at the end of iteration k, all variables indexed with round $k-\mathsf{v}$ can be forgotten. It is thus sufficient to store the variables of $\mathsf{v}+1$ consecutive rounds.

To conclude, observe also that the maximum length of any sequence F_k is $\mathsf{d}(\mathsf{v}+1)$. Therefore each F_k has at most $\mathsf{d}(\mathsf{v}+1)+1$ prefixes, and there are at most $(\mathsf{d}(\mathsf{v}+1)+1)(\mathsf{v}+1)$ sets $S_r(f)$ with $r \in [k-\mathsf{v}, k]$ for a fixed round number k. We also do not need to store the value of k. All in all, the algorithm can be implemented in space complexity $O(Q \cdot \mathsf{d} \cdot \mathsf{v}^2)$.

Ensuring termination. To exhibit a stopping criterion, we apply the pigeonhole principle to conclude that after a number of iterations at most exponential in $Q \cdot \mathsf{d} \cdot \mathsf{v}^2$, the elements stored in memory repeat from a previous iteration, so that the algorithm starts looping. If q_{err} was not covered at that point, it cannot be covered in further iterations. One can thus use an iteration counter, encoded in polynomial space in the size of the protocol, to count iterations and return a decision when the counter reaches its largest value.

Note that, for negative instances of the safety problem, this gives an exponential upper bound on the round number at which q_{err} is covered. Combined with Corollary 8, it yields an exponential upper bound on the cutoff too. Both match the lower bounds established in Propositions 9 and 12.

▶ **Corollary 22.** *Let $\mathcal{P}$ be a round-based register protocol, and q_{err} an error state. If $(\mathcal{P}, q_{\mathsf{err}})$ is a negative instance of the safety problem, then there exist $K, N \in \mathbb{N}$ both exponential in $|\mathcal{P}|$ such that there exist $k \leq K$ and a concrete execution $\pi : \mathsf{init}_N \xrightarrow{*} \gamma$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\gamma)$.*

With the space constraints and stopping criterion discussed above, the nondeterministic algorithm decides the safety problem for round-based register protocols. Indeed, it suffices to execute Algorithm 2 up until iteration K and check whether q_{err} appears in one the sets $S_k(F_k)$. If q_{err} is found in some $S_k(F_k)$ with $k \leq K$, then $q_{\mathsf{err}} \in Q\mathsf{cover}(\mathcal{F}, k)$, where $(\mathcal{F}_r)_{r \leq k}$ is the family of projections picked by the computation of the algorithm. Thus, the protocol is unsafe. Conversely, if the protocol is unsafe, then there exist $k \leq K$ and $\rho : \sigma_{\mathsf{init}} \xrightarrow{*} \sigma$ such that $(q_{\mathsf{err}}, k) \in \mathsf{loc}(\sigma)$. Letting $\mathcal{F} = (\mathsf{proj}_{[r-\mathsf{v},r]}(\mathsf{fwo}(\rho)))_{r \in \mathbb{N}}$, the $\mathcal{F}$-computation of the algorithm is non-rejecting, and since $q_{\mathsf{err}} \in Q\mathsf{cover}(\mathcal{F}, k)$, one has $q_{\mathsf{err}} \in S_k(F_k)$.

3.4 PSPACE lower bound

▶ **Theorem 23.** *The safety problem for round-based register protocols is PSPACE-hard, even for fixed* $\mathsf{v} = 0$ *and fixed* $\mathsf{d} = 1$.

Proof. The proof is by reduction from the validity of QBF.

From a 3-QBF instance, we define a round-based register protocol $\mathcal{P}_{\mathsf{QBF}}$ with an error state q_{err} so that the answer to the safety problem is no if and only if the answer to QBF-validity is yes, i.e., state q_{err} is coverable if, and only if, the QBF instance is valid. This proves that the safety problem is coPSPACE-hard, and therefore that it is PSPACE-hard since PSPACE = coPSPACE.

The protocol $\mathcal{P}_{\mathsf{QBF}}$ that we construct from a QBF instance is partly inspired by the binary counter from Figure 3. Recall that in $\mathcal{BC}_m$, each bit is represented by a subprotocol, and every round corresponds to an increment of the counter value. In $\mathcal{P}_{\mathsf{QBF}}$, each variable is represented by a subprotocol, and every round corresponds to considering a different valuation and evaluating whether it makes the inner SAT formula true. $\mathcal{P}_{\mathsf{QBF}}$ uses a single register per round ($\mathsf{d} = 1$), and the subprotocol corresponding to variable x writes at each round the truth value of x in the considered valuation. The protocol is designed to enumerate all relevant valuations, and take the appropriate decision about the validity.

We fix an instance ϕ of 3-QBF over the $2m$ variables $\{x_0, \cdots, x_{2m-1}\}$

$$\phi = \forall x_{2m-1} \exists x_{2m-2} \forall x_{2m-3} \exists x_{2m-4} \dots \forall x_1 \exists x_0 \bigwedge_{1 \leq j \leq p} a_j \vee b_j \vee c_j \ ,$$

with for every $j \in [1, p]$, $a_j, b_j, c_j \in \{x_i, \neg x_i \mid i \in [0, 2m-1]\}$ are the literals and write ψ for the inner 3-SAT formula.

From ϕ we construct a round-based register protocol on the data alphabet

$$\mathrm{D} := \{\mathsf{wait}_i, \mathsf{yes}_i, \mathsf{no}_i \mid i \in [0, 2m]\} \cup \{\mathtt{x_i}, \neg\mathtt{x_i} \mid i \in [0, 2m-1]\} \cup \{d_0\} \ ,$$

that in particular contains two symbols $\mathtt{x_i}$ and $\neg\mathtt{x_i}$ for each variable x_i. Moreover, we let $\mathsf{v} = 0$ and $\mathsf{d} = 1$.

Thanks to Proposition 10, when $\mathsf{v} = 0$ and $\mathsf{d} = 1$, all coverable locations are compatible, for every finite number of coverable locations, there exists an execution that covers all these locations. We therefore do not have to worry about with which execution a location is coverable, and we will simply write that a location *is coverable* or *is not coverable* and that a symbol *can be written* or *cannot be written* to a given register.

The protocol we construct is represented in Figure 5; it contains several gadgets that we detail in the sequel. Before that we provide a high-level view of $\mathcal{P}_{\mathsf{QBF}}$. In $\mathcal{P}_{\mathsf{QBF}}$, each variable x_i is represented by a subprotocol $\mathcal{G}_i$, and every round corresponds to considering a different valuation and evaluating whether it makes the inner SAT formula true with the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$. The gadget $\mathcal{G}_i$ writes at each round the truth value of x_i in the considered evaluation. The protocol enumerates all valuations: a given round k will correspond to one

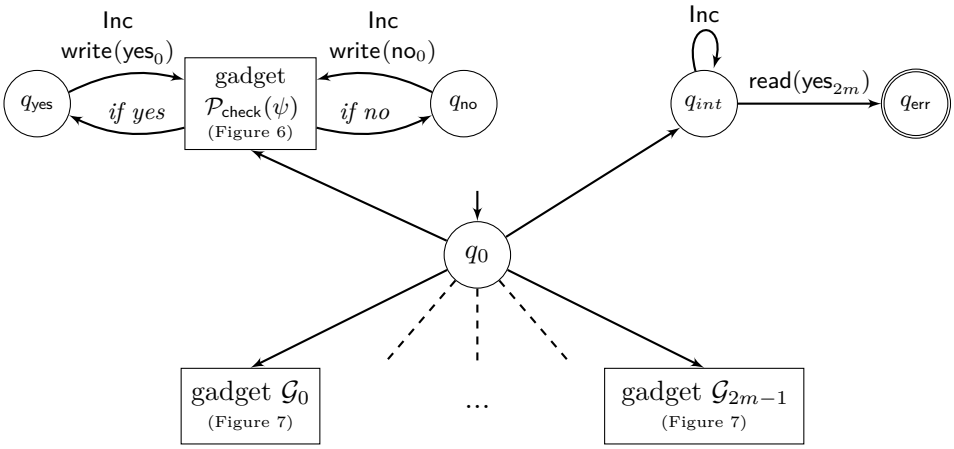


Figure 5 Overview of the protocol $\mathcal{P}_{\mathsf{QBF}}$. All transitions to gadgets go to their initial states.

valuation of the variables of ψ, in which variable x is true if $\mathtt{x}$ can be written to $\mathsf{rg}[k]$, and false if $\neg\mathtt{x}$ can be written to $\mathsf{rg}[k]$. The enumeration of the valuations and corresponding evaluations of ψ are performed so as to take the appropriate decision about the validity of the global formula ϕ.

We start by describing the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$, depicted in Figure 6, that checks whether ψ is satisfied by the valuation under consideration. State q_{yes} corresponds to ψ evaluated to

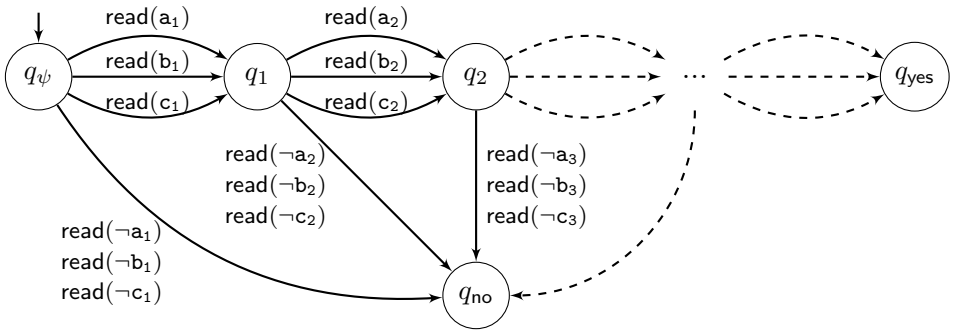


Figure 6 Gadget $\mathcal{P}_{\mathsf{check}}(\psi)$ that checks whether ψ is satisfied by the current valuation.

true and q_{no} corresponding to ψ evaluated to false. Note that we allow transitions labelled by sequences of actions; for instance the transition from state q_ψ to state q_{no} consists of three consecutive reads. The following lemma proves that the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$ indeed checks how ψ evaluates for the current valuation.

▶ **Lemma 24.** *Let $k \in \mathbb{N}$. Suppose that (q_ψ, k) is coverable and that we have a valuation ν of the variables of ψ such that, for every $i \in [0, 2m-1]$:*

- *if $\nu(x_i) = 1$, then $\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$, and $\neg\mathtt{x_i}$ cannot,*
- *if $\nu(x_i) = 0$, then $\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$, and $\mathtt{x_i}$ cannot.*

Then (q_{yes}, k) is coverable if and only if $\nu \models \psi$, and (q_{no}, k) is coverable if and only if $\nu \models \neg\psi$.

We now explain how valuations are enumerated, and how the different quantifiers are handled. The procedure next, given valuation ν, computes the next valuation $\mathsf{next}(\nu)$ that needs to be checked. Eventually, the validity of the formula will be determined by checking whether $\nu_0 \models \psi$ (where ν_0 assigns 0 to all variables) and $\mathsf{next}^k(\nu_0) \models \psi$ for increasing values of $k \geq 1$.

Let ν a valuation of all variables, and define the valuation $\mathsf{next}(\nu)$. Let ϕ_i denote the subformula $Qx_i \dots \forall x_1 \exists x_0 \psi$ where $Q = \exists$ if i is even, and $Q = \forall$ otherwise. We write $\nu \models \phi_i$ when ϕ_i is true when its free variables $x_{2m-1}, \dots, x_{i+1}$ are set to their values in ν. The procedure next uses variables $b_i \in \{\mathsf{yes}, \mathsf{no}, \mathsf{wait}\}$ for each $i \in [0, 2m]$, whose role is the following. We will set $b_0 = \mathsf{yes}$ if $\nu \models \psi$, and $b_0 = \mathsf{no}$ otherwise. For any $1 \leq i \leq 2m-1$, $b_i = \mathsf{yes}$ means $\nu \models \phi_i$; $b_i = \mathsf{no}$ means $\nu \not\models \phi_i$; while $b_i = \mathsf{wait}$ means that more valuations need to be checked to determine whether $\nu \models \phi_i$ or not. Given a valuation ν, the procedure next computes, at each iteration i, the truth value of x_i in valuation $\mathsf{next}(\nu)$ and the value of b_{i+1}. After $2m$ iterations, this provides the new valuation $\mathsf{next}(\nu)$ against which ψ must be checked. Formally, $b_0 = \mathsf{yes}$ if $\nu \models \psi$, and $b_0 = \mathsf{no}$ otherwise, and for all $i \in [0, 2m-1]$:

- If $b_i = \mathsf{wait}$, then $\mathsf{next}(\nu)(x_i) := \nu(x_i)$ and $b_{i+1} := \mathsf{wait}$.
- Otherwise
  - If i is even (existential quantifier).
    - if $b_i = \mathsf{yes}$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{yes}$,
    - if $b_i = \mathsf{no}$ and $\nu(x_i) = 0$, then $\mathsf{next}(\nu)(x_i) := 1$ and $b_{i+1} := \mathsf{wait}$,
    - if $b_i = \mathsf{no}$ and $\nu(x_i) = 1$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{no}$.
  - if i is odd (universal quantifier),
    - if $b_i = \mathsf{no}$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{no}$,
    - if $b_i = \mathsf{yes}$ and $\nu(x_i) = 0$, then $\mathsf{next}(\nu)(x_i) := 1$ and $b_{i+1} := \mathsf{wait}$,
    - if $b_i = \mathsf{yes}$ and $\nu(x_i) = 1$, then $\mathsf{next}(\nu)(x_i) := 0$ and $b_{i+1} := \mathsf{yes}$.

Note that variable b_{2m} is computed but not used in the computation. Its value will play the role of a result, e.g., in Lemma 25.

The following lemma formalizes how validity can be checked using next. It is easily proven by induction on m.

▶ **Lemma 25.** *ϕ is valid if and only if, when iterating* next *from valuation ν_0, one eventually obtains a computation of* next *that sets b_{2m} to* yes*. Otherwise, one eventually obtains a computation of* next *that sets b_{2m} to* no*.*

▶ **Example 26.** Let us illustrate the next operator and Lemma 25 on a small example. Assume

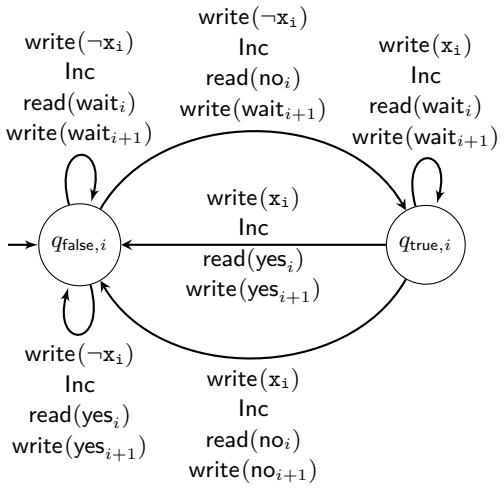
$$\phi = \exists x_2 \forall x_1 \exists x_0 \; \neg x_2 \wedge \neg x_1 \wedge (x_1 \vee \neg x_0),$$

which is not a valid formula. To determine that ϕ is not valid, we start by checking the valuation $\nu_0 = (0, 0, 0)$, writing ν_0 as the tuple $(\nu_0(x_0), \nu_0(x_1), \nu_0(x_2))$. Let $\nu = \mathsf{next}(\nu_0)$. ν_0 satisfies the inner formula, hence we set $b_0 = \mathsf{yes}$. By following the procedure of next, we obtain $\nu(x_0) = 0$, $b_1 = \mathsf{yes}$ in the first iteration (in fact, $\nu_0 \models \phi_0$); and $\nu(x_1) = 1$, $b_2 = \mathsf{wait}$ in the second iteration. In fact, even though $\nu_0 \models \psi$, because x_1 is quantified universally, we cannot yet conclude: we must also check whether ψ holds by setting x_1 to 1. This is what $b_2 = \mathsf{wait}$ means, and this is why $\nu(x_1)$ is set to 1. Lastly, we obtain $\nu(x_2) = 0$ and $b_3 = \mathsf{wait}$, therefore $\nu = (0, 1, 0)$.

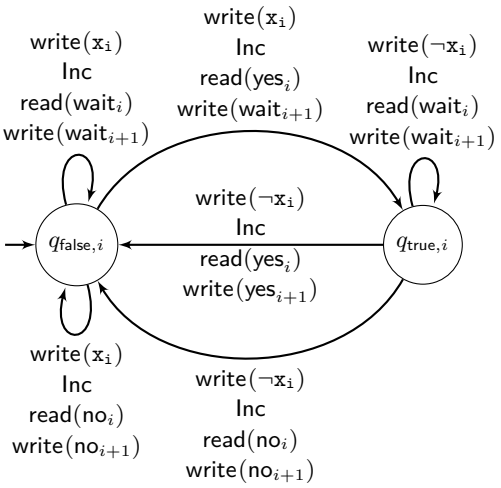
Let $\nu' = \mathsf{next}(\nu) = \mathsf{next}^2(\nu_0)$. We observe that $\nu \not\models \psi$ and set $b_0 = \mathsf{no}$. We then have $\nu'(x_0) = 1$, $b_1 = \mathsf{wait}$, and therefore $\nu'(x_1) = 1$ and $\nu'(x_2) = 0$. In the end, $\nu' = (0, 1, 1)$.

The computation of $\mathsf{next}^3(\nu_0)$ then sets x_2 to 1 because no valuation with $x_2 = 0$ satisfied the formula. We obtain $\mathsf{next}^3(\nu_0) = (1, 0, 0)$ and $\mathsf{next}^4(\nu_0) = (1, 0, 1)$. The computation of $\mathsf{next}^5(\nu_0)$ sets b_{2m} to no, establishing that ϕ is not valid. ⌟

Now, we define, for all $i \in [0, 2m-1]$, a gadget $\mathcal{G}_i$ that will play the role of variable x_i. At each round, gadget $\mathcal{G}_i$ receives from gadget $\mathcal{G}_{i-1}$ a value in $\{\mathsf{wait}_i, \mathsf{yes}_i, \mathsf{no}_i\}$ (except for gadget $\mathcal{G}_0$ which receives this value from $\mathcal{P}_{\mathsf{check}}(\psi)$). It transmits a value in $\{\mathsf{wait}_{i+1}, \mathsf{yes}_{i+1}, \mathsf{no}_{i+1}\}$ to $\mathcal{G}_{i+1}$, and modifies the value of variable x_i accordingly, writing either $\mathtt{x_i}$ or $\neg\mathtt{x_i}$ to the register. These gadgets $\mathcal{G}_i$ are given in Figure 7a if x_i is existentially quantified (i.e., i even),



(a) Gadget $\mathcal{G}_i$ for existentially quantified variable x_i (i.e., i even).



(b) Gadget $\mathcal{G}_i$ for universally quantified variable x_i (i.e., i odd).

Figure 7 Illustration of the gadgets $\mathcal{G}_i$.

and Figure 7b if x_i is universally quantified (i.e., i odd). Using those gadgets $\mathcal{G}_i$ and $\mathcal{P}_{\mathsf{check}}(\psi)$ together with the earlier described gadget $\mathcal{P}_{\mathsf{check}}(\psi)$, we define the protocol $\mathcal{P}_{\mathsf{QBF}}$ represented in Figure 5.

Finally, the following lemma justifies the correctness of our construction by formalising the relation between next and $\mathcal{P}_{\mathsf{QBF}}$.

▶ **Lemma 27.** *Let $k \in \mathbb{N}$ and $\nu_k := \mathsf{next}^k(\nu_0)$, the valuation obtained by applying next k times from $\nu_0 := 0^{2m}$. For all $i \in [0, 2m-1]$:*

- *$(q_{\mathsf{false},i}, k)$ is coverable if and only if $\nu_k(x_i) = 0$,*
- *$(q_{\mathsf{true},i}, k)$ is coverable if and only if $\nu_k(x_i) = 1$,*
- *$\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 0$,*
- *$\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 1$.*

Moreover, if $k > 0$, then for all $j \in [0, 2m]$:

- *yes_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to yes,*
- *no_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to no,*
- *wait_j can be written to $\mathsf{rg}[k]$ if and only if computation $\nu_k = \mathsf{next}(\nu_{k-1})$ sets b_j to wait.*

Combining Lemma 27 with Lemma 25 proves that there exists a register to which yes_{2m} can be written if and only if ϕ is valid. Also, q_{err} is coverable in $\mathcal{P}_{\mathsf{QBF}}$ if and only if there exists a register to which yes_{2m} can be written, concluding the proof of Theorem 23. ◀

It may seem surprising that the safety problem is PSPACE-hard already for $\mathsf{d} = 1$ and $\mathsf{v} = 0$, i.e., with a single register and no visibility on previous rounds. For single register protocols without rounds, safety properties can be verified in polynomial time with a simple saturation algorithm. This complexity blowup highlights the expressive power of rounds, independently of the visibility on previous rounds.

Theorems 19 and 23 yield the precise complexity of the safety problem.

▶ **Corollary 28.** *The safety problem for round-based register protocols is PSPACE-complete.*

4 Conclusion

This paper makes a first step towards the automated verification of round-based shared-memory distributed algorithms. We introduce the model of round-based register protocols and solves its parameterized safety verification problem. Precisely, we prove that this problem is PSPACE-complete, providing in particular a non-trivial polynomial space decision algorithm. We also establish exponential lower and upper bounds on the cutoff and on the minimal round at which an error is reached.

Many interesting extensions could be considered, such as assuming the presence of a leader as in [13], or considering other properties than safety. In particular, for algorithms such as Aspnes', beyond validity and agreement that are safety properties, one would need to be able to handle liveness properties (possibly under a fairness assumption) to prove termination.

References

1 C. Aiswarya, Benedikt Bollig, and Paul Gastin. An automata-theoretic approach to the verification of distributed algorithms. *Information and Computation*, 259(3):305–327, 2018. doi:10.1016/j.ic.2017.05.006.

2 Krzysztof Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, May 1986. doi:10.1016/0020-0190(86)90071-2.

3 James Aspnes. Fast deterministic consensus in a noisy environment. *Journal of Algorithms*, 45(1):16–39, 2002. doi:10.1016/S0196-6774(02)00220-1.

4 James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003. doi:10.1007/s00446-002-0081-5.

5 Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder. Verification of randomized consensus algorithms under round-rigid adversaries. *International Journal on Software Tools Technology Transfer*, 23(5):797–821, 2021. doi:10.1007/s10009-020-00603-x.

6 Nathalie Bertrand, Bastien Thomas, and Josef Widder. Guard automata for the verification of safety and liveness of distributed algorithms. In *Proceedings of the 32nd International Conference on Concurrency Theory (CONCUR'21)*, volume 203 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.15.

7 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.2200/S00658ED1V01Y201508DCT013.

8 Benedikt Bollig, Fedor Ryabinin, and Arnaud Sangnier. Reachability in distributed memory automata. In *Proceedings of the 29th EACSL Annual Conference on Computer Science Logic (CSL'21)*, volume 183 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CSL.2021.13.

9 Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP'16)*, volume 55 of *LIPIcs*, pages 106:1–106:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.106.

10 Antoine Durand-Gasselin, Javier Esparza, Pierre Ganty, and Rupak Majumdar. Model checking parameterized asynchronous shared-memory systems. *Formal Methods Systems Design*, 50(2-3):140–167, 2017. doi:10.1007/s10703-016-0258-3.

11 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.1.

12 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *Lecture Notes in Computer Science*, pages 124–140. Springer-Verlag, July 2013. doi:10.1007/978-3-642-39799-8_8.

13 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *Journal of the ACM*, 63(1):10:1–10:48, 2016. doi:10.1145/2842603.

14 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, July 1992. doi:10.1145/146637.146681.

15 Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. Fair termination for parameterized probabilistic concurrent systems. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 499–517, 2017. doi:10.1007/978-3-662-54577-5_29.

16 Michel Raynal and Julien Stainer. A Simple Asynchronous Shared Memory Consensus Algorithm Based on Omega and Closing Sets. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 357–364, 2012. doi:10.1109/CISIS.2012.198.

Passive Learning of Deterministic Büchi Automata by Combinations of DFAs

León Bohn
RWTH Aachen University, Germany

Christof Löding
RWTH Aachen University, Germany

Abstract

We present an algorithm that constructs a deterministic Büchi automaton in polynomial time from given sets of positive and negative example words. This learner constructs multiple DFAs using a polynomial-time active learning algorithm on finite words as black box using an oracle that we implement based on the given sample of ω-words, and combines these DFAs into a single DBA. We prove that the resulting algorithm can learn a DBA for each DBA-recognizable language in the limit by providing a characteristic sample for each DBA-recognizable language. We can only guarantee completeness of our algorithm for the full class of DBAs through characteristic samples that are, in general, exponential in the size of a minimal DBA for the target language. But we show that for each fixed k these characteristic samples are of polynomial size for the class of DBAs in which each subset of pairwise language-equivalent states has size at most k.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases deterministic Büchi automata, learning from examples, learning in the limit, active learning

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.114

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding *León Bohn*: Supported by DFG grant LO 1174/7-1.

1 Introduction

The problem of constructing finite automata from example words (also referred to as passive learning of automata) has been investigated since the 1970ies [8, 26, 14], see [17] for a survey. The task is to develop an algorithm that infers a finite automaton from a given sample $S = (S_+, S_-)$ such that all words from the finite set S_+ of positive examples are accepted, and all words from the finite set S_- are rejected. Such an algorithm can be viewed as a way of learning an automaton from a given set of examples, and we therefore refer to such algorithms as learners in the following.

Besides the running time of a learner, it is also of interest for which languages it can learn a finite automaton. In order to characterize learners that are robust and generalize from the sample, Gold proposed the notion of "learning in the limit" [13]. Given a class $\mathcal{C}$ of regular languages, a learner is said to learn every language in $\mathcal{C}$ in the limit, if for each language $L \in \mathcal{C}$ there is a characteristic sample S^L that is consistent with L and such that the learner returns a DFA for L for each sample that is consistent with L and contains all examples from S^L. In this case, we also say that the learner is complete for the class $\mathcal{C}$.

In [14] Gold presents a learner that constructs in polynomial time a DFA for a given sample and that can learn every regular language in the limit. Moreover, for each regular language L there is a characteristic sample of size polynomial in the minimal DFA for L. Such a learner is said to learn every regular language in the limit from polynomial data.

The key property that is used in most learning algorithms for regular languages is the characterization of regular languages by the Myhill/Nerode congruence: For a language L, two words u, v are equivalent if they cannot be distinguished by the language, that is, if for each word w, either both uw, vw are in L, or both are not in L. It is a basic result from automata theory that L is regular iff the Myhill/Nerode congruence has finitely many classes, and that these classes can be used as state set of the minimal DFA for L (see basic textbooks on automata theory, e.g. [15]). The idea for Gold's algorithm is to infer the Myhill/Nerode congruence from the sample $S = (S_+, S_-)$ based on the following idea. Two words u, v cannot be equivalent for any language that is consistent with S, if there is a word w such that $uw \in S_+$ iff $vw \in S_-$. Roughly speaking, Gold's algorithm uses such "obviously distinguishable" words as states for a DFA. If the sample does not contain enough information, it can happen that the resulting DFA is not consistent with the sample, and then the algorithm simply defaults to returning a DFA for S_+. The algorithm RPNI [21] avoids this problem by starting from a tree-shaped DFA for S_+, and then trying to merge states of this DFA in a specific order. If a merge leads to a DFA that is inconsistent with the sample, then the merge is discarded. Otherwise, the two states are merged, and the algorithm continues with this smaller DFA. This algorithm runs in polynomial time and learns every regular language in the limit from polynomial data [21]. Since then, many variations of such state merging techniques have been proposed and implemented, e.g., in the framework learnlib [16] and the library flexfringe [27]

While there is a lot of work on construction of DFAs from samples, very little is known about this problem for automata on infinite words, so called ω-automata. These have been studied since the early 1960s as a tool for solving decision problems in logic [11] (see also [24]), and are nowadays used in procedures for formal verification and synthesis of reactive systems (see, e.g., [6, 25, 19] for surveys and recent work). Syntactically, ω-automata are very similar to NFA resp. DFA (standard nondeterministic resp. deterministic finite automata on finite words), and they also share many closure and algorithmic properties. However, while the definition of the Myhill/Nerode congruence can easily be lifted to ω-languages, it does not give a characterization of the regular ω-languages. In particular, deterministic ω-automata may need several different language equivalent states in order to accept some regular ω-languages (as opposed to DFAs). As a consequence, deterministic ω-automata do not share the good properties of DFAs, e.g., minimization of deterministic Büchi automata is NP-hard [23], and also the methods for learning DFAs cannot be directly transferred to ω-automata. In fact, the current algorithms for constructing deterministic ω-automata from examples are adaptions of the algorithm of Gold [5] and of RPNI [9], and they are only known to learn ω-languages with informative right congruence (IRC) in the limit. The class IRC [4] consists of those languages that can be accepted by deterministic ω-automata without different language equivalent states, and thus can be handled by an adaption of the methods from finite words.

In this paper we propose a passive learning algorithm that is complete for the class of ω-languages that can be accepted by deterministic Büchi automata (DBA) (meaning that it can learn every language form that class in the limit). To the best of our knowledge, this is the first learning algorithm that is complete for a relevant class of ω-languages beyond languages with IRC. While DBA languages still form a strict subclass of the regular ω-languages, each regular ω-language is a finite Boolean combination of DBA languages [24], and therefore the DBA languages form an important class for understanding learning problems for ω-automata.

Our algorithm uses a learning algorithm for DFAs as sub-procedure, but interestingly an active learning algorithm. Such algorithms have to infer the DFA of a target language based on queries that are answered by an oracle. Angluin proposed an algorithm that learns the minimal DFA for a target language L based on membership and equivalence queries

in polynomial time [1]. A membership query asks for a specific word if it is in the target language, and the oracle answers "yes" or "no". An equivalence query asks if a hypothesis DFA accepts the target language, and the oracle provides a word as counterexample if it does not. We make use of such an active learning algorithm as black box in order to infer a set of DFAs that are then used to build the DBA. Roughly, our algorithm works as follows for a given ω-sample $S = (S_+, S_-)$ consisting of ultimately periodic ω-words (the use of ultimately periodic words is standard in learning of ω-automata [18, 3, 5, 9]):

- Infer a right congruence $\sim$ that is consistent with S. This can be done using the same methods as for finite words.
- For each class of $\sim$ learn a DFA using an active learning algorithm as black box, answering the queries of this algorithm based on the information in S.
- Combine the DFAs into a DBA: Start in the DFA for the initial class of $\sim$. Whenever a DFA reaches an accepting state, redirect the transition into the initial state of the DFA for the current class, and make this transition accepting for the Büchi condition.[1]

This algorithm runs in polynomial time and can learn every DBA language in the limit. The characteristic sample for a DBA language L that we use for showing the learning in the limit result can be of size exponential in a smallest DBA for L. But we also show that it is polynomial if we fix the size of sets of pairwise language equivalent states in the DBAs. This generalizes known results for learning in the limit from polynomial data for languages with IRC [5, 9].

Besides the work on passive learning, there are also some papers on active learning of ω-languages from queries. In general, it seems that efficient active learning of deterministic omega-automata is more difficult than efficient passive learning. This is witnessed by the fact that a polynomial time active learner (with membership and equivalence queries) yields an efficient passive learner, and that polytime active learning IRC ω-languages is not easier than polytime active learning general omega-languages (see [9] for both results). For this reason, the currently known active learning algorithms either learn different representations for ω-languages, as for example families of DFAs in [3], or include syntactic information on the target automaton in the form of loop index queries [20].

The remainder of the paper is structured as follows. In Section 2 we introduce basic terminology and definitions used in the paper. Subsequently, in Section 3 we present our learning algorithm and prove that it is a consistent polynomial time DBA-learner. In Section 4 we prove the completeness result that our algorithm can learn every DBA language in the limit. In Section 5 we analyze the size of the characteristic samples from the completeness proofs, and we conclude in Section 6.

2 Preliminaries

For a finite alphabet Σ we denote by Σ^* and Σ^ω the set of all finite and infinite words over Σ, respectively and define $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$, where ε denotes the empty word. A language is a subset of Σ^* and an ω-language a subset of Σ^ω. We sometimes simply write language instead of ω-language, the meaning should always be clear from the context. We use $\preceq$ to denote the canonical or length-lexicographic order on finite words over Σ, in which words are first ordered by length, and words of same length are ordered lexicographically for some underlying linear order of the alphabet. For a finite word u and a finite or infinite word v,

[1] We use Büchi acceptance on transitions, i.e., a run is accepting if it passes infinitely many accepting transitions.

we write $u \sqsubseteq v$ if $ux = v$ for some $x \in \Sigma^*$ or $x \in \Sigma^\omega$, respectively, and call u a *prefix* of v. We use $\mathsf{Prf}(w)$ to denote the set of all prefixes of a finite or infinite word w and extend this notation to a set X of words in the natural way, i.e. $\mathsf{Prf}(X) = \bigcup_{w \in X} \mathsf{Prf}(w)$.

We say that $(u, v) \in \Sigma^* \times \Sigma^+$ is a *representation of* $w \in \Sigma^\omega$ if $uv^\omega = w$. If an ω-word w has such a representation, we call w *ultimately periodic*, and we denote the set of all possible representations $\Sigma^* \times \Sigma^+$ with $\mathbb{UP}$. For a finite word u and a (ω)-language L, we define $u^{-1}L = \{w \mid uw \in L\}$. Note that if L is an ω-language, then $u^{-1}L$ is an ω-language as well. For a word w, we use $w[i, j]$ to denote the infix of w from position i to position j and set $w[i, j] = \varepsilon$ if $i > j$. We write $(u, v) \preceq (u', v')$ if $u \preceq u'$ or $u = u'$ and $v \preceq v'$. For a set $X \subseteq \mathbb{UP}$ of representations, we define $[\![X]\!]_\omega = \{uv^\omega \mid (u, v) \in X\}$.

A *deterministic transition system* (TS) is given as a tuple $\mathcal{T} = (Q, \Sigma, \iota, \delta)$, where Q is a finite, non-empty set of states, Σ is the finite alphabet, $\iota \in Q$ is the initial state and $\delta : Q \times \Sigma \to Q$ is the transition function. We use $|\mathcal{T}|$ to refer to the size of $\mathcal{T}$, which is the number of states in Q. The run of $\mathcal{T}$ on some word $w = w_0 w_1 \ldots \in \Sigma^* \cup \Sigma^\omega$ is the unique (possibly infinite) sequence $\rho = \mathsf{run}(\mathcal{T}, w) = q_0 w_0 q_1 w_1 \ldots$ with $q_0 = \iota$ and $q_{i+1} = \delta(q_i, w_i)$ for $i < |w|$. For a finite word w, we use $\mathcal{T}(w)$ to denote the last state of $\mathsf{run}(\mathcal{T}, w)$. We define the *infinity set*, referred to as $\mathsf{inf}(\rho)$, of a run $\rho = q_0 w_0 q_1 w_1 \ldots$, as the set containing all pairs (q, a) such that there exist infinitely many positions i with $q_i = q$ and $w_i = a$. For a finite run $\rho = q_0 w_0 \ldots w_{n-1} q_n$ and a set $X \subseteq Q \times \Sigma$, we use $\rho \cap X$ to denote the transitions of ρ that are in X, i.e. $\rho \cap X = \{(q_i, w_i) \in X \mid i < n\}$.

By equipping $\mathcal{T}$ with a set of *final states* $F \subseteq Q$, we obtain a *deterministic finite automaton* (DFA) $\mathcal{A} = (Q, \Sigma, \iota, \delta, F)$. The language accepted by $\mathcal{A}$ is denoted as $L(\mathcal{A})$ and contains all words $w \in \Sigma^*$ such that $\mathcal{T}(w) \in F$. By combining a transition system $\mathcal{T}$ with a *Büchi condition* (on transitions), which is a set $\beta \subseteq Q \times \Sigma$, we obtain a *deterministic Büchi automaton* (DBA) $\mathcal{A} = (Q, \Sigma, \iota, \delta, \beta)$. Its accepted language $L(\mathcal{A})$ contains all infinite words w such that $\mathsf{inf}(\mathsf{run}(\mathcal{T}, w)) \cap \beta \neq \emptyset$. When we depict a DBA with acceptance condition β in a figure, we underline the label a of a transition leaving the state q, if $(q, a) \in \beta$. We can use the terminology for transition systems also for automata by simply ignoring the acceptance component. Similarly, the size of an automaton is equal to the size of the underlying transition system.

The class of DBA-recognizable languages is a strict subclass of the class of the regular ω-languages, which are defined in terms of nondeterministic Büchi automata. We do not detail the definition here because it is not relevant for our paper (see [24] for a survey of the topic). It is well known that two regular ω-languages, and hence also two DBA-recognizable languages, are equal iff they coincide on the ultimately periodic words [11] (see also [12]).

We call a relation $\sim \subseteq \Sigma^* \times \Sigma^*$ a *right congruence* if $u \sim w$ implies $ua \sim wa$ for all $a \in \Sigma$. For $u \in \Sigma^*$ we write $[u]_\sim$ for the set of all $v \in \Sigma^*$ with $u \sim v$, also referred to as the *equivalence class* of u. For a language $L \subseteq \Sigma^*$ or $L \subseteq \Sigma^\omega$, we define the right congruence of L, denoted by $\sim_L$, as $u \sim_L v$ iff $u^{-1}L = v^{-1}L$.

A right congruence $\sim$ defines the canonical TS $\mathcal{T}_\sim = (Q_\sim, \Sigma, [\varepsilon]_\sim, \delta_\sim)$ where $Q_\sim$ is the set of classes in $\sim$ and $\delta([u]_\sim, a) = [ua]_\sim$. Due to this correspondence, we write $c \in Q_\sim$ to denote that c is a class of $\sim$. Similarly, we can associate with every TS $\mathcal{T}$ a congruence $\sim_\mathcal{T} \subseteq \Sigma^* \times \Sigma^*$, where $u \sim_\mathcal{T} v$ iff $\mathcal{T}(u) = \mathcal{T}(v)$. Let p, q be states of some automaton $\mathcal{A}$ that accepts the language L, and let c be a class of $\sim_L$. By abuse of notation, we use $q \in c$ if $\mathcal{A}$ reaches q on some word $u \in c$. Further, we write $p \sim q$ if $p \in c$ and $q \in c$ for some class c. We use L_c for a class c of $\sim_L$ to denote the language $u^{-1}L$ for a $u \in c$.

An *ω-sample* is a pair $S = (S_+, S_-)$ with $S_+, S_- \subseteq \mathbb{UP}$ and $[\![S_+]\!]_\omega \cap [\![S_-]\!]_\omega = \emptyset$. We refer to words in S_+ as *positive*, while those in S_- are called negative. Since we only use ω-samples in our algorithm, we often simply write sample instead of ω-sample. We only consider finite samples, and the size of S, denoted as $|S|$, is defined as the sum of all $|u| + |v|$ for $(u, v) \in S_+ \cup S_-$. Furthermore, we call a sample $S = (S_+, S_-)$ consistent with a language L, if $L \cap [\![S_-]\!]_\omega = \emptyset$ and $[\![S_+]\!]_\omega \subseteq L$. Similarly, an automaton $\mathcal{A}$ is called *consistent* with S if $L(\mathcal{A})$ is consistent with S. We say that $u, v \in \Sigma^*$ are *separated by* S if there exists a word $w \in \Sigma^\omega$ such that $uw, vw \in \big([\![S_+]\!]_\omega \cup [\![S_-]\!]_\omega\big)$ and $\big(uw \in [\![S_+]\!]_\omega \Leftrightarrow vw \in [\![S_-]\!]_\omega\big)$. A DFA $\mathcal{D}$ is called *prefix-consistent* with the ω-sample $S = (S_+, S_-)$ if for all $w \in [\![S_+]\!]_\omega$ we have that $\mathsf{Prf}(w) \cap L(\mathcal{D}) \neq \emptyset$ and $\mathsf{Prf}([\![S_-]\!]_\omega) \cap L(\mathcal{D}) = \emptyset$.

A right congruence $\sim$ is consistent with S if $u \not\sim v$ for all $u, v \in \Sigma^*$ that are separated by S. In a *partial* TS $\mathcal{T}$, the transition function δ can have undefined values. While the definition of run also applies to partial TS, it is possible that the run of a partial TS on some words is undefined. A partial TS $\mathcal{T}$ has a conflict with a sample S if there are $u, v \in \Sigma^*$ that are separated by S but lead to the same state in $\mathcal{T}$ (which in particular means that the runs of $\mathcal{T}$ on u and v must exist). Note that $\sim$ is consistent with S iff the corresponding TS $\mathcal{T}_\sim$ has no conflict with S.

A passive learner (for DBAs) is a function f that maps ω-samples to DBAs. f is called a polynomial-time learner if f can be computed in polynomial time. A learner f is *consistent* if it constructs from each ω-sample $S = (S_+, S_-)$ a DBA $\mathcal{A}$ such that S is consistent with $L(\mathcal{A})$. We say that f can learn every DBA language in the limit if for each DBA language L there is a *characteristic sample* S^L such that $L(f(S^L)) = L$ and $f(S^L) = f(S)$ for each sample S that is consistent with L and contains S^L (so f produces the same DBA for L for all samples containing S^L). For a class $\mathcal{C}$ of DBA languages we say that f can learn every language in $\mathcal{C}$ in the limit *from polynomial data* if the characteristic samples for the languages in $\mathcal{C}$ are of polynomial size (in the smallest DBA for the corresponding language).

We also consider the standard active learning scenario for DFAs, in which the learning algorithm can obtain information on the target language L by posing membership and equivalence queries to an oracle [1]. In the following let L be a regular language. An oracle T for L answers a membership query $\mathsf{mem}(w)$ with either "yes" or "no", depending on whether the w is in the target language L or not. In an equivalence query $\mathsf{equiv}(\mathcal{A}_H)$, the active learner proposes a hypothesis $\mathcal{A}_H$ to T. If $\mathcal{A}_H$ accepts precisely the target language L, the oracle returns "yes". Otherwise, T answers "no" and additionally provides the learner with a word from the symmetric difference of $L(\mathcal{A}_H)$ and L, which serves as a counterexample. We make use of the following result.

▶ **Theorem 1** ([1], Theorem 6). *Let L be a regular language and T be an oracle for L. There exists an active learner* AL *that returns the minimal DFA $\mathcal{A}_L$ for L. Moreover, the runtime of* AL *is polynomial in the size of the minimal DFA and the length of the longest counterexample provided by T, and the size of all hypotheses used in equivalence queries is polynomial in the size of $\mathcal{A}_L$.*

3 A Learner for DBAs

In this section we introduce DBAInf (for DBA inference), a passive learner for deterministic Büchi automata, and describe how it constructs a DBA from a finite ω-sample. We start with an informal description of the underlying idea, then provide a formal description of the learning algorithm, and finally show that DBAInf is a consistent polynomial time learner for the class of DBAs.

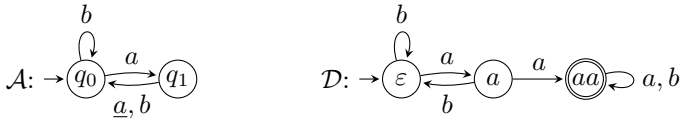


Figure 1 Illustrations for the high-level description of DBAInf. On the left a DBA $\mathcal{A}$ accepting the words in which the infix aa appears infinitely often. The DFA $\mathcal{D}$ on the right is constructed in an intermediate step of DBAInf from the ω-sample given in Example 2.

High-Level Description of the Idea

DFA-learners for languages of finite words are based on the fact that for the minimal DFA of a regular language, there is a one-to-one correspondence between states and equivalence classes of the right congruence of the language. DFA-learners basically extract a right congruence from the sample (consisting of finite words) that is then used for defining the transition system for the resulting DFA. The central obstacle in passive learning for ω-automata is the lack of such a one-to-one correspondence between states and equivalence classes. For example, the language L over the alphabet $\{a, b\}$ consisting of all words in which the infix aa occurs infinitely often cannot be recognized by a DBA with only one state. However, L has only one equivalence class, since for all $u \in \Sigma^*, w \in \Sigma^\omega$ we have $uw \in L$ if and only if $w \in L$. Thus, given an ω-sample that is consistent with L, the extracted right congruence will only have one class since the ω-sample does not separate any words.

We explain the idea underlying DBAInf for languages with only one equivalence class such as L, that is, if the sample does not separate any two words. The formal description then also covers the general case.

Basically, DBAInf attempts to identify specific subwords, also referred to as the *positive patterns of* L in the following description, that appear infinitely often precisely in words belonging to L. Consider, for example, the DBA $\mathcal{A}$ depicted in Figure 1 on the left, which accepts L. Regardless of which state $\mathcal{A}$ is in, reading the pattern aa makes $\mathcal{A}$ use an accepting transition, which means aa is a positive pattern of L. On the right-hand side of Figure 1 a DFA $\mathcal{D}$ that accepts all positive patterns of L is depicted. From $\mathcal{D}$ we can now obtain a DBA for L, which we refer to as $\mathsf{DBA}(\mathcal{D})$, by replacing each transition leading into a final state (state aa in the example) with an accepting transition that leads to the initial state (state ε in the example). In the example in Figure 1, the operation $\mathsf{DBA}(\mathcal{D})$ returns a DBA isomorphic to $\mathcal{A}$.

The core idea of DBAInf is to learn such a DFA for the positive patterns. For this purpose, given some ω-sample $S = (S_+, S_-)$, DBAInf uses an active learning algorithm AL for DFAs as black-box, and answers the queries of this algorithm based on S. To ensure that $\mathsf{DBA}(\mathcal{D})$ is consistent with S, we require that the active learning algorithm learns a DFA $\mathcal{D}$ that does not accept any infix of a loop of a negative example, and that for any position in a positive example an infix starting at this position is accepted. So whenever AL asks an equivalence query for a hypothesis $\mathcal{D}$, DBAInf checks whether $\mathcal{D}$ satisfies this condition and stops if yes. Otherwise, a positive or negative example is found on which $\mathcal{D}$ violates the condition and a corresponding finite word is returned.

▶ **Example 2.** Consider an ω-sample $S = (S_+, S_-)$ with $[\![S_+]\!]_\omega = \{(aba)^\omega\}$ and $[\![S_-]\!]_\omega = \{b^\omega, (ab)^\omega\}$. Recall that $[\![S_+]\!]_\omega$ refers to the ω-words, while S_+ itself is specified by pairs (u, v) of finite words representing uv^ω. The precise representation of the ultimately periodic words in $[\![S_+]\!]_\omega$ and $[\![S_-]\!]_\omega$ does not play any role. In the first step, DBAInf infers a right congruence that is consistent with S. Since S does not separate any finite words, this

Input: An ω-sample $S = (S_+, S_-)$.
Output: The deterministic Büchi automaton $\mathcal{A}$.
1. Extract a right congruence $\sim$ from S by constructing a TS $\mathcal{T}_\sim$
2. Build a sample $S^{(c)}$ for each $\sim$-class c
3. Compute a DFA $\mathcal{D}_c = \mathsf{PrfCons}(S^{(c)})$ for each class c of $\sim$
4. Return the DBA $\mathcal{A} := \mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in \sim})$

Figure 2 An overview of the algorithm DBAInf. The individual steps are explained in the text.

right congruence has only one class $c = [\varepsilon]_\sim$. For this class, DBAInf now uses an active learning algorithm AL for DFAs. In order to answer the queries of AL, DBAInf constructs an ω-sample $S^{(c)}$ with $S^{(c)}_+ = \{(\varepsilon, aab), (\varepsilon, aba), (\varepsilon, baa)\}$, representing all the suffixes of $[\![S_+]\!]_\omega$, and $S^{(c)}_- = \{(\varepsilon, b), (\varepsilon, ab), (\varepsilon, ba)\}$, representing all the suffixes of $[\![S_-]\!]_\omega$ starting at a position in a loop (which is the same in this example as all suffixes of $[\![S_-]\!]_\omega$). The precise execution of DBAInf now depends on the active learning algorithm AL that is used. We do not detail a precise active learner here but simply explain how queries would be answered. The queries that we use in this example correspond to a version of Angluin's algorithm by Rivest and Schapire [22], see also [7, Section 19.4].

AL starts by asking membership queries for ε, a, b, which are all answered negatively because these are all prefixes of $[\![S^{(c)}_-]\!]_\omega$ (and hence infixes of loops of negative examples from S). Then AL asks an equivalence query for the DFA that rejects all words. Since this DFA does not accept prefixes of all words in $[\![S^{(c)}_+]\!]_\omega$, DBAInf selects the smallest $(u, v) \in S^{(c)}_+$ such that no prefix of uv^ω is accepted by the DFA. In this example this is (ε, aab). The counterexample that is given to AL is the shortest prefix of $[\![(\varepsilon, aab)]\!]_\omega = (aab)^\omega$ that is not a prefix of $[\![S^{(c)}_-]\!]_\omega$, in this case aa. After this counterexample, AL asks a few more membership queries, namely for $ba, aaa, ab, aba, aaaa, aaba$ which are answered negatively for prefixes of $[\![S^{(c)}_-]\!]_\omega$ (that is, ba, ab, aba), and positively for all other words. With this information, AL asks an equivalence query for the DFA $\mathcal{D}$ shown in Figure 1. Since $\mathcal{D}$ accepts a prefix for each word in $[\![S^{(c)}_+]\!]_\omega$ and rejects all prefixes of words in $[\![S^{(c)}_-]\!]_\omega$, the execution of AL ends. Then DBAInf returns the DBA obtained by the operation $\mathsf{DBA}(\mathcal{D})$, which is isomorphic to $\mathcal{A}$ in Figure 1. ◀

This idea can be generalized to more than one equivalence class by first extracting a right congruence $\sim$ from S that is consistent with S, and then learning one DFA for each equivalence class c of $\sim$ with a more refined definition of the samples $S^{(c)}$ that is illustrated for a single class in Example 2. These DFAs are combined into a DBA by taking the product of $\mathcal{T}_\sim$ with the union of the $\mathcal{D}_c$, and redirecting transitions of the DFAs that lead to an accepting state into the initial state of the $\mathcal{D}_c$ for the current class c given by $\mathcal{T}_\sim$. The details are given in the formal description of the algorithm.

Formal Description of the Learner

The overall structure of the algorithm DBAInf in shown in Figure 2. The individual steps are described in more detail below. In the description, $S = (S_+, S_-)$ always refers to the ω-sample that is the input for DBAInf. After the description, we illustrate the steps with an example.

Step 1. The algorithm DBAInf starts by constructing a right congruence $\sim$ that is consistent with S, represented by a TS $\mathcal{T}_\sim = (Q_\sim, \Sigma, [\varepsilon]_\sim, \delta_\sim)$. For obtaining an algorithm that can learn every DBA language in the limit, it is important to use a method that can infer the right

congruence $\sim_L$ of any DBA language L in the limit. This is possible by slightly modifying one of the algorithms described in [5, 10], which learn ω-automata with an acceptance condition, while we are only interested in a right congruence in this step. Because of this different setting, we briefly describe a possible construction for $\mathcal{T}_\sim$ similar to the algorithm in [10]. The transition system $\mathcal{T}_\sim$ is built incrementally by considering prefixes of positive examples in $[\![S_+]\!]_\omega$ that exit the transition system (via an undefined transition). In order to guarantee termination in polynomial time, only prefixes up the length $k \cdot \ell^2$ are considered, with $k = \max\{|u| \mid (u, v) \in S_+\}$ and $\ell = \max\{|v| \mid (u, v) \in S_+\}$. The construction of $\mathcal{T}_\sim$ proceeds as follows:

- Start with only the initial state $\varepsilon \in Q_\sim$ and no transitions. Then repeat:
  - Pick the smallest u and a in canonical order such that ua is a prefix of a positive example word, $|ua| \leq k \cdot \ell^2$, and $\delta_\sim(u, a)$ is not yet defined. If no such ua exists, exit the loop.
  - If there is $v \in Q_\sim$ such that setting $\delta_\sim(u, a) = v$ does not lead to a conflict of $\mathcal{T}_\sim$ with S, then define $\delta_\sim(u, a) = v$ for the least such v in canonical order. Otherwise, add ua to $Q_\sim$ and let $\delta_\sim(u, a) = ua$.
- If there remain positive examples in $[\![S_+]\!]_\omega$ for which the run in $\mathcal{T}_\sim$ is not defined (because of missing transitions), complete $\mathcal{T}_\sim$ by adding appropriate paths to disjoint loops for all such elements of $[\![S_+]\!]_\omega$.
- Finally, add a sink state as target for all remaining undefined transitions.

This construction runs in polynomial time (because of the limit on the length of the considered ua, and since all tests can be carried out in polynomial time). By construction, $\mathcal{T}_\sim$ has no conflict with S, and hence the associated right congruence $\sim$ is consistent with S, as required. In the following, we identify states in $Q_\sim$ with the corresponding equivalence classes of $\sim$.

Step 2. For each equivalence class $c \in Q_\sim$, we define a sample $S^{(c)} = (S_+^{(c)}, S_-^{(c)})$ based on the infixes of example words starting in positions i such that $\mathcal{T}_\sim(w[1, i-1]) = c$. Formally, we have that $S_+^{(c)}$ contains for each $xw \in [\![S_+]\!]_\omega$ with $\mathcal{T}_\sim(x) = c$ the minimal $(u, v) \in \mathbb{UP}$ such that $w = uv^\omega$. We define $S_-^{(c)}$ to contain for each $xw \in [\![S_-]\!]_\omega$ with $\mathcal{T}_\sim(x) = c$ the minimal (ε, v) such that $w = v^\omega$ and $\mathcal{T}_\sim(xv) = c$, if such a word v exists.

Note that this definition formally ranges over infinitely many x, but for each (u, v) in S it suffices to consider the prefixes x of length at most $|u| + |v||Q_\sim|$ because the run of uv^ω in $\mathcal{T}_\sim$ has an initial part of length at most u followed by a period of length at most $|v||Q_\sim|$. This observation gives a polynomial bound on the number of elements in $S^{(c)}$ and also on their size.

Step 3. In the third step, DBAInf constructs for each class c of $\sim$ a DFA $\mathcal{D}_c$ that is prefix-consistent with $S^{(c)}$. This construction is achieved by passing $S^{(c)}$ to the algorithm PrfCons, which is depicted in Algorithm 1. Given an ω-sample $R = (R_+, R_-)$, PrfCons uses a polynomial-time active learning algorithm AL for DFAs, and answers the queries of AL based on R. We assume that R_- only contains periodic words, which is satisfied by the samples $S^{(c)}$ on which PrfCons is applied.

▶ **Lemma 3.** *For every ω-sample $R = (R_+, R_-)$ with $R_- \subseteq \{\varepsilon\} \times \Sigma^+$, PrfCons terminates in polynomial time and returns a DFA that is prefix-consistent with R.*

Proof. We first show that all answers given to AL during the execution of PrfCons are consistent with the language $P = \Sigma^* \setminus \mathsf{Prf}([\![R_-]\!]_\omega)$: If a membership query $\mathsf{mem}(x)$ is answered positively, then $x \notin \mathsf{Prf}([\![R_-]\!]_\omega)$ and hence $x \in P$. Analogously, a negative answer

Algorithm 1 PrfCons.

Input: An ω-sample $R = (R_+, R_-)$ with $R_- \subseteq (\{\varepsilon\} \times \Sigma^+)$.
Output: A DFA $\mathcal{D}$ that is prefix-consistent with R.
Simulate an active learning algorithm AL for DFAs that satisfies the properties of Theorem 1, and answer its queries as follows:

Query `mem`(x)
  if $x \in \mathsf{Prf}([\![R_-]\!]_\omega)$ **then**
    answer "no"
  else
    answer "yes"
Query `equiv`$(\mathcal{D})$
  if $\mathcal{D}$ *is prefix-consistent with* R **then**
    stop simulation and output $\mathcal{D}$
  else if *there exists a* $w \in [\![R_+]\!]_\omega$ *with* $\mathsf{Prf}(w) \cap L(\mathcal{D}) = \emptyset$ **then**
    pick the minimal $(u, v) \in R_+$ such that $uv^\omega \cap L(\mathcal{D}) = \emptyset$
    return *shortest* $x \sqsubseteq uv^\omega$ *with* $x \notin \mathsf{Prf}([\![R_-]\!]_\omega)$ *as counterexample*
  else
    let $(\varepsilon, v) \in R_-$ be minimal such that $\mathsf{Prf}(v^\omega) \cap L(\mathcal{D}) \neq \emptyset$
    return *shortest prefix* x *of* v^ω *with* $x \in L(\mathcal{D})$ *as counterexample*

to $\mathsf{mem}(x)$ implies $x \in \mathsf{Prf}([\![R_-]\!]_\omega)$, meaning $x \notin P$. For an equivalence query $\mathsf{equiv}(\mathcal{D})$, we distinguish the two cases in PrfCons. In the first case, the returned counterexample x is not accepted by $\mathcal{D}$ and is not in $\mathsf{Prf}([\![R_-]\!]_\omega)$. So $x \in P$ and giving x as counterexample means that it should be accepted. In the other case, the query is answered with a prefix $x \in \mathsf{Prf}(v^\omega) \cap L(\mathcal{D})$ for some $(\varepsilon, v) \in R_-$ such that x is accepted by $\mathcal{D}$. So $x \notin P$ and should be accepted. Hence, all answers provided to AL are consistent with the language P.

Since a DFA that accepts P is prefix-consistent with R, the execution of AL either terminates with a DFA for P, or it terminates earlier if a prefix-consistent DFA $\mathcal{D}$ is used in an equivalence query. Therefore, all hypotheses $\mathcal{D}$ used by AL are of size polynomial in the size of a minimal DFA for P (see Theorem 1). It is not hard to verify that P can be accepted by a DFA whose size is polynomial in $|R|$ using the prefixes of $[\![R_-]\!]_\omega$ as non-accepting states, and introducing a loop on v for each $(\varepsilon, v) \in R_-$ when reaching a prefix of v^ω that is not a prefix of any other word in $[\![R_-]\!]_\omega$. Adding an accepting sink for all missing transitions results in a DFA for P. (A similar construction is described in more detail in [5] for the construction of the "table look-up DPA".)

To conclude that the execution of AL terminates in polynomial time, it remains to verify that the lengths of the provided counterexamples are indeed polynomial in $|R|$. The counterexample for an equivalence query on $\mathcal{D}$ is a shortest word that is prefix of an example from R and is accepted/rejected by $\mathcal{D}$ (depending on the case). These words are clearly polynomial in the size of R and $\mathcal{D}$, and hence polynomial in the size of R. ◀

Step 4. The constructed DFAs are combined into a DBA by taking the product of $\mathcal{T}_\sim$ with the union of the $\mathcal{D}_c$, and redirecting transitions of the DFAs that lead to an accepting state into the initial state of the $\mathcal{D}_c$ for the current class c given by $\mathcal{T}_\sim$.

Formally, we assume that the state sets of the $\mathcal{D}_c$ are pairwise disjoint, and define $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim}) = (Q, \Sigma, \iota, \delta, \beta)$ as follows:

- $Q := Q_\sim \times \bigcup_{c \in Q_\sim} Q_c$
- $\iota := ([\varepsilon]_\sim, \iota_{[\varepsilon]_\sim})$ (note that $\iota_{[\varepsilon]_\sim}$ is the initial state of $\mathcal{D}_{[\varepsilon]_\sim}$)

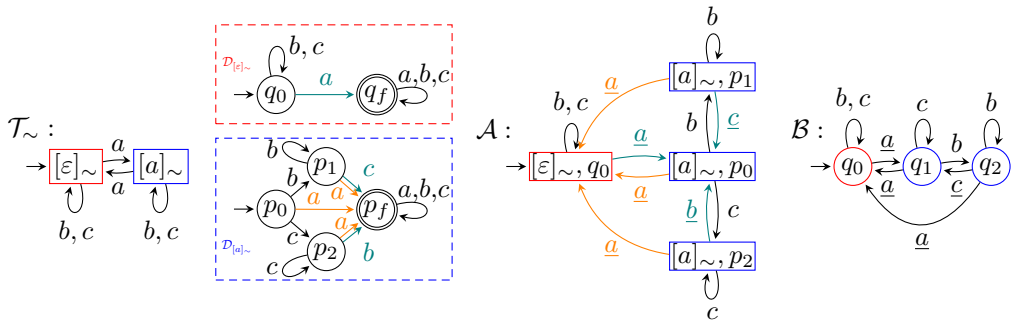


Figure 3 Illustrations for the execution of DBAInf on the sample S given in Example 4.

- For $(c, q) \in Q$ and $a \in \Sigma$ let $q' = \delta_{\hat{c}}(q, a)$ for the unique $\hat{c} \in Q_\sim$ with $q \in Q_{\hat{c}}$, and let $c' = \delta_\sim(c, a)$. Then
  - $\delta((c, q), a) = \begin{cases} (c', q') & \text{if } q' \notin F_{\hat{c}} \\ (c', \iota_{c'}) & \text{if } q' \in F_{\hat{c}} \end{cases}$
  - $((c, q), a) \in \beta$ if $q' \in F_{\hat{c}}$.

This DBA is returned by the algorithm DBAInf. Note that syntactically, a tuple of the form $(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim})$ is the same as a family of DFAs, which are considered for active learning of ω-languages in [3]. The semantics is, however, very different. Families of DFAs can represent all regular ω-languages but turning them into a deterministic ω-automaton for the represented language involves an exponential blow-up, in general [2]. So the way we use these tuples of DFAs here is not related to these families of DFAs. Before we prove that DBAInf is a consistent polynomial time DBA-learner, we illustrate it with an example.

▶ **Example 4.** As an example for the execution of DBAInf, consider $S = (S_+, S_-)$ with

$[\![S_+]\!]_\omega = \{a^\omega, (ba)^\omega, (ca)^\omega, a(bc)^\omega, a(bcb)^\omega, a(cbc)^\omega\}$ and
$[\![S_-]\!]_\omega = \{b^\omega, c^\omega, (bc)^\omega, aa(bc)^\omega, ab^\omega, ac^\omega\}$.

DBAInf first extracts a right congruence $\sim$ consisting of two equivalence classes. The corresponding transition system $\mathcal{T}_\sim$ is shown in Figure 3 on the left. The words ε and a are separated by S, since $a(bc)^\omega \in S_+$ and $(bc)^\omega \in S_-$. The b, c-loops on $[\varepsilon]_\sim$ do not introduce a conflict, as well as the a-transition from $[a]_\sim$ to $[\varepsilon]_\sim$. The b-transition from $[a]_\sim$ cannot go back to $[\varepsilon]_\sim$ because this would introduce a conflict for the words b and ab, which are separated by S, again because of $a(bc)^\omega \in S_+$ and $(bc)^\omega \in S_-$. Hence, there is a b-loop on $[a]_\sim$. Similarly, for the c-transition from $[a]_\sim$. The resulting transition system is complete, so no further states have to be added.

In the second step, the samples $S^{[\varepsilon]_\sim}$ and $S^{[a]_\sim}$ are computed. The positive components of these samples are $S_+^{[\varepsilon]_\sim} = \{(\varepsilon, a), (\varepsilon, ba), (\varepsilon, ab), (\varepsilon, ca), (\varepsilon, ac), (a, bc), (a, bcb), (a, cbc)\}$ and $S_+^{[a]_\sim} = \{\varepsilon\} \times \{a, ba, ab, ca, ac, bc, cb, bcb, bbc, cbb, cbc, bcc, ccb\}$.

To illustrate how these samples are constructed, consider $(ba)^\omega \in S_+$. It induces the sequence of classes $([\varepsilon]_\sim[\varepsilon]_\sim[a]_\sim[a]_\sim)^\omega$ in $\mathcal{T}_\sim$ and thus contributes (ε, ba) and (ε, ab) to both $S_+^{[\varepsilon]_\sim}$ and $S_+^{[a]_\sim}$. The example $a(bcb)^\omega$ induces the sequence $[\varepsilon]_\sim([a]_\sim)^\omega$, and hence contributes (a, bcb) to $S_+^{[\varepsilon]_\sim}$, and $\{\varepsilon\} \times \{bcb, cbb, bbc\}$ to $S_+^{[a]_\sim}$. Similarly, for the other positive examples.

The negative components are $S_-^{[\varepsilon]_\sim} = \{\varepsilon\} \times \{b, c, bc, cb\}$ and $S_-^{[a]_\sim} = \{\varepsilon\} \times \{b, c\}$. For example, $aa(bc)^\omega$ induces the class sequence $[\varepsilon]_\sim[a]_\sim([\varepsilon]_\sim)^\omega$. This adds (ϵ, bc) and (ε, cb) to $S_-^{[\varepsilon]_\sim}$ and nothing to $S_-^{[a]_\sim}$ because the only position with class $[a]_\sim$ after the first a is not in the periodic part, and hence there is no v such that $av^\omega = aa(bc)^\omega$. Similarly, for the other negative examples.

Executing PrfCons on these samples gives rise to two DFAs $\mathcal{D}_{[\varepsilon]_\sim}$ and $\mathcal{D}_{[a]_\sim}$ with 2 and 4 states, respectively, which are depicted in the middle of Figure 3. Next to that (on the right) in Figure 3, the DBA $\mathcal{A}$ that is the result of $\mathsf{DBA}(\mathcal{T}_\sim, \mathcal{D}_{[\varepsilon]_\sim}, \mathcal{D}_{[a]_\sim})$ is shown. The states of $\mathcal{A}$ are pairs of states of $\mathcal{T}_\sim$ and of the union of $\mathcal{D}_{[\varepsilon]_\sim}$ and $\mathcal{D}_{[a]_\sim}$. Accepting transitions that reset into $\mathcal{D}_{[\varepsilon]_\sim}$ are depicted in orange and those that reset into $\mathcal{D}_{[a]_\sim}$ are colored teal. On the very right of Figure 3, a minimal DBA $\mathcal{B}$ that accepts the same language as $\mathcal{A}$ is depicted. The states of $\mathcal{B}$ are colored according to the class of $\sim$ that they belong to.

Consistency

We now prove that DBAInf always infers a DBA that is consistent with the given ω-sample.

▶ **Theorem 5.** DBAInf *is a consistent polynomial time DBA-learner.*

Proof. Let $S = (S_+, S_-)$ be an ω-sample and $\sim$ be a right congruence that is consistent with S, as constructed by DBAInf in the first step (represented by $\mathcal{T}_\sim = (Q_\sim, \Sigma, [\varepsilon]_\sim, \delta_\sim)$). As in Figure 2, let $\mathcal{D}_c$ be the DFA constructed from $S^{(c)}$ for each class c of $\sim$. Further, let $\mathcal{A} = \mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim})$ be the DBA returned by DBAInf. The extraction of $\mathcal{T}_\sim$ and the construction of the samples $S^{(c)}$ can be done in polynomial time. By Lemma 3 PrfCons runs in polynomial time for each class c and returns a DFA $\mathcal{D}_c$ whose size is polynomial in $|S|$. Hence, the construction of $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c \in Q_\sim})$ and thus the overall procedure DBAInf can be completed in polynomial time.

For consistency, first note that in the construction of $\mathcal{A}$, the first component of the states is always updated according to $\delta_\sim$. Therefore, for each $u \in \Sigma^*$, the state (c, q) reached by $\mathcal{A}$ after reading u is such that c is the class of u, i.e., $\mathcal{T}_\sim$ reaches c when reading u.

We start by showing that $\mathcal{A}$ rejects all words in S_-. Let $w = uv^\omega$ for a $(u, v) \in S_-$. As w is ultimately periodic, we can write the run of $\mathcal{A}$ on w as $\rho\pi^\omega$. Pick a position $i \geq \max\{|u|, |\rho|\}$ such that $\mathcal{A}$ uses an accepting transition when reaching position i in w. If no such position exists, then $\mathcal{A}$ uses only finitely many accepting transitions and hence rejects w. It is easily verified that if such a position i exists, we can write $uv^\omega = xy^\omega$ and $\rho\pi^\omega = \hat{\rho}(\hat{\pi})^\omega$ such that $|x| = |\hat{\rho}| = i$ and $|y| = |\hat{\pi}|$. Let c be the class reached after reading x, meaning $\mathcal{T}_\sim(x) = c$ and $\mathcal{A}$ is in state (c, ι_c) after reading x. This means $\hat{\pi}$ starts and ends in (c, ι_c) and thus $\delta_\sim(c, y) = c$. Since $xy^\omega = uv^\omega \in [\![S_-]\!]_\omega$, we have $(\varepsilon, y) \in S_-^{(c)}$ by the definition of $S^{(c)}$. Because the DFA $\mathcal{D}_c$ is prefix-consistent with $S^{(c)}$ by Lemma 3, it does not accept any prefix of y^ω. Thus, $\mathcal{D}_c$ will never reach a final state when reading the suffix of w starting at position i. By definition this means $\mathcal{A}$ uses no accepting transitions after position i in w, and hence rejects w.

Now let us show that $\mathcal{A}$ accepts all words in S_+. Let $w = uv^\omega \in [\![S_+]\!]_\omega$ and let ρ be the run of $\mathcal{A}$ on w. Let i be a position such that ρ is in a state of the form (c, ι_c) at position i. Then $\mathcal{A}$ uses an accepting transition after position i: If ρ is in state (c, ι_c) at position i, this means that also $\mathcal{T}_\sim(w[1, i-1]) = c$, and we can find words $u \in \Sigma^*, w' \in \Sigma^\omega$ such that $|u| = i$ and $w = uw'$. By definition of $S_+^{(c)}$, we have that $w' \in [\![S_+^{(c)}]\!]_\omega$. Since $\mathcal{D}_c$ is prefix-consistent with $S^{(c)}$, there exists a prefix $x \sqsubseteq w'$ that is accepted by $\mathcal{D}_c$. Hence, $\mathcal{A}$ uses an accepting transition on the prefix x of w'. From that we can conclude that ρ uses infinitely many accepting transitions since ρ starts in $([\varepsilon]_\sim, \iota_{[\varepsilon]_\sim})$ and each accepting transition ends in a state of the form (c, ι_c) for a class c of $\sim$. ◀

4 Completeness of the Learning Algorithm

In this section we establish that the class of DBA languages can be learned in the limit by DBAInf. For obtaining this completeness result, we show how to construct a sample S^L for a DBA language L such that DBAInf constructs a DBA for L from each sample S that

contains S^L and is consistent with L. We separate the construction of S^L in two parts. The first part ensures that DBAInf infers the right congruence $\sim_L$ in the first step. The second part ensures that, based on the correct right congruence $\sim_L$, a DBA for L is constructed. We keep the first part short as the idea for this is the same as for RPNI on finite words [21] and the passive learner for ω-automata from [9]. Roughly, the sample has to cover all transitions and states of $\mathcal{T}_{\sim_L}$ and provide examples that separate all different states. This has to be done in a specific way using minimal words reaching the states and transitions of $\mathcal{T}_{\sim_L}$.

▶ **Lemma 6.** *For every DBA language L there exists an ω-sample $S^{\sim_L}$ such that $|S^{\sim_L}|$ is polynomial in the size of L and* DBAInf *extracts $\sim_L$ from every sample that is consistent with L and contains $S^{\sim_L}$.*

The remainder of this section is about the second part of the construction of S^L: DBAInf returns a DBA that consists of smaller DFAs $\mathcal{D}_c$ for each class c of the extracted congruence $\sim$. We want to ensure that these DFAs satisfy certain conditions that are captured in the definition of "safe" below.

▶ **Definition 7.** *Let L be a DBA language and*

$$\mathcal{K}(L) = \{u \in \Sigma^+ \mid \text{for all } y \in \Sigma^* \text{ if } (uy)^{-1}L = L \text{ then } (uy)^\omega \in L\}.$$

We call a DFA $\mathcal{D}$ safe for *L, if*
1. *for each word $w \in L$, $\mathcal{D}$ accepts some prefix of w, and*
2. *$L(\mathcal{D}) \subseteq \mathcal{K}(L)$.*

Our goal is to build the sample S^L in such a way that the samples $S^{(c)}$ constructed in the second step of DBAInf ensure that AL learns a DFA that is safe for L_c according to Definition 7 above. Then Lemma 10 further below shows that the DBA returned by DBAInf accepts L. For this purpose, we consider for each class c an execution of AL in which the answers given to the queries of AL are consistent with $\mathcal{K}(L_c)$. The answers used in this run of AL are used to define the part of the sample that ensures that $\mathsf{PrfCons}(S^{(c)})$ returns a DFA that is safe for L_c (see Algorithm 2, Lemma 11, and Lemma 12). In order to ensure that the execution of AL terminates, we need to show that $\mathcal{K}(L_c)$ is regular, which we do first in Lemma 8 and Lemma 9. For this we assume that $\mathcal{A} = (Q, \Sigma, \iota, \delta, \beta)$ is some DBA that accepts L. Let

$$\mathcal{K}_q(\mathcal{A}) = \{u \in \Sigma^* \mid \text{ for all } v \in \Sigma^* : \text{ if } q \xrightarrow{uv} q \text{ then } \mathsf{run}(\mathcal{A}, uv) \cap \beta \neq \emptyset\}$$

be the set of all words u, that do not lie on a rejecting loop starting in the state q. As an example consider the DBA $\mathcal{B}$ on the right of Figure 3. We have $\mathcal{K}_{q_0}(\mathcal{B}) = \Sigma^* a \Sigma^*$, $\mathcal{K}_{q_1}(\mathcal{B}) = \Sigma^*(a+b)\Sigma^*$, and $\mathcal{K}_{q_2}(\mathcal{B}) = \Sigma^*(a+c)\Sigma^*$. In general, a DFA for $\mathcal{K}_q(\mathcal{A})$ can be constructed by using $\mathcal{A}$ with initial state q, and one new accepting sink state to which all accepting transitions of $\mathcal{A}$ are redirected, and all transitions from states p such that each path from p to q contains an accepting transition.

▶ **Lemma 8.** *For every DBA $\mathcal{A}$ and each state $q \in Q$, the language $\mathcal{K}_q(\mathcal{A})$ is regular and can be recognized by a DFA of size at most $|\mathcal{A}| + 1$.*

Proof. To construct a DFA that recognizes $\mathcal{K}_q(\mathcal{A})$, we first extract the transition system $\mathcal{T}$ underlying $\mathcal{A}$. Then we remove from $\mathcal{T}$ all transitions which are accepting in $\mathcal{A}$. Subsequently, we build the set G consisting of all transitions $q \xrightarrow{a} p$ such that p and q lie in different SCCs of $\mathcal{T}$. This allows us to finally define the DFA $\mathcal{B} = (Q \cup \{\top\}, \Sigma, q, \delta', \{\top\})$ with

$$\delta'(q,a) = \begin{cases} p & \text{if } (q \xrightarrow{a} p) \notin (\beta \cup G) \\ \top & \text{otherwise.} \end{cases}$$

It is easily verified that the size of $\mathcal{B}$ is indeed linear in $|Q|$ as only one state is added. Further, we can show that $\mathcal{B}$ accepts $\mathcal{K}_q(\mathcal{A})$ by considering both directions of the mutual inclusion:

- Let $w \in L(\mathcal{B})$, then the run of $\mathcal{B}$ on w must end in the only final state, $\top$. This state can clearly only be reached if a transition $\tau \in beta \cup G$ is used. If $\tau \in \beta$, then $\mathcal{A}$ must use the accepting transition τ on w and hence any extension of w leading back to q is also guaranteed to use an accepting transition. Otherwise, $\tau \in G$ and thus from q the word w reaches a state $p \in \mathcal{T}$, which lies in a different SCC. This either happens if p and q lie in different SCCs in $\mathcal{A}$ or if all paths leading from p to q use at least one accepting transition in G. Clearly in both cases we have $w \in \mathcal{K}_q(\mathcal{A})$.
- On any word $w \in \mathcal{K}_q(\mathcal{A})$, the DBA $\mathcal{A}$ reaches from q some state p such that either p is in a different SCC or every path returning to q uses an accepting transition. In the former case the two states also lie in different SCCs and hence there exists some transition τ connecting the SCCs. By construction, however, we know that $\tau \in G$ and hence it is redirected to the final state $\top$ in $\mathcal{B}$. On the other hand if both states lie in the same SCC, then removing all accepting transitions guarantees that q can no longer be reached from p, as every path uses an accepting transition. Thus, p and q are in different SCCs in $\mathcal{T}$ and thus by an analogous argument, $\mathcal{B}$ accepts w. ◀

We can now express $\mathcal{K}(L_c)$ as intersection of the languages $\mathcal{K}_q(\mathcal{A})$ for states q in class c. As example, consider again the DBA $\mathcal{B}$ on the right of Figure 3. The states q_1 and q_2 belong to the class $[a]_\sim$. The intersection $\mathcal{K}_{q_1}(\mathcal{B}) \cap \mathcal{K}_{q_2}(\mathcal{B})$ consists of all words that contain a or contain both b and c. This is the language $\mathcal{K}(L(\mathcal{B})_{[a]_\sim})$.

▶ **Lemma 9.** *Let $\mathcal{A}$ be a DBA, $L = L(\mathcal{A})$ and c be a class of $\sim_L$. Then $\mathcal{K}(L_c) = \bigcap_{q \in c} \mathcal{K}_q(\mathcal{A})$, and in particular $\mathcal{K}(L_c)$ is regular.*

Proof. For the first inclusion of the equality, let $u \in \mathcal{K}(L_c)$. Assume to the contrary that there is some $q \in c$ with $u \notin \mathcal{K}_q(\mathcal{A})$. Then there is $v \in \Sigma^*$ such that in $\mathcal{A}$ we have $q \xrightarrow{uv} q$ without using an accepting transition. Note that $q \in c$ implies that $\mathcal{A}$ with initial state q accepts L_c. But then $(uv)^{-1}L_c = L_c$ and $(uv)^\omega \notin L_c$, which is a contradiction to $u \in \mathcal{K}(L_c)$.

For the inclusion from right to left, let $u \in \mathcal{K}_q(\mathcal{A})$ for all states q with $q \in c$. Consider an arbitrary $v \in \Sigma^*$ with $(uv)^{-1}L_c = L_c$. We need to show that $(uv)^\omega \in L_c$ because then $u \in \mathcal{K}(L_c)$. We pick some state $q_0 \in c$ and consider the run $\rho = q_0 \xrightarrow{uv} q_1 \xrightarrow{uv} q_2 \xrightarrow{uv} \dots$ of $\mathcal{A}$ on $(uv)^\omega$ from q_0. For all i, we have $q_i \in c$ since $(uv)^{-1}L_c = L_c$. There is some state q such that the set of indices $I = \{i \in \mathbb{N} \mid q = q_i\}$ is infinite. But then for any $i, j \in I$ with $i < j$ there must be an accepting transition between q_i and q_j in ρ by definition of $\mathcal{K}_q(\mathcal{A})$, and hence $(uv)^\omega \in \mathcal{K}_c(L)$.

Regularity of $\mathcal{K}(L_c)$ follows as it is a finite intersection of languages that are regular by Lemma 8. ◀

In the following Lemma, we show that if each constructed DFA $\mathcal{D}_c$ for a class c satisfies the conditions of Definition 7 with respect to the language L_c, then the resulting DBA accepts precisely L.

▶ **Lemma 10.** *Let $L \subseteq \Sigma^\omega$ be a DBA-recognizable language, and for each $c \in Q_{\sim_L}$ let $\mathcal{D}_c$ be a DFA that is safe for L_c. Then $\mathsf{DBA}(\mathcal{T}_{\sim_L}, (\mathcal{D}_c)_{c \in Q_{\sim_L}})$ accepts L.*

Proof. In the following let $\mathcal{A} = \mathsf{DBA}(\mathcal{T}_{\sim_L}, (\mathcal{D}_c)_{c \in Q_{\sim_L}})$. For the first inclusion consider a word $w \in L$. The run of $\mathcal{A}$ on w begins in (c_0, ι_{c_0}), where $c_0 = [\varepsilon]_{\sim_L}$ and ι_{c_0} refers to the initial state of $\mathcal{D}_{c_0}$. As $w \in L$, we clearly also have that $w \in L_{c_0}$, which by the first condition in the definition of "safe" (Definition 7) implies that $v_0 \in L(\mathcal{D}_{c_0})$ for some prefix v_0 of w. Thus, on the prefix v_0, $\mathcal{A}$ uses an accepting transition and reaches some state (c_1, ι_{c_1}) for

$c_1 = \mathcal{T}_{\sim_L}(v_0)$. Now let w_1 be the infinite word such that $w = v_0 w_1$. Since $\mathcal{T}_{\sim_L}$ reaches c_1 on v_0, we have that $w_1 \in L_{c_1}$ and thus by the first condition there exists a prefix $v_1 \sqsubseteq w_1$ such that $v_1 \in L(\mathcal{D}_{c_1})$. Thus, the run of $\mathcal{A}$ from (c_1, ι_{c_1}) on v_1 uses an accepting transition and reaches the state (c_2, ι_{c_2}) for some class c_2 with $\mathcal{T}_{\sim_L}(v_0 v_1) = c_2$. Repeated application of this argument yields a factorization $w = v_0 v_1 \dots$ and an infinite sequence $(c_i)_{i \in \mathbb{N}}$ with $c_0 = [\varepsilon]_{\sim_L}$, such that $c_{i+1} = \delta^*_{\sim_L}(c_i, v_i)$ and $v_i \in L(\mathcal{D}_{c_i})$ for all $i \in \mathbb{N}$. The run ρ of $\mathcal{A}$ on $w = v_0 v_1 \dots$ can similarly be factorized into $\rho = \rho_0 \rho_1 \dots$, such that ρ_i begins in (c_i, ι_{c_i}) and ends in $(c_{i+1}, \iota_{c_{i+1}})$. Additionally, each of these factors ρ_i uses an accepting transition. This means that the run of $\mathcal{A}$ on w uses infinitely many accepting transitions and thus $w \in L(\mathcal{A})$.

For the other inclusion, let $w \in L(\mathcal{A})$ and ρ be the run of $\mathcal{A}$ on w. Since ρ uses infinitely many accepting transitions, there exists a factorization $w = v_0 v_1 \dots$ and a sequence $(c_i)_{i \in \mathbb{N}}$ with $c_0 = [\varepsilon]_{\sim_L}$ such that $(c_0, \iota_{c_0}) \xrightarrow{v_0} (c_1, \iota_{c_1}) \xrightarrow{v_1} \dots$ in $\mathcal{A}$ and on each v_i the last transition is accepting and all the other transitions are non-accepting. By construction of $\mathcal{A}$, each v_i is accepted by the DFA $\mathcal{D}_{c_i}$ and hence by the second condition of "safe" (Definition 7) also $v_i \in \mathcal{K}(L_c)$. Consider now any DBA $\mathcal{B}$ that accepts L. By Lemma 9 we know that $\mathcal{K}(L_c) = \bigcap_{q \in c} \mathcal{K}_q(\mathcal{B})$ for all classes $c \in Q_{\sim_L}$. Let π be the unique run of $\mathcal{B}$ on $w = v_0 v_1 \dots$, then π can be written as $\iota_B = q_0 \xrightarrow{v_0} q_1 \xrightarrow{v_1} \cdots$. As $\mathcal{B}$ has a finite number of states, we know that there exists an infinite set I of indices such that $q_i = q_j$ for all $i, j \in I$. Note that because $\mathcal{B}$ accepts L, we have $q_i \in c_i$ for all $i \in \mathbb{N}$. But then since $v_i \in \mathcal{K}_{c_i}(L)$ and $\mathcal{K}_{c_i}(L) \subseteq \mathcal{K}_{q_i}(\mathcal{B})$ (by Lemma 9) for all $i \in I$, between any two visits to a state q_i with $i \in I$, $\mathcal{B}$ uses an accepting transition. As I is an infinite set, this guarantees that the run of $\mathcal{B}$ on w uses infinitely many accepting transitions and hence $w \in L(\mathcal{B}) = L$. ◀

In the following, we describe how a sample can be constructed to guarantee that PrfCons yields a DFA that is safe for some DBA language L. For this we use a specific execution of the active learning algorithm AL, shown in Algorithm 2. We refer to this specific execution as the L-run of AL and denote the sample that it produces with $S^{\mathsf{AL},L}$. We present the L-run in form of an algorithm, but we are only interested in the definition of the sample $S^{\mathsf{AL},L}$ and not its computation. Therefore, we do not go into further detail regarding the computation of each individual operation. Note that the L-run of AL is defined along the same lines as the algorithm PrfCons shown in Algorithm 1, but now the queries are answered based on the language L and not based on a sample. The ω-words added to the sample $S^{\mathsf{AL},L}$ ensure that PrfCons will give the same answers to the queries of AL for any sample that includes $S^{\mathsf{AL},L}$ and is consistent with L.

▶ **Lemma 11.** *For a DBA-recognizable language L, the size of $S^{\mathsf{AL},L}$ is polynomial in the size of a minimal DFA for $\mathcal{K}(L)$. Furthermore, the DFA $\mathcal{D}$ returned by the L-run of* AL *is safe for L.*

Proof. We first explain why all the answers of the L-run to the queries of AL are consistent with $\mathcal{K}(L)$. For the membership queries this is obvious from the definition of $\mathcal{K}(L)$. For equivalence queries, consider the first case. The word x that is given as counterexample satisfies the definition of $\mathcal{K}(L)$ and is not accepted by the current hypothesis $\mathcal{D}$, so giving this counterexample is consistent with $\mathcal{K}(L)$. Let us argue that such a word x always exists. Let $\mathcal{A} = (Q, \Sigma, \iota, \delta, \beta)$ be a DBA with $L(\mathcal{A}) = L$. Since $uv^\omega \in L$, it is accepted from all states that are equivalent to the initial state. So there is a prefix $z \sqsubseteq uv^\omega$ such that for every q that is equivalent to ι, the run from q on uv^ω uses an accepting transition on the prefix z. Now assume that y is such that $(zy)^{-1}L = L$. Then the run of $\mathcal{A}$ on $(zy)^\omega$ is accepting

Algorithm 2 Definition of $S^{\mathsf{AL},L}$ by a specific execution of AL, called L-run of AL.

Input: A DBA accepting $L \subseteq \Sigma^\omega$.
Output: The ω-sample $S^{\mathsf{AL},L} = (S_+^{\mathsf{AL},L}, S_-^{\mathsf{AL},L})$ and the DFA $\mathcal{D}$
$S_+^{\mathsf{AL},L} \leftarrow \emptyset, S_-^{\mathsf{AL},L} \leftarrow \emptyset$
Simulate an active learning algorithm AL that satisfies the properties of Theorem 1, and answer its queries as follows:
Query `mem(`x`)`
  if $(xy)^\omega \notin L$ *for some* $y \in \Sigma^*$ *with* $(xy)^{-1}L = L$ **then**
    $S_-^{\mathsf{AL},L} \leftarrow S_-^{\mathsf{AL},L} \cup \{(\varepsilon, xy)\}$ for the shortest such y and answer "no"
  else
    answer "yes"
Query `equiv(`$\mathcal{D}$`)`
  if *there exists a word* $w \in L$ *with* $\mathsf{Prf}(w) \cap L(\mathcal{D}) = \emptyset$ **then**
    pick the minimal (u, v) such that $uv^\omega \in L$ and $\mathsf{Prf}(uv^\omega) \cap L(\mathcal{D}) = \emptyset$
    $S_+^{\mathsf{AL},L} \leftarrow S_+^{\mathsf{AL},L} \cup \{(u, v)\}$
    choose minimal $x \sqsubseteq uv^\omega$ with $(xy)^{-1}L = L \Rightarrow (xy)^\omega \in L$ for all $y \in \Sigma^*$
    forall $x' \sqsubseteq x$ *with* $x' \neq x$ **do**
      choose the minimal y' such that $(x'y')^\omega \notin L$ and $(x'y')^{-1}L = L$
      $S_-^{\mathsf{AL},L} \leftarrow S_-^{\mathsf{AL},L} \cup \{(\varepsilon, x'y')\}$
    return x *as counterexample*
  else if *there is some* $v \in \Sigma^+$ *with* $v^\omega \notin L, v^{-1}L = L$ *and* $\mathsf{Prf}(v^\omega) \cap L(\mathcal{D}) \neq \emptyset$ **then**
    pick v to be the minimal word with this property
    $S_-^{\mathsf{AL},L} \leftarrow S_-^{\mathsf{AL},L} \cup \{(\varepsilon, v)\}$
    return *minimal* $x \sqsubseteq v^\omega$ *such that* $x \in L(\mathcal{D})$ *as counterexample*
  else
    terminate the execution of AL and output $S^{\mathsf{AL},L}$, $\mathcal{D}$

because it uses an accepting transition on each z-segment. Hence, z is in $\mathcal{K}(L)$ and there exists a minimal prefix x of uv^ω with this property that can be given as counterexample. In the second case of an equivalence query, the selected $x \sqsubseteq v^\omega$ is accepted by $\mathcal{D}$, but it is not in $\mathcal{K}(L)$ because there is a y such that $xy = v^k$ for some k.

Thus, all answers to AL are consistent with $\mathcal{K}(L)$, and therefore all hypothesis DFAs $\mathcal{D}$ are of polynomial size in $\mathcal{A}_{\mathcal{K}(L)}$ (since AL satisfies the conditions of Theorem 1). From that one can derive that the counterexamples given to AL, and also the examples added to $S^{\mathsf{AL},L}$ for equivalence queries are of polynomial size in $\mathcal{A}_{\mathcal{K}(L)}$. Hence, the computation time taken by AL is also polynomial in $\mathcal{A}_{\mathcal{K}(L)}$, and the size of the words in membership queries is polynomial in $\mathcal{A}_{\mathcal{K}(L)}$. This implies that also the size of the examples added to $S^{\mathsf{AL},L}$ for the answers of membership queries is polynomial in $\mathcal{A}_{\mathcal{K}(L)}$.

It remains to show that the DFA $\mathcal{D}$ computed by the L-run of AL is safe for L (see Definition 7). The first case of an equivalence query guarantees that for each $w \in L$, some prefix $x \sqsubseteq w$ is accepted by $\mathcal{D}$. To verify that the second condition of Definition 7 is satisfied, let $u \in L(\mathcal{D})$. If $u \notin \mathcal{K}(L)$, then there exists a word $y \in \Sigma^*$ such that $(uy)^{-1}L = L$ and $(uy)^\omega \notin L$. This means that the second case of the equivalence query matches with $v = uy$, and hence the simulation of $\mathcal{D}$ will not stop as long as $\mathcal{D}$ accepts a word outside $\mathcal{K}(L)$. ◀

For a DBA-recognizable language L, we now define the sample

$$S^L = S^{\sim_L} \cup \Big(\bigcup_{c \in Q_{\sim_L}} r_c \cdot S^{\mathsf{AL},L_c} \Big) \text{ for the smallest } r_c \in \Sigma^* \text{ such that } \mathcal{T}_{\sim_L}(r_c) = c$$

where the union and concatenation operation on samples is done component wise, and "smallest" refers to the length-lexicographic ordering. The next lemma establishes that each DFA $\mathcal{D}_c$ for a class c that DBAInf constructs from a consistent sample containing S^L is safe for L_c.

▶ **Lemma 12.** *Let L be a DBA language with right congruence $\sim$. If $S = (S_+, S_-)$ is consistent with L and contains S^L, then for each class c, $\mathsf{PrfCons}(S^{(c)})$ returns a DFA $\mathcal{D}_c$ that is safe for L_c.*

Proof. Since S is consistent with L, we know that $[\![S_+^{(c)}]\!]_\omega \subseteq L_c$ and $[\![S_-^{(c)}]\!]_\omega \cap L_c = \emptyset$. We show that $\mathsf{PrfCons}(S^{(c)})$ gives the same answers to AL as the L_c-run.

- For a membership query $\mathsf{mem}(x)$:
  - If "no" is answered during the L_c-run of AL, then there exists some $(\varepsilon, xy) \in S_-^{\mathsf{AL},L_c}$ with $(xy)^\omega \notin L_c$ and $(xy)^{-1}L_c = L_c$. By definition, we have $r_c(xy)^\omega \in [\![S_-^L]\!]_\omega$ for some $r_c \in \Sigma^*$ with $\mathcal{T}_\sim(r_c) = c$. Because $(xy)^{-1}L_c = L_c$, it further holds that also $\mathcal{T}_\sim(r_c xy) = c$. This means $(xy)^\omega \in [\![S_-^{(c)}]\!]_\omega$ and thus $\mathsf{PrfCons}(S^{(c)})$ also answers "no".
  - If the L_c-run of AL answers "yes", then $(xy)^\omega \in L_c$ for all $y \in \Sigma^*$ with $(xy)^{-1}L_c = L_c$. But then x cannot be a prefix of $[\![S_-^{(c)}]\!]_\omega$ because $S_-^{(c)}$ contains only periodic words and $[\![S_-^{(c)}]\!]_\omega \cap L_c = \emptyset$.
- Consider an equivalence query $\mathsf{equiv}(\mathcal{D})$:
  - Whenever the L_c-run goes to the first case, the minimal (u, v) with $uv^\omega \in L_c$ and $\mathsf{Prf}(uv^\omega) \cap L(\mathcal{D}) = \emptyset$ is in S_+^{AL,L_c}. By definition, this means $(r_c u, v) \in S_+^L \subseteq S_+$ and thus $\mathsf{PrfCons}(S^{(c)})$ also goes to the first case. Note that there cannot be a $(\hat{u}, \hat{v}) \in S_+^{(c)}$ with $\mathsf{Prf}(\hat{u}\hat{v}^\omega) \cap L(\mathcal{D}) = \emptyset$ and $(\hat{u}, \hat{v}) \prec (u, v)$, as otherwise $(\hat{u}, \hat{v})$ would have been selected by the L_c-run. Further, $\mathsf{PrfCons}(S^{(c)})$ picks the same prefix $x \sqsubseteq uv^\omega$: For all strict prefixes $x' \sqsubset x$, we have $(r_c, x'y') \in S_-^{\mathsf{AL},L}$ for the minimal $y' \in \Sigma^*$ such that $(x'y')^{-1}L_c = L_c$ and some $r_c \in \Sigma^*$ with $\mathcal{T}_\sim(r_c) = c$. But then $x' \in \mathsf{Prf}([\![S_-^{(c)}]\!]_\omega)$ and thus $\mathsf{PrfCons}(S^{(c)})$ cannot return x' as a counterexample.
  - If the L_c-run goes to the second case, then $\mathcal{D}$ accepts a prefix of all words in L_c and hence for each $w \in [\![S_+^{(c)}]\!]_\omega$ we have $\mathsf{Prf}(w) \cap L(\mathcal{D}) \neq \emptyset$. Thus, $\mathsf{PrfCons}(S^{(c)})$ also goes to the second case. By construction, we have that $(\varepsilon, v) \in S_-^{\mathsf{AL},L}$ for the minimal v with $v^\omega \notin L_c, v^{-1}L_c = L_c$ and $\mathsf{Prf}(v^\omega) \cap L(\mathcal{D}) \neq \emptyset$. Thus, $(r_c, v) \in S_-^L \subseteq S_-$, which guarantees that $(\varepsilon, v) \in S_-^{(c)}$. There can be no $(\varepsilon, \hat{v}) \in S_-^{(c)}$ with $\hat{v} \prec v$ and $\mathsf{Prf}(\hat{v}^\omega) \cap L(\mathcal{D}) \neq \emptyset$, since $(\varepsilon, \hat{v}) \in S_-^{(c)}$ implies $\hat{v}^\omega \notin L_c$ and $\hat{v}^{-1}L_c = L_c$, which would contradict the minimality of v. Therefore, $\mathsf{PrfCons}(S^{(c)})$ returns the same counterexamples as the L_c-run.

So $\mathsf{PrfCons}(S^{(c)})$ computes the same DFA as the L_c-run and therefore this DFA is safe for L_c by Lemma 11. ◀

By combining the previous results, we are now able to establish that DBAInf can learn all DBA-recognizable languages in the limit.

▶ **Theorem 13.** DBAInf *is a polynomial-time DBA-learner that learns every DBA-recognizable language in the limit.*

Proof. Let L be a DBA-recognizable language and $S = (S_+, S_-)$ be a sample that is consistent with L and contains S^L. Polynomial runtime of DBAInf was already established in Theorem 5, so it remains to show that DBAInf constructs a DBA for L from the sample S. By Lemma 6, we know that DBAInf extracts $\sim_L$ from S. In the third step, DBAInf constructs a DFA $\mathcal{D}_c$ from $S^{(c)}$ for each $\sim_L$-class c. It is guaranteed by Lemma 12 that each $\mathcal{D}_c$ is safe for L_c. Thus, we can conclude from Lemma 10 that $L(\mathsf{DBA}(\mathcal{T}_{\sim_L}, (\mathcal{D}_c)_{c \in Q_{\sim_L}})) = L$ and hence DBAInf returns a DBA which accepts precisely L. ◀

5 Sample Size

As stated in Theorem 13, DBAInf can learn every DBA language in the limit, i.e., for each DBA language L there is a characteristic ω-sample S^L such that DBAInf constructs a DBA for L when executed on an ω-sample S that is consistent with L and contains S^L. In this section, we analyze the size of this sample in terms of the size of L, which is the size of a smallest DBA for L. The characteristic sample S^L that is constructed in Section 4 consists of the following components:

- The sample $S^{\sim_L}$ whose size is polynomial in the size of L, see Lemma 6.
- The samples S^{AL,L_c} (prefixed by a word r_c) that ensure that the DFAs $\mathcal{D}_c$ satisfy the properties of Lemma 10. The size of these samples is polynomial in the size of a minimal DFA for $\mathcal{K}(L_c)$ according to Lemma 11.

This raises the question on the size of the minimal DFA for $\mathcal{K}(L_c)$ compared to the size of L. It turns out that this size can be exponential in the size of L.

▶ **Proposition 14.**

1. *Let L be recognizable by a DBA $\mathcal{A}$ with n states, and let c be a class of $\sim_L$. The number of states of the minimal DFA for $\mathcal{K}(L_c)$ is in $O((n+1)^k)$ where k is the number of states of $\mathcal{A}$ that are in class c.*
2. *There is a family $(L^{(k)})_{k \geq 1}$ of DBA languages such that $\sim_{L^{(k)}}$ has only one class, L_k can be accepted by a DBA with k states, and the minimal DFA for $\mathcal{K}(L^{(k)})$ has 2^k many states for each $k \geq 1$.*

Proof (sketch). The first claim directly follows from $\mathcal{K}(L_c) = \bigcap_{q \in c} \mathcal{K}_q(\mathcal{A})$ (see Lemma 9) and Lemma 8. For the second claim, let $\Sigma_k = \{1, \ldots, k\}$ for $k > 0$, and define the language $L^{(k)} \subseteq \Sigma_k^\omega$ as the set of all ω-words that contain infinitely many occurrences of each symbol from Σ_k. Since membership in $L^{(k)}$ does not depend on any finite prefix, each $\sim_{L^{(k)}}$ has only one class. Each $L^{(k)}$ can be accepted by a DBA $\mathcal{A}^{(k)}$ with k states that repeatedly verifies the occurrence of each symbol from Σ_k in ascending order. The language $\mathcal{K}(L^{(k)})$ consists of all u that contain all letters from Σ_k. It is not difficult to check that the minimal DFA for $\mathcal{K}(L^{(k)})$ has 2^k states. ◀

This means that the size of the characteristic sample for L that we construct in our completeness proof can be of size exponential in the size of L. But if we fix the number of states in each equivalence class, then we obtain a class of DBA languages that can be learned in the limit from polynomial data by DBAInf. For this purpose, we say that a DBA language L has a k-informative right congruence if it can be accepted by a DBA with at most k different states per $\sim_L$ equivalence class. By k-IRC(DBA) we denote the corresponding class of languages. The DBA languages with informative right congruence from [4] correspond to the class 1-IRC(DBA). As a direct consequence of Proposition 14 and the fact that the characteristic sample for L is polynomial in the size of L and $\mathcal{K}(L)$, we obtain:

▶ **Theorem 15.** *For every fixed k,* DBAInf *can learn every language in k-IRC(DBA) in the limit from polynomial data (the degree of the polynomial depends on k).*

The existing polynomial time learners for deterministic ω-automata used with a Büchi condition are known to learn every language in 1-IRC(DBA) from polynomial data [5, 9]. The algorithm in [5] cannot learn any DBA language outside of 1-IRC(DBA), and while the algorithm in [9] can learn DBA languages outside 1-IRC(DBA), there are very simple DBA languages that it cannot learn.

For understanding the worst-case exponential size of the characteristic sample for L, we take a closer look at the operation $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c\in Q_\sim})$ that is used for constructing the DBA $\mathcal{A}$ from the DFAs $(\mathcal{D}_c)_{c\in Q_\sim}$ computed in Step 3. (see Figure 2). Lemma 10 asserts that $\mathcal{A}$ accepts L if each $\mathcal{D}_c$ is safe for L_c (see Definition 7). In the completeness proof we use DFAs $\mathcal{D}_c$ for the languages $\mathcal{K}(L_c)$, which are safe of L_c but can be of exponential size in $\mathcal{A}$ as shown in Proposition 14. This raises the question whether DFAs that are safe for L_c need to be, in general, of exponential size. We show that this is not the case, formally stated in Lemma 17 further below. For the proof of Lemma 17 we use the following lemma.

▶ **Lemma 16.** *Let $\mathcal{A}$ be a DBA with n states and $w \in \Sigma^*$ be word. If $\mathcal{A}$ uses n^2+1 accepting transitions on the run on w starting in a state q, then $\mathcal{A}$ uses at least one accepting transition on its run on w from each state p with $p \sim_{L(\mathcal{A})} q$.*

Proof (sketch). If this is not the case, one can find two language equivalent states p, q and a word $u \in \Sigma^*$ such that u loops on p without an accepting transition, and u loops on q with an accepting transition in the loop, contradicting the language equivalence of p and q. ◀

▶ **Lemma 17.** *Let $\mathcal{A}$ be a DBA with n states, $L = L(\mathcal{A})$ and $\sim$ be the right congruence of L. For each class c of $\sim$ there is a DFA D_c of size polynomial in $|\mathcal{A}|$ such that $\mathcal{D}_c$ is safe for L_c, and thus $\mathsf{DBA}(\mathcal{T}_\sim, (\mathcal{D}_c)_{c\in Q_\sim})$ accepts L.*

Proof (sketch). For a class c of $\sim$ pick a state $q_c \in c$ as initial state of $\mathcal{D}_c$, and simulate $\mathcal{A}$ while incrementing a counter each time $\mathcal{A}$ uses an accepting transition. If this counter exceeds n^2+1, then go to an accepting sink. From Lemma 16 and Lemma 9 it follows that $\mathcal{D}_c$ is safe for L_c. ◀

This shows that the way how we compose a DBA from DFAs can, in principle, produce polynomial size DBAs for each DBA language L. So the reason for the exponential size of the characteristic sample of L in the completeness proof is not enforced by the operation that is used to build the DBA from the DFAs, but is rather coming from the way how we extract information from that sample to obtain the DFAs.

6 Conclusion

We have presented a passive learning algorithm DBAInf for DBAs that constructs a consistent DBA in polynomial time for a given ω-sample, and can learn every DBA language in the limit. Previously, the only known class of ω-languages learnable in the limit was the class of languages with informative right congruence [5, 9], whose definition eliminates one of the most difficult properties of ω-automata, namely that deterministic ω-automata often need several language equivalent states for accepting a language. While the characteristic samples for DBAInf that are used in the completeness proof are of exponential size in the worst-case, we obtain learnability in the limit from polynomial data for the class of DBAs that have no more than k states that are all pairwise language equivalent, for each fixed k. This includes the class of DBA languages with informative right congruence for $k = 1$.

Our algorithm uses an active learning algorithm for DFAs as black-box. Our first attempts to build a DBA learner, following the same basic idea but using a passive learner for DFAs instead of an active one, failed. The reason for this seems to be that one carefully needs to select the information from the ω-sample that used for building the DFAs, in order to obtain a robust DBA learner with the learning in the limit property. An active learning algorithm selects this information with its queries. In future work we plan to investigate whether these ideas can be extended to deal with all regular ω-languages by learning deterministic parity automata. It is also an open question whether our approach can be improved in order to obtain learnability in the limit from polynomial data for all DBA languages.

References

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.
2. Dana Angluin, Udi Boker, and Dana Fisman. Families of DFAs as acceptors of omega-regular languages. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, volume 58 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. URL: http://www.dagstuhl.de/dagpub/978-3-95977-016-3.
3. Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi:10.1016/j.tcs.2016.07.031.
4. Dana Angluin and Dana Fisman. Regular omega-languages with an informative right congruence. In *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018*, volume 277 of *EPTCS*, pages 265–279, 2018. doi:10.4204/EPTCS.277.19.
5. Dana Angluin, Dana Fisman, and Yaara Shoval. Polynomial identification of ω-automata. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part II*, volume 12079 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2020. doi:10.1007/978-3-030-45237-7_20.
6. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
7. Therese Berg and Harald Raffelt. Model checking. In *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, chapter 19. Springer, 2005. doi:10.1007/11498490_25.
8. Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, 21(6):592–597, 1972. doi:10.1109/TC.1972.5009015.
9. León Bohn and Christof Löding. Constructing deterministic ω-automata from examples by an extension of the RPNI algorithm. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.20.
10. León Bohn and Christof Löding. Constructing deterministic ω-automata from examples by an extension of the rpni algorithm, 2021. arXiv:2108.03735.
11. J Richard Büchi. On a decision method in restricted second order arithmetic, logic, methodology and philosophy of science (proc. 1960 internat. congr.), 1962.
12. Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω-languages. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Mathematical Foundations of Programming Semantics*, pages 554–566, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
13. E Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. doi:10.1016/S0019-9958(67)91165-5.

14 E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.

15 John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.

16 Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib - A framework for active automata learning. In *Computer Aided Verification - 27th International Conference, CAV 2015*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer, 2015. doi:10.1007/978-3-319-21690-4_32.

17 Damian López and Pedro Garcíía. On the inference of finite state automata from positive and negative data. In Sempere J. In: Heinz J., editor, *Topics in Grammatical Inference*. Springer, 2016. doi:10.1007/978-3-662-48395-4_4.

18 Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995. doi:10.1006/inco.1995.1070.

19 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 578–586, 2018. doi:10.1007/978-3-319-96145-3_31.

20 Jakub Michaliszyn and Jan Otop. Learning deterministic automata on infinite words. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2370–2377. IOS Press, 2020. doi:10.3233/FAIA200367.

21 Jose Oncina and Pedro García. Inferring regular languages in polynomial update time. *World Scientific*, January 1992. doi:10.1142/9789812797902_0004.

22 Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Machine Learning: From Theory to Applications*, volume 661 of *Lecture Notes in Computer Science*, pages 51–73. Springer, 1993.

23 Sven Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2010.400.

24 Wolfgang Thomas. *Automata on Infinite Objects*, pages 133–191. MIT Press, Cambridge, MA, USA, 1991.

25 Wolfgang Thomas. Facets of synthesis: Revisiting Church's problem. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2009*, volume 5504 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2009.

26 Boris A. Trakhtenbrot and Y.M. Barzdin. *Finite Automata: Behavior and Synthesis*. North-Holland Publishing Company, Amsterdam, 1973.

27 Sicco Verwer and Christian A. Hammerschmidt. flexfringe: A passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, pages 638–642. IEEE Computer Society, 2017. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8090480.

Strategy Synthesis for Global Window PCTL

Benjamin Bordais
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190 Gif-sur-Yvette, France

Damien Busatto-Gaston
Université libre de Bruxelles, Brussels, Belgium

Shibashis Guha
Tata Institute of Fundamental Research, Mumbai, India

Jean-François Raskin
Université libre de Bruxelles, Brussels, Belgium

Abstract

Given a Markov decision process (MDP) M and a formula Φ, the strategy synthesis problem asks if there exists a strategy σ s.t. the resulting Markov chain $M[\sigma]$ satisfies Φ. This problem is known to be undecidable for the probabilistic temporal logic PCTL. We study a class of formulae that can be seen as a fragment of PCTL where a local, bounded horizon property is enforced all along an execution. Moreover, we allow for linear expressions in the probabilistic inequalities. This logic is at the frontier of decidability, depending on the type of strategies considered. In particular, strategy synthesis is decidable when strategies are deterministic while the general problem is undecidable.

2012 ACM Subject Classification Mathematics of computing → Stochastic processes

Keywords and phrases Markov decision processes, synthesis, PCTL

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.115

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: http://arxiv.org/abs/2204.14107

Funding This work is partially supported by the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS & FWO), the COST Action 16228 GAMENET (European Cooperation in Science and Technology), by the PDR project Subgame perfection in graph games (F.R.S.- FNRS), and the DST-SERB SRG/2021/000466 "Zero-sum and Nonzero-sum Games for Controller Synthesis of Reactive Systems" project.

1 Introduction

Given an MDP M and a probabilistic temporal logic formula Φ, the strategy synthesis problem is to determine if there exists a strategy σ to resolve the nondeterminism in M such that the resulting Markov chain (MC) $M[\sigma]$ satisfies Φ, and if so, to construct one such strategy. The probabilistic temporal logic that we study in this paper allows us to express rich probabilistic global temporal constraints over a *bounded* horizon that must be enforced along all computations. Let us illustrate our logic with a few examples. The formula $\mathrm{A}\,\mathrm{G}(\mathbb{P}(\mathrm{F}^5\,\mathsf{Good}) \geq 0.95)$ expresses that it must always be the case, under the strategy σ, that along all computations, the probability to reach a good state within 5 steps is at least 0.95. This is a quantitative bounded horizon Büchi property. In addition, our logic allows for comparing the probability of different events: $\mathrm{A}\,\mathrm{G}\,\mathbb{P}(\mathrm{F}^5\,\mathsf{Good}) \geq 2 \times \mathbb{P}(\mathrm{F}^{10}\,\mathsf{Bad}))$ expresses that under the strategy σ, along all computations, it is always the case that the probability to reach a good state within 5 steps is at least twice the probability of reaching a bad state within 10 steps. The ability to compare probabilities of different events, while not present in classical

logics like PCTL, is necessary to express properties like probabilistic noninterference [13]. This feature has been introduced and studied in probabilistic hyperlogics, e.g. [2], where the ability to compare probabilities plays a central role in describing applications. While hyperlogics are very expressive and highly undecidable, we study here the ability to compare probabilities in a weaker logical setting in order to understand more finely the decidability border that probabilistic comparisons, and more generally linear expressions, induce.

While the model-checking problem for PCTL and MDPs is decidable [6], the synthesis problem is in general undecidable [15].[1] Synthesis for PCTL [15], HyperPCTL [2], and as well as for our logic (as shown in Theorem 37) is undecidable. To recover decidability, we explore two options. First, we consider subclasses of strategies: memoryless deterministic strategies (MD), memoryless randomized strategies (MR), and history-dependant deterministic strategies (HD) are important classes to be considered. Second, we identify syntactically defined sublogics with better decidability properties. For instance, while for PCTL objectives the synthesis problem for HD strategies is highly undecidable (Σ_1^1-complete) [15], it has been shown that the problem is decidable for the cases of MD and MR strategies [15]. The synthesis problem for the qualitative fragment of PCTL, where probabilistic operator can only be compared to constant 0 and 1, is decidable for HD strategies. An important contribution of this paper is to show that the synthesis problem for our sublogics is decidable for HD strategies. To the best of our knowledge, this is the first decidability result for a class of unbounded memory strategies (here HD) and quantitative probabilistic temporal properties.

Main technical contributions. We introduce the logic L-PCTL and two sublogics. L-PCTL extends PCTL with linear constraints over probability subformulae. We first study the window L-PCTL fragment that only allows bounded until or bounded weak until operators in the path formulae. The results for this fragment are presented in Table 1(a) where columns distinguish between memoryless (M) and history-dependent strategies (H), and rows between deterministic (D) and randomized strategies (R). Second, we study the *global* window L-PCTL extension of this logic in which window formulae appear in the scope of an A G operator that imposes the window formula to hold on every state of every computation. The results for this fragment are presented in Table 1(b). Third, we adapt results from the literature to the full logic as summarized in Table 1(c). An L-PCTL formula is *flat* if it does not have nested probabilistic operators, while it is *non-strict* if it does not contain strict comparison operators ($>$ or $<$) for comparing probability expressions.

Our two main technical contributions are focused on the synthesis problem for the global window L-PCTL logic. First, we introduce a fixpoint characterization of the set of strategies that enforces an L-PCTL window property globally. This characterization is effective for HD strategies, leads to a 2EXPTIME algorithm, and we provide an EXPTIME lower bound. Furthermore, the fixpoint characterization allows us to prove that the synthesis problem is in coRE for the class of history-dependent randomized (HR) strategies for the flat and non-strict fragment of global window L-PCTL. Second, we prove that the synthesis problem for HR strategies is undecidable with an original technique that reduces the halting problem of 2-counter Minsky machines (2CM) to our synthesis problem. We believe that the fixpoint characterization and the 2CM encoding are of independent interest.

[1] The difference between the two problems is essentially as follows: in the model-checking problem, each probabilistic operator in the formula is associated with one strategy (or scheduler) while in the synthesis problem, a *unique* strategy is fixed and used for all the probabilistic operators.

Table 1 A summary of our results for the synthesis problem on MDPs for L-PCTL formulae.

(a) synthesis for window L-PCTL.

	M	H
D	NP-complete [5]	PSPACE-complete Prop. 27
R	PSPACE SQRT-SUM-hard Prop. 29, [15]	EXPSPACE PSPACE-hard Thm. 17

(b) synthesis for global window L-PCTL.

	M	H
D	NP-complete	2EXPTIME EXPTIME-hard Prop. 28
R	PSPACE SQRT-SUM-hard Prop. 30	coRE-complete[a] Σ^1_1-hard Thm. 24, 37

[a] if the formula is flat and non-strict

(c) synthesis for L-PCTL.

strategies:	Memoryless	History-dependent
Deterministic	NP-complete [5]	Σ^1_1-complete [15]
Randomized	EXPTIME SQRT-SUM-hard [16], [15]	Σ^1_1-hard [15]

Finally, the satisfiability problem [7] for PCTL (and its variants) can be reduced to the synthesis problem. The decidability of the satisfiability problem for PCTL is a long standing open problem. Our decidability result for the synthesis problem for HD strategies and global window PCTL formulae can be transferred to the following version of the satisfiability problem: given a granularity g for the probabilities, and a global window PCTL formula Φ, does there exist an MC with granularity g that satisfies Φ? (Theorem 31). This gives a new positive decidability result for the satisfiability problem with an unbounded horizon fragment of PCTL and unbounded MCs.

Related work. The model-checking problem for PCTL is decidable [6] and should not be confused with the synthesis problem. In [15], the authors study the synthesis problem for PCTL on MDPs and stochastic games. In [5] it is shown that both randomization and memory in strategies are necessary even for flat window PCTL formulae. Further, [5] shows that the synthesis of MD strategies for PCTL objectives is NP-complete, and [16] shows that MR synthesis is in EXPTIME.[2] For the qualitative fragment of PCTL, deciding the existence of MR and HD strategies have been shown to be NP-complete and EXPTIME-complete, respectively [15]. As previously mentioned, the synthesis problems for HD and HR strategies in the general case of (quantitative) PCTL objectives are highly undecidable [15].

In [11], a probabilistic hyperlogic (PHL) has been introduced to study hyperproperties of MDPs. PHL allows quantification over strategies, and includes PCTL* and temporal logics for hyperproperties such as HYPERCTL* [10]. Hence the model-checking problem in PHL can ask for the existence of a strategy for hyperproperties, and has been shown to be

[2] For the existence of MR strategies for PCTL objectives, in the introduction of [15], it is claimed that the problem is in PSPACE, with a reference to [16]. However, in [16] only an EXPTIME upper bound is proven, for the more general problem of stochastic games. The proof encodes the problem as a polynomial-size formula in the first-order theory of the reals with a fixed alternation of quantifiers so that deciding it is in EXPTIME. The claim seems to be that the complexity of their approach drops to PSPACE when all states are controllable. There is no convincing argument there for that claim, in particular their formula still contains universal quantifiers.

undecidable [11]. Another related work is [1], where HyperPCTL [2] has been extended with strategy quantifiers, and studies hyperproperties over MDPs. The model-checking problem for this logic is also undecidable. In both of these undecidability proofs, the constructed formula contains unbounded finally (F) properties that cannot be expressed in the global window fragment of PCTL that we study here. In both [11] and [1], the model-checking problem is decidable when restricted to MD strategies but is undecidable for HD strategies.

The PCTL satisfiability problem is open for decades. In [14], decidability of finite and infinite satisfiability has been considered for several fragments of PCTL using unbounded finally (F) and unbounded always (G) operators. In [9], satisfiability for bounded PCTL has been considered where the number of steps or horizon used in the operators is restricted by a bound. In [4], a problem related to the satisfiability problem called the feasibility problem has been studied. Given a PCTL formula φ, and a family of Markov chains defined using a set of parameters and with a fixed number of states, the feasibility problem is to identify a valuation for the parameters such that the realized Markov chain satisfies φ. In the satisfiability problem that we study here, the number of states is however not fixed a priori and can be arbitrarily large.

2 Preliminaries

A *probability distribution* on a finite set S is a function $d : S \to [0, 1]$ such that $\sum_{s \in S} d(s) = 1$. We denote the set of all probability distributions on set S by $\mathsf{Dist}(S)$.

▶ **Definition 1.** *A* Markov chain *(MC) is a tuple* $M = \langle S, s_{\mathsf{init}}, \mathbb{P}, \mathsf{AP}, L\rangle$ *where* S *is a countable set of states,* $s_{\mathsf{init}} \in S$ *is an initial state,* $\mathbb{P} : S \to \mathsf{Dist}(S)$ *is a transition function,* AP *is a non-empty finite set of atomic propositions, and* $L : S \to 2^{\mathsf{AP}}$ *is a labelling function.*

If $\mathbb{P}$ maps a state s to a distribution d so that $d(s') > 0$, we write $s \xrightarrow{d(s')} s'$ or simply $s \to s'$, and we denote $\mathbb{P}(s, s')$ the probability $d(s')$. We say that the atomic proposition p holds on a state s if $p \in L(s)$.

A *finite path* $\rho = s_0 s_1 \cdots s_i$ in an MC M is a sequence of consecutive states, so that for all $j \in [0, i-1]$, $s_j \to s_{j+1}$. We denote $|\rho| = i$ the length of ρ, $\mathsf{last}(\rho) = s_i$ and $\mathsf{first}(\rho) = s_0$. We also consider states to be paths of length 0. Similarly, an *infinite path* is an infinite sequence $\rho = s_0 s_1 \cdots$ so that for all $j \in \mathbb{N}$, $s_j \to s_{j+1}$. If ρ is a finite (resp. infinite) path $s_0 s_1 \cdots$, we let $\rho[i]$ denote s_i, $\rho[:i]$ denote the finite prefix $s_0 \cdots s_i$, and $\rho[i:]$ denote the finite (resp. infinite) suffix $s_i s_{i+1} \cdots$.

We denote the set of all finite paths in M by FPaths_M. We introduce notations for the subsets FPaths^{i}_{M} (resp. $\mathsf{FPaths}^{\leq i}_{M}$, $\mathsf{FPaths}^{<i}_{M}$) of paths of length i (resp. of length at most or less than i). Let $\mathsf{FPaths}_M(s)$ denote the set of paths ρ in FPaths_M such that $\mathsf{first}(\rho) = s$. More generally, $\mathsf{FPaths}_M(\rho)$ denotes the set of paths which admit ρ as a prefix. Similarly, we let Paths_M be the set of infinite paths of M, and extend the previous notations for fixing an initial state or a shared prefix. In particular, $\mathsf{Paths}_M(\rho)$ is called the cylinder of ρ.

If $\rho = s_0 \dots s_i$ is a finite path and $\rho' = s_i s_{i+1} \dots$ is a finite or infinite path so that $\mathsf{first}(\rho') = \mathsf{last}(\rho)$, let $\rho \cdot \rho' = s_0 \dots s_i s_{i+1} \dots$ denote their concatenation.

▶ **Definition 2.** *Let* s *be a state of an MC* M*. The MC* M *naturally defines a* probability measure μ^s_M *on* $(\mathsf{Paths}_M(s), \Omega^s_M)$*, where* Ω^s_M *is the* σ*-algebra of cylinders, i.e. the sets* $\mathsf{Paths}_M(\rho)$ *with* $\rho \in \mathsf{FPaths}_M(s)$*, their complements and countable unions.*

The measure of a cylinder $\mathsf{Paths}_M(\rho)$ is the product of the probabilities of each transition in the finite path ρ, and by Carathéodory's extension theorem we get a measure μ^s_M over Ω^s_M. As s is always obvious from context (first state of the paths being considered), we omit it

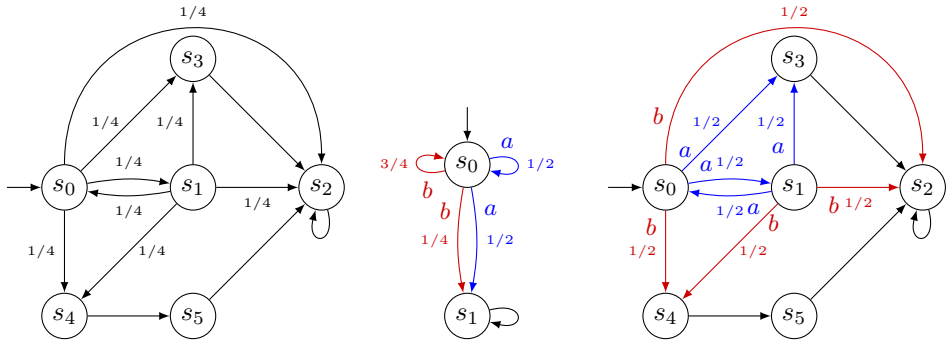


Figure 1 An MC and two MDPs, with states s_i and actions $\{a, b\}$. In MCs, transitions are labelled by their probability, and the probability is 1 if unspecified. In MDPs, transitions are labelled by their action and probability. If the action is unspecified (black transitions), then every action allows this transition. The initial state is s_0. In our examples, the set of atomic propositions is the set of states, so that the proposition s_i holds on state s_i only.

from the measure notation, in favour of μ_M. We note that $\mathsf{FPaths}_M(s)$ is a set of finite words over a countable alphabet, and as such is countable. In particular, if $\mathcal{C} \subseteq \mathsf{FPaths}_M(s)$ is a set of prefixes forming disjoint cylinders,[3] then $\mu_M(\bigcup_{\rho\in\mathcal{C}} \mathsf{Paths}_M(\rho)) = \sum_{\rho\in\mathcal{C}} \mu_M(\mathsf{Paths}_M(\rho))$. Moreover, if $\Pi \subseteq \mathsf{Paths}_M(s)$ is the complement of a measurable set Π', $\mu_M(\Pi) = 1 - \mu_M(\Pi')$.

▶ **Definition 3.** *A* Markov decision process *(MDP) is a tuple* $\mathcal{M} = \langle S, A, s_{\mathsf{init}}, \mathbb{P}, \mathsf{AP}, L\rangle$, *where* S *is a finite set of states,* A *is a finite set of actions,* $s_{\mathsf{init}} \in S$ *is an initial state,* $\mathbb{P} : S \times A \to \mathsf{Dist}(S)$ *is a transition function*[4], AP *is a non-empty finite set of atomic propositions, and* $L : S \to 2^{\mathsf{AP}}$ *is a labelling function.*

If $\mathbb{P}$ maps a state s and an action a to a distribution d so that $d(s') > 0$, we write $s \xrightarrow{a,d(s')} s'$ or simply $s \xrightarrow{a} s'$, and we denote $\mathbb{P}(s, a, s')$ the probability $d(s')$. We extend from MCs to MDPs the definitions and notations of finite and infinite paths, now labelled by actions and denoted $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$. Moreover, for a finite path ρ, we denote by $\rho \cdot as$ (resp. $sa \cdot \rho$) the concatenation of ρ with $\mathsf{last}(\rho) \xrightarrow{a} s$ (resp. of $s \xrightarrow{a} \mathsf{first}(\rho)$ with ρ).

We say that $\mathcal{M}$ is stored in size $|\mathcal{M}|$ if the number of states $|S|$, the number of actions $|A|$ and the number of transitions $s \xrightarrow{a} s'$ in $\mathcal{M}$ are bounded by $|\mathcal{M}|$. Then, $|\mathsf{FPaths}^{\leq i}_{\mathcal{M}}|$, the number of paths of horizon at most i, is in $|\mathcal{M}|^{\mathcal{O}(i)}$. Moreover, the probabilities in $\mathbb{P}$ are assumed to be rational numbers stored as pairs of integers $\frac{a}{b}$ in binary, so that $a, b \leq 2^{|\mathcal{M}|}$.

A (probabilistic) *strategy* is a function $\sigma : \mathsf{FPaths}_{\mathcal{M}} \to \mathsf{Dist}(A)$ that maps finite paths ρ to distributions on actions. A strategy σ is *deterministic* if the support of the distribution $\sigma(\rho)$ has size 1 for every ρ, it is *memoryless* if $\sigma(\rho)$ depends only on the last state of ρ, *i.e.* if σ satisfies that for all $\rho, \rho' \in \mathsf{FPaths}_{\mathcal{M}}$, $\mathsf{last}(\rho) = \mathsf{last}(\rho')$ implies $\sigma(\rho) = \sigma(\rho')$. We denote by $\sigma(\rho, a)$ the probability of the action a in the distribution $\sigma(\rho)$.

An MDP $\mathcal{M} = \langle S, A, s_{\mathsf{init}}, \mathbb{P}, \mathsf{AP}, L\rangle$ equipped with a strategy σ defines an MC, denoted $\mathcal{M}[\sigma]$, obtained intuitively by unfolding $\mathcal{M}$ and using σ to define the transition probabilities. Formally $\mathcal{M}[\sigma] = \langle \mathsf{FPaths}_{\mathcal{M}}, s_{\mathsf{init}}, \mathbb{P}_\sigma, \mathsf{AP}, L'\rangle$, with finite paths of $\mathcal{M}$ as states, transitions defined for all $\rho \in \mathsf{FPaths}_{\mathcal{M}}$, $a \in A$ and $s \in S$ by $\mathbb{P}_\sigma(\rho, \rho \cdot as) = \sigma(\rho, a)\mathbb{P}(\mathsf{last}(\rho), a, s')$, and

[3] $\mathsf{Paths}_M(\rho)$ and $\mathsf{Paths}_M(\rho')$ share a path if and only if either ρ is a prefix of ρ' or ρ' is a prefix of ρ.
[4] This formalism implies that every action is available from every state. This is w.l.o.g., as one can model illegal actions by sending them to a special state.

atomic propositions assigned by $L'(\rho) = L(\mathsf{last}(\rho))$. In particular, note that since S is finite $\mathsf{FPaths}_{\mathcal{M}}$ is infinite but countable. We say that a finite path ρ in $\mathcal{M}$ *matches* a finite path ρ' in $\mathcal{M}[\sigma]$ if $\mathsf{last}(\rho') = \rho$, so that they follow the same sequence of states and actions. We say that a path $\rho \in \mathsf{FPaths}_{\mathcal{M}}$ has probability m in $\mathcal{M}[\sigma]$ if ρ matches $\rho' \in \mathsf{FPaths}_{\mathcal{M}[\sigma]}$ and m is the measure of $\mathsf{Paths}_{\mathcal{M}[\sigma]}(\rho')$. It corresponds to the likelihood of having ρ as a prefix when following σ and starting from $\mathsf{first}(\rho)$.

We may omit M or $\mathcal{M}$ from all previous notations when they are clear from the context. MC notations may use σ as shorthand for $\mathcal{M}[\sigma]$, *e.g.* μ_σ is the probability measure induced by $\mathcal{M}[\sigma]$, and FPaths_σ refers to finite paths of non-zero probability under σ.

▶ **Example 4.** Consider the MC on the left of Figure 1, and the property asking to reach the state s_2 in at most two steps. Consider the set of paths of length at most two from s_0 to s_2. Let $\Pi = \mathsf{Paths}(s_0 s_2) \uplus \mathsf{Paths}(s_0 s_1 s_2) \uplus \mathsf{Paths}(s_0 s_3 s_2)$ be the infinite paths obtained from their cylinders. Then, the probability of reaching s_2 in two steps when starting from s_0 is $\mu(\Pi) = \frac{1}{4} + \frac{1}{16} + \frac{1}{4} = \frac{9}{16}$. Note that the probability of reaching s_2 in two steps when starting from s_1 is also $\frac{9}{16}$. Every other state reaches s_2 with probability 1 in two steps. Consider now the MDP in the middle of Figure 1, and the property asking that the state reached after the first transition is s_1. For every strategy σ, the probability that this property holds in $\mathcal{M}[\sigma]$ is equal to $\sigma(s_0, a)\frac{1}{2} + \sigma(s_0, b)\frac{1}{4} = \sigma(s_0, a)\frac{1}{2} + (1 - \sigma(s_0, a))\frac{1}{4} = \sigma(s_0, a)\frac{1}{4} + \frac{1}{4}$.

Probabilistic CTL with Linear expressions. A formula of L-PCTL is generated by the nonterminal Φ in the following grammar:

▶ **Definition 5** (L-PCTL in normal form, syntax)**.**

$$\Phi := p \mid \neg p \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \sum_{i=1}^{n} c_i \mathbb{P}\left[\varphi_i\right] \succcurlyeq c_0$$
$$\varphi := \mathrm{X}^{\ell}\, \Phi \mid \Phi_1\, \mathrm{U}^{\ell}\, \Phi_2 \mid \Phi_1\, \mathrm{W}^{\ell}\, \Phi_2 \mid \Phi_1\, \mathrm{U}^{\infty}\, \Phi_2 \mid \Phi_1\, \mathrm{W}^{\infty}\, \Phi_2$$

where p ranges over the atomic propositions in AP, ℓ ranges over $\mathbb{N}$, and $n \in \mathbb{N}_{>0}$, $(c_0, \cdots, c_n) \in \mathbb{Z}^n$, $\succcurlyeq \in \{\geq, >\}$ define linear inequalities.

We call a formula generated by Φ a *state formula*, and a formula generated by φ a *path formula*. The *horizon label* of a path formula is the label of its root operator, *i.e.* either ℓ or ∞. Intuitively, the Next operator $\mathrm{X}^{\ell}\, \Phi$ means that Φ holds in exactly ℓ steps, the (unbounded) Until and Weak until operators U^{∞} and W^{∞} are defined as usual in CTL, and their bounded version $\Phi_1\, \mathrm{U}^{\ell}\, \Phi_2$ and $\Phi_1\, \mathrm{W}^{\ell}\, \Phi_2$ impose a horizon on the reachability of Φ_2. We will use the standard notations X, U and W, defined by X^1, U^{∞} and W^{∞}, respectively.

▶ **Definition 6** (L-PCTL in normal form, semantics)**.** *For a fixed MC M of states S, we inductively define $[\![\Phi]\!]_M$ as a set of states, and for each state s we define $[\![\varphi]\!]^s_M$ as a measurable set of infinite paths starting from s:*

$$[\![p]\!]_M = \{s \in S \mid p \in L(s)\} \qquad [\![\neg p]\!]_M = \{s \in S \mid p \notin L(s)\}$$
$$[\![\Phi_1 \wedge \Phi_2]\!]_M = [\![\Phi_1]\!]_M \cap [\![\Phi_2]\!]_M \qquad [\![\Phi_1 \vee \Phi_2]\!]_M = [\![\Phi_1]\!]_M \cup [\![\Phi_2]\!]_M$$
$$\left[\!\!\left[\sum_{i=1}^{n} c_i \mathbb{P}\left[\varphi_i\right] \succcurlyeq c_0\right]\!\!\right]_M = \{s \in S \mid \sum_{i=1}^{n} c_i\ \mu_M([\![\varphi_i]\!]^s_M) \succcurlyeq c_0\}$$
$$\left[\!\!\left[\mathrm{X}^{\ell}\, \Phi\right]\!\!\right]^s_M = \{\rho \in \mathsf{Paths}(s) \mid \rho[\ell] \in [\![\Phi]\!]_M\}$$
$$\left[\!\!\left[\Phi_1\, \mathrm{U}^{\ell}\, \Phi_2\right]\!\!\right]^s_M = \{\rho \in \mathsf{Paths}(s) \mid \exists j \leq \ell, \rho[j] \in [\![\Phi_2]\!]_M \wedge \forall i < j, \rho[i] \in [\![\Phi_1]\!]_M\}$$

$$\begin{aligned}
\left[\!\left[\Phi_1 \,\mathrm{W}^{\ell}\, \Phi_2\right]\!\right]^{s}_{M} &= \{\rho \in \mathsf{Paths}(s) \mid \forall j \leq \ell, (\rho[j] \in [\![\Phi_1]\!]_M \vee \rho[j] \in [\![\Phi_2]\!]_M) \\
&\qquad \vee \exists i < j, \rho[i] \in [\![\Phi_2]\!]_M\} \\
[\![\Phi_1 \,\mathrm{U}\, \Phi_2]\!]^{s}_{M} &= \{\rho \in \mathsf{Paths}(s) \mid \exists j \in \mathbb{N}, \rho[j] \in [\![\Phi_2]\!]_M \wedge \forall i < j, \rho[i] \in [\![\Phi_1]\!]_M\} \\
[\![\Phi_1 \,\mathrm{W}\, \Phi_2]\!]^{s}_{M} &= \{\rho \in \mathsf{Paths}(s) \mid \forall j \in \mathbb{N}, (\rho[j] \in [\![\Phi_1]\!]_M \vee \rho[j] \in [\![\Phi_2]\!]_M) \\
&\qquad \vee \exists i < j, \rho[i] \in [\![\Phi_2]\!]_M\}
\end{aligned}$$

Then, we write $s \models_M \Phi$ (resp. $\rho \models_M \varphi$) if $s \in [\![\Phi]\!]_M$ (resp. $\rho \in [\![\varphi]\!]_M^{\mathsf{first}(\rho)}$), and say that s satisfies Φ (resp. ρ satisfies φ). We denote by $\equiv$ the semantic equivalence of state or path formulae (that holds on all MCs). Finally, we write $M \models \Phi$ if $s_{\mathsf{init}} \models_M \Phi$. Note that by restricting the linear inequalities to $n = 1$ and $\ell = 1$ in X^{ℓ}, we recover the standard definition of PCTL (see *e.g.* [6]).

We define usual notions as syntactic sugar, so that state formulae allow for $\bot := p \wedge \neg p$ and $\top := p \vee \neg p$ (for any $p \in \mathsf{AP}$). We allow rational constants c_1 and all comparison symbols in $\{\leq, <, =, \neq, >, \geq\}$ in linear expressions, with $\sum_{i=1}^{n} c_i \mathbb{P}[\varphi_i] \preccurlyeq c_0 := \sum_{i=1}^{n} (-c_i)\mathbb{P}[\varphi_i] \succcurlyeq -c_0$ and $=, \neq$ defined as conjunctions or disjunctions. Moreover, path formulae allow for $\mathrm{F}^{\ell}\,\Phi := \top \,\mathrm{U}^{\ell}\, \Phi$ and $\mathrm{G}^{\ell}\,\Phi := \Phi \,\mathrm{W}^{\ell}\, \bot$. We allow the negation operation $\neg$ in state and path formulae, and recover a formula in normal form using De Morgan's laws, the negation of inequalities ($\geq$ becomes $<$ and $>$ becomes $\leq$), and the duality rule $\neg(\Phi_1 \,\mathrm{W}^{\ell}\, \Phi_2) \equiv (\neg\Phi_1 \wedge \neg\Phi_2) \,\mathrm{U}^{\ell}\, \neg\Phi_1$. Finally, boolean implication and equivalence are defined as usual. A notable property is $\Phi_1 \,\mathrm{W}^{\ell}\, \Phi_2 \equiv (\Phi_1 \,\mathrm{U}^{\ell}\, \Phi_2) \vee \mathrm{G}^{\ell}\, \Phi_1$.

We encode L-PCTL formulae as trees, whose internal nodes are labelled by state or path operators and whose leaves are labelled by atomic propositions. Let $\ell_{\max} \geq 1$ denote an upper bound on horizon labels ℓ of subformula of Φ where ℓ is finite. The constants c_i in linear expressions are encoded in binary, and *the horizon labels ℓ are encoded in unary*, so that if the overall encoding of Φ is of size $|\Phi|$, we shall have $\ell_{\max} \leq |\Phi|$. We argue that this choice is justified from a larger point of view that extends PCTL to PCTL* by allowing boolean operations in path formulae, as the bounded horizon operators $\mathrm{X}^{\ell}, \mathrm{U}^{\ell}, \mathrm{W}^{\ell}$ can be seen as syntactic sugar for a disjunction of nested X operators of size $\mathcal{O}(\ell)$.

▶ **Definition 7.** *An L-PCTL formula Φ (in normal form) is a* window formula *if the horizon label ℓ of every path operator in Φ is finite, so that the unbounded* U *and* W *are not used. It is a* non-strict *formula if $\succcurlyeq$ is always $\geq$ in its linear inequalities. It is a* flat *formula if the measure operator $\mathbb{P}$ is never nested, so that if Φ is seen as a tree, every branch has at most one node labelled by a linear inequality $\sum_{i=1}^{n} c_i \mathbb{P}[\varphi_i] \succcurlyeq c_0$.*

▶ **Definition 8.** *A* global window *formula is a formula of the shape $\mathrm{A}\,\mathrm{G}\,\Phi$, with Φ a window L-PCTL formula. It is satisfied by a state s of M if every infinite path in $\mathsf{Paths}_M(s)$ satisfies the path formula $\mathrm{G}\,\Phi$, or equivalently if every state reachable from s satisfies Φ.*

▶ **Lemma 9.** *The* global window *formula $\mathrm{A}\,\mathrm{G}\,\Phi$ is satisfied on a state s of M if and only if s satisfies the L-PCTL formula $\mathbb{P}[\mathrm{G}\,\Phi] = 1$.*

Proof. If $\mathrm{A}\,\mathrm{G}\,\Phi$ holds on s, then $\mu_M([\![\mathrm{G}\,\Phi]\!]^s_M) = \mu_M(\mathsf{Paths}_M(s)) = 1$. If $\mathrm{A}\,\mathrm{G}\,\Phi$ is not satisfied on s, then there exists a finite path ρ leading to a state that violates Φ, so that the entire cylinder $\mathsf{Paths}_M(\rho)$ satisfies the path formula $\mathrm{F}\,\neg\Phi$. It follows that $\mu_M([\![\mathrm{G}\,\Phi]\!]^s_M) = 1 - \mu_M([\![\mathrm{F}\,\neg\Phi]\!]^s_M) \leq 1 - \mu_M(\mathsf{Paths}_M(\rho)) < 1$. ◀

▶ **Example 10.** Consider the MC M to the left of Figure 1. Let Φ be the L-PCTL formula $\mathbb{P}\left[\mathrm{F}^2\, s_2\right] \geq \frac{9}{16}$. It is a window formula, that is flat and non-strict. As detailed in Example 4, every state of M satisfies Φ. Therefore, M satisfies the global window formula $\mathrm{A}\,\mathrm{G}\,\Phi$.

Consider now the MDP $\mathcal{M}$ to the right of Figure 1. Let σ denote the memoryless strategy that chooses, in s_0 and s_1, action a with probability $\frac{1}{2}$ and action b with probability $\frac{1}{2}$. While $\mathcal{M}[\sigma]$ is an infinite MC by definition, it is bisimilar to the MC on the left of Figure 1 and must satisfy the same PCTL formulae, so that $\mathcal{M}[\sigma] \models \mathrm{A}\,\mathrm{G}\,\Phi$. In Section 3, we will show that σ is the only strategy on $\mathcal{M}$ that satisfies $\mathrm{A}\,\mathrm{G}\,\Phi$.

Model checking and synthesis problems. The *model-checking problem* of an L-PCTL formula Φ and of a finite MC M is the decision problem asking if $M \models \Phi$. The *synthesis problem* of an L-PCTL formula Φ and of an MDP $\mathcal{M}$ asks if there exists a strategy σ so that $\mathcal{M}[\sigma] \models \Phi$. We also consider the sub-problems that restrict the set of strategies to subsets defined by constraints on the memory or on determinism. For example, the memoryless (resp. deterministic) synthesis problem asks for a memoryless (resp. deterministic) strategy satisfying the formula. They are indeed distinct problems:

▶ **Example 11.** Consider the MDP in the middle of Figure 1. Let Φ be the window formula $(\mathbb{P}\left[\mathrm{F}^2\, s_1\right] = \frac{5}{8}) \wedge (\mathbb{P}\left[\mathrm{X}\, s_1\right] \geq \frac{1}{2} \vee \mathbb{P}\left[\mathrm{X}\, s_1\right] \leq \frac{1}{4})$. First, $s_0 \models \mathbb{P}\left[\mathrm{X}\, s_1\right] \geq \frac{1}{2} \Leftrightarrow \sigma(s_0, a) = 1$ and $s_0 \models \mathbb{P}\left[\mathrm{X}\, s_1\right] \leq \frac{1}{4} \Leftrightarrow \sigma(s_0, a) = 0$, so that the first move must be deterministic. If the first action is a, and the transition $s_0 \xrightarrow{a} s_0$ is chosen, then the next choice must be b to ensure $\mathbb{P}\left[\mathrm{F}^2\, s_1\right] = \frac{5}{8}$. Similarly, if the first action is b, the next choice on s_0 must be a. Moreover, $s_1 \models \Phi$ under any strategy. Thus, the only strategies that satisfy $\mathrm{A}\,\mathrm{G}\,\Phi$ are the strategies that alternate between a and b as long as we are in s_0, while no memoryless strategy satisfies $\mathrm{A}\,\mathrm{G}\,\Phi$. On the other hand, in Example 10 randomisation is needed. An example that require both randomisation and memory can be constructed by combining both examples.

▶ **Proposition 12.** *The* model-checking *problem for L-PCTL formulae and finite MCs can be solved in* PTIME. *This comes at no extra cost when compared to standard PCTL [6].*

Proof. This problem is detailed in [6, Thm. 10.40] for a PCTL definition that only allows expressions of the shape $c_1\mathbb{P}\left[\varphi\right] \succcurlyeq c_0$ to quantify over path formulae. Extending to $\sum_{i=1}^{n} c_i\mathbb{P}\left[\varphi_i\right] \succcurlyeq c_0$ is straight-forward, as their algorithm computes the measure of $\mathbb{P}\left[\varphi\right]$, and then checks if the comparison holds. The intuition is that the measure of bounded operators X^ℓ, U^ℓ and W^ℓ are obtained by $\mathcal{O}(\ell)$ vector-matrix multiplications, while unbounded U and W are seen as linear equation systems. Overall, the complexity is in $|M|^{\mathcal{O}(1)}|\Phi|\ell_{\max}$. ◀

3 Synthesis for global window PCTL

In this section, we detail complexity results on the synthesis problem for global window L-PCTL formulae. We fix a Markov decision process $\mathcal{M}$, a formula $\mathrm{A}\,\mathrm{G}\,\Phi$ where Φ is a window L-PCTL formula, and ask if there exists a strategy σ so that $\mathcal{M}[\sigma] \models \mathrm{A}\,\mathrm{G}\,\Phi$. We also address the sub-problems concerning deterministic or memoryless strategies.

Solving window formulae. We start by constructing a strategy σ so that $\mathcal{M}[\sigma] \models \Phi$. The formula Φ can be seen syntactically as a tree with state or path operators on internal nodes and atomic propositions on leaves. The window length of a branch of this tree is the sum of the horizon labels ℓ of path operators in the branch. The *window length* of the formula Φ is an integer $\mathcal{L}$ obtained as the maximum over every branch of Φ of their respective window lengths. In particular, $\mathcal{L} \leq |\Phi|\ell_{\max}$. For example, if $\Phi = \mathbb{P}\left[\mathrm{X}\,\mathbb{P}\left[\mathrm{X}^2\, p_1\right] \geq \frac{1}{2}\right] \leq \frac{1}{2} \vee \mathbb{P}\left[\mathrm{F}^2\, p_3\right] > 0$, then $\ell_{\max} = 2$ and $\mathcal{L} = \max(1+2, 2) = 3$.

▶ **Definition 13.** *Let s be a state of $\mathcal{M}$ and Φ be a window L-PCTL formula of window length $\mathcal{L}$. A* window strategy *for s of horizon $\mathcal{L}$ is a mapping $\partial : \mathsf{FPaths}_{\mathcal{M}}^{\leq \mathcal{L}}(s) \to \mathsf{Dist}(A)$.*

A window strategy ∂ for state s can be seen as a partial strategy, only defined on paths of length under $\mathcal{L}$ that start from s. Formally, ∂ defines a set of strategies $\sigma : \mathsf{FPaths}_{\mathcal{M}} \to \mathsf{Dist}(A)$, where the first $\mathcal{L}$ steps from s are specified by ∂, and the subsequent steps are not. This set of strategies is called the *cylinder* of the window strategy ∂. In particular, if two strategies σ and σ' are in the cylinder of the window strategy ∂, then the MCs $\mathcal{M}[\sigma]$ and $\mathcal{M}[\sigma']$ coincide for the first $\mathcal{L}$ steps from s, in the sense that every path $\rho \in \mathsf{FPaths}^{\leq\mathcal{L}}_{\mathcal{M}}(s)$ has the same probability m in $\mathcal{M}[\sigma]$ and in $\mathcal{M}[\sigma']$. In this case, we say that m is the probability of ρ under ∂.

We may conflate a window strategy ∂ with an arbitrary strategy σ in its cylinder, so that $\mathsf{FPaths}^{\leq\mathcal{L}}_{\partial}(s)$ is a set of paths in $\mathcal{M}[\sigma]$. Then, we say that the window strategy ∂ for state s satisfies Φ, noted $s \models_{\partial} \Phi$, if $s \models_{\sigma} \Phi$ for all σ in the cylinder of ∂. Conversely, a strategy $\sigma : \mathsf{FPaths}_{\mathcal{M}} \to \mathsf{Dist}(A)$ naturally defines a window strategy ∂_ρ for every fixed prefix ρ, so that for all $\rho' \in \mathsf{FPaths}^{\leq\mathcal{L}}_{\mathcal{M}}(\mathsf{last}(\rho))$, $\partial_\rho(\rho') = \sigma(\rho \cdot \rho')$.

▶ **Lemma 14.** *Let Φ be a window L-PCTL formula of window length $\mathcal{L}$, σ be a strategy for $\mathcal{M}$, and let ∂_s be the window strategy defined by σ on state s and horizon $\mathcal{L}$ (the fixed prefix is s). Then, it holds that $s \models_{\sigma} \Phi \Leftrightarrow s \models_{\partial_s} \Phi$.*

Thus, the synthesis problem on window formulae reduces to finding a window strategy ∂ for s_{init} so that $s_{\mathsf{init}} \models_{\partial} \Phi$. Let ∂ be a window strategy for state s and horizon $\mathcal{L}$. Let $\mathcal{X}_s$ denote a finite set of variables $x_{\rho,a}$, with $\rho \in \mathsf{FPaths}^{\leq\mathcal{L}}_{\mathcal{M}}(s)$, and $a \in A$. The window strategy ∂ can be seen as a point in the real number space $\mathbb{R}^{\mathcal{X}_s}$, where $x_{\rho,a}$ encodes $\partial(\rho, a)$. Conversely, every point in $\mathbb{R}^{\mathcal{X}_s}$ so that $\forall x \in \mathcal{X}_s$, we have $x \in [0, 1]$, and $\forall \rho \in \mathsf{FPaths}^{\leq\mathcal{L}}_{\mathcal{M}}(s)$, we have $\sum_{a\in A} x_{\rho,a} = 1$ represents a window strategy. Therefore, the points of $\mathbb{R}^{\mathcal{X}_s}$ that encode a window strategy can be described by a finite conjunction of linear inequalities $x \geq 0$, $x \leq 1$ and $x_{\rho,a_1} + \cdots + x_{\rho,a_k} = 1$ over the variables $\mathcal{X}_s$.

We want to similarly characterise the set of window strategies satisfying a given window L-PCTL formula. As will become apparent later on, we will need polynomial inequalities.

▶ **Definition 15.** *The* first-order theory of the reals *(FO-$\mathbb{R}$) is the set of all well-formed sentences of first-order logic that involve universal and existential quantifiers and logical combinations of equalities and inequalities of real polynomials.*[5]

We allow the use of strict comparison operators $\{<, \neq, >\}$ as the negation of their non-strict versions. We also assume that the formula is written in *prenex normal form* (PNF), *i.e.* as a sequence of alternating blocks of quantifiers followed by a quantifier-free formula. Finally, if $\mathcal{S} = \{x_1, \cdots, x_k\}$ is a finite set of variables, we use the notation $\exists\, \mathcal{S}$ as shorthand for the quantifier sequence $\exists x_1 \cdots \exists x_k$.

This theory is decidable, and admits a doubly-exponential quantifier elimination procedure [18]. Of particular interest is the existential fragment of FO-$\mathbb{R}$, denoted $\exists$-$\mathbb{R}$, where only $\exists$ is allowed. It is decidable in PSPACE [8].

We say that an FO-$\mathbb{R}$ formula of free variables $\mathcal{X}$ is *non-strict* if it is satisfied by a closed set of points in $\mathbb{R}^{\mathcal{X}}$. In particular, an FO-$\mathbb{R}$ formula that only uses non-strict comparison symbols $\{\leq, =, \geq\}$ and that is negation-free[6] is non-strict.

▶ **Proposition 16.** *Let s be a state of $\mathcal{M}$ and Φ be a window L-PCTL formula. The set of window strategies ∂ such that $s \models_{\partial} \Phi$ can be represented in $\exists$-$\mathbb{R}$ as a PNF formula of free variables $\mathcal{X}_s$. This formula is of size $|\Phi||\mathcal{M}|^{\mathcal{O}(\mathcal{L})}$, and can be computed in* EXPTIME*. If Φ is flat and non-strict then the $\exists$-$\mathbb{R}$ formula is non-strict.*

[5] The primitives operations are multiplication and addition, the comparison symbols are $\{\leq, =, \geq\}$.
[6] A formula is negation-free if the Boolean negation operator $\neg$ is not used.

Proof sketch. We encode the problem in the theory of the reals, by using free variables $x_{\rho,a} \in [0,1]$ that have the value of $\partial(\rho, a)$, existential variables $y_{\rho,\Phi'} \in \{0,1\}$ that are true if the state subformula Φ' is satisfied when one follows ∂ after a fixed history of ρ, and existential variables $z_{\rho,\varphi} \in [0,1]$ having the probability that the path subformula φ is satisfied when one follows ∂ after a fixed history of ρ. The z variables use "local consistency equations" that equate the probability of a path formula on the current state as a linear combination of its probability on the successor states. For the $\mathrm{X}^\ell\,\Phi'$ formula this translates into $z_{\rho,\mathrm{X}^\ell\,\Phi'} = \sum_{s \xrightarrow{a} s'} x_{\rho,a}\mathbb{P}(s,a,s') z_{\rho\cdot a s',\mathrm{X}^{\ell-1}\,\Phi'}$ for example. The y variables can then be defined, so that $y_{\rho,(\sum_{i=1}^n c_i\mathbb{P}[\varphi_i] \succcurlyeq c_0)} = 1$ if and only if $\sum_{i=1}^n c_i z_{\rho,\varphi_i} \succcurlyeq c_0$. Lastly we ask that $y_{s,\Phi} = 1$. To maintain the non-strict property, some subtlety is needed in the way nested probabilistic operators are dealt with. ◀

If we use a PSPACE decision procedure for $\exists$-$\mathbb{R}$ on the formula of Proposition 16, we get:

▶ **Theorem 17.** *The synthesis problem for* window *L-PCTL formulae is in* EXPSPACE.

▶ **Example 18.** Consider the MDP $\mathcal{M}$ to the right of Figure 1, and $\Phi = \mathbb{P}\left[\mathrm{F}^2\, s_2\right] \geq \frac{9}{16}$. Let us describe the formula obtained by Proposition 16. $s_0 \models_\partial \Phi$ can be encoded schematically as the formula $\exists z_{s_0,\mathrm{F}^2\, s_2} \exists z_{s_0 a s_1,\mathrm{F}^1\, s_2}$, s.t. $z_{s_0,\mathrm{F}^2\, s_2} = \frac{x_{s_0,a}}{2} z_{s_0 a s_1,\mathrm{F}^1\, s_2} + \frac{x_{s_0,a}}{2} + \frac{x_{s_0,b}}{2} \wedge z_{s_0 a s_1,\mathrm{F}^1\, s_2} = \frac{x_{s_0 a s_1,b}}{2} \wedge z_{s_0,\mathrm{F}^2\, s_2} \geq \frac{9}{16}$. For readability reasons, we simplified boolean expressions involving $\top$ or $\bot$ when appropriate, and we omitted the variables that can be simplified out immediately, as well as the constraints making sure that the variables x encode probabilities.

After quantifier elimination, and using $x_{s_0,a} + x_{s_0,b} = 1$, we get $x_{s_0,a}\, x_{s_0 a s_1,b} \geq \frac{1}{4}$. Observe that, as mentioned in Example 10, a window strategy ∂ that sets $x_{s_0,a} = x_{s_0 a s_1,b} = \frac{1}{2}$ satisfies Φ. Similarly, $s_1 \models_\partial \Phi$ can be encoded as $x_{s_1,a}\, x_{s_1 a s_0,b} \geq \frac{1}{4}$.

Fixed point characterisation of global window formulae. Let Φ be a window L-PCTL formula of window length $\mathcal{L}$. In this subsection, we describe a fixed point characterisation of the synthesis problem for the global window formula $\mathrm{A}\,\mathrm{G}\,\Phi$.

A *window strategy portfolio* Π of horizon $\mathcal{L}$ (in short, a portfolio Π) maps each state s to a set Π_s of window strategies for s of horizon $\mathcal{L}$. A window strategy portfolio can be seen as a set of points in $\mathbb{R}^{\mathcal{X}_s}$ for every state s. Given two window strategy portfolios Π and Π' of horizon $\mathcal{L}$, we write $\Pi \subseteq \Pi'$ if for all $s \in S$, it holds that $\Pi_s \subseteq \Pi'_s$. Then, the set of all window strategy portfolios of horizon $\mathcal{L}$ is a complete lattice w.r.t. $\subseteq$, where for a set $\mathcal{S}$ of portfolios, the meet $\bigsqcap \mathcal{S}$ (resp. the join $\bigsqcup \mathcal{S}$) maps s to $\bigcap_{\Pi\in\mathcal{S}} \Pi_s$ (resp. $\bigcup_{\Pi\in\mathcal{S}} \Pi_s$).

Let $s \xrightarrow{a} s'$ be a transition in $\mathcal{M}$, and let ∂, ∂' be window strategies for s and s', respectively, of horizon $\mathcal{L}$. We say that ∂ and ∂' are *compatible* w.r.t. $s \xrightarrow{a} s'$ if they make the same decisions on shared paths, *i.e.* for all $\rho \in \mathsf{FPaths}_{\mathcal{M}}^{<\mathcal{L}}(s')$ the probability of $sa\cdot\rho$ under ∂ equals the probability of ρ under ∂' multiplied by $\partial(s,a)\mathbb{P}(s,a,s')$. In particular, whenever $sa\cdot\rho$ has non-zero probability under ∂ and $|\rho| < \mathcal{L} - 1$, we have $\partial(sa\cdot\rho) = \partial'(\rho)$. Similarly, we say that ∂ and a set $\Pi_{s'}$ of window strategies for s' are compatible w.r.t. $s \xrightarrow{a} s'$ if either $\partial(s,a) = 0$ or there exists a window strategy ∂' in $\Pi_{s'}$ so that ∂ and ∂' are compatible w.r.t. $s \xrightarrow{a} s'$.

Let f map portfolios to portfolios, so that $\mathsf{f}(\Pi)$ maps $s \in S$ to the set $\mathsf{f}(\Pi)_s$ of window strategies $\partial \in \Pi_s$ so that for each $s \xrightarrow{a} s'$ in $\mathcal{M}$, we have that ∂ and $\Pi_{s'}$ are compatible w.r.t. $s \xrightarrow{a} s'$. Intuitively, f removes from Π_s the window strategies ∂ that are not compatible with any continuation after a transition $s \xrightarrow{a} s'$ for some action a. Then, f is expressible in the theory of the reals:

▶ **Lemma 19.** *Let* Π *be a portfolio, encoded as an* $\exists$-$\mathbb{R}$ *formula* R^s*, of free variables* $\mathcal{X}_s$*, for every state* s*. Assume that each* R^s *is a PNF formula of size* F*. Then,* $\mathsf{f}(\Pi)_s$ *can also be encoded as a PNF formula, of size in* $\mathcal{O}(|\mathcal{M}|F) + |\mathcal{M}|^{\mathcal{O}(\mathcal{L})}$*. Moreover, if the formulae associated with* Π *are non-strict, so are the formulae of* $\mathsf{f}(\Pi)$*.*

Proof. Let $s \xrightarrow{a} s'$ be a transition in $\mathcal{M}$, and let ∂, ∂' be window strategies for s and s', encoded as points in $\mathbb{R}^{\mathcal{X}_s}$ and $\mathbb{R}^{\mathcal{X}_{s'}}$, respectively. If $\rho \in \mathsf{FPaths}_{\mathcal{M}}^{\leq \mathcal{L}}(s')$, then let $\mathrm{POLY}(\rho)$ denote the polynomial $\prod_{0 \leq i < |\rho|} x_{\rho[:i],a_i} \mathbb{P}(s_i, a_i, s_{i+1})$. Then, the strategies ∂ and ∂' are compatible w.r.t. $s \xrightarrow{a} s'$ if for all $\rho \in \mathsf{FPaths}_{\mathcal{M}}^{\leq \mathcal{L}-1}(s')$ so that $\mathrm{POLY}(\rho) > 0$, for all $a' \in A$ we have $x_{sa\cdot\rho,a'} = x_{\rho,a'}$. Then, if $\Pi_{s'}$ is encoded as the formula $R^{s'}$, we get that ∂ and $\Pi_{s'}$ are compatible w.r.t. $s \xrightarrow{a} s'$ if there exists a valuation of $\mathcal{X}_{s'}$ that encodes a window strategy ∂' so that ∂ and ∂' are compatible w.r.t. $s \xrightarrow{a} s'$. This property corresponds to the formula defined by

$$\mathcal{F}(s,a,s') := \exists \mathcal{X}_{s'}, R^{s'} \wedge \bigwedge_{\rho \in \mathsf{FPaths}_{\mathcal{M}}^{\leq \mathcal{L}-1}(s')} \mathrm{POLY}(\rho) = 0 \vee \bigwedge_{a' \in A} x_{sa\cdot\rho,a'} = x_{\rho,a'}$$

Therefore, if Π_s is encoded as R^s then the formula $R^s \wedge \bigwedge_{s \xrightarrow{a} s'} x_{s,a} = 0 \vee \mathcal{F}(s,a,s')$ encodes $\mathsf{f}(\Pi)_s$. Observe that it introduces non-strict comparisons, existential quantifiers, and no negation operations, and is of size in $F + |\mathcal{M}|(F + |\mathcal{M}|^{\mathcal{O}(\mathcal{L})})$. ◀

▶ **Example 20.** Consider again the MDP $\mathcal{M}$ to the right of Figure 1. Let Π be the portfolio where Π_{s_0} is defined by $x_{s_0,a}\ x_{s_0as_1,b} \geq \frac{1}{4} \wedge x_{s_0as_1,b} \leq c$ with $c \in \left[\frac{1}{2}, 1\right]$, Π_{s_1} is defined by $x_{s_1,a}\ x_{s_1as_0,b} \geq \frac{1}{4} \wedge x_{s_1as_0,b} \leq c$, and Π is $\top$ on every other state of $\mathcal{M}$.[7] Then, using Lemma 19 yields that a formula equivalent to $x_{s_0,a}\ x_{s_0as_1,b} \geq \frac{1}{4} \wedge x_{s_0as_1,b} \leq 1 - \frac{1}{4c}$ encodes $\mathsf{f}(\Pi)_{s_0}$. Symmetrically, $\mathsf{f}(\Pi)_{s_1}$ can be encoded as $x_{s_1,a}\ x_{s_1as_0,b} \geq \frac{1}{4} \wedge x_{s_1as_0,b} \leq 1 - \frac{1}{4c}$.

▶ **Lemma 21** (Knaster-Tarski, Kleene)**.** *The operator* f *is Scott-continuous (upwards and downwards), and is thus monotone. Let* Q *be a set of window strategy portfolios of horizon* $\mathcal{L}$ *that forms a complete lattice w.r.t.* $\subseteq$*. Then, the set of fixed points of* f *in* Q *forms a complete lattice w.r.t.* $\subseteq$*. Moreover,* f *has a greatest fixed point in* Q *equal to* $\bigsqcap\{\mathsf{f}^n(\bigsqcup Q) \mid n \in \mathbb{N}\}$*.*

Let Φ be a window L-PCTL formula of window length $\mathcal{L}$. Let $Q^\Phi = \{\Pi \mid \forall s \in S, \forall \partial \in \Pi_s, s \models_\partial \Phi\}$ be the set of portfolios containing window strategies of horizon $\mathcal{L}$ that ensure Φ. It is closed by $\bigsqcap$ and $\bigsqcup$, and therefore forms a complete lattice. The greatest element $\bigsqcup Q^\Phi$ is the full portfolio mapping every s to all window strategies ∂ so that $s \models_\partial \Phi$. We denote Π^Φ the greatest fixed point of f in Q^Φ, that must exist by Lemma 21.

▶ **Proposition 22.** *Let* s_0 *be a state, and let* Φ *be a window L-PCTL formula. Then,* $\Pi^\Phi_{s_0} \neq \emptyset$ *if and only if there exists a strategy* σ *so that* $s_0 \models_\sigma \mathrm{A\,G}\,\Phi$*.*

Proof sketch. On the one hand, we show that if σ is a strategy so that $s_0 \models_\sigma \mathrm{A\,G}\,\Phi$, and if $\Pi^{s_0,\sigma}$ is the set of window strategies obtained for state s and horizon $\mathcal{L}$ from fixed prefixes of non-zero probability in σ, then $\Pi^{s_0,\sigma}$ is a fixed point of f in Q^Φ so that $\Pi^{s_0,\sigma}_{s_0} \neq \emptyset$. On the other hand, we show that from every fixed point Π of f in Q^Φ that is non-empty on a state s_0, we can inductively construct a strategy σ so that $s_0 \models_\sigma \mathrm{A\,G}\,\Phi$, that intuitively consists in picking successive window strategies from Π that are compatible with each other. ◀

[7] We omitted the constraints that ensure that all variables encode probabilities.

Therefore, computing Π^Φ solves the synthesis problem for global window L-PCTL formulae. By Lemma 21, we have that Π^Φ is the limit of the non-increasing sequence $(\mathsf{f}^i(\bigsqcup Q^\Phi))_{i\in\mathbb{N}}$, with $\bigsqcup Q^\Phi$ being the full portfolio that can be obtained as an $\exists$-$\mathbb{R}$ formula by Proposition 16, so that $\mathsf{f}^i(\bigsqcup Q^\Phi)$ is computable by Lemma 19 as an $\exists$-$\mathbb{R}$ formula of size in $|\Phi||\mathcal{M}|^{\mathcal{O}(\mathcal{L}+i)}$.

▶ **Example 23.** Let $\mathcal{M}$ be the MDP to the right of Figure 1, and $\Phi = \mathbb{P}\left[\mathrm{F}^2\, s_2\right] \geq \frac{9}{16}$. As detailed in Example 18, the set of strategies $(\bigsqcup Q^\Phi)_{s_0}$ is described by $x_{s_0,a}x_{s_0as_1,b} \geq \frac{1}{4}$, the set of strategies $(\bigsqcup Q^\Phi)_{s_1}$ is described by $x_{s_1,a}x_{s_1as_0,b} \geq \frac{1}{4}$, and the set $\bigsqcup Q^\Phi$ is described by $\top$ on all other states.[8] By Example 20, $\mathsf{f}^i(\bigsqcup Q^\Phi)_{s_0}$ is described by $x_{s_0,a}x_{s_0as_1,b} \geq \frac{1}{4} \wedge x_{s_0as_1,b} \leq c_i$, where the constant c_i is defined by $c_0 = 1$ and $c_{i+1} = 1 - \frac{1}{4c_i}$. Similarly, $\mathsf{f}^i(\bigsqcup Q^\Phi)_{s_1}$ is described by $x_{s_1,a}x_{s_1as_0,b} \geq \frac{1}{4} \wedge x_{s_1as_0,b} \leq c_i$, and $\mathsf{f}^i(\bigsqcup Q^\Phi)$ is $\top$ on all other states. The sequence $(c_i)_{i\in\mathbb{N}}$ is a decreasing sequence that converges towards $\frac{1}{2}$ (but never reaches it).

The limit of this sequence is the greatest fixpoint $\Pi^\Phi_{s_0}$, described by $x_{s_0,a}x_{s_0as_1,b} \geq \frac{1}{4} \wedge x_{s_0as_1,b} \leq \frac{1}{2}$ on s_0, $x_{s_1,a}x_{s_1as_0,b} \geq \frac{1}{4} \wedge x_{s_1as_0,b} \leq \frac{1}{2}$ on s_1, $\top$ everywhere else. If we follow the proof of Proposition 22, we can recover the only choice on s_0 and s_1 that ensures $\mathrm{A\,G}\,\Phi$: play a and b with probability $\frac{1}{2}$.

We note that this fixed point computation is not an algorithm: as we have seen in Example 23 the fixed point may not be reachable in finitely many steps. In this case, we do not know if the limit will be empty or not. Nonetheless, this characterisation yields multiple corollary results, that we detail in the remainder of this section.

Flat, non-strict formulae. If Φ is flat and non-strict then $\mathsf{f}^i(\bigsqcup Q^\Phi)$ maps every state to a compact set.[9] The limit of an infinite decreasing sequence of non-empty compact sets in $\mathbb{R}^{\mathcal{X}_s}$ is non-empty. Therefore, if the limit of a decreasing sequence of compact sets is the empty set, it must be reached after finitely many steps. Thus, if $\Pi^\Phi_s = \emptyset$, then there exists $i \in \mathbb{N}$ so that $\mathsf{f}^i(\bigsqcup Q^\Phi)_s = \emptyset$.

▶ **Theorem 24.** *The synthesis problem for flat, non-strict* global window *L-PCTL formulae is in* coRE.

As we will detail in Section 4, the synthesis problem for flat, non-strict global window formulae is undecidable (coRE-hard), and therefore coRE-complete.

▶ Remark 25. From the proof of Proposition 16, it follows that if Φ is non-flat, that is, it contains nested probabilistic operators, then the set of window strategies ∂ such that $s \models_\partial \Phi$ may not be closed and hence $\bigsqcup Q^\Phi$ is not necessarily a compact set.

▶ Remark 26. Note that in Example 23 we were able to compute by hand the limit of the sequence of $\exists$-$\mathbb{R}$ formulae describing $\mathsf{f}^i(\bigsqcup Q^\Phi)$, and obtained an $\exists$-$\mathbb{R}$ formula for the greatest fixed point Π^Φ_s. This is not always possible: there exists an MDP $\mathcal{M}$ and a flat, non-strict global window formula $\mathrm{A\,G}\,\Phi$ so that Π^Φ_s cannot be expressed in FO-$\mathbb{R}$. Indeed, FO-$\mathbb{R}$ formulae can be seen as finite words over a countable alphabet, so that there are countably many of them. If by contradiction Π^Φ_s was always expressible in FO-$\mathbb{R}$, we could enumerate all FO-$\mathbb{R}$ formulae and check for each of them if it describes a fixed-point of f where s_{init} is mapped to a non-empty set, two properties also expressible in FO-$\mathbb{R}$ by using Lemma 19. This would show that the synthesis problem is recursively enumerable, therefore in $\mathsf{RE} \cap \mathsf{coRE}$ *i.e.* decidable, which is absurd as it is coRE-complete as we will see in Section 4.

[8] Once again, we omit the constraints that ensure that all variables encode probabilities.
[9] Non-strict formulae describe closed sets, and all variables are in $[0, 1]$ as they encode probabilities.

Deterministic strategies. In this paragraph, we study the synthesis problem for deterministic strategies. First, note that the window strategy defined by a deterministic strategy for a given prefix and horizon is also deterministic. Conversely, if ∂ is a deterministic window strategy then there exists a deterministic strategy in its cylinder. Therefore, Lemma 14 carries over, and finding a deterministic strategy satisfying a window formula reduces to finding a deterministic window strategy for it. Then, note that for a fixed state s, each deterministic window strategy can be seen as a boolean assignment over the set $\mathcal{X}_s$ of variables, and we have that $|\mathcal{X}_s| = |\mathcal{M}|^{\mathcal{O}(\mathcal{L})}$. Therefore, the set of deterministic window strategies is finite, of doubly-exponential size $2^{|\mathcal{M}|^{\mathcal{O}(\mathcal{L})}}$. We denote by W the number of deterministic window strategies. By guessing a window strategy and verifying it in EXPTIME, we get a NEXPTIME upper bound on the synthesis problem for window formulae. By guessing a strategy in an online manner we can lower this complexity, and show that the problem is in fact PSPACE-complete.

▶ **Proposition 27.** *The synthesis problem for* window *L-PCTL formulae is* PSPACE*-complete when restricted to* deterministic *strategies.*

Proof. We present a non-deterministic algorithm, running in polynomial space, that accepts all positive instances of the synthesis problem with deterministic strategies. We will guess a deterministic window strategy ∂ and check that Φ holds on the resulting MC. In order to avoid guessing an exponential certificate (the entire strategy ∂), we will perform a depth-first search (DFS) traversal of the MDP, starting from s_{init} and of horizon $\mathcal{L}$, where we guess every decision of ∂ in an online manner ($\partial(\rho)$ is guessed when the search path, that is the path from s_{init} to the current state, is ρ for the first time). We will compute along the way information that ultimately lets us evaluate if Φ holds on the root node of the search. At any point in the DFS, when the current path traversed from s_{init} is ρ, this information represents partial evaluations of subformulae of Φ on states along ρ, according to the strategy ∂. Formally, we equip each state in the DFS with a set of formulae to be evaluated. For each path formula φ in the set, we store the probability of satisfying the formula according to paths previously visited by the DFS. Once all of the subtree below a state has been seen by the search, this value matches the probability $\mathbb{P}[\varphi]$, and we can then use this value to evaluate the state formulae that needs to know $\mathbb{P}[\varphi]$ on the current state. In order to define the sets of subformulae to evaluate, we can use the same induction rules as in the proof of Proposition 16, that reduce the evaluation of a formula such as $\mathrm{X}^{\ell}\,\Phi_1$ or $\Phi_1\,\mathrm{U}^{\ell}\,\Phi_2$ on a given state to the evaluation of Φ_1, Φ_2, $\mathrm{X}^{\ell-1}\,\Phi_1$ or $\Phi_1\,\mathrm{U}^{\ell-1}\,\Phi_2$ on the current or the next state. Overall, we need to remember a path of length at most $\mathcal{L}$, a set of subformulae of Φ on each state in this path, and a probability for each such path formula. Assuming that the probabilities can be stored in polynomial space, this is indeed a PSPACE algorithm.

We argue that these probabilities can always be stored in polynomial space. Indeed, they correspond to the measure, in the MC defined by ∂, of a finite union of cylinders defined by prefixes of length at most $\mathcal{L}$. Since ∂ is deterministic, these measures are finite sums of real numbers obtained as the product of at most $\mathcal{L}$ constants appearing as transition probabilities on $\mathcal{M}$. Using a standard binary representation of rational numbers as irreducible fractions, we get that since $\mathcal{L}$ is polynomial in $|\Phi|$ these probabilities are always rational and of polynomial size.

We show a reduction from the synthesis problem with a generalized reachability objective in a two-player game. Given an arena with a set V of vertices that are partitioned into vertices belonging to Player 1 and Player 2, given an initial vertex v_0, and reachability sets $F_1, \ldots, F_k$, the problem asks for a (deterministic) Player 1 strategy that ensures reaching each of the sets against any Player 2 strategies. The generalized reachability problem is PSPACE-complete [12].

We construct an MDP $\mathcal{M}$ with $Q = V$ set of states and transitions which are the same as the edges of the two-player game arena. A Player 1 vertex corresponds to a state in the MDP such that for every outgoing edge (v, v_i) from v, we have an action a_i labelling the transition (v, v_i) in $\mathcal{M}$. For a Player 2 vertex v, all the outgoing edges (v, v_i) correspond to transitions for the same action to vertices v_i with equal probability. Also a state $v \in Q$ in $\mathcal{M}$ is labelled x_i for $i \leq i \leq k$ if and only if the corresponding vertex $v \in F_i$ in the two-player game. Further, if Player 1 has a wining strategy in the generalized reachability game, then she can visit all the reachability sets within a total of nk steps with a deterministic strategy.

Now consider the property Φ defined as $\bigwedge_{i=1}^{k} \mathbb{P}\left[\mathrm{F}^{nk}\, x_i\right] = 1$. There exists a deterministic strategy from v_0 in $\mathcal{M}$ satisfying Φ if and only if Player 1 has a winning strategy for the generalized reachability objective. ◀

As there are finitely many deterministic window strategies of horizon $\mathcal{L}$, the fixed point computation always terminates and thus provides decidability. We also reduce the problem asking if an alternating Turing machine running in polynomial space accepts a given word to deterministic strategy synthesis.

▶ **Proposition 28.** *The synthesis problem for* global window *L-PCTL formulae is in* 2EXPTIME *when restricted to* deterministic *strategies. Moreover, it is* EXPTIME*-hard.*

Proof. For global window formulae, we need to change the set Q^Φ, defined in Section 3, to only contain deterministic window strategies. It is still a complete lattice, and Proposition 22 carries over for deterministic strategies. Moreover, for every portfolio Π, there are finitely many strictly smaller portfolios, at most $|\mathcal{M}|W$. As the sequence $(\mathsf{f}^i(\bigsqcup Q^\Phi))_{i\in\mathbb{N}}$ is non-increasing, the fixed point is reached in at most $|\mathcal{M}|W$ steps. Each step can be performed without relying on the theory of the reals, by representing the window strategies explicitly as trees of depth $\mathcal{L}$. Applying the operator f on a portfolio amounts to checking if a tree is a prefix of another. Overall, the fixed point computation is doubly-exponential.

Note that if we rely on computing $\exists$-$\mathbb{R}$ formulae for $(\mathsf{f}^i(\bigsqcup Q^\Phi))_{i\in\mathbb{N}}$ instead of these explicit sets of strategies, the formulae could a priori grow to sizes in $|\Phi||\mathcal{M}|^{\mathcal{O}(\mathcal{L}+|\mathcal{M}|W)}$, so that we end up with a triply-exponential upper bound.

For the EXPTIME-hardness, we use APSPACE = EXPTIME and present a polynomial reduction from alternating polynomial-space Turing machines. We consider a Turing machine of states Q and tape alphabet Σ, so that each state q is equipped with a label $L(q) \in \{\forall, \exists\}$, except for the accepting and rejecting states $q_\top$ and $q_\bot$, where $L(q) = q$. Let $w \in \Sigma^*$ be an input word, and let $n \in \mathbb{N}$ be a bound on the length of the tape used when running w on the machine. Since we considered a polynomial space machine, n is polynomial. W.l.o.g., we assume that for every input, the Turing machine we consider above halts, and the input is accepted if and only if it halts in $q_\top$.

Let $\mathcal{M}$ be the MDP from Figure 2, where each named state is assigned an identical label. We describe a window formula Φ that ensures that controller only makes choices that faithfully represent an execution of the alternating Turing machine. Let $N = 2n + 5$ be the number of steps needed to follow a cycle from s to s in $\mathcal{M}$. If l is a label in $\mathcal{M}$, we shorten the L-PCTL formula $\mathbb{P}\left[\mathrm{F}^N\, l\right] = 1$ as $\mathrm{F}_1^N\, l$. It means that every path of length N must reach l. Intuitively, a run of the Turing machine can be described as a path in this MDP, where one full loop around s describes a configuration of the Turing machine: visiting a_i means that cell number i contains a, visiting a_i^H means that additionally the reading head is in position i, and entering the gadget q^a means that we are in state q and will read an a. Either controller or the environment gets to pick the next transition, and then we go back to s.

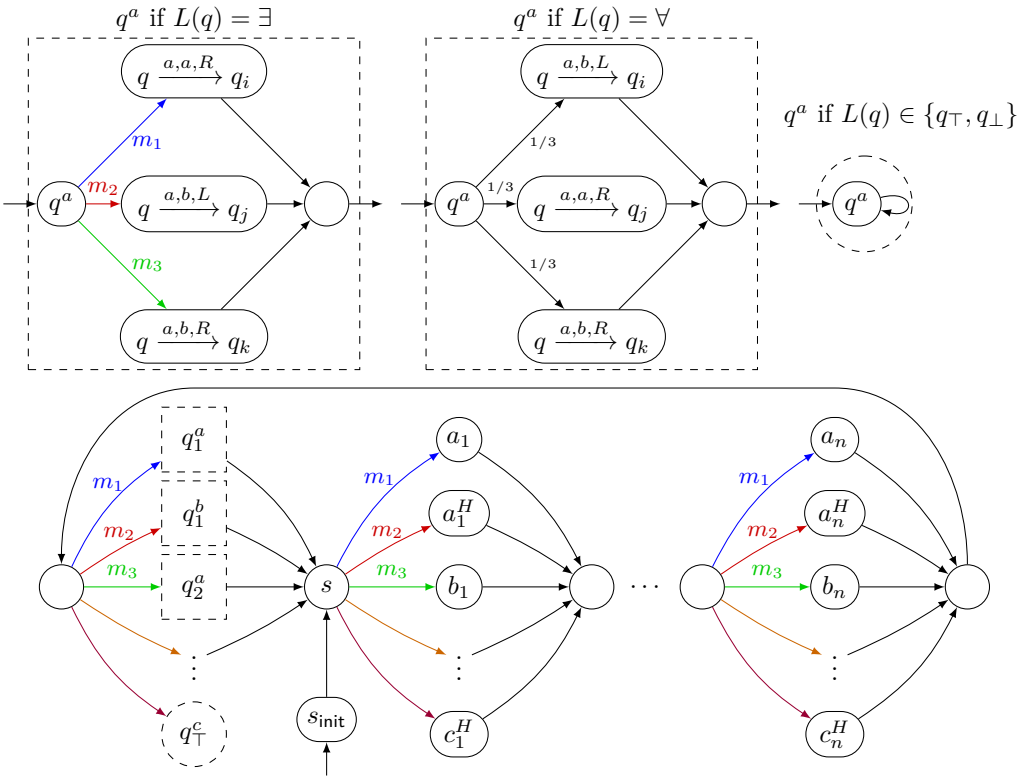


Figure 2 The MDP used in the reduction from APSPACE to deterministic synthesis for global window formulae. The colored transitions m_1, m_2, ... represent different actions, black transitions are available for every action, and the probability of a transition is 1 if unspecified. The letters a, b, c enumerate the tape alphabet Σ, while q_1, q_2, ... enumerate the states in Q, with $q_\top$ the accepting state. The only randomized transition is in the universal gadget q^a, and uses a uniform distribution over reachable states.

The formula Φ is obtained as the conjunction of the following constraints:

In order to ensure that the tape is initialized appropriately, we ask for every letter $a \in \Sigma$ in position $i > 1$ in the input word w that $s_{\mathsf{init}} \Rightarrow \mathrm{F}_1^N\, a_i$. If the first letter in w is a and the initial state of the Turing machine is q_1, we also ask $s_{\mathsf{init}} \Rightarrow \mathrm{F}_1^N\, a_1^H \wedge \mathrm{F}_1^N\, q_1^a$.

In order to simulate the transitions on the tape cells correctly, we ask for every $1 \leq i < n$, $a, c \in \Sigma$ and transition $q \xrightarrow{a,b,L} q'$ that

$$(s \wedge \mathrm{F}_1^N\, c_i \wedge \mathrm{F}_1^N\, a_{i+1}^H) \Rightarrow \mathbb{P}\Big[\mathrm{X}^{N-2}((q \xrightarrow{a,b,L} q') \Rightarrow \mathrm{F}_1^N\, c_i^H \wedge \mathrm{F}_1^N\, b_{i+1})\Big] = 1\,.$$

We similarly ask for every $1 \leq i < n$, $a, c \in \Sigma$ and transition $q \xrightarrow{a,b,R} q'$ that

$$(s \wedge \mathrm{F}_1^N\, a_i^H \wedge \mathrm{F}_1^N\, c_{i+1}) \Rightarrow \mathbb{P}\Big[\mathrm{X}^{N-2}((q \xrightarrow{a,b,R} q') \Rightarrow \mathrm{F}_1^N\, b_i \wedge \mathrm{F}_1^N\, c_{i+1}^H)\Big] = 1\,.$$

The other tape cells should be left untouched, so that for every $1 \leq i < n$, $a, b \in \Sigma$ and transition $q \xrightarrow{c,d,L} q'$, we ask that

$$(s \wedge \mathrm{F}_1^N\, a_i \wedge \mathrm{F}_1^N\, b_{i+1}) \Rightarrow \mathbb{P}\Big[\mathrm{X}^{N-2}((q \xrightarrow{c,d,L} q') \Rightarrow \mathrm{F}_1^N\, a_i)\Big] = 1\,.$$

Similarly for every transition transition $q \xrightarrow{c,d,R} q'$, we ask that

$$(s \wedge \mathrm{F}_1^N\, a_i \wedge \mathrm{F}_1^N\, b_{i+1}) \Rightarrow \mathbb{P}\Big[\mathrm{X}^{N-2}((q \xrightarrow{c,d,R} q') \Rightarrow \mathrm{F}_1^N\, b_{i+1})\Big] = 1\,.$$

Finally, in order to update the state, we ask for every transition $q \xrightarrow{a,b,D} q'$ with $D \in \{L, R\}$ and every $1 \leq i \leq n$, $c \in \Sigma$ that $(q \xrightarrow{a,b,D} q') \wedge \mathrm{F}_1^N\, c_i^H \Rightarrow \mathrm{F}_1^N\, q'^c$.

Then, we let $q_\top$ and $q_\bot$ be the labels that hold on all states $q_\top^a$ and $q_\bot^a$, respectively, for all $a \in \Sigma$. We consider the global window formula $\mathrm{A}\,\mathrm{G}[(\Phi \vee q_\top) \wedge \neg q_\bot]$, and show that there is a winning strategy for this formula in $\mathcal{M}$ if and only if the alternating Turing machine accepts the input word w. Indeed, an alternating Turing machine equipped with an initial word can be seen as a turn-based two-player zero-sum reachability game played on the execution tree of the machine, where we ask if there exists a strategy for player $\exists$ that ensures against every strategy of $\forall$ the state $q_\top$ is reached. ◀

Memoryless strategies. We study the synthesis of memoryless strategies. The window strategy defined by a memoryless strategy for a given prefix and a horizon is also memoryless. Conversely, a memoryless window strategy has a memoryless strategy in its cylinders. By Lemma 14, finding a memoryless strategy satisfying a window formula reduces to finding a memoryless window strategy for it. Let s be a state. As usual, a window strategy ∂ for state s can be seen as an assignment in $[0, 1]$ for variables $\mathcal{X}_s$. However, the memoryless property asks that $\partial(\rho) = \partial(\rho')$ for all ρ, ρ' that share the same last state s', or equivalently $x_{\rho,a} = x_{\mathsf{last}(\rho),a}$ for all ρ. Thus, we can replace every instance of $x_{\rho,a}$ by $x_{\mathsf{last}(\rho),a}$ in the $\exists$-$\mathbb{R}$ formula of Proposition 16, so that the set of free variables used to represent a memoryless window strategy for s is $\mathcal{X}_s = \{x_{s',a} \mid \exists \rho \in \mathsf{FPaths}_{\mathcal{M}}^{\leq \mathcal{L}}(s), s' = \mathsf{last}(\rho)\}$. Similarly, the variables $y_{\rho,\Phi}$ and $z_{\rho,\varphi}$ can be replaced by by $y_{\mathsf{last}(\rho),\Phi}$ and $z_{\mathsf{last}(\rho),\varphi}$ respectively, as the satisfaction of a state formula, or the probability of satisfying a path formula, only depend on the current state. The formula is now of polynomial size, so that we obtain as a corollary:

▶ **Proposition 29.** *The synthesis problem for* window *L-PCTL formulae is in* PSPACE *when restricted to* memoryless *strategies.*

Further, following a reduction in [15], it can be shown that the MR synthesis problem for window L-PCTL objectives is at least as hard as the Square-Root-Sum problem which is known to be in PSPACE, but whose exact complexity is a longstanding open problem.

We now study the memoryless synthesis problem for global window formulae. For each state s, let R^s denote the $\exists$-$\mathbb{R}$ formula encoding the window formula Φ for state s, as per Proposition 29. The free variables are the variables in $\mathcal{X}_s \subseteq \mathcal{X} = \{x_{s',a} \mid s' \in S, a \in A\}$. A memoryless strategy σ can be seen as a point in $\mathbb{R}^{\mathcal{X}}$, so that $\sigma(s, a)$ is assigned to $x_{s,a}$. For all states s and s', we define a variable $r_{s,s'} \in \{0, 1\}$ quantified existentially, and construct a formula ensuring that if $r_{s,s'} = 0$ then s' is not reachable from s under the strategy σ. This formula states $r_{s,s} = 1$ for all states s, and asks that the variables r are a solution to the system of equations asking, for all s, s', s'' and a, that if $r_{s,s'} = 1$ and $x_{s',a}\mathbb{P}(s', a, s'') > 0$ then $r_{s,s''} = 1$. Therefore, the set of states s' so that $r_{s,s'} = 1$ is an over-approximation.[10]

Then, the formula asking that there exists a value for each variable r so that $R^{s'}$ holds whenever $r_{s,s'} = 1$ represents the memoryless strategies that satisfy Φ on an over-approximation of the states reachable from s, which is equivalent to satisfying $\mathrm{A}\,\mathrm{G}\,\Phi$ when

[10] For example, the formula is satisfied if $r_{s,s'}$ is 1 for all s, s', which represents an over-approximation of the set of states reachable from s where every state is reachable.

starting from state s. Note that since the variables $r_{s,s'}$ are existentially quantified, and Φ is only required to be satisfied on states reachable from s, then there always exists a valuation for these r variables that sets $r_{s,s'}$ to 1 if and only if s' is reachable from s. It follows that:

▶ **Proposition 30.** *The synthesis problem for* global window *L-PCTL formulae is in* PSPACE *when restricted to* memoryless *strategies.*

PCTL satisfiability. We now consider the satisfiability problem, that asks, given a formula Φ, if there a exists an MC M so that $M \models \Phi$. This is a longstanding open problem for PCTL formulae. One can also consider variants of the problem, that either restrict Φ to a sublogic of PCTL or limit M to MCs that belong to a particular set, such as finite MCs or MCs where all probabilities are rational numbers. The decidability of these variants is also open and, as noted in [7], some PCTL formulae are only satisfiable by infinite MCs. In particular, we say that an MC M has granularity bounded by $N \in \mathbb{N}$ if every probability in the transition function $\mathbb{P}$ is equal to a rational $\frac{a}{b}$ with $b \leq N$. The *bounded granularity satisfiability problem* asks, given Φ and N, if there exists an MC of granularity bounded by N that satisfies Φ. The bounded granularity satisfiability problem for global window L-PCTL formulae can be reduced to the HD strategy synthesis problem for global window L-PCTL formulae. Therefore, we obtain the following result as a corollary of Proposition 28:

▶ **Theorem 31.** *The bounded granularity satisfiability problem for global window L-PCTL formulae is decidable in complexity doubly-exponential in* $|\Phi|$ *and* N*. Moreover, finite MCs are sufficient, in the sense that for every formula* $\mathrm{A}\,\mathrm{G}\,\Phi$ *that admits a model* M *of granularity bounded by* N*, there exists a finite MC* M' *of granularity bounded by* N *so that* $M' \models \mathrm{A}\,\mathrm{G}\,\Phi$*.*

Proof sketch. Given a formula with atomic propositions AP, and a granularity bound N, we intuitively consider an MC of states 2^{AP} with an action for every distribution over 2^{AP} whose granularity is bounded by N, so that this action describes the next states and their probabilities. Then, every MC of granularity bounded by N can be seen as a deterministic strategy in this MDP, so that strategy synthesis and MC satisfiability are equivalent. We can then apply Proposition 28. Moreover, finite MCs are sufficient as finite-memory strategies are sufficient for global window PCTL when restricted to deterministic strategies. ◀

4 Undecidability

In Section 3, we have shown that the synthesis problem for flat, non-strict global window L-PCTL formulae is in coRE. In this section, we argue that it is coRE-hard and that it becomes Σ^1_1-hard when relaxing the hypothesis that the formulae considered are non-strict.

When considering flat non-strict formulae, we proceed via a reduction from the non-halting problem of a two-counter Minsky machine. A two-counter Minsky machine consists of a list of instructions $l_1 : \mathsf{ins}_1, \ldots, l_n : \mathsf{ins}_n$ and two counters c_2 and c_3 (the indices 2 and 3 are chosen to ease the notations) where, for all $i \leq n$, we have ins_i an instruction in one the following types, for $j \in \{2, 3\}$ and $1 \leq k, m \leq n$: $\mathsf{Inc}_j(k)$: $c_j := c_j + 1$; goto k; $\mathsf{Branch}_j(k, m)$: if $c_j = 0$ then goto k; else $c_j := c_j - 1$, goto m; H: halt. The semantics of these instructions is straightforward. The non-halting problem for Minsky machine, denoted MinskyNotStop, is to decide, given a machine Msk, if its execution is infinite. This problem is undecidable, as stated in the theorem below.

▶ **Theorem 32** ([17])**.** *MinskyNotStop is* coRE*-complete.*

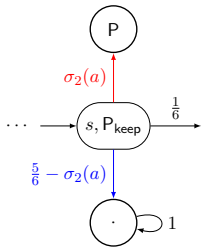


Figure 3 The end of a gadget.

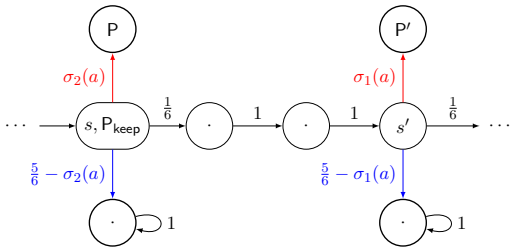


Figure 4 The end of a gadget on the left, and the beginning of another one on the right.

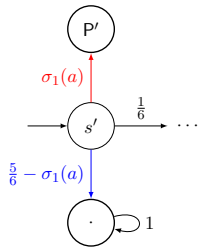


Figure 5 The beginning of a gadget.

Given $\mathsf{Msk} = l_1 : \mathsf{ins}_1, \ldots, l_n : \mathsf{ins}_n$ on two counters c_2 and c_3, we build an MDP $\mathcal{M}$ and an L-PCTL formula Φ such that there exists a strategy σ for $\mathcal{M}$ s.t. $\mathcal{M}[\sigma] \models \Phi$ if and only if $\mathsf{Msk} \in \mathsf{MinskyNotStop}$. The crucial point of the reduction is to encode the values of the counters that may take unbounded values. It is done in $\mathcal{M}$ by encoding these values in the probability (chosen by the strategy σ) to see a given predicate in the next few steps. More specifically, in the situation where the counters are such that $\{c_2 \mapsto x_2; c_3 \mapsto x_3\}$, we consider the probability $p(x_2, x_3) = \frac{5}{6} \times \frac{1}{2^{x_2}} \times \frac{1}{3^{x_3}}$. We then associate to each different instruction a gadget, i.e. an MDP, and a formula encoding the update of probability $p(x_2, x_3)$ according to how the counters are changed by the corresponding instruction. Inside a gadget, one can find predicates of the shape $(\mathsf{P}.)$. They are used to define the formulae specifying the expected behavior of the strategy. Furthermore, there is also an entering and an exiting probability which correspond to the encoding of the counters respectively before and after the effect of the instructions. We define below formally the notion of well-placed gadgets.

▶ **Definition 33** (Gadgets). *A gadget* Gd *is an MDP with an* entering probability *and an* exiting probability. *Consider Figure 3 that represents how every gadget* Gd *ends. The exiting probability* $p^{\mathsf{ex}}_{\mathsf{Gd}}$ *is the probability* $\sigma_2(a)$ *to visit the state on the top. It is equal to* $p^{\mathsf{ex}}_{\mathsf{Gd}} = \mathbb{P}_s(\mathrm{F}^1\, \mathsf{P})$, i.e. *the probability that* $\mathrm{F}^1\, \mathsf{P}$ *holds on state* s. *Consider Figure 5. All gadgets begin as in this figure: a state* s' *with a successor satisfying the predicate* P'. *The* entering probability $p^{\mathsf{en}}_{\mathsf{Gd}}$ *is the probability* $\sigma_1(a)$ *to see* P', *that is:* $p^{\mathsf{en}}_{\mathsf{Gd}} = \mathbb{P}_{s'}(\mathrm{F}^1\, \mathsf{P}')$. *A gadget is* well-placed *if, as for the gadget on the right of Figure 4, it is preceded by two dummy states, themselves immediately preceded by a gadget.*

Before looking at how specific instructions are encoded in the counters and the formula, we have to ensure that the exiting probability of a gadget is equal to the entering probability of the following well-placed gadget. This is done with the formula: $\Phi_{\mathsf{keep}} := \mathsf{P}_{\mathsf{keep}} \Rightarrow (\mathbb{P}(\mathrm{F}^1\, \mathsf{P}) = 6 \cdot \mathbb{P}(\mathrm{F}^4\, \mathsf{P}'))$. These definitions ensure the following proposition:

▶ **Proposition 34** (Entering probability of a well-placed gadget). *Assume that a well-placed gadget* Gd' *follows a gadget* Gd. *Then, for a strategy* σ *s.t. the formula* $\mathrm{A}\, \mathrm{G}\, \Phi_{\mathsf{keep}}$ *is satisfied, the exiting probability of gadget* Gd *is equal to the entering probability of gadget* Gd'*:* $p^{\mathsf{ex}}_{\mathsf{Gd}} = p^{\mathsf{en}}_{\mathsf{Gd}'}$.

Due to lack of space, we only exhibit the gadgets encoding the increment of a counter and for testing if a counter value is 0. Consider the increment of counter c_2. By definition of $p(x_2, x_3)$, incrementing that counter is simulated by multiplying the probability by $\frac{1}{2}$.We define the gadget $\mathsf{Gd}_{c_2{+}{+}}$ and the formula $\Phi_{c_2{+}{+}}$ ensuring that the probability is indeed multiplied by $\frac{1}{2}$. The gadget $\mathsf{Gd}_{c_2{+}{+}}$ is depicted in Figure 6.In addition, we define the L-PCTL formula $\Phi_{c_2{+}{+}}$ such that $\Phi_{c_2{+}{+}} := \mathsf{P}_{c_2{+}{+}} \Rightarrow (\mathbb{P}(\mathrm{F}^2\, \mathsf{P}_{c_2}) = 6 \cdot 2 \cdot \mathbb{P}(\mathrm{F}^3\, \mathsf{P}_+))$. The interest of these definitions lies in the proposition below.

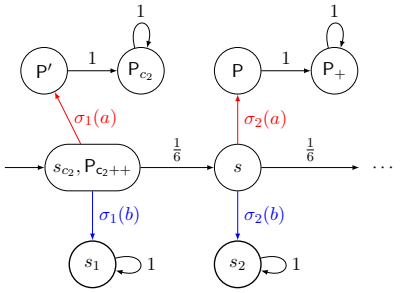


Figure 6 The gadget Gd_{c_2++} for the operation $c_2 := c_2 + 1$.

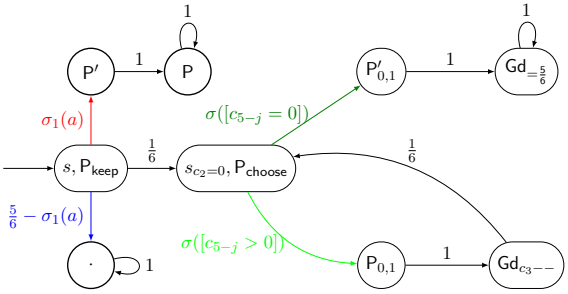


Figure 7 The gadget $\mathsf{Gd}_{c_2=0}$ for testing that the counter $c_2 = 0$.

▶ **Proposition 35** (Incrementing Gadget Specification). *If the entering probability* $p^{\mathsf{en}}_{\mathsf{Gd}_{c_2++}}$ *of the gadget* Gd_{c_2++} *is equal to* $p(x_2, x_3)$ *with* $x_2, x_3 \in \mathbb{N}$*, then whenever the formula* $\mathrm{A\,G}\,\Phi_{c_2++}$ *is satisfied, the exiting probability* $p^{\mathsf{ex}}_{\mathsf{Gd}_{c_2++}}$ *of this gadget is equal to* $p^{\mathsf{ex}}_{\mathsf{Gd}_{c_2++}} = p(x_2 + 1, x_3)$.

We now consider the gadget that tests if a counter value is 0, let us exemplify it with counter c_2 in gadget $\mathsf{Gd}_{c_2=0}$ depicted in Figure 7. The gadget $\mathsf{Gd}_{=5/6}$ used on the right tests that both counters have value 0 (i.e. the entering probability is equal to 5/6).Then, the idea is as follows: as long as counter c_3 has a positive value, the strategy σ has to take the bottom branch to decrement it and once this counter has reached 0, it can take the top branch to check that the probability is indeed equal to 5/6. Note that one cannot decrement counter c_2 in this gadget, hence if its value is positive, there is no way to pass the test of the comparison to 5/6. To ensure that the choice at state $s_{c_2=0}$ is deterministic, we consider the formula $\Phi_{0,1} := \mathsf{P}_{\mathsf{choose}} \Rightarrow (\mathbb{P}(\mathrm{X}\,\mathsf{P}_{0,1}) = 1 \vee \mathbb{P}(\mathrm{X}\,\mathsf{P}'_{0,1}) = 1)$. We have the proposition below:

▶ **Proposition 36** (Testing Gadget Specification). *Assume that the entering probability* $p^{\mathsf{en}}_{\mathsf{Gd}_{c_2=0}}$ *of the gadget* $\mathsf{Gd}_{c_2=0}$ *is equal to* $p(x_2, x_3)$ *for some* $x_2, x_3 \in \mathbb{N}$*. Then, there is a strategy* σ *such that the formula* $\mathrm{A\,G}[\Phi_{\mathsf{keep}} \wedge \Phi_{0,1} \wedge \Phi_{c_3--} \wedge \Phi_{=5/6}]$ *is satisfied if and only if* $x_2 = 0$.

We can similarly test that a counter is different from 0. A gadget corresponding to a branching instruction can then be defined by using these gadgets.Finally, a gadget corresponding to the Halt instruction only consists of a gadget where no strategy σ can satisfy the formula considered. Overall, we can combine all these gadgets to encode all the instructions of the Minsky machine Msk.We obtain the theorem below.

▶ **Theorem 37.** *The synthesis problem for flat, non-strict global window L-PCTL formulae is* coRE*-hard.*

When considering arbitrary flat formulae without the non-strict constraint, we can adapt the proof to reduce from the problem asking if there is an execution of a Minsky Machine that visits infinitely often the first instruction [3], so that the strategy synthesis problem becomes highly undecidable.

▶ **Theorem 38.** *The synthesis problem for flat global window L-PCTL formulae is* Σ^1_1*-hard.*

The construction here is similar to the case with the non-strict constraint, except that whenever the first instruction is seen, a choice is given to the strategy which can set in how many number of steps n the first instruction will be seen again (note that this number may be arbitrarily large). This choice is encoded by resetting a new counter c_5 to value n, which is then decremented each time the first instruction is not seen, and a problem arises if this

counter ever reaches 0. In terms of probability, the value $p(x_2, x_3)$ is initially multiplied by $\frac{1}{5^n}$ and then multiplied by 5 each time the first instruction is not seen. Hence, the probabilities chose by σ may be arbitrarily close to 0, but cannot be equal to 0. This is where we need the non-strict comparison with 0.

References

1 E. Ábrahám, E. Bartocci, B. Bonakdarpour, and O. Dobe. Probabilistic hyperproperties with nondeterminism. In *ATVA*, pages 518–534. Springer, 2020.
2 E. Ábrahám and B. Bonakdarpour. Hyperpctl: A temporal logic for probabilistic hyperproperties. In *QEST*, pages 20–35. Springer, 2018.
3 Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.
4 R. Andriushchenko, M. Ceska, S. Junges, and J-P Katoen. Inductive synthesis for probabilistic programs reaches new horizons. In *TACAS, Part I*, pages 191–209. Springer, 2021.
5 C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *Exploring New Frontiers of Theoretical Informatics*, pages 493–506. Springer US, 2004.
6 C. Baier and J-P. Katoen. *Principles of model checking*. MIT Press, 2008.
7 T. Brázdil, V. Forejt, J. Kretínský, and A. Kucera. The satisfiability problem for probabilistic CTL. In *LICS*, pages 391–402. IEEE Computer Society, 2008.
8 John Canny. Some algebraic and geometric computations in pspace. In *STOC*, pages 460–467, 1988.
9 S. Chakraborty and J-P Katoen. On the satisfiability of some simple probabilistic logics. In *LICS*, pages 56–65. Association for Computing Machinery, 2016.
10 M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In *POST*, pages 265–284. Springer, 2014.
11 R. Dimitrova, B. Finkbeiner, and H. Torfah. Probabilistic hyperproperties of markov decision processes. In *ATVA*, pages 484–500. Springer, 2020.
12 N. Fijalkow and F. Horn. The surprizing complexity of reachability games. *CoRR*, abs/1010.2420, 2010. `arXiv:1010.2420`.
13 J. W. Gray III. Toward a mathematical foundation for information flow security. *J. Comput. Secur.*, 1(3-4):255–294, 1992.
14 Jan Kretínský and Alexej Rotar. The satisfiability problem for unbounded fragments of probabilistic CTL. In *CONCUR*, pages 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
15 A. Kucera, V. Forejt, V. Brozek, and T. Brazdil. Stochastic games with branching-time winning objectives. In *LICS*, pages 349–358. IEEE Computer Society, 2006.
16 A. Kučera and O. Stražovský. On the controller synthesis for finite-state markov decision processes. In *FSTTCS*, pages 541–552. Springer Berlin Heidelberg, 2005.
17 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
18 J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. part iii: Quantifier elimination. *Journal of Symbolic Computation*, 13(3):329–352, 1992.

The Complexity of SPEs in Mean-Payoff Games

Léonard Brice
Université libre de Bruxelles, Brussels, Belgium

Jean-François Raskin
Université libre de Bruxelles, Brussels, Belgium

Marie van den Bogaard
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Abstract

We establish that the subgame perfect equilibrium (SPE) threshold problem for mean-payoff games is **NP**-complete. While the SPE threshold problem was recently shown to be decidable (in doubly exponential time) and **NP**-hard, its exact worst case complexity was left open.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Logic and verification; Theory of computation → Solution concepts in game theory

Keywords and phrases Games on graphs, subgame-perfect equilibria, mean-payoff objectives

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.116

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2202.08499` [4]

Funding This work is partially supported by the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS & FWO), the COST Action 16228 GAMENET (European Cooperation in Science and Technology), and by the PDR project *Subgame perfection in graph games* (F.R.S.- FNRS).

Acknowledgements We wish to thank the anonymous reviewers for their useful comments, in particular for the question that led us to add Lemma 12 to this paper.

1 Introduction

Nash equilibria (NEs), a fundamental solution concept in game theory, are defined as strategy profiles such that no player can improve their payoff by changing unilaterally their strategy. So NEs can be interpreted as self-inforcing contracts from which there is no incentive to deviate unilaterally. Unfortunately, NEs are known to suffer, in sequential games like infinite duration games played on graphs, from the issue of *non-credible threats*: to enforce a NE, some players may threaten other players to play irrationally in order to punish deviations. This is allowed by the definition of NEs, as in case of deviation from one player, the other players are not bound to rational behaviors anymore and they can therefore play irrationally w.r.t. their own objectives in order to sanction the deviating player. This drawback of NEs has triggered the introduction of the notion of *subgame-perfect equilibria* (SPEs) [22], a more complex but more natural solution concept for sequential games. A strategy profile is an SPE if after every history, i.e. in every *subgame*, the strategies of the players still form an NE. Thus, SPEs impose rationality even after a deviation and only rational behaviors can be used to coerce the behavior of other players.

In this paper, we study the complexity of SPE problems in infinite-duration sequential games played on graphs with mean-payoff objectives. While NEs always exist in those games, as proved in [8], SPEs do not always exist as shown in [23, 7]. The *SPE threshold problem*, i.e. the problem of deciding whether a given mean-payoff game admits an SPE satisfying some constraints on the payoffs it grants to the players, has recently been proved to be decidable


in [3]. However, its worst-case computational complexity is open: [3] provides only an **NP** lower bound and a 2**ExpTime** upper bound. In this paper, we close this complexity gap and prove that the problem is actually **NP**-complete.

Contributions. The starting point of our algorithm is the characterization of SPEs recently presented in [3], based on the notions of *requirement* and *negotiation function*. A requirement on a game G is a function $\lambda : V \to \mathbb{R} \cup \{\pm\infty\}$, where V is the state space of G. For a given state v, the value $\lambda(v)$ should be understood as the minimal payoff that the player controlling the state v will require in a play traversing v in order to avoid deviating. A requirement captures, therefore, some level of *rationality* of the players. The negotiation function transforms each requirement λ into a (possibly stronger) requirement $\mathsf{nego}(\lambda)$, such that $\mathsf{nego}(\lambda)(v)$ is the best payoff that the player controlling v can ensure, while playing against a coalition of the other players that play rationally with regards to the requirement λ. A play is the outcome of an SPE if and only if it satisfies the requirement λ^*, the least fixed point of the negotiation function – or equivalently, one of its fixed points. We recall that result in Lemma 29. In order to obtain our nondeterministic polynomial time algorithm, the rest of the paper constructs a notion of witness recognizing the positive instances of the SPE threshold problem. Such witnesses admit three pieces. First, we show that the size of λ^* can be bounded by a polynomial function of the size of the game (Theorem 37). This result is obtained by showing that the set of fixed points of the negotiation function can be characterized by a finite union of polyhedra that in turn can be represented by linear inequations. While the number of inequations that are needed for that characterization may be large (it cannot be bounded polynomially), we show that each of those inequations have coefficients and constants whose binary representations can be bounded polynomially. As the least fixed point is the minimal value in this set, it is represented by a vertex of one of those polyhedra. Then this guarantees, using results that bounds the solutions of linear equalities, that the least fixed point has a binary representation that is polynomial and so it can be guessed in polynomial time by a nondeterministic algorithm: it will be the first piece of our notion of witness, in the non-deterministice algorithm we design to solve the SPE threshold problem. Second, we define a witness of polynomial size for the existence of a play, consistent with a given requirement, which generates a payoff vector between the desired thresholds (Theorem 38). This play is not guaranteed to be regular. Third, we define a witness of polynomial size to prove that a requirement is indeed a fixed point of the negotiation function. This notion of certificate relies on a new and more compact game characterization of the negotiation function called the *reduced negotiation game* (Definition 41, Theorem 44). These results are far from trivial as we also show that SPEs may rely on strategy profiles that are not regular and require infinite memory. As both the least fixed point and its two certificates can be guessed and verified in polynomial time, we obtain **NP** membership for the threshold problem, closing the complexity gap left open in [3] (Theorem 48).

Additionally, all the previous results do also apply to ε-SPEs, a quantitative relaxation of SPEs. In particular, Theorem 48 does also apply to the ε-SPE threshold problem.

Related works. Non-zero sum infinite duration games have attracted a large attention in recent years, with applications targeting reactive synthesis problems, see e.g. [1, 9, 10, 14, 19] and their references. We now detail other works more closely related to our contributions.

In [7], Brihaye et al. introduce and study the notion of weak SPE, which is a weakening of the classical notion of SPE. This weakening is equivalent to the original SPE concept on reward functions that are *continuous*. This is the case for example for the quantitative

reachability reward function, on which Brihaye et al. solve the SPE threshold problem in [6]. The mean-payoff cost function is not continuous and the techniques used in [7], and generalized in [11], cannot be used to characterize SPEs for the mean-payoff reward function.

In [21], Meunier develops a method based on Prover-Challenger games to solve the problem of the existence of SPEs on games with a finite number of possible payoffs. In mean-payoff games, the number of possible payoffs is uncountably infinite.

In [15], Flesch and Predtetchinski present another characterization of SPEs on games with finitely many possible payoffs, based on a game structure with infinite state space. In [3], Brice et al. define the notions of requirements and negotiation function. They prove that the negotiation function is characterized by a zero-sum two-player game called *abstract negotiation game*, which is similar to the game introduced in the characterization of Flesch and Predtetchinski. As a starting point for algorithms, they also provide an effective representation of this game, called *concrete negotiation game*, which turns out to be a zero-sum finite state multi-mean-payoff games [24]. Finally, they use those tools to prove that the SPE threshold problem is decidable for mean-payoff games. They left open the question of its precise complexity: they provide a **NP** lower bound and a 2**ExpTime** upper bound. In [5], the same authors use those tools to close the complexity gap for the SPE threshold problem in parity games, which had been proved to be **ExpTime**-easy and **NP**-hard by Ummels and Grädel in [16]. They prove that the problem is actually **NP**-complete. The techniques used in that paper heavily rely on the fact that parity objectives are ω-regular, which is not the case of mean-payoff games in general.

In [13], Chatterjee et al. study mean-payoff *automata*, and give a result that can be translated into an expression of all the possible payoff vectors in a mean-payoff game. In [2], Brenguier and Raskin give an algorithm to build the Pareto curve of a multi-dimensional two-player zero-sum mean-payoff game. To do so, they study systems of equations and of inequations, and they prove that they always admit simple solutions (with polynomial size). Those technical results will be used along this paper.

Structure of the paper. In Section 2, we introduce the necessary background. Section 3 recalls the notions of requirement and negotiation function, and link them to NEs and SPEs. Section 4 recalls results about the size of solutions of systems of equations or inequations, and use them to bound the size of the least fixed point of the negotiation function. Section 5 defines a witness for the existence of a λ-consistent play between two given thresholds. Section 6 introduces the reduced negotiation game that is a new compact characterization of the negotiation function. Finally, Section 7 applies those results to prove the **NP**-completeness of the SPE threshold problem on mean-payoff games. The proofs that are not presented here can be found in appendices of [4], the full version of this paper.

2 Background

Games, strategies, equilibria. In all what follows, we study infinite duration turn-based quantitative games on finite graphs with complete information.

▶ **Definition 1** (Game). A *non-initialized game* is a tuple $G = (\Pi, V, (V_i)_{i\in\Pi}, E, \mu)$, where:

- Π is a finite set of *players*;
- (V, E) is a directed graph, called the *underlying graph* of G, whose vertices are sometimes called *states* and whose edges are sometimes called *transitions*, and in which every state has at least one outgoing transition. For the simplicity of writing, a transition $(v, w) \in E$ will often be written vw;

- $(V_i)_{i\in\Pi}$ is a partition of V, in which V_i is the set of states *controlled* by player i;
- $\mu : V^\omega \to \mathbb{R}^\Pi$ is an *payoff function*, that maps each infinite word ρ to the tuple $\mu(\rho) = (\mu_i(\rho))_{i\in\Pi}$ of the players' *payoffs*.

An *initialized game* is a tuple (G, v_0), often written $G_{\restriction v_0}$, where G is a non-initialized game and $v_0 \in V$ is a state called *initial state*. We often use the word *game*, alone, for both initialized and non-initialized games.

▶ **Definition 2** (Play, history). A *play* (resp. history) in the game G is an infinite (resp. finite) path in the graph (V, E). It is also a play (resp. history) in the initialized game $G_{\restriction v_0}$, when v_0 is its first vertex. The set of plays (resp. histories) in the game G (resp. the initialized game $G_{\restriction v_0}$) is denoted by $\mathsf{Plays}G$ (resp. $\mathsf{Plays}G_{\restriction v_0}, \mathsf{Hist}G, \mathsf{Hist}G_{\restriction v_0}$). We write Hist_iG (resp. $\mathsf{Hist}_iG_{\restriction v_0}$) for the set of histories in G (resp. $G_{\restriction v_0}$) of the form hv, where v is a vertex controlled by player i.

Given a play ρ (resp. a history h), we write $\mathsf{Occ}(\rho)$ (resp. $\mathsf{Occ}(h)$) the set of vertices that appear in ρ (resp. h), and $\mathsf{Inf}(\rho)$ the set of vertices that appear infinitely often in ρ. For a given index k, we write $\rho_{\leq k}$ (resp. $h_{\leq k}$), or $\rho_{<k+1}$ (resp. $h_{<k+1}$), the finite prefix $\rho_0 \dots \rho_k$ (resp. $h_0 \dots h_k$), and $\rho_{\geq k}$ (resp. $h_{\geq k}$), or $\rho_{>k-1}$ (resp. $h_{>k-1}$), the infinite (resp. finite) suffix $\rho_k\rho_{k+1}\dots$ (resp. $h_kh_{k+1}\dots h_{|h|-1}$). Finally, we write $\mathsf{first}(\rho)$ (resp. $\mathsf{first}(h)$) the first vertex of ρ (and $\mathsf{last}(h)$ the last vertex of h).

▶ **Definition 3** (Strategy, strategy profile). A *strategy* for player i in the initialized game $G_{\restriction v_0}$ is a function $\sigma_i : \mathsf{Hist}_iG_{\restriction v_0} \to V$, such that $v\sigma_i(hv)$ is an edge of (V, E) for every hv. A history h is *compatible* with a strategy σ_i if and only if $h_{k+1} = \sigma_i(h_0 \dots h_k)$ for all k such that $h_k \in V_i$. A play ρ is compatible with σ_i if all its prefixes are.

A *strategy profile* for $P \subseteq \Pi$ is a tuple $\bar\sigma_P = (\sigma_i)_{i\in P}$, where each σ_i is a strategy for player i in $G_{\restriction v_0}$. A play or a history is *compatible* with $\bar\sigma_P$ if it is compatible with every σ_i for $i \in P$. A *complete* strategy profile, usually written $\bar\sigma$, is a strategy profile for Π. Exactly one play is compatible with a complete strategy profile: we write it $\langle\bar\sigma\rangle$, and call it the *outcome* of $\bar\sigma$.

When i is a player and when the context is clear, we will often write $-i$ for the set $\Pi \setminus \{i\}$. When $\bar\tau_P$ and $\bar\tau'_Q$ are two strategy profiles with $P \cap Q = \emptyset$, we write $(\bar\tau_P, \bar\tau'_Q)$ the strategy profile $\bar\sigma_{P\cup Q}$ such that $\sigma_i = \tau_i$ for $i \in P$, and $\sigma_i = \tau'_i$ for $i \in Q$.

Before moving on to SPEs, let us recall that an NE is a strategy profile such that no player has an incentive to deviate unilaterally.

▶ **Definition 4** (Nash equilibrium). Let $G_{\restriction v_0}$ be a game. The strategy profile $\bar\sigma$ is a *Nash equilibrium* – or *NE* for short – in $G_{\restriction v_0}$ if and only if for each player i and for every strategy σ'_i, called *deviation of* σ_i, we have the inequality $\mu_i\left(\langle\sigma'_i, \bar\sigma_{-i}\rangle\right) \leq \mu_i\left(\langle\bar\sigma\rangle\right)$.

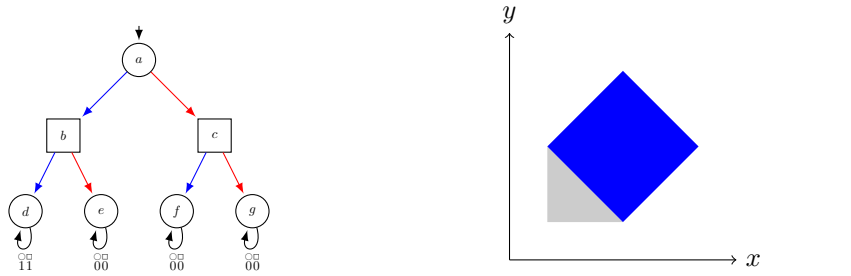
An SPE is a strategy profile whose all substrategy profiles are NEs.

▶ **Definition 5** (Subgame, substrategy). Let hv be a history in the game G. The *subgame* of G after hv is the game $\left(\Pi, V, (V_i)_i, E, \mu_{\restriction hv}\right)_{\restriction v}$, where $\mu_{\restriction hv}$ maps each play to its payoff in G, assuming that the history hv has already been played: formally, for every $\rho \in \mathsf{Plays}G_{\restriction hv}$, we have $\mu_{\restriction hv}(\rho) = \mu(h\rho)$. If σ_i is a strategy in $G_{\restriction v_0}$, its *substrategy* after hv is the strategy $\sigma_{i\restriction hv}$ in $G_{\restriction hv}$, defined by $\sigma_{i\restriction hv}(h') = \sigma_i(hh')$ for every $h' \in \mathsf{Hist}_iG_{\restriction hv}$.

▶ Remark. The initialized game $G_{\restriction v_0}$ is also the subgame of G after the one-state history v_0.

▶ **Definition 6** (Subgame-perfect equilibrium). Let $G_{\restriction v_0}$ be a game. The strategy profile $\bar\sigma$ is a *subgame-perfect equilibrium* – or *SPE* for short – in $G_{\restriction v_0}$ if and only if for every history h in $G_{\restriction v_0}$, the strategy profile $\bar\sigma_{\restriction h}$ is a Nash equilibrium in the subgame $G_{\restriction h}$.

The notion of subgame-perfect equilibrium refines the notion of Nash equilibrium and excludes coercion by non-credible threats.



(a) Two Nash equilibria. **(b)** An example for the operator $\llcorner$.

Figure 1 Illustration for preliminary notions.

▶ Example 7. Consider the game pictured in Figure 1a. It is initialized with initial state a, and has two players, player ○ and player □, who own respectively the circle and the square vertices. The payoff function assigns to each player a payoff of 1 for the play abd^ω, and 0 for all the other plays. Two different strategy profiles are represented here, one by the blue colored transitions, which has outcome abd^ω, one by the red colored ones, which has outcome acg^ω. Both are NEs: clearly, no player can increase their payoff by deviating from the blue choices, and in the case of the red profile, a deviation of player □ can only lead to the play acf^ω, and a deviation of player ○ to abe^ω – both plays give to both player the payoff 0. However, for player □, going from b to e is not a rational choice, hence the red profile is not an SPE, while the blue profile is one.

An ε-SPE is a strategy profile which is *almost* an SPE: if a player deviates after some history, they will not be able to improve their payoff by more than a quantity $\varepsilon \geq 0$. Note that a 0-SPE is an SPE, and conversely.

▶ **Definition 8** (ε-SPE)**.** Let $G_{\restriction v_0}$ be a game, and $\varepsilon \geq 0$. A strategy profile $\bar{\sigma}$ from v_0 is an ε-SPE if and only if for every history h, for every player i and every strategy σ'_i, we have $\mu_i(\langle \bar{\sigma}_{-i\restriction h}, \sigma'_{i\restriction h}\rangle) \leq \mu_i(\langle \bar{\sigma}_{\restriction h}\rangle) + \varepsilon$.

Mean-payoff games. We now turn to the definition of mean-payoff objectives.

▶ **Definition 9** (Mean-payoff, mean-payoff game)**.** In a graph (V, E), we associate to each mapping $r : E \to \mathbb{Q}$ the *mean-payoff function*:

$$\mathsf{MP}_r : h_0 \dots h_n \mapsto \frac{1}{n}\sum_{k=0}^{n-1} r_i\left(h_k h_{k+1}\right).$$

A game $G = (\Pi, V, (V_i)_i, E, \mu)$ is a *mean-payoff game* if its underlying graph is finite, and if there exists a tuple $(r_i)_{i\in\Pi}$ of reward functions, such that for each player i and every play ρ:

$$\mu_i(\rho) = \liminf_{n\to\infty} \mathsf{MP}_{r_i}(\rho_{\leq n}).$$

The mapping r_i is called *reward function* of player i: it represents the immediate reward that each action grants to player i. The final payoff of player i is their average payoff along the play, classically defined as the limit inferior[1] over n (since the limit may not be defined) of the average payoff after n steps. When the context is clear, we liberally write $\mathsf{MP}_i(h)$ for $\mathsf{MP}_{r_i}(h)$, and $\mathsf{MP}(h)$ for the tuple $(\mathsf{MP}_i(h))_i$, as well as $r(uv)$ for the tuple $(r_i(uv))_i$.

[1] An alternative definition of mean-payoff games exists, with a limit superior instead of inferior. While in zero-sum one dimensional games, the two definitions lead to the same notion of optimality, this is not the case when considering multiple dimensions, see e.g. [24]. All the results presented in this paper apply only on mean-payoff games defined with a limit inferior.

In the sequel, we develop a worst-case optimal algorithm to solve the *ε-SPE threshold problem*, which is a generalization of the *SPE threshold problem*, defined as follows.

▶ **Definition 10** (ε-SPE threshold problem)**.** Given a rational number $\varepsilon \geq 0$, a mean-payoff game $G_{\restriction v_0}$ and two thresholds $\bar{x}, \bar{y} \in \mathbb{Q}^\Pi$, does there exist an ε-SPE $\bar{\sigma}$ in $G_{\restriction v_0}$ such that $\bar{x} \leq \mu(\langle\bar{\sigma}\rangle) \leq \bar{y}$?

That problem is already known, by [3], to be 2**ExpTime** and **NP**-hard. The proof given in that paper does also show that **NP**-hardness still holds when ε is fixed to 0. Let us also add that the existence of an SPE in a given mean-payoff game, i.e. the same problem with no thresholds and with $\varepsilon = 0$, is itself **NP**-hard.

▶ **Definition 11** (SPE existence problem)**.** Given a mean-payoff game $G_{\restriction v_0}$, does there exist an SPE in $G_{\restriction v_0}$?

▶ **Lemma 12.** *The SPE existence problem is* **NP***-hard.*

Set of possible payoffs. A first important result that we need is the characterization of the set of possible payoffs in a mean-payoff game, which has been introduced in [13]. Given a graph (V, E), we write $\mathsf{SC}(V, E)$ the set of simple cycles it contains. Given a finite set D of dimensions and a set $X \subseteq \mathbb{R}^D$, we write $\mathsf{Conv}X$ the convex hull of X. We will often use the subscript notation $\mathsf{Conv}_{x \in X} f(x)$ for the set $\mathsf{Conv} f(X)$.

▶ **Definition 13** (Downward sealing)**.** Given a set $Y \subseteq \mathbb{R}^D$, the *downward sealing* of Y is the set $\llcorner Y = \left\{ (\min_{\bar{z} \in Z} z_d)_{d \in D} \;\middle|\; Z \text{ is a finite subset of } Y \right\}$.

▶ Example 14. In $\mathbb{R}^2$, if Y is the blue area in Figure 1b, then $\llcorner Y$ is the union of the blue area and the gray area.

▶ **Lemma 15** ([13])**.** *Let G be a mean-payoff game, whose underlying graph is strongly connected. The set of the payoffs $\mu(\rho)$, where ρ is a play in G, is exactly the set:*

$$\llcorner \left(\mathop{\mathsf{Conv}}_{c \in \mathsf{SC}(V,E)} \mathsf{MP}(c) \right).$$

Two-player zero-sum games. We now recall several definitions and two classical results about two-player zero-sum games.

▶ **Definition 16** (Two-player zero-sum game)**.** A *two-player zero sum game* is a game G with $\Pi = \{1, 2\}$ and $\mu_2 = -\mu_1$.

▶ **Definition 17** (Borel game)**.** A game G is *Borel* if the function μ, from the set V^ω equipped with the product topology to the Euclidian space $\mathbb{R}^\Pi$, is Borel, i.e. if, for every Borel set $B \subseteq \mathbb{R}^\Pi$, the set $\mu^{-1}(B)$ is Borel.

▶ Remark. Mean-payoff games are Borel (see [12]).

▶ **Lemma 18** (Determinacy of Borel games, [20])**.** *Let $G_{\restriction v_0}$ be a zero-sum Borel game, with $\Pi = \{1, 2\}$. Then, we have the following equality:*

$$\sup_{\sigma_1} \inf_{\sigma_2} \mu_1(\langle\bar{\sigma}\rangle) = \inf_{\sigma_2} \sup_{\sigma_1} \mu_1(\langle\bar{\sigma}\rangle).$$

That quantity is called *value* of $G_{\restriction v_0}$, denoted by $\mathsf{val}_1(G_{\restriction v_0})$.

▶ **Definition 19** (Optimal strategy). Let $G_{\restriction v_0}$ be a zero-sum Borel game, with $\Pi = \{1, 2\}$. The strategy σ_1 is *optimal* in $G_{\restriction v_0}$ if $\inf_{\sigma_2} \mu_1(\langle \sigma_1, \sigma_2 \rangle) = \mathsf{val}_1(G_{\restriction v_0})$.

Let us now define memoryless strategies, and a condition under which they can be optimal.

▶ **Definition 20** (Memoryless strategy). A strategy σ_i in a game $G_{\restriction v_0}$ is *memoryless* if for every vertex $v \in V_i$ and for all histories h and h', we have $\sigma_i(hv) = \sigma_i(h'v)$.

We usually write $\sigma_i(\cdot v)$ for the state $\sigma_i(hv)$ for every h. For every game $G_{\restriction v_0}$, we write $\mathsf{ML}(G_{\restriction v_0})$ for the set of memoryless strategies in $G_{\restriction v_0}$.

▶ **Definition 21** (Shuffling). Let ρ, η and θ be three plays in a game G. The play θ is a *shuffling* of ρ and η if there exist two sequences of indices $k_0 < k_1 < \dots$ and $\ell_0 < \ell_1 < \dots$ such that $\eta_0 = \rho_{k_0} = \eta_{\ell_0} = \rho_{k_1} = \eta_{\ell_1} = \dots$, and:

$$\theta = \rho_0 \dots \rho_{k_0-1} \eta_0 \dots \eta_{\ell_0-1} \rho_{k_0} \dots \rho_{k_1-1} \eta_{\ell_0} \dots \eta_{\ell_1-1} \dots .$$

▶ **Definition 22** (Convexity, concavity). A function $f : \mathsf{Plays} G \to \mathbb{R}$ is *convex* if every shuffling θ of two plays ρ and η satisfies $f(\theta) \geq \min\{f(\rho), f(\eta)\}$. It is *concave* if $-f$ is convex.

▶ Remark. Mean-payoff functions, defined with a limit inferior, are convex.

▶ **Lemma 23.** *In a two-player zero-sum game played on a finite graph, every player whose payoff function is concave has an optimal strategy that is memoryless.*

Proof. According to [18], this result is true for qualitative objectives, i.e. when μ can only take the values 0 and 1. It follows that for every $\alpha \in \mathbb{R}$, if a player i, whose payoff function is concave, has a strategy that ensures $\mu_i(\rho) \geq \alpha$ (understood as a qualitative objective), then they have a memoryless one. Hence the equality:

$$\mathsf{val}_1(G_{\restriction v_0}) = \sup_{\sigma_1 \in \mathsf{ML}(G_{\restriction v_0})} \inf_{\sigma_2} \mu_1(\langle \bar{\sigma} \rangle).$$

Since the underlying graph (V, E) is finite, memoryless strategies exist in finite number, hence the supremum above is realized by a memoryless strategy σ_1 that is, therefore, optimal. ◀

3 Requirements and negotiation

We now recall some notions and results from [3], which are the starting point of our algorithm.

Requirements. In the sequel, we write $\overline{\mathbb{R}}$ the set $\mathbb{R} \cup \{\pm\infty\}$.

▶ **Definition 24** (Requirement). A *requirement* on the game G is a mapping $\lambda : V \to \overline{\mathbb{R}}$.

For a given state v, the quantity $\lambda(v)$ represents the minimal payoff that the player controlling v will require in a play traversing the state v.

▶ **Definition 25** (λ-consistency). Let λ be a requirement on a game G. A play ρ in G is λ-*consistent* if and only if, for all $i \in \Pi$ and $n \in \mathbb{N}$ with $\rho_n \in V_i$, we have $\mu_i(\rho_{\geq n}) \geq \lambda(\rho_n)$. The set of λ-consistent plays from a state v is denoted by $\lambda\mathsf{Cons}(v)$.

▶ Remark. The set $\lambda\mathsf{Cons}(v)$ can be empty, and is not regular in general.

▶ **Definition 26** (λ-rationality). Let λ be a requirement on a mean-payoff game G. Let $i \in \Pi$. A strategy profile $\bar{\sigma}_{-i}$ is λ-*rational* if and only if there exists a strategy σ_i such that, for every history hv compatible with $\bar{\sigma}_{-i}$, the play $\langle \bar{\sigma}_{\restriction hv} \rangle$ is λ-consistent. We then say that the strategy profile $\bar{\sigma}_{-i}$ is λ-rational *assuming* σ_i. The set of λ-rational strategy profiles in $G_{\restriction v}$ is denoted by $\lambda\mathsf{Rat}(v)$.

Negotiation. In mean-payoff games, as well as in a wider class of games (see [3] and [15]), SPEs are characterized by the fixed points of the *negotiation function*, a function from the set of requirements into itself. We always use the convention $\inf \emptyset = +\infty$.

▶ **Definition 27** (Negotiation function)**.** Let G be a game. The *negotiation function* is the function that transforms each requirement λ on G into a requirement $\mathsf{nego}(\lambda)$ on G defined, for each $i \in \Pi$ and $v \in V_i$, by:

$$\mathsf{nego}(\lambda)(v) = \inf_{\bar{\sigma}_{-i} \in \lambda\mathsf{Rat}(v)} \sup_{\sigma_i} \mu_i(\langle \bar{\sigma} \rangle).$$

The quantity $\mathsf{nego}(\lambda)(v)$ is the best payoff the player controlling the state v can enforce if the other players play rationally with regards to the requirement λ.

▶ Remark. The negotiation function satisfies the following properties.

- It is monotone: if $\lambda \leq \lambda'$ (for the pointwise order), then $\mathsf{nego}(\lambda) \leq \mathsf{nego}(\lambda')$.
- It is also non-decreasing: for every λ, we have $\lambda \leq \mathsf{nego}(\lambda)$.
- There exists a λ-rational strategy profile from v against the player controlling v if and only if $\mathsf{nego}(\lambda)(v) \neq +\infty$.

Link with SPEs. The SPE outcomes in a mean-payoff game are characterized by the fixed points of the negotiation function, or equivalently by its least fixed point. That result can be extended to ε-SPEs. To that end, we recall the notion of ε-fixed points of a function.

▶ **Definition 28** (ε-fixed point)**.** Let $\varepsilon \geq 0$, let D be a finite set and let $f : \overline{\mathbb{R}}^D \to \overline{\mathbb{R}}^D$ be a mapping. A tuple $\bar{x} \in \mathbb{R}^D$ is a *ε-fixed point* of f if for each $d \in D$, for $\bar{y} = f(\bar{x})$, we have $y_d \in [x_d - \varepsilon, x_d + \varepsilon]$.

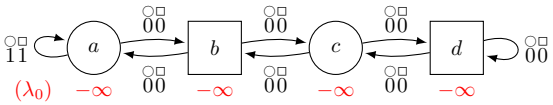
▶ Remark. A 0-fixed point is a fixed point, and conversely.

▶ **Lemma 29** ([3])**.** *Let $G_{\restriction v_0}$ be a mean-payoff game, and let $\varepsilon \geq 0$. The negotiation function on G has a least ε-fixed point λ^*, and given a play ρ in $G_{\restriction v_0}$, the three following assertions are equivalent: (1) the play ρ is an ε-SPE outcome; (2) the play ρ is λ-consistent for some ε-fixed point λ of the negotiation function; (3) the play ρ is λ^*-consistent.*

The abstract negotiation game. Given λ and $u \in V$, the quantity $\mathsf{nego}(\lambda)(u)$ can be characterized as the value of a *negotiation game*, a two-player zero-sum game opposing the player *Prover*, who simulates a λ-rational strategy profile and wants to minimize player i's payoff, and the player *Challenger*, who simulates player i's reaction by accepting or refusing Prover's proposals. Two negotiation games were defined in [3]. Conceptually simpler, the *abstract negotiation game* $\mathsf{Abs}_{\lambda i}(G)_{\restriction u}$ unfolds as follows:

- from the state v, Prover chooses a λ-consistent play ρ from the state v and proposes it to Challenger. If Prover has no play to propose, the game is over and Challenger gets the payoff $+\infty$.
- Once a play ρ has been proposed, Challenger can accept it. Or he can *deviate*, and choose a prefix $\rho_{\leq k}$ with $\rho_k \in V_i$ and a new transition $\rho_k w \in E$.
- In the former case, the game is over. In the latter, it starts again from the state w.

If Challenger finally accepts a proposal ρ, then his payoff is $\mu_i(\rho)$. If he deviates infinitely often, then Prover's proposals and his deviations construct a play $\dot{\pi} = \rho^{(0)}_{\leq k_0} \rho^{(1)}_{\leq k_1} \rho^{(2)}_{\leq k_2} \cdots$. Then, Challenger's payoff is $\mu_i(\dot{\pi})$. It has been proved in [3] that the equality $\mathsf{nego}(\lambda)(u) = \mathsf{val}_{\mathbb{C}}(\mathsf{Abs}_{\lambda i}(G)_{\restriction u})$ holds. Thus, the abstract negotiation game captures a first intuition on how the negotiation function can be computed.



(λ_0)	$-\infty$	$-\infty$	$-\infty$	$-\infty$
(λ_1)	1	0	0	0
(λ_2)	1	1	0	0
(λ_3)	1	1	1	0
(λ_4)	1	1	1	1

Figure 2 Iterations of the negotiation function.

▶ Example 30. Let G be the game of Figure 2, where each edge is labelled by the rewards $r_{\bigcirc}$ and $r_{\square}$. Below the states, we present the requirements $\lambda_0 : v \mapsto -\infty$, $\lambda_1 = \mathsf{nego}(\lambda_0)$, $\lambda_2 = \mathsf{nego}(\lambda_1)$, $\lambda_3 = \mathsf{nego}(\lambda_2)$, and $\lambda_4 = \mathsf{nego}(\lambda_3)$. Let us explicate those computations, using the abstract negotiation game. From λ_0 to λ_1: since every play is λ_0-consistent, Prover can always propose whatever she wants. From the state a, whatever she (trying to minimize player $\bigcirc$'s payoff) proposes, Challenger can always make player $\bigcirc$ deviate in order to loop on the state a. Then, in the game G, player $\bigcirc$ gets the payoff 1, hence $\lambda_1(a) = 1$. From the state b, Prover (trying to minimize player $\square$'s payoff) can propose the play $(bc)^\omega$. If Challenger makes player $\square$ deviate to go to the state a, then Prover can propose the play $a(bc)^\omega$. Even if Challenger makes player $\square$ deviate infinitely often, he cannot give him more than the payoff 0, hence $\lambda_1(b) = 0$. Similar situations happen from the states c and d, hence $\lambda_1(c) = \lambda_1(d) = 0$. From λ_1 to λ_2: now, from the state b, whatever Prover proposes at first, Challenger can make player $\square$ deviate and go to the state a. From there, since we have $\lambda_1(a) = 1$, Prover has to propose a play in which player $\bigcirc$ gets the payoff 1. The only such plays do also give the payoff 1 to player $\square$, hence $\lambda_2(b) = 1$. Similar situations explain $\lambda_3(c) = 1$, and $\lambda_4(c) = 1$. Finally, plays ending with the loop a^ω are all λ_4-consistent, hence Prover can always propose them, hence the requirement λ_4 is a fixed point of the negotiation function – and therefore the least. By Lemma 29, the SPE plays in G are exactly the plays in which both player $\bigcirc$ and player $\square$ get the payoff 1.

The concrete negotiation game. The abstract negotiation game cannot be directly used for an algorithmic purpose, since it has an infinite state space. However, it can be turned into a game on a finite graph if Prover does not propose plays as a whole, but edge by edge. In the *concrete negotiation game* $\mathsf{Conc}_{\lambda i}(G)_{\restriction (u,\{u\})}$, the states controlled by Prover have the form (v, M), where $M \subseteq V$ memorizes the states seen since the last time Challenger did deviate, in order to control that the play Prover is constructing since that moment is effectively λ-consistent: for each $u \in M$, Prover has to give to the player controlling u at least the payoff $\lambda(u)$. Similarly, the states controlled by Challenger are of the form (vv', M), where $vv' \in E$ is an edge proposed by Prover. The game unfolds as follows:

- from the state (v, M), Prover chooses a transition vv' and proposes it to Challenger.
- Once a transition vv' has been proposed, Challenger can accept it. Or, if $v \in V_i$, he can *deviate*, and choose a new transition vw.
- If the former case, the game starts again from the state $(v', M \cup \{v'\})$. In the latter, it starts from the state $(w, \{w\})$.

For every play $\pi = (\rho_0, M_0)(\rho_0\rho'_0, M_0)(\rho_1, M_1)(\rho_1\rho'_1, M_1)\dots$ in $\mathsf{Conc}_{\lambda i}(G)$, we write $\dot{\pi} = \rho_0\rho_1\dots$ the play in G constructed by Prover's proposals and Challenger's deviations. Then, Challenger's payoff in the play π is either $+\infty$ if there exists an index k such that the suffix $\pi_{\geq 2k}$ contains no deviation and $\dot{\pi}_{\geq k}$ is not λ-consistent, and $\mu_i(\dot{\pi})$ otherwise.

In [3], a first algorithm was proposed to solve the ε-SPE threshold problem, using the fact that in the concrete negotiation game, Challenger has a memoryless optimal strategy, to design a complete representation of the negotiation function, and to compute its least ε-fixed point. However, that algorithm requires doubly exponential time, because it needs to enumerate all the memoryless strategies available for Challenger, whose number is exponential in the size of the concrete game, itself exponential in the size of G. Here, we make use of the concrete negotiation game only to bound the size of the least ε-fixed point: our algorithm will use a third negotiation game, the *reduced negotiation game*.

4 Size of the least ε-fixed point

In this section, after having recalled some results about the sizes of solutions to linear equations and inequations, we prove that the least ε-fixed point of the negotiation function in a game G has a size that is polynomial in the size of G and ε. The first piece of the witnesses identifying positive instances of the ε-SPE threshold problem will then be an ε-fixed point of the negotiation function of polynomial size.

About size, equations and inequations. We define here the notion of size that we use.

▶ **Definition 31** (Size)**.** The *size* of a rational number $r = \frac{p}{q}$, where $p, q \in \mathbb{Z}$ are co-prime, is the quantity $\|r\| = 1 + \lceil \log_2(|p| + 1) \rceil + \lceil \log_2(|q| + 1) \rceil$. The size of an irrational number is $+\infty$. The size of the infinite numbers is $\|+\infty\| = \|-\infty\| = 1$. The size of a tuple $\bar{x} \in O^D$, where D is a set and O is a set of objects for which the notion of size has been defined, is the quantity $\mathsf{card} D + \sum_{d \in D} \|x_d\|$. Similarly, the size of a function $f : D \to X$ is the quantity $\mathsf{card} D + \sum_{d \in D} \|f(d)\|$, and the size of a set $X \subseteq O$ is the quantity $\mathsf{card} X + \sum_{x \in X} \|x\|$.

The proof of Theorem 37 below requires the manipulation of *polytopes*, e.g. downward sealings of convex hulls (from Lemma 15), expressed as solution sets of *systems of linear inequations*.

▶ **Definition 32** (Linear equations, inequations, systems)**.** Let D be a finite set. A *linear equation* in $\mathbb{R}^D$ is a pair $(\bar{a}, b) \in \left(\mathbb{R}^D \setminus \{\bar{0}\}\right) \times \mathbb{R}$. The *solution set* of the equation $(\bar{a}, b)$ is the set $\mathsf{Sol}_=(\bar{a}, b) = \{\bar{x} \in \mathbb{R}^D \mid \bar{a} \cdot \bar{x} = b\}$, where $\cdot$ denotes the canonical scalar product on the euclidian space $\mathbb{R}^D$. A set $X \subseteq \mathbb{R}^D$ is a *hyperplane* of $\mathbb{R}^D$ if it is the solution set of some linear equation. A *system of linear equations* is a finite set Σ of linear equations. The *solution set* of the system Σ is the set $\mathsf{Sol}_=\Sigma = \bigcap_{(\bar{a},b) \in \Sigma} \mathsf{Sol}_=(\bar{a}, b)$. A set $X \subseteq \mathbb{R}^D$ is a *linear subspace* of $\mathbb{R}^D$ if it is the solution set of some system of linear equations.

A *linear inequation* in $\mathbb{R}^D$ is a pair $(\bar{a}, b) \in \left(\mathbb{R}^D \setminus \{\bar{0}\}\right) \times \mathbb{R}$. The *solution set* of the inequation $(\bar{a}, b)$ is the set $\mathsf{Sol}_\geq(\bar{a}, b) = \{\bar{x} \in \mathbb{R}^D \mid \bar{a} \cdot \bar{x} \geq b\}$. A set $X \subseteq \mathbb{R}^D$ is a *half-space* of $\mathbb{R}^D$ if it is the solution set of some linear inequation. A *system of linear inequations* is a finite set Σ of linear inequations. The *solution set* of the system Σ is the set $\mathsf{Sol}_\geq\Sigma = \bigcap_{(\bar{a},b) \in \Sigma} \mathsf{Sol}_\geq(\bar{a}, b)$. A set $X \subseteq \mathbb{R}^D$ is a *polyhedron* of $\mathbb{R}^D$ if it is the solution set of some system of linear inequations Σ. A *vertex* of X is a point $\bar{x} \in \mathbb{R}^D$ such that $\{\bar{x}\} = \mathsf{Sol}_=(\Sigma')$ for some subset $\Sigma' \subseteq \Sigma$. A *polytope* is a bounded polyhedron.

▶ Remark. Polyhedra are closed sets. The polytopes of $\mathbb{R}^D$ are exactly the sets of the form $\mathsf{Conv}(S)$, where S is a finite subset of $\mathbb{R}^D$.

▶ **Lemma 33** ([13])**.** *Let Σ be a system of inequations, and let $X = \mathsf{Sol}_\geq(\Sigma)$. The set $\llcorner X$ is itself a polyhedron, and there exists a system of inequations Σ' such that $\llcorner X = \mathsf{Sol}_\geq(\Sigma')$ and that for every $(\bar{a}', b') \in \Sigma'$, there exists $(\bar{a}, b) \in \Sigma$ with $\|(\bar{a}', b')\| \leq \|(\bar{a}, b)\|$.*

▶ **Lemma 34** ([2], Theorem 1). *There exists a polynomial P_1 such that, for every system of equations Σ, there exists a point $\bar{x} \in \mathsf{Sol}_{=}\Sigma$, such that $\|\bar{x}\| \leq P_1\left(\max_{(\bar{a},b)\in\Sigma}\|(\bar{a},b)\|\right)$.*

▶ **Corollary 35.** *For every system of inequations Σ, each vertex $\bar{x}$ of the polyhedron $\mathsf{Sol}_{\geq}(\Sigma)$ has size $\|x\| \leq P_1\left(\max_{(\bar{a},b)\in\Sigma}\|(\bar{a},b)\|\right)$.*

Note that in Lemma 33, in Lemma 34 and in Corollary 35, the number of equations or inequations has no influence. A consequence of Lemma 34 is the following result.

▶ **Lemma 36.** *There exists a polynomial P_2 such that, for each finite set D and every finite subset $X \subseteq \mathbb{R}^D$, there exists a system of linear inequations Σ, such that $\mathsf{Sol}_{\geq}(\Sigma) = \mathsf{Conv}(X)$ and $\|(\bar{a},b)\| \leq P_2(\|X\|)$ for every $(\bar{a},b) \in \Sigma$.*

Size of the least ε-fixed point. The following theorem bounds the size of the least fixed point of the negotiation function. As we used the notation $\bar{x}$ for tuples so far, we use the notation $\bar{\bar{x}}$ for tuples of tuples.

▶ **Theorem 37.** *There exists a polynomial P_3 such that for every mean-payoff game G, the least ε-fixed point λ^* of the negotiation function has size $\|\lambda^*\| \leq P_3(\|G\| + \|\varepsilon\|)$.*

Proof sketch. It has been proved in [3] that Challenger has a memoryless optimal strategy in every concrete negotiation game. Given a requirement λ, a player i, a state $v \in V_i$ and a memoryless strategy $\tau_{\mathbb{C}}$, we can construct the set of payoff vectors $\mu(\dot{\pi})$, where π is a play in $\mathsf{Conc}_{\lambda i}(G)_{\restriction(v,\{v\})}$ compatible with $\tau_{\mathbb{C}}$, as a union of polytopes defined using Lemma 15. If we intersect the upward closures of those sets, then $\mathsf{nego}(\lambda)(v)$ is equal to the least value x_i, where $\bar{x}$ belongs to that intersection. Therefore, if $X_\lambda \subseteq \mathbb{R}^{V\times\Pi}$ is the product of those intersections, then for each i and $v \in V_i$, we have $\mathsf{nego}(\lambda) = \inf\{x_{vi} \mid \bar{\bar{x}} \in X_\lambda\}$.

To each tuple of tuples $\bar{\bar{x}} \in \mathbb{R}^{V\times\Pi}$, we associate the requirement $\lambda_{\bar{\bar{x}}}$ defined by $\lambda_{\bar{\bar{x}}}(v) = x_{vi} - \varepsilon$ for each $i \in \Pi$ and $v \in V_i$. Then, we define $X = \left\{\bar{\bar{x}} \;\middle|\; \bar{\bar{x}} \in X_{\lambda_{\bar{\bar{x}}}}\right\}$, and we show that any requirement λ is an ε-fixed point of the negotiation function if and only $\lambda = \lambda_{\bar{\bar{x}}}$ for some $\bar{\bar{x}} \in X$. Then, the set X is itself a union of polyhedra, hence the linear mapping $\bar{\bar{x}} \mapsto \sum_v \lambda_{\bar{\bar{x}}}(v)$ has its minimum over X on some vertex $\bar{\bar{x}}$ of one of those polyhedra. The requirement λ^* is equal to $\lambda_{\bar{\bar{x}}}$, hence its size can be bounded using Corollary 35. ◀

5 Constrained existence of a λ-consistent play

We claim that a non-deterministic algorithm can recognize the positive instances of the ε-SPE threshold problem by guessing an ε-fixed point λ of the negotiation function. Once λ has been guessed, according to Lemma 29, two assertions must be proved: on the one hand, that there exists a λ-consistent play between the two desired thresholds, and on the other hand, that λ is actually an ε-fixed point of the negotiation function. The latter will be handled later through the concept of *reduced negotiation game*. Now, we tackle the former, and provide the second piece of our notion of witness: to prove the existence of a λ-consistent play ρ with $\bar{x} \leq \mu(\rho) \leq \bar{y}$, we need to guess the sets $W = \mathsf{Inf}(\rho)$ and $W' = \mathsf{Occ}(\rho)$, and a tuple of tuples $\bar{\bar{\alpha}} \in [0,1]^{\Pi\times\mathsf{SC}(W)}$ indicating how ρ combines the cycles of W, i.e. such that:

$$\mu(\rho) = \left(\min_{j\in\Pi} \sum_{c\in\mathsf{SC}(W)} \alpha_{jc}\mathsf{MP}_i(c)\right)_i.$$

▶ **Theorem 38.** *There exists a polynomial P_4 such that for every mean-payoff game $G_{\restriction v_0}$, for every $\bar{x},\bar{y} \in \mathbb{R}^V$, and for every requirement λ on G, there exists a λ-consistent play ρ in $G_{\restriction v_0}$ satisfying $\bar{x} \leq \mu(\rho) \leq \bar{y}$ if and only if there exist two sets $W \subseteq W' \subseteq V$ and a tuple of tuples $\bar{\bar{\alpha}} \in [0,1]^{\Pi\times\mathsf{SC}(W)}$ such that:*

- *the set W is strongly connected in (V, E), and accessible from the state v_0 using only and all the states of W';*
- *for each player i, we have $\sum_c \alpha_{ic} = 1$, and:*

$$x_i \leq \min_{j\in\Pi} \sum_{c\in\mathsf{SC}(W)} \alpha_{jc}\mathsf{MP}_i(c) \leq y_i;$$

- *for each player i and $v \in W \cap V_i$, we have:*

$$\min_{j\in\Pi} \sum_{c\in\mathsf{SC}(W)} \alpha_{jc}\mathsf{MP}_i(c) \geq \lambda(v);$$

- $\|\bar{\bar{\alpha}}\| \leq P_4(\|G, \bar{x}, \bar{y}, \lambda\|)$.

Proof. Let us first notice that given a set $X \subseteq \mathbb{R}^\Pi$, the elements of the set $\llcorner(\mathsf{Conv}X)$ are exactly the tuples of the form:

$$\left(\min_{j\in\Pi} \sum_{x\in X} \alpha_{jx} x\right)_{i\in\Pi}$$

for some tuple $\bar{\bar{\alpha}} \in \mathbb{R}^{\Pi\times X}$ satisfying $\sum_x \alpha_{ix} = 1$ for each x.

Now, let us assume that W, W' and $\bar{\bar{\alpha}}$ exist. Then, there exists a play η with $\mathsf{Occ}(\eta) = \mathsf{Inf}(\eta) = W$ with payoff vector:

$$\mu(\eta) = \left(\min_{j\in\Pi} \sum_{x\in X} \alpha_{jx} x\right)_{i\in\Pi}.$$

Moreover, since W is accessible from v_0 using all and only the vertices of W', there exists a history $h\eta_0$ from v_0 to η_0 with $\mathsf{Occ}(h) = W'$. Then, the play $\rho = h\eta$ is λ-consistent and satisfies $\bar{x} \leq \mu(\rho) \leq \bar{y}$.

Conversely, if the play ρ exists: let $W = \mathsf{Inf}(\rho)$ and $W' = \mathsf{Occ}(\rho)$. The polytope:

$$Z = \left\{ \mu(\eta) \;\middle|\; \begin{array}{c} \eta \in \lambda\mathsf{Cons}(G_{\restriction v_0}), \\ \mathsf{Inf}(\eta) = W, \\ \mathsf{Occ}(\eta) = W', \\ \text{and } \bar{x} \leq \mu(\eta) \leq \bar{y} \end{array} \right\} = \left\{ \bar{z} \in \llcorner \left(\underset{c\in\mathsf{SC}(W)}{\mathsf{Conv}} \mathsf{MP}(c) \right) \;\middle|\; \begin{array}{c} \bar{x} \leq \bar{z} \leq \bar{y}, \text{ and} \\ \forall i, \forall v \in W' \cap V_i, z_i \geq \lambda(v) \end{array} \right\}$$

(the equality holds by Lemma 15) is nonempty (it contains at least $\mu(\rho)$). By Lemma 36, the set $\mathsf{Conv}_{c\in SC(W)}\mathsf{MP}(c)$ is defined by a system of inequations which all have size $\|(\bar{a}, b)\| \leq P_2\left(\max_c \|\mathsf{MP}(c)\|\right)$. Since by Lemma 33, the inequations defining $\llcorner\left(\mathsf{Conv}_{c\in SC(W)}\mathsf{MP}(c)\right)$ are not larger, there exists a polynomial P_6, independent of $G, \bar{x}, \bar{y}$ and λ, such that Z is defined by a system of inequations Σ such that for every $(\bar{a}, b) \in \Sigma$, we have $\|(\bar{a}, b)\| \leq P_6(\|(G, \bar{x}, \bar{y}, \lambda)\|)$. Therefore, by Corollary 35, the polytope Z admits a vertex $\bar{z}$ of size $\|\bar{z}\| \leq P_1(P_6(\|(G, \bar{x}, \bar{y}, \lambda)\|))$.

Then, since we have $\bar{z} \in \llcorner\left(\underset{c\in\mathsf{SC}(W)}{\mathsf{Conv}} \mathsf{MP}(c)\right)$, that vertex is, according to Definition 13, of the form:

$$\bar{z} = \left(\min_j \sum_c \alpha_{jc}\mathsf{MP}_j(c)\right)_i$$

for some tuple of tuples $\bar{\bar{\alpha}} \in [0,1]^{\Pi\times\mathsf{SC}(W)}$ with $\sum_c \alpha_{ic} = 1$ and having, by Corollary 35 again, size $\|\bar{\bar{\alpha}}\| \leq P_1\left(\max_{i\in\Pi} \sum_{c\in\mathsf{SC}(W)} \|\mathsf{MP}_i(c)\| + \|z_i\|\right)$, i.e. $\|\bar{\bar{\alpha}}\| \leq P_4(\|(G, \bar{x}, \bar{y}, \lambda)\|)$ for some polynomial P_4 independent of $G, \bar{x}, \bar{y}$ and λ. ◀

Now, we need the third piece of our witness, which will be evidence of the fact that the requirement λ is an ε-fixed point of the negotiation function.

6 The reduced negotiation game

The abstract negotiation game has an infinite (and uncountable) state space in general, and the concrete negotiation game has an exponential one. In [5], the infiniteness of the abstract negotiation game has been handled in the case of parity games, by proving that Prover has an optimal strategy that is memoryless, and that proposes only simple plays with a finite representation. Unfortunately, this result does not apply to mean-payoff games, where Prover needs infinite memory in general. That fact is illustrated in the next example.

▶ Example 39. In the game of Figure 3a, the requirement λ defined by $\lambda(a) = \lambda(b) = 1$ is a fixed point of the negotiation function (it is actually the least fixed point). Indeed, from the state a (the situation is symmetrical from the state b), consider the strategy for Prover that proposes always, from the state v, the play $vb^{|h|^2}(a^3b^3)^\omega$, where h is the history that has already been constructed by her proposals and Challenger's deviations. If Challenger accepts such a play, then he gets the payoff 1. If he deviates infinitely often, then Prover loops longer and longer on the state b, and he also gets the payoff 1. The loop on b corresponds to what we will call later a *punishing cycle*. Now, if Prover uses only finite memory, Challenger can get a payoff better than 1 by always deviating and go to b as soon as he can: then, edges giving to player ○ the reward 2 will occur with a nonzero frequency.

However, the plays proposed by Prover in the previous example are very similar: only the number of repetitions of the loop b does increase. More generally, one observes that Prover can play optimally while always proposing a play of the form $hc^n\rho$, where h, c and ρ are constant, and only the number n increases, quadratically with the time – so that Challenger's payoff is dominated by the mean-payoff $\mathsf{MP}_i(c)$ if he deviates infinitely often.

▶ **Definition 40** (Punishment family)**.** A *punishment family* is a set of plays of the form:

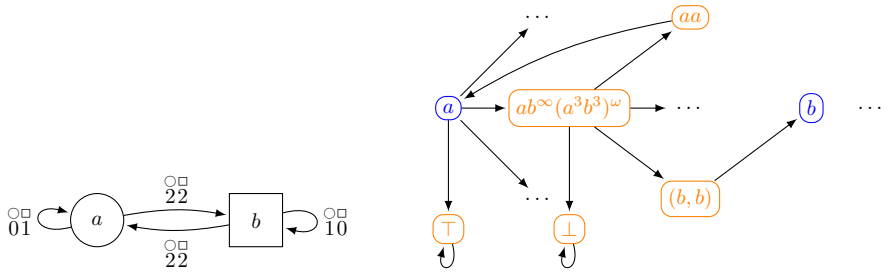
$$\{hc^n\rho \mid n > 0, \mu(\rho) = \bar{x}, \mathsf{Occ}(\rho) = W\}$$

where h is a simple history, c is a nonempty simple cycle, and where $W \subseteq V$ and $\bar{x} \in \mathbb{R}^\Pi$. The cycle c is called its *punishing cycle*. For every $\beta \in \mathbb{N}$, a *β-punishment family* is a punishment family with $\|\bar{x}\| \leq \beta$. A β-punishment family is represented by the data h, c, $\mu(\rho)$ and $\mathsf{Occ}(\rho)$, and that representation has a size smaller than or equal to the quantity $3\mathsf{card}V\lceil\log_2(\mathsf{card}V + 1)\rceil + \beta$.

We write $hc^\infty\rho$ for the punishment family $\{hc^n\rho' \mid n > 0, \mu(\rho') = \mu(\rho), \mathsf{Occ}(\rho') = \mathsf{Occ}(\rho)\}$. Beware that the play ρ matters only for its payoff vector and the vertices it traverses: if $\mathsf{Occ}(\rho) = \mathsf{Occ}(\rho')$ and $\mu(\rho) = \mu(\rho')$, then $hc^\infty\rho = hc^\infty\rho'$. We write $\mu(hc^\infty\rho)$ for the common payoff vector of all elements of $hc^\infty\rho$, and we will say that $hc^\infty\rho$ is λ-consistent if all its elements are (or equivalently, if one of its elements is). Let us clarify that a punishment family is not an equivalence class: for example, in the game of Figure 3a, the play ab^ω belongs to both $a^\infty b^\omega$ and $ab^\infty b^\omega$, which are distinct. We can now define the *reduced negotiation game*, where Prover proposes β-punishment families instead of plays.

▶ **Definition 41** (Reduced negotiation game)**.** Let G be a mean-payoff game, let λ be a requirement, let i be a player, let $v_0 \in V_i$ and let β be a natural integer. The corresponding *reduced negotiation game* is the game $\mathsf{Red}^\beta_{\lambda i}(G)_{\restriction v_0} = (\{\mathbb{P}, \mathbb{C}\}, S, (S_\mathbb{P}, S_\mathbb{C}), \Delta, \nu)_{\restriction v_0}$, where:

- the player $\mathbb{P}$ is called *Prover*, and the player $\mathbb{C}$ *Challenger*;
- the states controlled by Prover are the states of G, i.e. $S_\mathbb{P} = V$;
- the states controlled by Challenger are the states of the form $[hc^\infty\rho]$, (c, u) or $[h'v]$, where:



(a) The game G.

(b) The reduced negotiation game.

Figure 3 A game on which Prover needs infinite memory.

- $hc^\infty\rho$ is a λ-consistent β-punishment family,
- there exists a state $\rho_k \in V_i$ along the play ρ such that $\rho_k u \in E$,
- and $h'v$ is a history such that h' is a prefix of the history hc, and $\mathsf{last}(h') \in V_i$;

plus two additional states, written $\top$ and $\bot$;

- with the same notations, the set Δ contains the transitions of the form:
  - $v[hc^\infty\rho]$ (Prover proposes a punishment family);
  - $v\bot$ (Prover gives up);
  - $[hc^\infty\rho]\top$ (Challenger accepts Prover's proposal);
  - $[hc^\infty\rho][h'v]$ (Challenger deviates before the punishing cycle – *pre-cycle deviation*);
  - $[hc^\infty\rho](c,u)$ (Challenger deviates after the punishing cycle – *post-cycle deviation*);
  - $[h'v]v$ and $(c,u)u$ (Prover has now to propose a new play);
  - $\top\top$ and $\bot\bot$ (the play is over);
- given a history $H = H_0 \dots H_n \in \mathsf{HistRed}^\beta_{\lambda i}(G)$ that does not reach the state $\bot$, we write $\dot{H} = h^{(1)} \dots h^{(n)}$ the history or play in G defined by, for each k:
  - if $H_{k-1}H_k = v[hc^\infty\rho]$, then $h^{(k)}$ is empty;
  - if $H_{k-1}H_k = [hc^\infty\rho]\top$, then $h^{(k)} \dots h^{(n)} = hc^{\left|h^{(1)}\dots h^{(k-1)}h\right|^2}\rho$ (the number of times the cycle c is repeated depends quadratically on the time);
  - if $H_{k-1}H_k = [hc^\infty\rho][h'v]$, then $h^{(k)} = h'$;
  - if $H_{k-1}H_k = [hc^\infty\rho](c,v)$, then $h^{(k)} = hc^{\left|h^{(1)}\dots h^{(k-1)}h\right|^2}h'$, where h' is among the shortest histories such that $\mathsf{Occ}(h') \subseteq \mathsf{Occ}(\rho)$, $\mathsf{last}(h') \in V_i$ and $\mathsf{last}(h')v \in E$;
  - if $H_{k-1}H_k = [h'v]v$ or $(c,v)v$, then $h^{(k)}$ is empty;

  and that definition is naturally extended to plays: for example, if G is the game of Figure 3a and if $\pi = a[ab^\infty a^\omega][aa]a[ab^\infty a^\omega](b,b)b[b^\infty a^\omega]\top^\omega$, then $\dot{\pi} = a \cdot ab^{2^2} \cdot ab^{7^2}a^\omega = a^2b^4ab^{49}a^\omega$;
- the payoff function ν is defined, for each play π, by $\nu_{\mathbb{C}}(\pi) = -\nu_{\mathbb{P}}(\pi) = +\infty$ if π reaches the state $\bot$, and $\nu_{\mathbb{C}}(\pi) = -\nu_{\mathbb{P}}(\pi) = \mu_i(\dot{\pi})$ otherwise.

▶ Example 42. Figure 3b illustrates a (small) part of the game $\mathsf{Red}^2_{\lambda\bigcirc}(G)_{\restriction a}$, where G is the game of Figure 3a, and $\lambda(a) = \lambda(b) = 1$. Blue states are owned by Prover, orange ones by Challenger. When Prover proposes the punishment family $ab^\infty(a^3b^3)^\omega$, the function $\nu_{\mathbb{C}}$ interprets it as the play $ab^{|h|^2}(a^3b^3)^\omega$, where h is the history that has already been constructed so far.

▶ Remark. Reduced negotiation games are Borel, and are played on a finite graph.

Link with the negotiation function. We will now prove that the reduced negotiation game captures the negotiation function, as do the abstract and concrete ones. For that purpose, we first need the following key result.

▶ **Lemma 43.** *In a reduced negotiation game, Prover has a memoryless optimal strategy.*

Proof. This lemma is a consequence of Lemma 23: the payoff function $\nu_{\mathbb{C}}$ is concave. Indeed, let ξ be a shuffling of two plays π and χ. If either π or χ reaches the state $\bot$ (in which case both do), then we immediately have $\nu_{\mathbb{C}}(\dot{\xi}) \leq \max\{\nu_{\mathbb{C}}(\dot{\pi}), \nu_{\mathbb{C}}(\dot{\chi})\} = +\infty$. Otherwise, the play $\dot{\xi}$ is a shuffling of $\dot{\pi}$ and $\dot{\chi}$, and since mean-payoff objectives defined with a limit inferior are convex, we have $\nu_{\mathbb{C}}(\dot{\xi}) \leq \max\{\nu_{\mathbb{C}}(\dot{\pi}), \nu_{\mathbb{C}}(\dot{\chi})\}$. ◀

This lemma enables us to prove that the reduced negotiation game is equivalent to the other negotiation games.

▶ **Theorem 44.** *There exists a polynomial P_4 such that for every mean-payoff game G, every requirement λ with rational values, each player i and each $v_0 \in V_i$, for every $\beta \geq P_4(\|G\| + \|\lambda\|)$, we have* $\mathsf{nego}(\lambda)(v_0) = \mathsf{val}_{\mathbb{C}}\left(\mathsf{Red}^{\beta}_{\lambda_i}(G)_{\restriction v_0}\right)$.

Proof. For every mean-payoff game G and every requirement λ, we assume:

$$\beta \geq P_1\left(P_2\left(\|\{\mathsf{MP}(c) \mid c \in \mathsf{SC}(G)\}\|\right)\right)$$

and for each $v \in V$:

$$\beta \geq \|\lambda(v)\| + 3,$$

which are indeed quantities that are bounded by a polynomial of $\|G\| + \|\lambda\|$.

- *First direction:* $\mathsf{nego}(\lambda)(v_0) \geq \mathsf{val}_{\mathbb{C}}\left(\mathsf{Red}^{\beta}_{\lambda_i}(G)_{\restriction v_0}\right)$.
  Let $\bar{\sigma}_{-i}$ be a strategy profile in G that is λ-rational assuming a strategy σ_i, and let $x = \sup_{\sigma'_i} \mu_i(\langle \bar{\sigma}_{-i}, \sigma'_i \rangle)$. We wish to prove that there exists a strategy $\tau_{\mathbb{P}}$ in the reduced negotiation game such that $\sup_{\tau_{\mathbb{C}}} \nu_{\mathbb{C}}(\langle \bar{\tau} \rangle) \leq x$. Thus, we will have proved that the quantity $\mathsf{val}_{\mathbb{C}}\left(\mathsf{Red}^{\beta}_{\lambda_i}(G)_{\restriction v_0}\right)$ is smaller than or equal to every such x, and therefore smaller than or equal to $\mathsf{nego}(\lambda)(v_0)$.
  Let us define simultaneously the strategy $\tau_{\mathbb{P}}$ and a mapping $\varphi : \mathsf{Hist}_{\mathbb{P}}\mathsf{Red}^{\beta}_{\lambda_i}(G)_{\restriction v_0} \to \mathsf{Hist}G_{\restriction v_0}$, such that for each history H, the punishment family $\tau_{\mathbb{P}}(H)$ will be defined from the play $\langle \bar{\sigma}_{\restriction \varphi(H)} \rangle$. We guarantee inductively that if $H \in \mathsf{Hist}_{\mathbb{P}}\mathsf{Red}^{\beta}_{\lambda_i}(G)_{\restriction v_0}$ is compatible with $\tau_{\mathbb{P}}$, then $\varphi(H) \in \mathsf{Hist}G_{\restriction v_0}$ is compatible with $\bar{\sigma}_{-i}$. First, let us define $\varphi(v_0) = v_0$.
  Let $H \in \mathsf{Hist}_{\mathbb{P}}\mathsf{Red}^{\beta}_{\lambda_i}(G)_{\restriction v_0}$ be a history compatible with $\tau_{\mathbb{P}}$ as it has been defined so far, and such that $\varphi(H)$ has already been defined. Let $\eta^0 = \langle \bar{\sigma}_{\restriction \varphi(H)} \rangle$. By induction hypothesis, the history $\varphi(H)$ is compatible with $\bar{\sigma}_{-i}$, hence the play η^0 is λ-consistent, and satisfies $\mu_i(\eta^0) \leq x$.
  Let $\eta^0_{\leq \ell}$ be the shortest prefix of η^0 that is not simple, i.e. such that there exists $k < \ell$ with $\eta^0_k = \eta^0_\ell$. If $\mathsf{MP}_i\left(\eta^0_{k+1} \dots \eta^0_\ell\right) \leq x$, then we define $\tau_{\mathbb{P}}(H) = \left[\eta^0_{\leq k}\left(\eta^0_{k+1} \dots \eta^0_\ell\right)^{\infty} \rho\right]$, where ρ is a play such that $\mathsf{Occ}(\rho) = \mathsf{Occ}\left(\eta^0_{>\ell}\right)$, that $\mu_i(\rho) \leq x$, and that $\|\mu(\rho)\| \leq \beta$. Such a play exists, because the polytope:

$$Z = \left\{ \mu(\rho) \;\middle|\; \begin{array}{c} \forall j, \forall v \in V_j \cap \mathsf{Occ}(\eta^0), \mu_j(\rho) \geq \lambda(v), \\ \text{and } \mathsf{Occ}(\rho) = \mathsf{Occ}\left(\eta^0_{>\ell}\right) \end{array} \right\}$$

  is nonempty (it contains $\eta^0_{>\ell}$), and has at least one vertex $\bar{z}$ with $z_i \leq x$ (because $\mu_i(\eta^0_{>\ell}) \leq x$), which by Lemma 36 and Corollary 35 has size $\|\bar{z}\| \leq \beta$.

Otherwise, if $\mathsf{MP}_i\left(\eta^0_{k+1}\dots\eta^0_\ell\right) > x$, we define $\eta^1 = \eta^0_{\leq k}\eta^0_{>\ell}$, and we iterate the process, which does necessarily terminate – because $\mu_i(\eta^0) \leq x$. As a consequence, it effectively defines the proposal $\tau_{\mathbb{P}}(H) = \left[\eta^n_{\leq k}\left(\eta^n_{k+1}\dots\eta^n_\ell\right)^\infty\rho\right]$, for some n. Then, for each prefix hv, we define $\varphi\left(H\left[\eta^n_{\leq k}\left(\eta^n_{k+1}\dots\eta^n_\ell\right)^\infty\rho\right][hv]v\right) = \varphi(H)\eta^0_{\leq m}$, where $\eta^0_{\leq m}$ is the prefix of η^0 of which n simple cycles have been pulled out to obtain the prefix h of η^n; and similarly, for each pair (c,v), we define $\varphi\left(H\left[\eta^n_{\leq k}\left(\eta^n_{k+1}\dots\eta^n_\ell\right)^\infty\rho\right](c,v)v\right) = \varphi(H)\eta^0_{\leq m}$, where $\eta^0_{\leq m}$ is the prefix of η^0 from which n simple cycles have been pulled out to obtain the shortest prefix $\eta^n_{\leq p}$ of η^n such that $\eta^n_p \in V_i$ and $\eta^n_p v \in E$.

Thus, the mapping φ is defined on every history compatible with $\tau_{\mathbb{P}}$, and the image of such a history is always a history compatible with $\bar\sigma_{-i}$. We define it arbitrarily on other histories. Note that for each history H, the history $\dot H$ can be obtained from $\varphi(H)$ by pulling out cycles c satisfying $\mathsf{MP}_i(c) > x$, and adding cycles d with $\mathsf{MP}_i(d) \leq x$. As a consequence, if $\mathsf{MP}_i(\varphi(H)) \leq x$, then $\mathsf{MP}_i(\dot H) \leq x$ – and the same result is true when we naturally extend the mapping φ to plays.

Let us now prove that $\sup_{\tau_{\mathbb{C}}}\nu_{\mathbb{C}}(\langle\bar\tau\rangle) \leq \sup_{\sigma'_i}\mu_i(\langle\bar\sigma_{-i},\sigma'_i\rangle)$. Let π be a play compatible with $\tau_{\mathbb{P}}$:

- the state $\bot$ does not appear in π, because Prover's strategy does never use a transition to it.
- If π has the form $\pi = H[hc^\infty\rho]\top^\omega$: then, we have $\nu_{\mathbb{C}}(\pi) = \mu_i(\rho) \leq x$.
- If π is made of infinitely many deviations: the play $\varphi(\pi)$ is compatible with $\bar\sigma_{-i}$, hence $\mu_i(\varphi(\pi)) \leq x$; which implies $\mu_i(\dot\pi) \leq x$, i.e. $\nu_{\mathbb{C}}(\pi) \leq x$.

- *Second direction:* $\mathsf{nego}(\lambda)(v_0) \leq \mathsf{val}_{\mathbb{C}}\left(\mathsf{Red}^\beta_{\lambda i}(G)_{\restriction v_0}\right)$.

Let $\tau_{\mathbb{P}}$ be a memoryless strategy for Prover in the reduced negotiation game, and let $y = \sup_{\tau_{\mathbb{C}}}\nu_{\mathbb{C}}(\langle\bar\tau\rangle)$. We want to show that $\mathsf{nego}(\lambda)(v_0) \leq y$: by Lemma 43, it will be enough to conclude. If $y = +\infty$, it is clear. Let us assume that $y \neq +\infty$. Then, we will define a strategy profile $\bar\sigma$, where $\bar\sigma_{-i}$ is λ-rational assuming σ_i, such that $\sup_{\sigma'_i}\mu_i(\langle\bar\sigma_{-i},\sigma'_i\rangle) \leq y$: we proceed inductively by defining the play $\langle\bar\sigma_{\restriction hv}\rangle$ for each history hv compatible with $\bar\sigma_{-i}$ such that h is empty, or $\mathsf{last}(h) \in V_i$ and $v \neq \sigma_i(h)$. Such a history is called a *bud history*. After other histories, the strategy profile can be defined arbitrarily. To that end, we construct a mapping ψ which maps each bud history to a history $\psi(hv) \in \mathsf{Hist}_{\mathbb{P}}\mathsf{Red}^\beta_{\lambda i}(G)_{\restriction v_0}$ that is compatible with $\tau_{\mathbb{P}}$. This mapping will induce a definition of $\bar\sigma$: since $y \neq +\infty$, we have $\tau_{\mathbb{P}}(\psi(hv)) \neq \bot$: let then $[h'c^\infty\rho] = \tau_{\mathbb{P}}(\psi(hv))$. We then define $\langle\bar\sigma_{\restriction hv}\rangle = h'c^{|hh'|^2}\rho$, which is a λ-consistent play since $h'c^\infty\rho$ is a λ-consistent punishment family, by definition of the reduced negotiation game.

Let now h_0v be a bud history: we assume that $\bar\sigma$ has been defined on every prefix of h_0, but not on h_0v itself. If h_0 is empty, that is if $h_0v = v_0$, then we define $\psi(h_0v) = v_0$. Otherwise, let us write $h_0 = h_1wh_2$, where h_1w is the longest prefix of h_0 that is a bud history – that is, its longest prefix such that $\psi(h_1w)$ has been defined, or its shortest prefix such that wh_2 is compatible with $\bar\sigma_{\restriction h_1w}$. Let $H = \psi(h_1w)$, and let $[hc^\infty\rho] = \tau_{\mathbb{P}}(H)$. We have defined $\langle\bar\sigma_{\restriction h_1w}\rangle = hc^{|h_1h|^2}\rho$, and consequently, the history wh_2 is a prefix of that play. If it is a prefix of the history hc, then we define $\psi(h_0v) = H[hc^\infty\rho][wh_2v]v$. Otherwise, we define $\psi(h_0v) = H[hc^\infty\rho](c,v)v$.

Now, the strategy profile $\bar\sigma$ has been defined, and since all the punishment families proposed by Prover are λ-consistent, the strategy profile $\bar\sigma_{-i}$ is λ-rational assuming σ_i. Let η be a play compatible with $\bar\sigma_{-i}$, and let us prove that $\mu_i(\eta) \leq y$. If η has finitely many prefixes that are bud histories, then let $\eta_{\leq n}$ be the longest one: we have $\eta_{\geq n} = hc^{n+|h|}\rho$, where $[hc^\infty\rho] = \tau_{\mathbb{P}}(\psi(\eta_{\leq n}))$. Then, we have $\mu_i(\eta) = \mu_i(\rho) \leq y$.

Now, if η has infinitely many such prefixes, then there exists a unique play π in the reduced negotiation game such that for any prefix $\eta_{\leq n}$ of η that is a bud history, the history $\psi(\eta_{\leq n})$ is a prefix of π. Then, if π contains finitely many post-cycle deviations, then there exist two indices m and n such that $\eta_{\geq m} = \dot{\pi}_{\geq n}$, hence $\mu_i(\eta) = \mu_i(\dot{\pi}) \leq y$. Finally, if π contains infinitely many post-cycle deviations, i.e. infinitely many occurrences of a state of the form (c, v), then let us choose such state that minimizes the quantity $\mathsf{MP}_i(c)$. The play η has the form:

$$\eta = h_0 c^{k_0^2} h_1 c^{k_1^2} h_2 \dots,$$

where for each n, we have $k_n = \left| h_0 c^{k_0^2} \dots c^{k_{n-1}^2} h_n \right|$. Then, if we write $M = \max r_i$, we have:

$$\mathsf{MP}_i\left(h_0 c^{k_0^2} \dots h_n c^{k_n^2}\right) \leq \frac{1}{k_n + k_n^2|c| - 1}\left(k_n M + \left(k_n^2|c| - 1\right)\mathsf{MP}_i(c)\right),$$

which converges to $\mathsf{MP}_i(c)$ when n tends to $+\infty$, hence $\mu_i(\eta) \leq \mathsf{MP}_i(c)$. Now, since $\tau_{\mathbb{P}}$ is memoryless, there exists a play of the form HC^ω that is compatible with it, and such that $(c, v) \in \mathsf{Occ}(C) \subseteq \mathsf{Inf}(\pi)$; and by definition of y, we have $\mathsf{MP}_i(\dot{C}) = \nu_{\mathbb{C}}(HC^\omega) \leq y$. By minimality of $\mathsf{MP}_i(c)$, we have $\mathsf{MP}_i(\dot{C}) = \mathsf{MP}_i(c)$, hence $\mathsf{MP}_i(c) \leq y$, and therefore $\mu_i(\eta) \leq y$. ◀

Thus, a given requirement λ is an ε-fixed point of nego if and only if for each i and $v \in V_i$, there exists a memoryless strategy $\tau_{\mathbb{P}}$ in the game $\mathsf{Red}^\beta_{\lambda i}$, with $\beta = P_4(\|G\| + \|\lambda\|)$, such that $\sup_{\tau_{\mathbb{C}}} \nu_{\mathbb{C}}(\langle\bar{\tau}\rangle) \leq \lambda(v) + \varepsilon$. The reduced negotiation game has an exponential size, but it contains only $\mathsf{card} V$ states that are controlled by Prover: memoryless strategies for Prover are therefore objects of polynomial size. Thus, such memoryless strategies constitute the third and last piece of our notion of witness.

7 Algorithm and complexity

We are now in a position to define formally our notion of witness.

▶ **Definition 45** (Witness). Let $I = (G_{\restriction v_0}, \bar{x}, \bar{y}, \varepsilon)$ be an instance of the ε-SPE threshold problem. A *witness* for I is a tuple $\left(W, W', \bar{\bar{\alpha}}, \lambda, (\tau_{\mathbb{P}}^v)_v\right)$, where $W \subseteq W' \subseteq V$; $\bar{\bar{\alpha}} \in [0,1]^{\Pi \times \mathsf{SC}(W)}$; λ is a requirement; and each $\tau_{\mathbb{P}}^v$ is a memoryless strategy in the game $\mathsf{Red}^\beta_{\lambda i}(G)_{\restriction v}$, where $\beta = P_4(\|G\| + \|\lambda\|)$. A witness is *valid* if:

- each strategy $\tau_{\mathbb{P}}^v$ satisfies the inequality $\sup_{\tau_{\mathbb{C}}} \nu_{\mathbb{C}}(\langle \tau_{\mathbb{P}}^v, \tau_{\mathbb{C}}\rangle) \leq \lambda(v) + \varepsilon$;
- the sets W and W' and the tuple of tuples $\bar{\bar{\alpha}}$ satisfy the hypotheses of Theorem 38.

▶ Remark. The sets W and W', as well as the tuple of strategies $(\tau_{\mathbb{P}}^v)_v$, have polynomial size. In order to bound the size of witnesses by a polynomial, we only have to bound $\|\lambda\|$ and $\left\|\bar{\bar{\alpha}}\right\|$.

The ε-SPE threshold problem will be **NP**-easy if we show, first, that there exists a valid witness of polynomial size if and only if the instance is positive, and second, that the validity of a witness can be decided in polynomial time. The former is a consequence of Lemma 29, Theorem 37, Theorem 38, Theorem 44, and Lemma 43:

▶ **Lemma 46.** *There exists a polynomial P_5 such that an instance I of the ε-SPE threshold problem admits a valid witness of size $P_5(\|I\|)$ if and only if it is a positive instance.*

Let us now tackle the latter.

▶ **Lemma 47.** *Given an instance of the ε-SPE threshold problem and a witness for it, deciding whether that witness is valid is* **P***-easy.*

Proof. The validity of a witness is defined by two conditions. As regards the second one, all the hypotheses of Theorem 38 can be checked in polynomial time with classical algorithms. Let us now show how the first condition can also be checked in polynomial time.

Let $n = \mathsf{card} V$. Given a memoryless strategy $\tau_{\mathbb{P}}^v$ of Prover in a reduced negotiation game, one can construct in a time polynomial in $\|\tau_{\mathbb{P}}^v\|$ the graph $\mathsf{Red}_{\lambda i}^{\beta}(G)[\tau_{\mathbb{P}}^v]$, defined as the underlying graph of $\mathsf{Red}_{\lambda i}^{\beta}(G)$ where all the transitions that are not compatible with $\tau_{\mathbb{P}}^v$ have been omitted, as well as all the states that are, then, no longer accessible from the state v. That graph has indeed a polynomial size, because it is composed only of:

- at most n vertices of the form $w \in V$;
- at most n vertices of the form $\tau_{\mathbb{P}}(\cdot w)$ (either equal to $\bot$ or of the form $[hc^\infty\rho]$);
- at most n^2 vertices of the form (c, w);
- at most $2n^2$ vertices of the form $[h'w']$, where h' is a prefix of the history hc for some punishment family $[hc^\infty\rho] = \tau_{\mathbb{P}}(\cdot w)$;
- possibly the state $\top$.

We call this connected graph the *deviation graph*. Note that if among those vertices, there is the vertex $\bot$, then since the vertices that are not accessible have been removed, we have $\sup_{\tau_{\mathbb{C}}} \nu_{\mathbb{C}}(\langle\bar{\tau}\rangle) = +\infty$ and the problem can be solved immediately. In what follows, we assume that it is not the case, i.e. that for each w, the state $\tau_{\mathbb{P}}(\cdot w)$ has the form $[hc^\infty\rho]$. Deciding whether $\sup_{\tau_{\mathbb{C}}} \nu_{\mathbb{C}}(\langle\bar{\tau}\rangle) \leq \alpha$ is then equivalent to deciding whether there exists a path π, in that graph, such that $\nu_{\mathbb{C}}(\pi) > \alpha$. Such a play can have three forms.

- It can end in the state $\top$, i.e. with Challenger accepting Prover's proposal. The existence of such a play can be decided immediately, by checking whether in the deviation graph, there exists a vertex of the form $[hc^\infty\rho]$ with $\mu_i(hc^\infty\rho) > \alpha$.
- It can avoid the state $\top$, and comprise finitely many post-cycle deviations. This is the case if and only if there exists a cycle C in the deviation graph, without post-cycle deviations, such that $\mathsf{MP}_i(\dot{C}) > \alpha$. The existence of such a cycle can be decided in polynomial time with Karp's algorithm (see [17]).
- It can avoid the state $\top$, and comprise infinitely many post-cycle deviations. In that case, we have $\nu_{\mathbb{C}}(\pi) \leq \mathsf{MP}_i(c)$ for each state of the form (c, w) appearing infinitely often along π; then, there exists a cycle C in the deviation graph, such that every state of the form (c, w) along C satisfies $\mathsf{MP}_i(c) > \alpha$. Conversely, if such a cycle exists, then π exists. The existence of such a cycle can be decided in polynomial time with Karp's algorithm.

Therefore, the existence of such a play is decidable in polynomial time. ◀

Thus, given an instance of the ε-SPE threshold problem, a valid witness can be guessed and checked in polynomial time. Since the ε-SPE threshold problem has been proved to be **NP**-hard in [3], we finally obtain the following theorem:

▶ **Theorem 48.** *The ε-SPE threshold problem in mean-payoff games is* **NP***-complete.*

References

1 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016.

2 Romain Brenguier and Jean-François Raskin. Pareto curves of multidimensional mean-payoff games. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, volume 9207 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2015.

3 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Subgame-perfect equilibria in mean-payoff games. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 8:1–8:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.8.

4 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. The complexity of spes in mean-payoff games. *CoRR*, abs/2202.08499, 2022. arXiv:2202.08499.

5 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. On the complexity of spes in parity games. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.10.

6 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. In *CONCUR*, volume 140 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

7 Thomas Brihaye, Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Weak subgame perfect equilibria and their application to quantitative reachability. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 504–518. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

8 Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer cost games with simple nash equilibria. In *Logical Foundations of Computer Science, International Symposium, LFCS 2013, San Diego, CA, USA, January 6-8, 2013. Proceedings*, volume 7734 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2013.

9 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017.

10 Véronique Bruyère. Synthesis of equilibria in infinite-duration games on graphs. *ACM SIGLOG News*, 8(2):4–29, 2021.

11 Véronique Bruyère, Stéphane Le Roux, Arno Pauly, and Jean-François Raskin. On the existence of weak subgame perfect equilibria. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 145–161, 2017.

12 Krishnendu Chatterjee. Concurrent games with tail objectives. *Theor. Comput. Sci.*, 388(1-3):181–198, 2007.

13 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2010.

14 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010.

15 János Flesch and Arkadi Predtetchinski. A characterization of subgame-perfect equilibrium plays in borel games of perfect information. *Math. Oper. Res.*, 42(4):1162–1179, 2017.

16 Erich Grädel and Michael Ummels. Solution Concepts and Algorithms for Infinite Multiplayer Games. In Krzysztof Apt and Robert van Rooij, editors, *New Perspectives on Games and Interaction*, volume 4 of *Texts in Logic and Games*, pages 151–178. Amsterdam University Press, 2008.

17 Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discret. Math.*, 23(3):309–311, 1978.

18 Eryk Kopczynski. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2006.

19 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.

20 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, pages 363–371, 1975.

21 Noémie Meunier. *Multi-Player Quantitative Games: Equilibria and Algorithms.* PhD thesis, Université de Mons, 2016.

22 Martin J. Osborne. *An introduction to game theory.* Oxford Univ. Press, 2004.

23 Eilon Solan and Nicolas Vieille. Deterministic multi-player dynkin games. *Journal of Mathematical Economics*, 39(8):911–929, 2003.

24 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015.

On the Size of Good-For-Games Rabin Automata and Its Link with the Memory in Muller Games

Antonio Casares
LaBRI, Université de Bordeaux, France

Thomas Colcombet
CNRS, IRIF, Université Paris Cité, France

Karoliina Lehtinen
CNRS, Aix-Marseille Université, Université de Toulon, LIS, France

Abstract

In this paper, we look at good-for-games Rabin automata that recognise a Muller language (a language that is entirely characterised by the set of letters that appear infinitely often in each word). We establish that minimal such automata are exactly of the same size as the minimal memory required for winning Muller games that have this language as their winning condition. We show how to effectively construct such minimal automata. Finally, we establish that these automata can be exponentially more succinct than equivalent deterministic ones, thus proving as a consequence that chromatic memory for winning a Muller game can be exponentially larger than unconstrained memory.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Infinite duration games, Muller games, Rabin conditions, omega-regular languages, memory in games, good-for-games automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.117

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2204.11333

Funding *Thomas Colcombet*: ANR Delta and Duall

Acknowledgements We would like to thank Marthe Bonamy and Pierre Charbit for their help with graph theory.

1 Introduction

Games. Games, as considered in this work, are played by two antagonistic players, called the existential and universal players, who move a token around finite edge-coloured directed graphs. When the token lands on a position belonging to one of the players, this player moves it along an outgoing edge onto a new position. At the end of the day, the players have constructed an infinite path, called a play, and the winner is determined based on some language $\mathbb{W}$ of winning infinite sequences of colours, called the winning condition (we call $\mathbb{W}$-games the games which use the winning condition $\mathbb{W}$). Solving such games consists of deciding whether the existential player has a winning strategy, i.e. a way to guarantee, whatever the moves of the opponent are, that the play will end up in the winning condition. Solving infinite duration games is at the crux of many algorithms used in verification, synthesis, and automata theory [9, 19, 38, 30]. Difficulties in solving them are both theoretical and practical, and many questions pertaining to game resolution still remain unanswered.

Memory. Several parameters are relevant for solving a game: its size, of course, but also its winning condition and the complexity of winning strategies. A measure of this complexity is the memory used by a strategy. The simplest strategies are those that use no memory (positional strategies): decisions depend exclusively on the current position, and not on the past of the game. A strategy uses a finite amount of memory if the information that we need to retain from the past can be summarized by a finite state machine that processes the sequence of moves played in the game. In this case, the amount of memory used by the strategy is the number of states of this machine. Given a winning condition $\mathbb{W}$, a fundamental question is what is the minimal quantity m such that if the existential player wins a $\mathbb{W}$-game, there is a winning strategy using a memory of size m (we call m the memory requirements of $\mathbb{W}$). In addition to its size, a memory also has structure, which further elucidates the game dynamics. Understanding both the size and the structure of memories for $\mathbb{W}$ is a crucial step to design algorithms for solving $\mathbb{W}$-games.

Question A: Give a structural description of the optimal memory in $\mathbb{W}$-games.

Muller conditions. While there is a large zoo of winning conditions in the literature, here we are interested in ω-regular ones (described by finite state automata over infinite words), and, in particular, so called Muller conditions, for which the winner depends only on the colours that are seen infinitely often in the play. Memory requirements for Muller conditions have been studied in depth by Dziembowski, Jurdziński and Walukiewicz [15]. They provide a "formula" for computing the size of the minimal memory sufficient for winning in all games with a given Muller winning condition, based on the Zielonka tree [43], which describes the structure of a Muller condition. The Zielonka tree has also been used to characterise the memory requirements of Muller conditions when randomised strategies are allowed [21] and to provide minimal parity automata recognising a Muller condition [11]. This fundamental structure is also at the heart of our contribution.

Game reductions and good-for-gameness. When confronted with a $\mathbb{W}$-game, a standard solution is to reduce it to a game with a larger underlying graph, but a simpler winning condition. The typical way to do this (but not the only one) is to perform the composition of the game with a suitable automaton with another acceptance condition $\mathbb{W}'$ that accepts the language $\mathbb{W}$. The result is an $\mathbb{W}'$-game which has as size the product of the size of the original game and the size of the automaton. There is a subtlety here: not all automata can be used for this operation. For a non-deterministic automaton, this is in general incorrect, while using a deterministic automaton is always correct. So here, finding a minimal deterministic automaton for a given language improves the complexity of game resolution, and there is a large body of research in this direction (see [11, 26, 31] for Muller conditions, [1, 10, 41, 42] for minimisation of automata, and [32, 39, 36, 37, 40, 34, 31, 14, 29] for determinisation). However, some non-deterministic automata can also be used to perform this reduction. These are called good-for-games automata (GFG) [20, 12].

Some languages are known to be recognised by good-for-games automata that are exponentially more succinct than any equivalent deterministic automaton [27], and several lines of research concerning good-for-games automata are under study (how to decide "good-for-gameness" [3, 27, 4, 6], how expressive is "good-for-gameness" for pushdown automata [28, 18] what are good-for-games quantitative automata [5], etc). However, one key question that has not yet been addressed concerning good-for-games automata is how to design techniques as general as possible for building them. To the best of our knowledge, the only existing result in this direction is a polynomial-time algorithm to minimise co-Büchi GFG automata [1].

Question B: Provide general tools for constructing good-for-games automata.

In this paper, in the context of Muller conditions, we relate these two lines of study, and in particular give partial answers to the general questions A and B. Indeed, we show that the memory needed to win in L-games for a Muller language L coincides with the size of minimal GFG Rabin automata for L, and, in this sense, we give a structural description of the memory for Muller games, thus giving a refined answer to question A in this case. We also provide an optimal way to construct these minimal good-for-games automata, thus answering question B in the context of Muller conditions.

Contributions

1. We show that for all ω-regular languages L, the size (number of states) of a good-for-games Rabin-automaton for L is an upper bound on the memory that the existential player needs to implement winning strategies for L-games. This inequality is straightforward, but had not been stated explicitly prior to this work.
2. We establish that when L is a Muller language, the following two quantities are equal: the least size of a good-for-games Rabin-automaton for L and the least memory required for the existential player in all L-games in which she wins. Furthermore, we provide an efficient way to construct such a minimal automaton from the Zielonka tree of the condition [43]. This automaton can be seen, in a certain way, as a quotient of the minimal deterministic parity automaton for this language, as described in [11].
   Let us note that the least amount of memory needed to win a Muller game was described precisely by Dziembowski, Jurdziński and Walukiewicz [15]. We show here that the optimal strategy described in [15] can be implemented in a good-for-games Rabin-automaton. In combination with Item 1, this provides another proof of the upper bound in [15].
3. Finally, we provide a family of Muller languages such that the smallest GFG Rabin automata recognising it are of linear size in the number of letters, while equivalent deterministic Rabin automata grow exponentially. Note that the least size of a deterministic Rabin automaton for a Muller language L is known to coincide with the chromatic memory needed for winning L-games [10] (i.e. a memory that is updated based only on the letters seen, independently of the position in the game). The question of equivalence between chromatic memory and memory was asked by Kopcyński [23, 24], and an arbitrary difference between these two notions was established only recently by Casares [10]. Our new result, which is incomparable, shows that the chromatic memory can grow exponentially in the size of the alphabet, even when the general memory remains linear.

Together these three points develop techniques to solve Muller games in an optimal way by means of good-for-games Rabin automata reductions. The last point shows that an exponential gain can be achieved compared to using classical deterministic Rabin automata. Overall, our contribution supplements our understanding of Muller languages and highlights the – so far unexplored – fundamental role of GFG automata in the equation. Indeed, up to now GFG automata had mainly been studied for their succinctness, expressivity or algorithmic properties. Here, we shed light on a novel dimension of this automata class.

Related work. There is vast amount of literature on the memory requirements of different games. The first results in this direction where the proofs of the positionality of parity conditions and half-positionality of Rabin conditions [16, 22] and the finite-memory determinacy of Muller games [19]. The exact memory requirements of Muller conditions where characterised in [15]. In his PhD Thesis [23, 24], Kopczyński characterises several classes of

conditions that are half-positional, introduces the concept of chromatic memories (memories that are updated based only on colours seen) and provides an algorithm to decide the chromatic memory requirements of a winning condition. Conditions that are positional for both players over all graphs where characterised in [13] and those that are positional over finite graphs in [17]. More recently, these two results have been generalized to finite-memory conditions [8, 7]. The memory requirements have been proved to be different to the chromatic memory requirements in general [10], but conditions that are finite-memory determined are also chromatic-finite-memory determined [25].

Structure of this document. In Section 2, we describe the classical definitions related to our work such as games, automata and good-for-gamesness. In Section 3, we show why good-for-games Rabin automaton can be used as a memory structure for the existential player, the optimality of the construction for Muller conditions, and how to construct the least such automaton. In Section 4, we establish that this construction can be exponentially more succinct than deterministic Rabin automata. Section 5 concludes the paper.

2 Definitions

Notations. $|A|$ denotes the cardinality of a set A, $\mathcal{P}(A)$ its power set and $\mathcal{P}_+(A) = \mathcal{P}(A)\backslash\{\emptyset\}$. For a finite non-empty alphabet Σ, we write Σ^* and Σ^ω for the sets of finite and *infinite words* over Σ, respectively. The empty word is denoted by ε. Given $w = w_0w_1w_2\cdots \in \Sigma^\omega$, we denote $Inf(w) \subseteq \Sigma$ the set of letters that appear infinitely often in w. We let $w[i..j]$ be the finite word $w_iw_{i+1}...w_j$ if $i \leq j$, and ε if $j < i$.

We extend maps $\sigma : A \to B$ to A^* and A^ω component-wise and we denote these extensions by σ whenever no confusion arises. For a positive rational number $q \in \mathbb{Q}$ we denote by $\lfloor q \rfloor$ the greatest integer $n \in \mathbb{N}$ such that $n \leq q$.

2.1 Games and their memory

Games. We consider turn-based infinite duration games played between the existential and the universal player (referred to as *Exist* and *Univ*) over a directed graph. Formally, a Γ-coloured *game* is a tuple $\mathcal{G} = (V = V_E \uplus V_A, E, x_0, \mathbb{W})$, which consists of a set of vertices V partitioned into Exist's positions V_E and Univ's ones, V_A; a set of *transitions* (also called edges or moves) $E \subseteq V \times (\Gamma \cup \{\varepsilon\}) \times V$; an initial vertex $x_0 \in V$ and a subset $\mathbb{W} \subseteq \Gamma^\omega$ of winning sequences. We make the assumptions that there is at least one move from every position and that no cycle is labelled exclusively by ε. We will denote by $\gamma : E \to \Gamma \cup \{\varepsilon\}$ the function that assigns to each edge its colour. We write $Out(x)$ for the set of outgoing moves from x, that is, $Out(x) = \{e \in E \ : \ e = (x, c, x') \text{ for some } x' \in V, c \in \Gamma \cup \{\varepsilon\}\}$. If Γ is a game using the winning condition $\mathbb{W}$ we call it a $\mathbb{W}$-*game*.

Each player moves a pebble along an outgoing edge whenever it lands on a position belonging to that player, forming an infinite path $\pi \in E^\omega$ starting in x_0 called a *play*.

We denote $\gamma(\pi)$ sequence of colours labelling π omitting the ε labels (we remark that $\gamma(\pi) \in \Gamma^\omega$, since there are no cycles entirely labelled by ε). The play is *winning* for the existential player if $\gamma(\pi) \in \mathbb{W}$. A *partial play* is a finite path $\pi \in E^*$ in $\mathcal{G}$ starting in x_0. A *strategy for the existential player* is a function $\sigma\colon E^* \to E$ such that if a partial play π ends in a position $x \in V_E$, then $\sigma(\pi) \in Out(x)$. We say that a play π is *consistent with* the strategy σ if for every partial play π' that is a prefix of π ending in a position controlled by Exist, the next edge in π is $\sigma(\pi')$. The strategy σ is *winning* if every play consistent with σ is winning for the existential player. We say that the game $\mathcal{G}$ is *won by* the existential player

if that player has a winning strategy in $\mathcal{G}$. A strategy is *positional* if it can be represented by a function $\sigma\colon V_E \to E$ (that is, the choice of the next transition only depends on the current position, and not on the history of the path).

Winning conditions. We fix an alphabet Γ.

Muller. A *Muller condition* over the alphabet Γ is given by a family $\mathcal{F} \subseteq \mathcal{P}_+(\Gamma)$. A word $w \in \Gamma^\omega$ *satisfies the Muller condition* $\mathcal{F}$ if $Inf(w) \in \mathcal{F}$. The *language of the Muller condition* $\mathcal{L}_\mathcal{F} \subseteq \Gamma^\omega$ contains the ω-words that satisfy $\mathcal{F}$.

Rabin. A *Rabin condition* over the alphabet Γ is represented by a family of *Rabin pairs* $R = \{(G_1, R_1), \dots, (G_r, R_r)\}$, where $G_i, R_i \subseteq \Gamma$ and $G_i \cap R_i = \emptyset$. The Rabin pair j is said to be *green in* c if $c \in G_j$, to be *red in* c if $c \in R_j$, or to be *orange in* c if none of the previous occur. A word $w \in \Gamma^\omega$ *satisfies the Rabin condition* R if $Inf(w) \cap G_j \neq \emptyset$ and $Inf(w) \cap R_j = \emptyset$ for some index $j \in \{1, \dots, r\}$. Said differently, there is a Rabin pair j which is red in finitely many letters from w, and green for infinitely many letters of w. The *language of the Rabin condition* $\mathcal{L}_\mathcal{R} \subseteq \Gamma^\omega$ contains the ω-words that satisfy $\mathcal{R}$.

Parity. To define a *parity condition* we suppose that $\Gamma \subseteq \mathbb{N}$. A word $w \in \Gamma^\omega$ *satisfies the parity condition* if the maximum in $Inf(w)$ is even. The *language of the parity condition* contains the ω-words that satisfy it.

We say that a language $L \subseteq \Gamma^\omega$ is a *Muller language* if it is the language of some Muller condition $\mathcal{F}$. Equivalently, L is a Muller language if it can be described as a boolean combination of atomic propositions of the form "the letter "a" appears infinitely often" and their negations. Note that languages of Rabin conditions are languages of Muller conditions, and that languages of parity conditions are languages of Rabin conditions, but the converses do not hold. Given a Muller condition $\mathcal{F} \subseteq \mathcal{P}_+(\Gamma)$ and a subset $C \subseteq \Gamma$, we define the *restriction of* $\mathcal{F}$ *to* C as the Muller condition over C given by $\mathcal{F}|_C = \{A \subseteq C \,:\, A \in \mathcal{F}\}$.

Memory structures. A *memory structure for the game* $\mathcal{G}$ of moves E is a tuple $\mathcal{M} = (M, m_0, \mu, \sigma)$ where M is a *set of memory states*, $m_0 \in M$ is an *initial memory state*, $\mu : M \times E \to M$ is an *update function*, and $\sigma : M \times V_E \to E$ maps each position x owned by the existential player to a move from x. The *size* of $\mathcal{M}$ is the cardinal of M. We extend the function μ to paths by induction: $\mu(m, \varepsilon) = m$, and $\mu(m, \pi e) = \mu(\mu(m, \pi), e)$ for a path $\pi \cdot e \in E^*$. The memory structure $\mathcal{M}$ *induces a strategy* $\sigma_\mathcal{M}\colon E^* \to V_E$ given by $\sigma_\mathcal{M}(\pi) = \sigma(\mu(m_0, \pi), Last(\pi))$, where $Last(\pi)$ is the last position of the partial play π.

We say that $\mathcal{M}$ is a *chromatic memory structure* if there is a function $\mu_c\colon M \times (\Gamma \cup \{\varepsilon\}) \to M$ such that $\mu_c(m, \varepsilon) = m$ for every $m \in M$ and $\mu(m, e) = \mu_c(m, \gamma(e))$ for all edges $e \in E$.

The *memory requirements* of a winning condition $\mathbb{W}$ are defined as the least integer n such that if $\mathcal{G}$ is an $\mathbb{W}$-game won by the existential player, then she has a winning strategy given by a memory of size at most n. We denote this quantity by $\mathrm{mem}(\mathbb{W})$.

We say that a winning condition $\mathbb{W}$ is *exist-positional* (also called *half-positional*) if $\mathrm{mem}(\mathbb{W}) = 1$. Equivalently, $\mathbb{W}$ is exist-positional if Exist has a winning positional strategy whenever Exist has a winning strategy at all.

2.2 Automata and good-for-gameness

Automata. A *non-deterministic automaton* $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \mathbb{W})$ (or simply an *automaton*) consists of a finite set of *states* Q, an input alphabet Σ, a non-empty set of *initial states* $Q_0 \subseteq Q$, a *transition relation* $\Delta \subseteq Q \times \Sigma \times \Gamma \times Q$ and an *acceptance condition* $\mathbb{W} \subseteq \Gamma^\omega$. We will

write $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$ for the function $\delta(q,a) = \{q' \in Q \,:\, (q,a,c,q') \in \Delta \text{ for some } c \in \Gamma\}$. The *size* of the automaton, $|\mathcal{A}|$, is the number of its states. A *run of the automaton* over a word $a_0a_1a_2\cdots \in \Sigma^\omega$ is a sequence of transitions of the form:

$$\rho = (q_0,a_0,c_0,q_1)(q_1,a_1,c_1,q_2)\cdots \in \Delta^\omega, \text{ such that } q_0 \in Q_0 \text{ is an initial state.}$$

A run is *accepting* if $c_0c_1c_2\cdots \in \mathbb{W}$. If an accepting run over a word $w \in \Sigma^\omega$ exists, the automaton *accepts* w. The set of accepted words is the *language accepted by the automaton*, written $L(\mathcal{A})$. The automaton is *deterministic* if Q_0 is a singleton and Δ is such that for all states q and letter $a \in \Sigma$, there exists exactly one transition of the form (q,a,c,q'). In this case, for all words $w \in \Sigma^\omega$ there exists one and exactly one run of the automaton over w.

An automaton using an acceptance condition $\mathbb{W}$ (resp. an acceptance condition of type $X \in \{\text{Muller, Rabin, parity}\}$) is called an $\mathbb{W}$-*automaton* (resp. X-automaton).

Good-for-gameness. The automaton $\mathcal{A}$ is *good-for-games* (GFG) if there is a resolver for it, consisting of a choice of an initial state $r_0 \in Q_0$ and a function $r : \Sigma^* \times \Sigma \to \Delta$ such that for all words $w \in L(\mathcal{A})$, the run $t_0t_1... \in \Delta^\omega$, called the *run induced by* r and defined by $t_i = r(w[0..i-1], w[i])$, starts in r_0 and is an accepting run over w. In other words, r should be able to construct an accepting run in $\mathcal{A}$ letter-by-letter with only the knowledge of the word so far, for all words in $L(\mathcal{A})$.

2.3 The Zielonka tree of a Muller condition

A *tree* $T = (N, \sqsubseteq)$ is a nonempty finite set of *nodes* N equipped with an order relation $\sqsubseteq$ called the *ancestor relation* (x is an ancestor of y if $x \sqsubseteq y$), such that (1) there is a minimal node for $\sqsubseteq$, called *the root*, and (2) the ancestors of an element are totally ordered by $\sqsubseteq$. The converse relation is the *descendant* relation. Maximal nodes are called *leaves*, and the set of leaves of T is denoted by $Leaves(T)$. Given a node n of a tree T, the *subtree of* T *rooted at* n is the tree T restricted to the nodes that have n as ancestor. A node n' is a *child* of n if it is a minimal strict descendant of it. The set of children of n is written $Children_T(n)$. The *height* of a tree T is the maximal length of a chain for the ancestor relation. An A-*labelled tree* is a tree T together with a labelling function $\nu\colon N \to A$.

▶ **Definition 1** ([43]). *Let $\mathcal{F} \subseteq \mathcal{P}_+(\Gamma)$ be a Muller condition. A* Zielonka tree *for $\mathcal{F}$, denoted $\mathcal{Z}_\mathcal{F} = (N, \sqsubseteq, \nu : N \to \mathcal{P}_+(\Gamma))$ is a $\mathcal{P}_+(\Gamma)$-labelled tree with nodes partitioned into* round nodes *and* square nodes*, $N = N_\bigcirc \uplus N_\square$ such that:*

- *The root is labelled Γ.*
- *If a node is labelled $X \subseteq \Gamma$, with $X \in \mathcal{F}$, then it is a round node, and its children are labelled exactly with the maximal subsets $Y \subseteq X$ such that $Y \notin \mathcal{F}$.*
- *If a node is labelled $X \subseteq \Gamma$, with $X \notin \mathcal{F}$, then it is a square node, and its children are labelled exactly with the maximal subsets $Y \subseteq X$ such that $Y \in \mathcal{F}$.*

We remark that if n is a node of $\mathcal{Z}_\mathcal{F}$, then the subtree of $\mathcal{Z}_\mathcal{F}$ rooted at n is a Zielonka tree for $\mathcal{F}|_{\nu(n)}$, (the restriction of $\mathcal{F}$ to the label of n).

We equip trees with an order in order to navigate in them. An *ordered Zielonka tree* is a Zielonka tree for which the set of children of each node is equipped with a total order("from left to right"). The function $Next_n$ maps each child of n to its successor for this order, in a cyclic way. For each node $n \in N$ and each leaf l below n we define the set $Jump_n(l)$ containing a leaf l' if there are two children n_1, n_2 of n such that $n_1 \sqsubseteq l$, $n_2 \sqsubseteq l'$ and $n_2 = Next_n(n_1)$ (we remark that $n_1 = n_2$ if n has only one child). For $n = l$ we define $Jump_n(l) = \{l\}$. That

is, $l' \in Jump_n(l)$ if we can reach l' by the following procedure: we start at l, we go up the tree until finding the node n, we change to the next branch below n (in a cyclic way) and we re-descend to l'. From now on, we will suppose that all Zielonka trees are ordered, without explicitly mentionning it.

▶ **Example 2.** We will use the following Muller condition as a running example throughout the paper. Let $\Gamma = \{a, b, c\}$ and let $\mathcal{F}$ be the Muller condition defined by:

$$\mathcal{F} = \{\{a, b\}, \{a, c\}, \{b\}\}.$$

In Figure 1 we show the Zielonka tree for $\mathcal{F}$. We use Greek letters to name the nodes of the tree, N. We have that $Jump_\alpha(\delta) = \{\varepsilon, \zeta\}$ and $Jump_\gamma(\zeta) = \{\varepsilon\}$. The numbering of the branches will be used in Section 3.2.

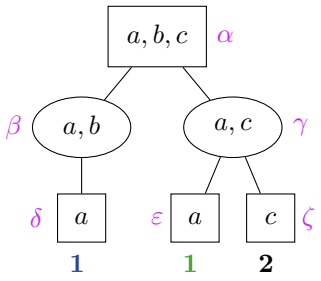


Figure 1 Zielonka tree $\mathcal{Z}_\mathcal{F}$ for $\mathcal{F} = \{\{a, b\}, \{a, c\}, \{b\}\}$.

▶ **Definition 3** ([15]). *Let T be a tree with nodes partitioned into round and square nodes, its* memory for the existential player *(*memory *for short), denoted* mem-tree(T)*, is defined inductively as:*

- 1 *if T has exactly one node.*
- *The sum of the memories of the subtrees of T rooted at the children of the root, if the root is round.*
- *The maximum of the memories of the subtrees of T rooted at the children of the root, if the root is square.*

For instance, for the Zielonka tree from Example 2, mem-tree$(\mathcal{Z}_\mathcal{F}) = 2$.

The key result justifying the introduction of this notion is that it characterises precisely the quantity of memory required for winning an $\mathcal{F}$-game, as shown by the next proposition.

▶ **Proposition 4** ([15]). *For all Muller conditions $\mathcal{F}$,* mem$(\mathcal{F})$ = mem-tree$(\mathcal{Z}_\mathcal{F})$.

3 GFG Rabin automata correspond to memory structures for Muller games

In this section, we prove the following result:

▶ **Theorem 5.** *Let L be a Muller language. The memory requirements for L coincide with the size of a minimal GFG Rabin automaton recognising L.*

In Section 3.1 we show the first direction: the size of a GFG Rabin automaton for a Muller language L is always an upper bound on the memory required by the existential player on L-games. In Section 3.2, we show how to construct a GFG Rabin automaton from

the Zielonka tree of a Muller condition of size mem-tree($\mathcal{Z}_L$) = mem(L), completing the equivalence. Moreover, we can build this minimal GFG Rabin automaton in polynomial time given the Zielonka tree of the Muller condition.

3.1 A GFG Rabin automaton induces a memory structure for any game

In this section, we establish Corollary 10 which states that the size of a GFG Rabin automaton $\mathcal{A}$ accepting a Muller language L is an upper bound on the memory required for winning all L-games. Concretely, given an L-game won by the existential player, we are able to construct a memory structure inducing a winning strategy based on $\mathcal{A}$. The argument is standard: we construct the product game of $\mathcal{A}$ and the L-game, which is a Rabin game in which the existential player enjoys a positional winning strategy; then we use the $\mathcal{A}$-component of this product as a memory structure for a strategy in the original L-game.

▶ **Lemma 6.** *Let $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \mathbb{W})$ be a GFG $\mathbb{W}$-automaton recognising a language $L \subseteq \Sigma^\omega$, with $\mathbb{W}$ exist-positional. Then if $\mathcal{G}$ is an L-game won by Exist, she can win it using a strategy given by a memory structure $\mathcal{M} = (Q, r_0, \mu, \sigma)$.*

Proof. Let $\mathcal{G} = (V = V_E \uplus V_A, E, x_0, L)$ and $\mathcal{A} = (Q, \Sigma, r_0, \Delta, \Gamma, \mathbb{W})$, where $r_0 \in Q_0$ is the initial state chosen by some resolver. We consider the product $\mathbb{W}$-game $\mathcal{G}'$ in which:

- Positions are elements in $V' = (V \times Q) \cup (V \times \Gamma \times \mathcal{P}(Q))$. The initial position is (x_0, r_0).
- Exist's positions are $V'_E = (V_E \times Q) \cup (V \times \Gamma \times \mathcal{P}(Q))$.
- There is an ε-coloured edge from (x, q) to $(x', c, \delta(q, c))$ if $(x, c, x') \in E$, $c \neq \varepsilon$, and to (x', q) if $(x, \varepsilon, x') \in E$.
- There is a c-labelled edge from (x, c, S) to (x, q') for all $q' \in S \subseteq Q$.
- The winning condition is $\mathbb{W}$.

In short, in this game the players still negotiate a play in G, but in addition, the existential player must simultaneously build an accepting run on the labelling of this play in $\mathcal{A}$.

If $\mathcal{A}$ is GFG then whenever the existential player wins in $\mathcal{G}$, she also wins in $\mathcal{G}'$ [20, Theorem 3] by playing a winning strategy on the $\mathcal{G}$ component of $\mathcal{G}'$ and using the resolver for $\mathcal{A}$ to choose the successor state in the $\mathcal{A}$ component.

This strategy is not necessarily positional. However, since $\mathbb{W}$ is exist-positional, the existential player also has a positional strategy $s : V'_E \to E'$. We can now build a memory structure $\mathcal{M}_s = (Q, r_0, \mu, \sigma)$ that projects the strategy s onto $\mathcal{G}$ (and updates itself with the projection of s onto $\mathcal{A}$):

- $\mu(q, e) = q$ if e is an ε-coloured move of $\mathcal{G}$ and
- $\mu(q, (x, c, x')) = q'$ where $q' = s(x', c, \delta(q, c))$ otherwise;
- $\sigma(q, x) = s(x, q)$.

Since s is a winning strategy in $\mathcal{G}'$, its projection onto $\mathcal{G}$ is a strategy that only agrees with plays with labels in $L(\mathcal{A})$, that is, winning plays. ◀

▶ Remark 7. Note that there is a slight subtlety here: the resolver, which induces a winning strategy in the product game, does not need to be positional. In fact, it might require exponential memory [27, Theorem 1]. Yet, for each L-game, a memory based on $\mathcal{A}$ suffices.

▶ **Lemma 8** ([22, 43])**.** *Rabin conditions are exist-positional.*

As a direct consequence we obtain the following Proposition.

▶ **Proposition 9.** *Let L be a Muller language accepted by a GFG Rabin automaton $\mathcal{A}$. Then, in every L-game $\mathcal{G}$ won by the existential player, she can win using a strategy given by a memory structure of size $|\mathcal{A}|$.*

▶ **Corollary 10.** *If $\mathcal{A}$ is a GFG Rabin automaton accepting a Muller language L, then* $\mathrm{mem}(L) \leqslant |\mathcal{A}|$.

3.2 An optimal construction of a GFG Rabin automaton

So far, we have seen that given a GFG Rabin automaton, it can serve as a memory structure for the games of which it accepts the winning condition. In this section we do the converse: we build a minimal GFG Rabin automaton for a Muller language L of the same size as the minimal memory required to win in L-games.

3.2.1 The construction

▶ **Proposition 11.** *Let $\mathcal{F} \subseteq \mathcal{P}_+(\Gamma)$ be a Muller condition. There exists a GFG Rabin automaton recognising $\mathcal{L}_\mathcal{F}$ of size* $\mathrm{mem}(\mathcal{F})$.

To prove Propositon 11, we build a GFG Rabin automaton $\mathcal{R}_\mathcal{F} = (Q, \Gamma, q_0, \Delta, \Gamma', R)$ for $\mathcal{L}_\mathcal{F}$ based on the Zielonka tree $\mathcal{Z}_\mathcal{F}$, as illustrated in Figure 2. We use a mapping from the leaves of $\mathcal{Z}_\mathcal{F}$ to the states of the automaton that guarantees that two leaves of which the last common ancestor is a round node cannot map to the same state. The number of states required to satisfy this condition ($\star$ below) coincides with mem-tree($\mathcal{Z}_\mathcal{F}$). Then, for each leaf of $\mathcal{Z}_\mathcal{F}$ and letter $c \in \Gamma$, we identify its last ancestor n in $\mathcal{Z}_\mathcal{F}$ containing c, and, using the $Jump_n$ function (defined in Section 2.3), pick a leaf below the next child of n. We add a c-transition with label n between the states mapped to from these leaves. This way, we can identify a run in the automaton with a promenade through the nodes of the Zielonka tree. If during this promenade a unique minimal node (for $\sqsubseteq$) is visited infinitely often, it is not difficult to see that the sequence of input colours belongs to $\mathcal{F}$ if and only if the label of this minimal node is an accepting set (it is a round node). We devise a Rabin condition over the set of nodes of the Zielonka tree accepting exactly these sequences of nodes.

We now describe the construction of the automaton $\mathcal{R}_\mathcal{F} = (Q, \Gamma, q_0, \Delta, N, R)$ formally, starting from the Zielonka tree $\mathcal{Z}_\mathcal{F} = (N, \sqsubseteq, \nu : N \to \mathcal{P}_+(\Gamma))$, and then proceed to prove its correctness.

States. First, we set $Q = \{1, 2, \ldots, \text{mem-tree}(\mathcal{Z}_\mathcal{F})\}$ and we label the leaves of $\mathcal{Z}_\mathcal{F}$ by a mapping $\eta : \mathit{Leaves}(\mathcal{Z}_\mathcal{F}) \to \{1, 2, \ldots, \text{mem-tree}(\mathcal{Z}_\mathcal{F})\}$ verifying the property:

> If $n \in \mathcal{Z}_\mathcal{F}$ is a round node with children $n_1 \neq n_2$, for any pair of leaves l_1 and l_2 below n_1 and n_2, respectively, $\eta(l_1) \neq \eta(l_2)$. ($\star$)

▶ **Lemma 12.** *For every Zielonka tree $\mathcal{Z}_\mathcal{F}$ there is a mapping verifying Property $\star$ of the form $\eta : \mathit{Leaves}(\mathcal{Z}_\mathcal{F}) \to \{1, 2, \ldots, \text{mem-tree}(\mathcal{Z}_\mathcal{F})\}$.*

Proof. We prove it by induction in the height of $\mathcal{Z}_\mathcal{F}$. Let $n_1, \ldots, n_k$ be the children of the root of $\mathcal{Z}_\mathcal{F}$. We write $\mathcal{F}_i$ to denote the Muller condition restricted to $\nu(n_i)$ and $\eta_i : \mathit{Leaves}(\mathcal{Z}_{\mathcal{F}_i}) \to \{1, 2, \ldots, \text{mem-tree}(\mathcal{Z}_{\mathcal{F}_i})\}$ be a labelling verifying Property $\star$, for $1 \leq i \leq k$. We distinguish two cases according to the shape of the root.

If the root of $\mathcal{Z}_\mathcal{F}$ is a square node ($\Gamma \notin \mathcal{F}$), then the mapping $\eta(l) = \eta_i(l)$ if the leaf l belongs to the subtree $\mathcal{Z}_{\mathcal{F}_i}$ verifies Property $\star$.

If the root of $\mathcal{Z}_\mathcal{F}$ is a round node ($\Gamma \in \mathcal{F}$), then mem-tree$(\mathcal{Z}_\mathcal{F}) = \sum_{i=1}^{k}$ mem-tree$(\mathcal{Z}_{\mathcal{F}_i})$, and we can partition $\{1, 2, \ldots,$ mem-tree$(\mathcal{Z}_\mathcal{F})\}$ into disjoint sets $C_1, \ldots, C_k$ of size $|C_i| =$ mem-tree$(\mathcal{Z}_{\mathcal{F}_i})$. We write σ_i for a bijection from $\{1, \ldots,$ mem-tree$(\mathcal{Z}_{\mathcal{F}_i})\}$ to C_i. Then, the mapping $\eta(l) = \sigma_i(\eta_i(l))$, if the leaf l belongs to the subtree $\mathcal{Z}_{\mathcal{F}_i}$ verifies Property $\star$. ◀

We suppose that the image of the leftmost leaf under η is 1 and choose the initial state q_0 to be 1. In Example 2, the labelling $\eta(\delta) = \eta(\varepsilon) = 1$, $\eta(\zeta) = 2$ verifies Property $\star$.

Transitions. For each leaf $l \in \mathit{Leaves}(\mathcal{Z}_\mathcal{F})$ and each letter $c \in \Gamma$, we define a c-transition from $\eta(l)$, with an output label from $\Gamma' = N$, as follows: let n be the maximal ancestor of l that contains the letter c in its label and let l' be the leftmost leaf in $\mathit{Jump}_n(l)$[1]. Then, $(\eta(l), c, n, \eta(l')) \in \Delta$. That is, if we are in a state $\eta(l)$, when we read the letter $c \in \Gamma$ we can choose to go up in the Zielonka tree from l until visiting a node n with c in its label. We produce the letter n as output, then we change to the next child of n (in a cyclic way) and we descend to the leftmost leaf below it. The destination is the η-label of this leaf[2].

Following the above definition, we obtain a mapping from transitions in the automaton to $\mathit{Leaves}(\mathcal{Z}_\mathcal{F}) \times \Gamma \times N \times \mathit{Leaves}(\mathcal{Z}_\mathcal{F})$. We say that l and l' are the leaves *corresponding to* the transition (q, a, n, q') if this transition is sent to (l, a, n, l') by this mapping. The node n produced as output is the last common ancestor of l and l'. The automaton obtained in this way might present multiple transitions labelled by the same input letter between two states. We will show in Proposition 18 that duplicated transitions can be removed.

Acceptance condition. We define a Rabin condition over the alphabet $\Gamma' = N$, that is the set of nodes of the Zielonka tree. We define a Rabin pair for each round node of $\mathcal{Z}_\mathcal{F}$ (that is, nodes whose label is an accepting set of letters for $\mathcal{F}$): $R = \{(G_n, R_n)\}_{n \in N_\bigcirc}$. Let n be a round node and n' be a general node of $\mathcal{Z}_\mathcal{F}$:

$$\begin{cases} n' \in G_n & \text{if } n' = n, \\ n' \in R_n & \text{if } n' \neq n \text{ and } n \text{ is not an ancestor of } n'. \end{cases}$$

That is, for the letter n', the Rabin pairs corresponding to round ancestors of n' are not affected by it (they are "orange in n'"). If this node n' is round, then it belongs to $G_{n'}$ (this pair is "green in n'"). For any other node $n \in N_\bigcirc$, we have $n' \in R_n$ (the pair is "red in n'").

▶ Remark 13. The construction presented depends on the order of the nodes of the Zielonka tree. However, the size of the resulting automaton is independent of this order.

▶ **Example 14.** Let $\mathcal{F} = \{\{a, b\}, \{a, c\}, \{b\}\}$ be the Muller condition from Example 2. The labelling of the leaves of the Zielonka tree given by $\eta(\delta) = \eta(\varepsilon) = 1$, $\eta(\zeta) = 2$ verifies Property $\star$. Figure 2 shows the GFG Rabin automaton obtained by following this procedure.

The Rabin condition of this automaton is given by two Rabin pairs (corresponding to the round nodes of the Zielonka tree in Figure 1):

$$G_\beta = \{\beta\}, \quad R_\beta = \{\alpha, \gamma, \varepsilon, \zeta\},$$
$$G_\gamma = \{\gamma\}, \quad R_\gamma = \{\alpha, \beta, \delta\}.$$

[1] We could add all transitions $\{(\eta(l), n, \eta(l')) : l' \in \mathit{Jump}_n(l)\}$ to Δ. However, a resolver for the GFG automaton just needs to make use of one of these transitions, so in order to simplify the automaton we make an arbitrary choice (the leftmost leaf).

[2] We remark that l' is the target of the transition of the Zielonka tree parity automaton from l reading letter "c" [11]. See also Section 3.2.3.

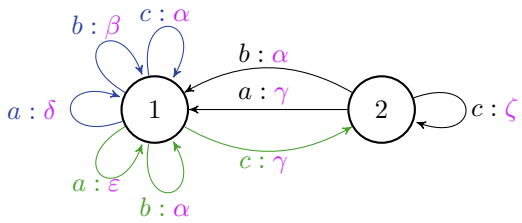


Figure 2 The GFG Rabin automaton obtained from the Zielonka tree $\mathcal{Z}_{\mathcal{F}}$.

3.2.2 Proof of correctness

▶ **Lemma 15.** *Let $w = n_0 n_1 n_2 \cdots \in N^\omega$ be an infinite sequence of nodes of the Zielonka tree. The word w satisfies the Rabin condition defined above if and only if there is a unique minimal node for the ancestor relation in $Inf(w)$ and this minimal node is round (recall that the root is the minimal element in $\mathcal{Z}_{\mathcal{F}}$).*

Proof. Suppose that there is a unique minimal node in $Inf(w)$, called n, and that n is round. We claim that w is accepted by the Rabin pair (G_n, R_n). It is clear that $Inf(w) \cap G_n \neq \emptyset$, because $n \in G_n$. It suffices to show that $Inf(w) \cap R_n = \emptyset$: By minimality, any other node $n' \in Inf(w)$ is a descendant of n (equivalently, n is an ancestor of n'), so $n' \notin R_n$.

Conversely, suppose that $w \in N^\omega$ satisfies the Rabin condition. Then, there is some round node $n \in N_\bigcirc$ such that $Inf(w) \cap G_n \neq \emptyset$ and $Inf(w) \cap R_n = \emptyset$. Since $G_n = \{n\}$, we deduce that $n \in Inf(w)$. Moreover, as $Inf(w) \cap R_n = \emptyset$, all nodes in $Inf(w)$ are descendants of n. We conclude that n is the unique minimal node in $Inf(w)$, and it is round. ◀

▶ **Lemma 16.** *The automaton $\mathcal{R}_{\mathcal{F}}$ recognises the language $\mathcal{L}_{\mathcal{F}}$ and is good-for-games.*

Proof. $\mathcal{L}(\mathcal{R}_{\mathcal{F}}) \subseteq \mathcal{L}_{\mathcal{F}}$: Let $u \in \mathcal{L}(\mathcal{R}_{\mathcal{F}})$ and let $w \in N^\omega$ be the sequence of nodes produced as output of an accepting run over u in $\mathcal{R}_{\mathcal{F}}$. By Lemma 15, there is a unique minimal node n for $\sqsubseteq$ appearing infinitely often in w and moreover n is round. Let $n_1, \ldots, n_k$ be an enumeration of the children of n (from left to right), with labels $\nu(n_i) \subseteq \Gamma$ (we remark that $\nu(n_i) \notin \mathcal{F}$, for $1 \leq i \leq k$). We will prove that $Inf(u) \subseteq \nu(n)$ and $Inf(u) \not\subseteq \nu(n_i)$ for $1 \leq i \leq k$. By definition of the Zielonka tree, as n is round, this implies that $Inf(u) \in \mathcal{F}$.

Since eventually all nodes produced as output are descendants of n (by minimality), $Inf(u)$ must be contained in $\nu(n)$ (by definition of the transitions of $\mathcal{R}_{\mathcal{F}}$).

We suppose, towards a contradiction, that $Inf(u) \subseteq \nu(n_j)$ for some $1 \leq j \leq k$. Let $Q_i = \{\eta(l) \,:\, l \text{ is a leaf below } n_i\}$ be the set of states corresponding to leaves under n_i, for $1 \leq i \leq k$. We can suppose that the leaves corresponding to transitions of an accepting run over u are all below n, and therefore, transitions of such a run only visit states in $\bigcup_{i=1}^{k} Q_i$. Indeed, eventually this is going to be the case, because if some of the leaves l, l' corresponding to a transition (q, a, n', q') are not below n, then n' would not be a descendant of n (since n' is the least common ancestor of l and l'). Also, by Property $\star$, we have $Q_i \cap Q_j = \emptyset$, for all $i \neq j$. By definition of the transitions of $\mathcal{R}_{\mathcal{F}}$, if $c \in \Gamma$ is a colour in $\nu(n)$ but not in $\nu(n_i)$, all transitions from some state in Q_i reading the colour c go to Q_{i+1}, for $1 \leq i \leq k-1$ (and to Q_1 if $i = k$). Also, if $c \in \nu(n_i)$, transitions from states in Q_i reading c stay in Q_i. We deduce that a run over u will eventually only visit states in Q_j, for some j such that $Inf(u) \subseteq \nu(n_j)$. However, the only transitions from Q_j that would produce n as output are those corresponding to a colour $c \notin \nu(n_j)$, so the node n is not produced infinitely often, a contradiction.

$\mathcal{L}_\mathcal{F} \subseteq \mathcal{L}(\mathcal{R}_\mathcal{F})$ and good-for-gameness: We will describe a strategy for a resolver in $\mathcal{R}_\mathcal{F}$ using as memory the set of leaves of the Zielonka tree[3]. It will verify at every step that if the memory is on the leaf l, then $\mathcal{R}_\mathcal{F}$ is on the state $\eta(l)$. The initial state of the memory is the leftmost leaf of $\mathcal{Z}_\mathcal{F}$. If we are on the memory state l and the letter $c \in \Gamma$ is read, we take the transition $(\eta(l), c, n, \eta(l')) \in \Delta$, where n is the maximal ancestor of l such that $c \in \nu(n)$ and l' is the leftmost leaf in $Jump_n(l)$; the memory state is updated to l'. Let us suppose that a word $u \in \mathcal{L}_\mathcal{F}$ is given as input to the automaton. We will see that the run produced by this strategy is accepting. We can suppose that the only colours appearing in u are those of $Inf(u)$. Let n be the leftmost $\sqsubseteq$-maximal node such that $Inf(u) \subseteq \nu(n)$. Since $Inf(u) \in \mathcal{F}$, n is a round node. We will prove that the run produced by the resolver above only produces nodes that are descendants of n (including n) infinitely often and that it produces n infinitely often and is therefore accepting. Let $n_1, \dots, n_k$ be the children of n from left to right, and let $L_1, \dots, L_k$ be the (disjoint) sets of leaves below them, respectively. By the definition of the transitions and the strategy, the memory will eventually only consider leaves in $\bigcup_{i=1}^{k} L_i$, and will produced as output nodes that are descendants of n (including n itself). Also, each time that the memory is in some state in L_i and a colour not in $\nu(n_i)$ is given, a transition leading to some state in $\eta(L_{i+1})$ ($\eta(L_1)$ if $i = k$) producing the node n as output is taken. Since $Inf(u)$ is not contained in $\nu(n_i)$ for $1 \leq i \leq k$ (by the maximality assumption), this occurs infinitely often. ◀

▶ Remark 17. We have shown that given as input the Zielonka tree of a Muller condition $\mathcal{F}$ we can build in polynomial time a minimal GFG Rabin automaton for $\mathcal{L}_\mathcal{F}$. On the other hand, with the same input, it is NP-complete to decide whether there is a deterministic Rabin automaton of size k recognising $\mathcal{L}_\mathcal{F}$ [10, Theorem 31]. Therefore, unless P = NP, there are Muller languages for which minimal deterministic Rabin automata are strictly greater than minimal GFG Rabin automata. We will explicitly show some of these languages in Section 4.

3.2.3 Relation with the Zielonka-tree parity automaton

The Zielonka tree has been previously used to provide a minimal deterministic parity automaton for a Muller condition [11, 33]. The automata states are the leaves of the Zielonka tree, and the transition from a leaf l reading colour c goes to the leftmost leave in $Jump_n(l)$, where n is the last ancestor of l containing colour c. For example, Figure 3 shows a parity automaton recognising the Muller condition $\mathcal{F} = \{\{a, b\}, \{a, c\}, \{b\}\}$ from Example 2.

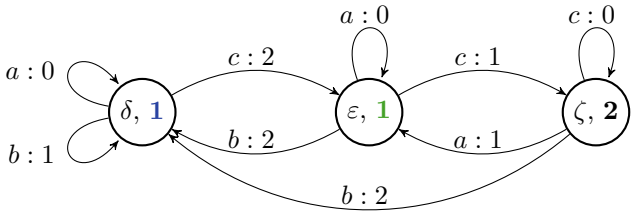


Figure 3 Parity automaton $\mathcal{P}_\mathcal{F}$ obtained from the Zielonka tree from Figure 1.

[3] This strategy is given by the (deterministic) Zielonka tree parity automaton $\mathcal{P}_\mathcal{F}$. It suffices to note that there is a morphism from $\mathcal{P}_\mathcal{F}$ to $\mathcal{R}_\mathcal{F}$ preserving all edges and the acceptance of loops.

This minimal parity automaton $\mathcal{P}_\mathcal{F}$ is closely related to the GFG Rabin automaton $\mathcal{R}_\mathcal{F}$ presented in Section 3.2. More precisely, the automaton $\mathcal{R}_\mathcal{F}$ can be regarded as a quotient of $\mathcal{P}_\mathcal{F}$ given by the numbering $\eta\colon \mathit{Leaves}(\mathcal{Z}_\mathcal{F}) \to \{1,\dots,\text{mem-tree}(\mathcal{Z}_\mathcal{F})\}$. That is, to obtain $\mathcal{R}_\mathcal{F}$ we merge the states $\eta^{-1}(i)$ for $1 \le i \le \text{mem-tree}(\mathcal{Z}_\mathcal{F})$ and we keep all transitions. Moreover, the strategy for a resolver for $\mathcal{R}_\mathcal{F}$ as presented in the proof of Lemma 16 is exactly given by the deterministic automaton $\mathcal{P}_\mathcal{F}$. However, we note that in general a parity condition is not sufficient in $\mathcal{R}_\mathcal{F}$ to accept $\mathcal{L}_\mathcal{F}$ and we need to replace it by a Rabin one.

We observe that the GFG Rabin automaton from Figure 2 is obtained as a quotient of the deterministic parity automaton in Figure 3.

3.2.4 Simplifications and optimisations

Given an automaton $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \mathbb{W})$ we say that it has *duplicated edges* if there are some pair of states $q, q' \in Q$ and two different transitions between them labelled with the same input letter: $(q, a, \alpha, q'), (q, a, \beta, q') \in \Delta$.

As remarked previously, the construction we have presented provides an automaton potentially having duplicated edges, which can be seen as an undesirable property (even if some automata models such as the HOA format [2] allow them). We show next that we can always derive an equivalent automaton without duplicated edges. Intuitively, in the Rabin case, if we want to merge two transitions having as output letters α and β, we add a fresh letter $(\alpha\beta)$ to label the new transition. For each Rabin pair, this new letter will simulate the best of either α or β depending upon the situation.

▶ **Proposition 18** (Simplification of automata). *Let $\mathcal{A}$ be a Muller (resp. Rabin) automaton presenting duplicated edges. There exists an equivalent Muller (resp. Rabin) automaton $\mathcal{A}'$ on the same set of states without duplicated edges. Moreover, if $\mathcal{A}$ is GFG, $\mathcal{A}'$ can be chosen GFG. In the Rabin case, the number of Rabin pairs is also preserved.*

Proof. For the Rabin case, let $\mathcal{A}'$ be an automaton that is otherwise as $\mathcal{A}$ except that instead of the transitions Δ of $\mathcal{A}$ it only has one a-transition $q \xrightarrow{a:x} q' \in \Delta'$ (with a fresh colour x per transition) per state-pair q, q' and letter $a \in \Sigma$. That is, $\Delta' = \{(q, a, x_j, q') : (q, a, y, q') \in \Delta \text{ for some } y\}$. The new Rabin condition $\{(G'_1, R'_1), \dots, (G'_r, R'_r)\}$ is defined as follows. For each transition $q \xrightarrow{a:x} q'$:

- $x \in G'_i$ if $q \xrightarrow{a:y} q' \in \Delta$ for some $y \in G_i$ (there is a green transition for the i^{th} pair)
- $x \in R'_i$ if for all $q \xrightarrow{a:y} q' \in \Delta$, $y \in R_i$ (there is no green or orange transition for the i^{th} pair).

We claim that $L(\mathcal{A}') = L(\mathcal{A})$. Indeed, if $u \in L(\mathcal{A})$, as witnessed by some run ρ and a Rabin pair (G_i, R_i), then the corresponding run ρ' in $\mathcal{A}'$ over u is also accepting with Rabin pair (G'_i, R'_i): the transitions of $\mathit{Inf}(\rho) \cap G_i$ induce transitions of $\mathit{Inf}(\rho') \cap G'_i$ and the fact that $\mathit{Inf}(\rho) \cap R_i = \emptyset$ guarantees that $\mathit{Inf}(\rho') \cap R'_i = \emptyset$.

Conversely, if $u \in L(\mathcal{A}')$ as witnessed by a run ρ' and Rabin pair (G'_i, R'_i), then there is an accepting run ρ over u in $\mathcal{A}$: such a run can be obtained by choosing for each transition $q \xrightarrow{a:x} q'$ of ρ' where $x \in G'_i$ a transition $q \xrightarrow{a:y} q' \in \Delta$ such that $y \in G_i$, which exists by definition of $\mathcal{A}'$, for each transition $q \xrightarrow{a:x} q'$ where $x \notin G_i \cup R_i$ a transition $q \xrightarrow{q,y} q' \in \Delta$ such that $y \notin R_i$, which also exists by definition of $\mathcal{A}'$, and for other transitions $q \xrightarrow{a:x} q'$ (that is, those for which $x \in R'_i$) an arbitrary transition $q \xrightarrow{a:y} q' \in \Delta$. Since ρ' is accepting, we have $\mathit{Inf}(\rho') \cap G_i \neq \emptyset$ and $\mathit{Inf}(\rho) \cap R_i = \emptyset$, that is, ρ is also accepting.

For the Muller case, the argument is even simpler. As above, we consider $\mathcal{A}'$ that is otherwise like $\mathcal{A}$ except that instead of the transitions Δ of $\mathcal{A}$, it only has one a-transition $q \xrightarrow{a:x} q' \in \Delta'$ (with a fresh colour per transition) per state-pair q, q' and the accepting

condition is defined as follows. A set of transitions T is accepting if and only if for each $t = q \xrightarrow{a:x} q' \in T$ there is a non-empty set $S_t \subseteq \{q \xrightarrow{a:y} q' \in \Delta\}$ such that $\bigcup_{t \in T} S_t$ is accepting in $\mathcal{A}$. In other words, a set of transitions in $\mathcal{A}'$ is accepting if for each transition we can choose a non-empty subset of the original transitions in $\mathcal{A}$ that form an accepting run in $\mathcal{A}$.

We claim that $L(\mathcal{A}') = L(\mathcal{A})$. Indeed if $u \in L(\mathcal{A})$, as witnessed by some run ρ, the run ρ' that visits the same sequence of states in $\mathcal{A}'$ is accepting as witnessed by the transitions that occur infinitely often in ρ.

Conversely, assume $u \in L(\mathcal{A}')$, as witnessed by a run ρ' and a non-empty subset S_t for each transitions t that occurs infinitely often in ρ' such that $\bigcup_{t \in Inf(\rho)} S_t$ is accepting in $\mathcal{A}$. Then there is an accepting run ρ over u in $\mathcal{A}$ that visits the same sequence of states as ρ' and chooses instead of a transition $t \in Inf(\rho)$ each transition in S_t infinitely often, and otherwise takes an arbitrary transition. The set of transitions ρ visits infinitely often is exactly $\bigcup_{t \in Inf(\rho)} S_t$, and is therefore accepting.

Finally, observe that in both cases, if $\mathcal{A}$ if GFG, then the automaton $\mathcal{A}'$ without duplicate edges is also GFG since $\mathcal{A}'$ is obtained from $\mathcal{A}$ by merging transitions. Indeed, the resolver r of $\mathcal{A}$ induces a resolver r' for $\mathcal{A}'$ by outputting the unique transition with the same letter and state-pair as r. By the same argument as above, the run induced by r' is accepting if and only if the run induced by r is. ◀

▶ **Example 19.** The GFG Rabin automaton from Figure 2 has duplicated transitions. In Figure 4 we present an equivalent GFG Rabin automaton without duplicates. For this, we have merged the self-loops in state 1 labelled with a and b respectively. We have added the output letters $(\alpha\beta)$ and $(\delta\varepsilon)$. The new Rabin pairs are given by:

$$G'_\beta = \{\beta, (\alpha\beta)\}, \quad R'_\beta = \{\alpha, \gamma, \varepsilon, \zeta\},$$
$$G'_\gamma = \{\gamma\}, \quad R'_\gamma = \{\alpha, \beta, (\alpha\beta), \delta\}.$$

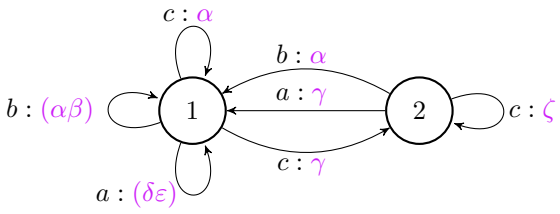


Figure 4 The simplified GFG Rabin automaton.

▶ Remark 20 (Optimisation on the number of Rabin pairs). An important parameter in the study of Rabin automata is the number of Rabin pairs used. The automaton presented in this Section uses a number of Rabin pairs that equals the number of round nodes in the Zielonka tree. This can be improved by using only the round nodes in the Zielonka directed acyclic graph (obtained from the tree by merging nodes with the same labels). However, even this latter option is not always optimal and we conjecture that minimising the number of Rabin pairs without increasing the size of the automaton is NP-complete.

4 GFG Rabin automata recognising Muller conditions can be exponentially more succinct than deterministic ones in number of states

On his PhD Thesis [23, 24], Kopczyński raised the question of whether the general and the chromatic memory requirements of winning conditions always coincide. By Theorem 5 and [10, Theorem 28], in the case of Muller conditions, this question is equivalent to the following:

> Is there a Muller language L such that minimal GFG Rabin automata recognising L are strictly smaller than deterministic Rabin automata for L?

In [10] this question is answered positively. It is shown that for every $n \in \mathbb{N}$ there is a Muller language L_n over an alphabet Γ_n such that a minimal GFG Rabin automaton for it has size 2, but a minimal deterministic Rabin automaton for it has size n. However, the size of the alphabet Γ_n in that example also has size n. A natural question is whether GFG Rabin automata recognising Muller conditions can be exponentially more succinct than deterministic ones, when also taking into account the alphabet size. This is indeed the case:

▶ **Theorem 21.** *There exists a constant $\alpha > 1$, a sequence of natural numbers $n_1 < n_2 < n_3 \dots$ and a sequence of Muller conditions $\mathcal{F}_{n_i}$ over $\Gamma_{n_i} = \{1, \dots, n_i\}$ such that*
- *a minimal GFG Rabin automaton for $\mathcal{L}_{\mathcal{F}_{n_i}}$ has size $\lfloor n_i/2 \rfloor$,*
- *a minimal deterministic Rabin automaton for $\mathcal{L}_{\mathcal{F}_{n_i}}$ has size at least α^{n_i}.*

A lower bound for such a constant is $\alpha = 1.116$.

We devote the rest of this Section to proving Theorem 21. In brief, the Muller conditions in question require half the colours to be seen infinitely often. The construction of the small GFG Rabin automaton follows from constructing the Zielonka tree of the condition. For the lower bound on the deterministic Rabin automaton, we reduce the problem to finding a lower bound on the chromatic number of a certain graph, which we finally show to be sufficiently large for a family of our Muller conditions.

Let $n \in \mathbb{N}$. We define the following Muller condition over $\Gamma_n = \{1, \dots, n\}$:

$$\mathcal{F}_n = \{C \subseteq \Gamma_n \,:\, |C| = \lfloor n/2 \rfloor\}.$$

The Zielonka tree of $\mathcal{F}_n$ is depicted in Figure 5 (for n even).

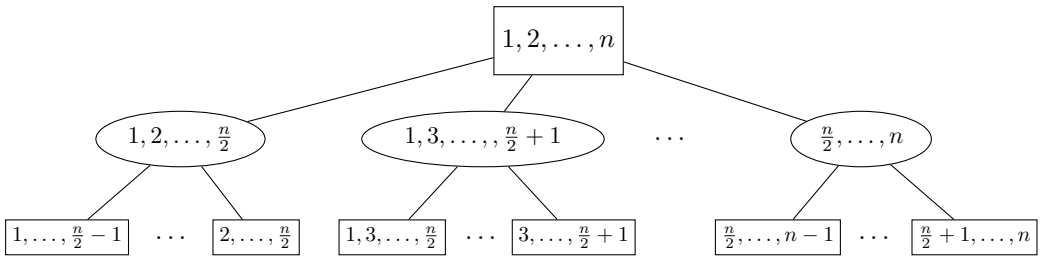


Figure 5 Zielonka tree $\mathcal{Z}_{\mathcal{F}_n}$ for $\mathcal{F}_n = \{C \subseteq \Gamma_n \,:\, |C| = \lfloor n/2 \rfloor\}$.

Each round node in $\mathcal{Z}_{\mathcal{F}_n}$ has exactly $\lfloor n/2 \rfloor$ children, and therefore mem-tree$(\mathcal{Z}_{\mathcal{F}_{n_i}}) = \lfloor n/2 \rfloor$. Thus, a minimal GFG Rabin automaton recognising $\mathcal{L}_{\mathcal{F}_n}$ has size $\lfloor n/2 \rfloor$ (by Proposition 4 and Theorem 5).

We now give a lower bound for deterministic Rabin automata recognising $\mathcal{L}_{\mathcal{F}_n}$. Our main tool will be Lemma 22, which uses the notion of *cycles*. A *cycle* of an automaton $\mathcal{A}$ is a set of transitions forming a closed path (not necessarily simple). The set of states of a cycle consists

of those states that are the source of some transition in it. If $\mathcal{A}$ is a Rabin automaton, we say that a cycle is accepting (resp. rejecting) if the colours $c_1, \dots, c_k$ appearing in its transitions form a word $(c_1 c_2 \dots c_k)^\omega$ that satisfies (resp. does not satisfy) the Rabin condition.

▶ **Lemma 22** ([11]). *Let $\mathcal{A}$ be a deterministic Rabin automaton. If ℓ_1 and ℓ_2 are two rejecting cycles in $\mathcal{A}$ with some state in common, then the union of ℓ_1 and ℓ_2 is also a rejecting cycle.*

For the following, let $\mathcal{A}$ be a deterministic Rabin automaton recognising $\mathcal{L}_{\mathcal{F}_n}$. For each subset of letters $C \subseteq \Gamma_n$, we define a *final C-Strongly Connected Component* (C-*FSCC* for short) as a set of states P of $\mathcal{A}$ such that:

- For every pair of states $p, q \in P$, there is a word $w \in C^*$ labelling a path from p to q.
- For every $p \in P$ and $w \in C^*$, the run over w starting in p remains in P.

It is easy to see that for every $C \subseteq \Gamma_n$ there exists some C-FSCC in $\mathcal{A}$.

▶ **Lemma 23.** *Let $C_1, C_2 \subseteq \Gamma_n$ such that $|C_i| < \lfloor n/2 \rfloor$, for $i = 1, 2$ and such that $|C_1 \cup C_2| = \lfloor n/2 \rfloor$. If P_1 and P_2 are two C_1 and C_2-FSCC, respectively, then $P_1 \cap P_2 = \emptyset$.*

Proof. For $i = 1, 2$, let ℓ_i be a cycle visiting all states of P_i and reading exactly the set of letters C_i. By definition of $\mathcal{L}_{\mathcal{F}_n}$, ℓ_i is a rejecting cycle. If P_1 and P_2 had some state in common, we could take the union of the cycles ℓ_1 and ℓ_2, producing an accepting cycle, which is impossible by Lemma 22. ◀

We associate the following (undirected) graph $\mathcal{G}_{\mathcal{F}_n} = (V_{\mathcal{F}_n}, E_{\mathcal{F}_n})$ to the Muller condition $\mathcal{F}_n$:

- $V_{\mathcal{F}_n} = \mathcal{P}(\Gamma_n)$.
- There is an edge between two subsets $C_1, C_2 \subseteq \Gamma_n$ if and only if $|C_i| < \lfloor n/2 \rfloor$, for $i = 1, 2$, and $|C_1 \cup C_2| = \lfloor n/2 \rfloor$.

That is, we connect two vertices if they correspond to rejecting sets but taking their union we obtain an accepting set.

We reduce finding lower bounds in the size of deterministic Rabin automata to giving lower bounds for the chromatic number of $\mathcal{G}_{\mathcal{F}_n}$. A *colouring* of an undirected graph $G = (V, E \subseteq V \times V)$ is a mapping $c : V \to \Lambda$ such that $c(v) = c(v') \Rightarrow (v, v') \notin E$ for every pair of nodes $v, v' \in V$. We say that such a colouring has size $|\Lambda|$. The *chromatic number* of G is the minimal number k such that G has a colouring of size k. We denote it $\chi(G)$.

▶ **Lemma 24.** *A lower bound for the size of a minimal deterministic Rabin automaton recognising $\mathcal{L}_{\mathcal{F}_n}$ is given by $\chi(\mathcal{G}_{\mathcal{F}_n})$.*

Proof. Let $\mathcal{A}$ be a deterministic Rabin automaton recognising $\mathcal{L}_{\mathcal{F}_n}$ with states Q. We define a colouring c of $\mathcal{G}_{\mathcal{F}_n}$ using Q as colours. For each $C \subseteq \Gamma_n$, we let P_C be a C-FSCC and we pick a state $q_C \in P_C$. We define $c(C) = q_C$. We prove that this is a correct colouring. Suppose that C_1 and C_2 are two vertices in $\mathcal{G}_{\mathcal{F}_n}$ connected by some edge, that is, $|C_i| < \lfloor n/2 \rfloor$ and $|C_1 \cup C_2| = \lfloor n/2 \rfloor$. If $q_{C_1} = q_{C_2}$, it means that $P_{C_1} \cap P_{C_2} \neq \emptyset$, contradicting Lemma 23. ◀

▶ Remark 25. The definition of $\mathcal{G}_{\mathcal{F}_n}$ is not specific to this Muller condition. It can be defined analogously for any other Muller condition and Lemma 24 holds by the same argument.

▶ **Proposition 26.** *There exists a constant $\alpha > 1$ and a sequence of natural numbers $n_1 < n_2 < n_3 \dots$ such that $\alpha^{n_i} \leq \chi(\mathcal{G}_{\mathcal{F}_{n_i}})$.*

In order to prove Proposition 26 we introduce some further graph-theoretic notions. Let $\mathcal{G} = (V, E)$ be an undirected graph. An *independent set* of $\mathcal{G}$ is a set $S \subseteq V$ such that $(v, v') \notin E$ for every pair of vertices $v, v' \in S$.

▶ **Lemma 27.** *Let $R \subseteq V$, and let $\mathcal{G}_R = (R, E|_{R\times R})$ be the subgraph of $\mathcal{G}$ induced by R. Then, $\chi(\mathcal{G}) \geq \chi(\mathcal{G}_R)$.*

▶ **Lemma 28.** *Let m be an upper bound on the size of the independent sets in $\mathcal{G}$. Then*

$$\chi(\mathcal{G}) \geq \frac{|V|}{m}.$$

Proof. Let $c\colon V \to \Lambda$ be a colouring of $\mathcal{G}$ with $|\Lambda| = \chi(\mathcal{G})$. Then, by definition of a colouring, for each $x \in \Lambda$, $c^{-1}(x)$ is an independent set in $\mathcal{G}$, so $|c^{-1}(x)| \leq m$. Also, $V = \bigcup_{x\in\Lambda} c^{-1}(x)$, so

$$|V| = \sum_{x\in\Lambda} |c^{-1}(x)| \leq \chi(\mathcal{G}) \cdot m. \qquad \blacktriangleleft$$

We will find a subgraph of $\mathcal{G}_{\mathcal{F}_n}$ for which we can provide an upper bound on the size of its independent sets. The upper bound is provided by the following theorem (adapted from [35, Theorem 15]).

▶ **Theorem 29** ([35], Theorem 15)**.** *Let $n > k > 2t$ such that $k - t$ is a prime number. Suppose that $\mathcal{B}$ is a family of subsets of size k of Γ_n such that $|A \cap B| \neq t$ for any pair of subsets $A, B \in \mathcal{B}$. Then,*

$$|\mathcal{B}| \leq \binom{n}{k-t-1}.$$

We conclude this section with the proof of Proposition 26.

Proof of Proposition 26. Let p be a prime number and let $n = 5p$. We will study the subgraph of $\mathcal{G}_{\mathcal{F}_n}$ formed by the subsets of size exactly $k = \lfloor 3n/10 \rfloor$. We denote this subgraph by $H_{n,k}$. Two subsets $A, B \subseteq \Gamma_n$ of size k verify that $|A \cup B| = \lfloor n/2 \rfloor$ if and only if $|A \cap B| = \lfloor n/10 \rfloor$. We set $t = \lfloor n/10 \rfloor$. We get $k - t = p$ so we can apply Theorem 29 and we obtain that any independent set in $H_{n,k}$ has size at most $\binom{n}{\frac{1}{5}n-1}$. By Lemma 28, $\chi(H_{n,k}) \geq \binom{n}{\lfloor \frac{3}{10}n \rfloor} / \binom{n}{\frac{1}{5}n-1}$. By Lemma 27 we know that this lower bound also holds for $\mathcal{G}_{\mathcal{F}_n}$. Using Stirling's approximation we obtain that

$$\chi(\mathcal{G}_{\mathcal{F}_n}) \geq \binom{n}{\lfloor \frac{3}{10}n \rfloor} \Big/ \binom{n}{\frac{1}{5}n-1} = \Omega\left(\left(\frac{(1/5)^{1/5}(4/5)^{4/5}}{(3/10)^{3/10}(7/10)^{7/10}}\right)^n\right) = \Omega(1.116^n).$$

To conclude, we take an enumeration of prime numbers, $p_1 < p_2 < \dots$ and we set $n_i = 5p_i$. $\blacktriangleleft$

▶ Remark 30 (Choices of k and t). The choice of $k = \lfloor 3n/10 \rfloor$ and $t = \lfloor n/10 \rfloor$ in the previous proof might appear quite enigmatic. We try to explain them now.

We want to find a number k such that there is not a big family of sets $\{A_i \subseteq \Gamma_n\}$ of size $|A_i| = k$ such that $|A_i \cup A_j| \neq n/2$, and express this fact in terms of $|A_i \cap A_j|$. Since $|A \cup B| = 2k - |A \cap B|$, if we define $t = 2k - n/2$, then $|A \cup B| \neq n/2$ if and only if $|A \cap B| \neq t$, so the value of t will be completely determined by the choice of k. Our objective is to minimise the upper bound given in Theorem 29 (what we do by minimising $k - t$) while making sure that the hypothesis $k > 2t$ is verified. In the boundary of this condition ($k = 2t$) we obtain $k = n/3$, so we express our choices as $k = (1/3 - \varepsilon)n$ and $t = (1/6 - 2\varepsilon)n$. Moreover, $k - t = (1/6 + \varepsilon)n$ has to be a prime number (for infinite n). If $1/6 + \varepsilon = 1/q$ for some $q \in \mathbb{N}$, we would succeed by considering n of the form $q \cdot p$, for p a prime number. We will therefore take $\varepsilon = \frac{6-q}{6q}$, for some q, $1 \leq q \leq 5$. With the optimal choice, $q = 5$, we obtain $k = 3n/10$, $t = n/10$ and $k - t = n/5$. Since k and t will not be integers for n of the form $5p$ (p a prime number) we are forced to take the integer part in the proof of Proposition 26.

5 Conclusion

We believe that our work is a significative advance in the understanding of the memory needed for winning ω-regular games. In combination with the literature, we can describe the current understanding of Muller languages as follows:

- The least memory necessary for winning all won L-games equals the least number of states of a GFG Rabin automaton for L.
- Computing this quantity can be done in polynomial time for L given by its Zielonka tree.
- The least chromatic memory necessary for winning all won L-games equals the least number of states of a deterministic Rabin automaton for L.
- Computing this quantity is NP-complete for L given by its Zielonka tree.
- The chromatic memory can be arbitrarily larger than the memory. It can be exponential in the size the alphabet, even while the memory remains linear.

This description shows that GFG automata play a key, and, up till now, unexplored role in understanding the complexity of Muller languages and that this role is – in some respect – even more important than that of the more classical deterministic automata.

References

1 Bader Abu Radi and Orna Kupferman. Minimizing GFG transition-based automata. In *ICALP*, volume 132, pages 100:1–100:16, 2019. doi:10.4230/LIPIcs.ICALP.2019.100.

2 Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi omega-automata format. In *CAV*, pages 479–486, 2015.

3 Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *FSTTCS*, page 16, 2018.

4 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michal Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *CoRR*, abs/2002.07278, 2020. arXiv:2002.07278.

5 Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata, 2021. arXiv:2110.14238.

6 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative-automata, 2022. To appear in proceedings of FoSSaCS'22. arXiv:2110.14308.

7 Patricia Bouyer, Mickael Randour, and Pierre Vandenhove. Characterizing omega-regularity through finite-memory determinacy of games on infinite graphs. *CoRR*, abs/2110.01276, 2021. arXiv:2110.01276.

8 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. In *CONCUR*, volume 171, pages 24:1–24:22, 2020. doi:10.4230/LIPIcs.CONCUR.2020.24.

9 J. Richard Büchi. Using determinancy of games to eliminate quantifiers. In *FCT*, volume 56 of *Lecture Notes in Computer Science*, pages 367–378. Springer, 1977. doi:10.1007/3-540-08442-8_104.

10 Antonio Casares. On the minimisation of transition-based Rabin automata and the chromatic memory requirements of Muller conditions. In *CSL*, volume 216, pages 12:1–12:17, 2022. doi:10.4230/LIPIcs.CSL.2022.12.

11 Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of games and automata using Muller conditions. In *ICALP*, volume 198, pages 123:1–123:14, 2021. doi:10.4230/LIPIcs.ICALP.2021.123.

12 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP*, pages 139–150, 2009. doi:10.1007/978-3-642-02930-1_12.

13 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theoretical Computer Science*, 352(1):190–196, 2006. doi:10.1016/j.tcs.2005.10.046.

14 Thomas Colcombet and Konrad Zdanowski. A tight lower bound for determinization of transition labeled Büchi automata. In *ICALP*, pages 151–162, 2009. doi:10.1007/978-3-642-02930-1_13.

15 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997. doi:10.1109/LICS.1997.614939.

16 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377, 1991. doi:10.1109/SFCS.1991.185392.

17 Hugo Gimbert and Wieslaw Zielonka. Games where you can play optimally without any memory. In *CONCUR*, volume 3653, pages 428–442, 2005. doi:10.1007/11539452_33.

18 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct. In *MFCS*, volume 202, pages 53:1–53:20, 2021. doi:10.4230/LIPIcs.MFCS.2021.53.

19 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC*, pages 60–65, 1982. doi:10.1145/800070.802177.

20 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic*, pages 395–410, 2006.

21 Florian Horn. Random fruits on the zielonka tree. In *STACS*, volume 3, pages 541–552, 2009. doi:10.4230/LIPIcs.STACS.2009.1848.

22 Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Annals of Pure and Applied Logic*, 69(2):243–268, 1994. doi:10.1016/0168-0072(94)90086-8.

23 Eryk Kopczyński. Half-positional determinacy of infinite games. In *ICALP*, pages 336–347, 2006. doi:10.1007/11787006_29.

24 Eryk Kopczyński. *Half-positional Determinacy of Infite Games*. Phd thesis, University of Warsaw, 2008.

25 Alexander Kozachinskiy. State complexity of chromatic memory in infinite-duration games. *CoRR*, abs/2201.09297, 2022. arXiv:2201.09297.

26 Jan Křetínský, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. Index appearance record for transforming Rabin automata into parity automata. In *TACAS*, pages 443–460, 2017. doi:10.1007/978-3-662-54577-5_26.

27 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *ICALP*, pages 299–310, 2015. doi:10.1007/978-3-662-47666-6_24.

28 Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. In *LICS*, pages 689–702, 2020. doi:10.1145/3373718.3394737.

29 Christof Löding and Anton Pirogov. Determinization of Büchi automata: Unifying the approaches of Safra and Muller-Schupp. In *ICALP*, pages 120:1–120:13, 2019. doi:10.4230/LIPIcs.ICALP.2019.120.

30 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, pages 3–36, 2020. doi:10.1007/s00236-019-00349-3.

31 Christof Löding. Optimal bounds for transformations of ω-automata. In *FSTTCS*, pages 97–109, 1999. doi:10.1007/3-540-46691-6_8.

32 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9:521–530, 1966.

33 Philipp Meyer and Salomon Sickert. On the optimal and practical conversion of Emerson-Lei automata into parity automata. *Personal Communication*, 2021.

34 Max Michel. Complementation is more difficult with automata on infinite words. *CNET, Paris*, 15, 1988.

35 Dhruv Mubayi and Vojtech Rödl. Specified intersections. *Transactions of the American Mathematical Society*, 366(1):491–504, 2014. URL: http://www.jstor.org/stable/23813142.

36 David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theor. Comput. Sci.*, 141(1–2):69–107, 1995. doi:10.1016/0304-3975(94)00214-4.

37 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006. doi:10.1109/LICS.2006.28.

38 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.

39 Schmuel Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327, 1988. doi:10.1109/SFCS.1988.21948.

40 Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, pages 167–181, 2009. doi:10.1007/978-3-642-00596-1_13.

41 Sven Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In *FSTTCS*, volume 8, pages 400–411, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.400.

42 Sven Schewe. Minimising Good-For-Games automata is NP-complete. In *FSTTCS*, volume 182, pages 56:1–56:13, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.56.

43 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

Dynamic Meta-Theorems for Distance and Matching

Samir Datta
Chennai Mathematical Institute, India

Chetan Gupta
Aalto University, Finland

Rahul Jain
Fernuniversität in Hagen, Germany

Anish Mukherjee
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland

Vimal Raj Sharma
Indian Institute of Technology, Kanpur, India

Raghunath Tewari
Indian Institute of Technology, Kanpur, India

Abstract

Reachability, distance, and matching are some of the most fundamental graph problems that have been of particular interest in dynamic complexity theory in recent years [8, 13, 11]. Reachability can be maintained with first-order update formulas, or equivalently in DynFO in general graphs with n nodes [8], even under $O(\frac{\log n}{\log \log n})$ changes per step [13]. In the context of how large the number of changes can be handled, it has recently been shown [11] that under a polylogarithmic number of changes, reachability is in DynFO$[\oplus](\leq, +, \times)$ in planar, bounded treewidth, and related graph classes – in fact in any graph where small *non-zero circulation weights* can be computed in NC.

We continue this line of investigation and extend the meta-theorem for reachability to distance and bipartite maximum matching with the same bounds. These are amongst the most general classes of graphs known where we can maintain these problems deterministically without using a majority quantifier and even maintain witnesses. For the bipartite matching result, modifying the approach from [15], we convert the static non-zero circulation weights to dynamic matching-isolating weights.

While reachability is in DynFO$(\leq, +, \times)$ under $O(\frac{\log n}{\log \log n})$ changes, no such bound is known for either distance or matching in any non-trivial class of graphs under non-constant changes. We show that, in the same classes of graphs as before, bipartite maximum matching is in DynFO$(\leq, +, \times)$ under $O(\frac{\log n}{\log \log n})$ changes per step. En route to showing this we prove that the rank of a matrix can be maintained in DynFO$(\leq, +, \times)$, also under $O(\frac{\log n}{\log \log n})$ entry changes, improving upon the previous $O(1)$ bound [8]. This implies a similar extension for the non-uniform DynFO bound for maximum matching in general graphs and an alternate algorithm for maintaining reachability under $O(\frac{\log n}{\log \log n})$ changes [13].

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory

Keywords and phrases Dynamic Complexity, Distance, Matching, Derandomization, Isolation, Matrix Rank

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.118

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2109.01875

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 118; pp. 118:1–118:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Samir Datta*: Partially funded by a grant from Infosys foundation and SERB-MATRICS grant MTR/2017/000480.
Chetan Gupta: Supported by Academy of Finland, Grant 321901.
Anish Mukherjee: Partally supported by the ERC CoG grant TUgbOAT no 772346.
Vimal Raj Sharma: Ministry of Electronics and IT, India, VVS PhD program.
Raghunath Tewari: Young Faculty Research Fellowship, Ministry of Electronics and IT, India.

Acknowledgements SD and AM would like to thank Nils Vortmeier and Thomas Zeume for their comments on Section 7.

1 Introduction

In traditional complexity theory it is assumed that the given input remains fixed throughout the computation. However in real-life scenarios, many situations involve an evolving input where parts of the data change frequently. Recomputing everything from scratch for these large datasets after every change is not an efficient option. Therefore, the goal is to dynamically maintain some auxiliary data structure to help us recompute the results quickly.

The dynamic complexity framework of Patnaik and Immerman [28] is one such approach that has its roots in descriptive complexity [23] and is closely related to the setting of Dong, Su, and Topor [14]. Here we would like to make the updates and queries by first-order logic formulas. For example, by maintaining some auxiliary relations, the reachability relation can be updated after every single edge modification in FO [8] i.e., the reachability query is contained in the dynamic complexity class DynFO [28]. The motivation to use first-order logic as the update method has connections to other areas. From the circuit complexity perspective, this implies that such queries are highly parallelizable, i.e., can be updated by polynomial-size circuits in constant-time due to the correspondence between FO and uniform AC^0 circuits [2]. From the perspective of database theory, such a program can be translated into equivalent SQL queries.

The area has seen renewed interest in proving further upper bounds results, partly after the resolution of the long-standing conjecture [28] that reachability is in DynFO under single edge modifications [8]. A natural direction to extend this result is to see which other fundamental graph problems also admit such efficient dynamic programs. The closely related problems of maintaining distance and matching are two such examples, though a DynFO bound for these problems in general graphs has been elusive so far. The best known bound for distance is DynTC^0 [20] and non-uniform $\mathsf{DynAC}^0[\oplus]$ [5]. Here, the updates are computed in FO formulas with majority quantifiers (uniform TC^0 circuits) and non-uniform FO formulas with parity quantifiers ($\mathsf{AC}^0[\oplus]$ circuits), respectively. For matching, we have a non-uniform DynFO bound for maintaining the size of the maximum matching [8]. The only non-trivial class of graphs where both these problems are in DynFO is bounded treewidth graphs [12].

At the same time progress has been made to understand how large a modification to the input can be handled by similar dynamic programs. It is of particular interest since, in applications, changes to a graph often come as a bulk set of edges. It was shown that reachability can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under changes of size $O(\log n / \log \log n)$ [13] in graphs with n nodes. Here, the class $\mathsf{DynFO}(\leq, +, \times)$ extends DynFO by access to built-in arithmetic, which is more natural for bulk changes. To handle larger changes, it is known that even for reachability, changes of size larger than polylogarithmic cannot be handled in DynFO [11]. And for changes of polylogarithmic size, the previous techniques seem to require extending DynFO by majority quantifiers [13]. Under bulk changes of polylog(n) size, we can even maintain distance and the size of a maximum matching in the uniform and the non-uniform version of $\mathsf{DynFO}[\mathsf{MAJ}](\leq, +, \times)$, respectively [13, 26].

Making further progress in this direction, recently it has been shown in [11] that reachability is in DynFO[$\oplus$]($\leq, +, \times$) (i.e., update formulas may use parity quantifiers) under polylog(n) changes in the class of graphs where polynomially bounded *non-zero circulation weights* can be computed statically in the parallel complexity class NC. A weight function for the edges of a graph has non-zero circulation if the (alternate) sum of the weights of the edges of every directed cycle is non-zero (see Section 2 for more details). Planar [29], bounded genus [10], bounded treewidth [11], single crossing minor-free [4] are some of the well-studied graph classes for which non-zero circulation weights can be computed in NC.

In this work, we first extend this result to prove similar meta-theorems for maintaining distance (including a shortest path witness) and the *search* version of minimum weight bipartite maximum matching (MinWtBMMSearch) in the same classes of graphs.

▶ **Theorem 1.** Distance *and* MinWtBMMSearch *are in* DynFO[$\oplus$]($\leq, +, \times$) *under polylog*(n) *edge changes on classes of graphs where non-zero circulation weights can be computed in* NC.

Note that these are the only classes of graphs known where we can maintain both these problems *deterministically* without using a majority quantifier and even maintain a witness to the solution (in other words, maintain a solution to the search problem).

While reachability can be maintained in DynFO($\leq, +, \times$) under bulk changes in general graphs, no such bound is known for either distance or matching in any non-trivial class of graphs under a non-constant number of changes. Since maintaining the size of maximum matching reduces to maintaining the rank of a matrix via bounded expansion first-order truth-table (bfo-tt) reduction [8], the following gives a non-uniform DynFO($\leq, +, \times$) bound for maintaining the size of maximum matching in *general* graphs to $O(\frac{\log n}{\log \log n})$ changes.

▶ **Theorem 2.** *Rank of a matrix from* $\mathbb{Z}_p^{n \times n}$ *is in* DynFO($\leq, +, \times$) *under* $O(\frac{\log n}{\log \log n})$ *entry changes.*

Earlier it was known that the rank of a matrix with small integer entries can be maintained in DynFO under changes that affect a single entry [8]. As reachability reduces to matrix rank via bfo-reduction [8], Theorem 2 also gives an alternative algorithm for maintaining reachability in DynFO($\leq, +, \times$) under $O(\frac{\log n}{\log \log n})$ changes. This is interesting in its own right as it generalizes the *rank-method* for maintaining reachability [8] even under bulk changes without going via the Sherman-Morrison-Woodbury identity [13].

Finally, building on Theorem 2, we show another meta theorem for maintaining the size of a maximum matching in bipartite graphs (BMMSize) in DynFO($\leq, +, \times$) under slightly sublogarithmic bulk changes, in the same class of graphs as in Theorem 1. Previously, no DynFO($\leq, +, \times$) bound was known even in planar graphs under single edge changes.

▶ **Theorem 3.** BMMSize *is in* DynFO($\leq, +, \times$) *under* $O(\frac{\log n}{\log \log n})$ *edge changes on classes of graphs for which non-zero circulation weights can be computed in* NC.

Main Technical Contributions. There are two major technical contributions of this work:

- *Converting the statically computed non-zero circulation weights for bipartite matchings to dynamically isolating weights for bipartite matchings.* Our main approach (described in detail in Section 3) is to assign polynomially bounded *isolating* weights to the edges of an evolving graph so that the minimum weight solution under these weights is *unique*. While static non-zero circulation weights guarantee this under deletions, for insertions, the dynamization is based on the seminal work of [15]. They construct isolating weights for perfect matching for arbitrary bipartite graphs, but which are quasipolynomially

Table 1 Previously known and new results in graphs with non-zero circulation weights in NC.

Problem	#changes		
	$O(1)$	$O(\frac{\log n}{\log \log n})$	$\log^{O(1)} n$
Reach	DynFO [8]	DynFO [13]	DynFO[⊕] [11]
Distance	DynFO[⊕]	DynFO[⊕]	DynFO[⊕]
BMMSize	DynFO	DynFO	DynFO[⊕]
BMMSearch	DynFO[⊕]	DynFO[⊕]	DynFO[⊕]

large in the size of the graphs. By assigning such weights only to the changed part of the graph and carefully combining with the previously assigned weights, we make sure the edge weights remain small as well as isolating throughout, using the Muddling Lemma (see Section 2). Our construction parallels that of [11] where dynamic isolating weights for reachability in non-zero circulation graphs were constructed based on the static construction from [24]. In addition to extending the reachability result (Theorem 1) this also enables us to prove a DynFO$(\leq, +, \times)$ bound (Theorem 3) for bipartite maximum matching (previously, a rather straightforward application of non-zero circulation weights in planar graphs could only achieve a DynFO[⊕] bound under single edge changes [26]).

- *Maintaining rank of a matrix under sublogarithmically many changes.* This involves non-trivially extending the technique from [8], which maintains rank under single entry changes, and combining it with [13] which shows how to compute the determinant of a small matrix of dimension $O(\frac{\log n}{\log \log n})$ in FO$(\leq, +, \times)$.

Organization. After some preliminaries in Section 2, in Section 3 we discuss the connection between dynamic isolation and static non-zero circulation and show its applications for matching and distance in Section 4 and Section 6, respectively. In Section 5, we describe the DynFO$(\leq, +, \times)$ algorithm for maximum matching, which is built on the rank algorithm under bulk changes from Section 7. Finally, we conclude with Section 8.

2 Preliminaries and Notations

Dynamic Complexity. The goal of a dynamic program is to answer a given query on an *input structure* subjected to insertion or deletion of tuples. The program may use an *auxiliary data structure* over the same domain. Initially, both input and auxiliary structures are empty; and the domain is fixed during each run of the program.

For a (relational) structure $\mathcal{I}$ over domain D and schema σ, a change $\Delta\mathcal{I}$ consists of sets R^+ and R^- of tuples for each relation symbol $R \in \sigma$. The result $\mathcal{I} + \Delta\mathcal{I}$ is the input structure where $R^{\mathcal{I}}$ is changed to $(R^{\mathcal{I}} \cup R^+) \setminus R^-$. The set of affected elements is the (active) domain of tuples in $\Delta\mathcal{I}$. A dynamic program $\mathcal{P}$ is a set of first-order formulas specifying how auxiliary relations are updated after a change. For a state $\mathcal{S} = (\mathcal{I}, \mathcal{A})$ with input structure $\mathcal{I}$ and auxiliary structure $\mathcal{A}$ we denote the state of the program after applying a change sequence α and updating the auxiliary relations accordingly by $\mathcal{P}_\alpha(\mathcal{S})$.

The dynamic program maintains a q-ary query Q under changes that affect k elements if it has a q-ary auxiliary relation ANS that at each point stores the result of Q applied to the current input structure. I.e., for each non-empty sequence α of changes affecting k elements, the relation ANS in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and the relation $Q(\alpha(\mathcal{I}_\emptyset))$ coincide, where the state $\mathcal{S}_\emptyset = (\mathcal{I}_\emptyset, \mathcal{A}_\emptyset)$ consists of an input structure $\mathcal{I}_\emptyset$ and an auxiliary structure $\mathcal{A}_\emptyset$ over some common domain that both have empty relations, and $\alpha(\mathcal{I}_\emptyset)$ is the input structure after applying α.

If a dynamic program maintains a query, we say that the query is in DynFO. Similar to DynFO, one can define the class of queries DynFO$(\leq, +, \times)$ that allows for auxiliary relations initialized as a linear order, and the corresponding addition and multiplication relations. One can further extend this class by allowing parity quantifiers to yield the class DynFO$[\oplus](\leq, +, \times)$ and majority quantifiers to yield the class DynFO[MAJ]$(\leq, +, \times)$. The parity and majority functions of n bits $a_1, \ldots, a_n$ are true if $\sum_{i=1}^{n} a_i = 1 \pmod 2$ and $\sum_{i=1}^{n} a_i \geq n/2$, respectively. As we focus on changes of non-constant size, we include arithmetic in our setting. See [13, 11] for more details.

The *Muddling Lemma* [11] states that to maintain many natural queries, it is enough to maintain the query for a bounded number of steps, that we crucially use in this paper. In the following, we first recall the necessary notions before stating the lemma.

A query Q is *almost domain-independent* if there is a $c \in \mathbb{N}$ such that $Q(\mathcal{A})[(\text{adom}(\mathcal{A}) \cup B)] = Q(\mathcal{A}[(\text{adom}(\mathcal{A}) \cup B)])$ for all structures $\mathcal{A}$ and sets $B \subseteq A \setminus \text{adom}(\mathcal{A})$ with $|B| \geq c$. Here, adom$(\mathcal{A})$ denotes the *active domain*, the set of elements that are used in some tuple of $\mathcal{A}$. A query Q is $(\mathcal{C}, f)$*-maintainable*, for some complexity class $\mathcal{C}$ and some function $f : \mathbb{N} \to \mathbb{R}$, if there is a dynamic program $\mathcal{P}$ and a $\mathcal{C}$-algorithm $\mathbb{A}$ such that for each input structure $\mathcal{I}$ over a domain of size n, each linear order $\leq$ on the domain, and each change sequence α of length $|\alpha| \leq f(n)$, the relation Q in $\mathcal{P}_\alpha(\mathcal{S})$ and $Q(\alpha(\mathcal{I}))$ coincide, where $\mathcal{S} = (\mathcal{I}, \mathbb{A}(\mathcal{I}, \leq))$. AC^i is the class of problems that can be solved using polynomial-size circuit of $O(\log^i n)$ depth and $\mathsf{NC} = \cup_i \mathsf{AC}^i$.

▶ **Lemma 4** (Muddling Lemma [11]). *Let Q be an almost domain independent query, and let $c \in \mathbb{N}$ be arbitrary. If the query Q is $(\mathsf{AC}^d, 1)$-maintainable under changes of size $\log^{c+d} n$ for some $d \in \mathbb{N}$, then Q is in* DynFO$(\leq, +, \times)$ *under changes of size $\log^c n$.*

There are several roughly equivalent ways to view the complexity class DynFO as capturing:

- The dynamic complexity of maintaining a Pure SQL database under fixed (first-order) updates and queries (the original formulation from [28]).
- The circuit dynamic complexity of maintaining a property where the updates and queries use uniform AC^0 circuits (see [2] for the equivalence of uniform AC^0 and FO).
- The parallel dynamic complexity of maintaining a property where the updates and queries use constant time on a CRCW PRAM (for the definition of Concurrent RAM, see [23]).

The first characterization is popular in the Logic and Database community, while the second is common in more complexity-theoretic contexts. The third one is useful to compare and contrast this class with dynamic algorithms, which essentially classify dynamic problems in terms of the sequential time for updates and queries. Operationally, our procedure is easiest to view in terms of the second or even the third viewpoint. We would like to emphasize that modulo finer variations based on built-in predicates (like arithmetic and order) in the first variation, uniformity in the second one and built-in predicates (like shift) in the third one, the three viewpoints are entirely equivalent.

We refer the readers to [8, 13, 11] for more discussion on the basics of the dynamic complexity framework.

Weight function and Circulation. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. By $G = (V, \vec{E})$ denote the corresponding graph where each of its edges is replaced by two directed edges, pointing in opposite directions. Let $|V| = n$ and we use the natural interpretation of the universe i.e., the set of vertices as the natural numbers from $[n]$.

A set system $\mathcal{M}$ on a universe U is a family of subsets of U i.e. $\mathcal{M} \subseteq 2^U$. Examples include the family of s, t-shortest paths and perfect matchings in a graph. A weight function $w : U \to \mathbb{Z}_{\geq 0}$ (the set of non-negative integers) induces a weight of $w(M) = \sum_{e \in M} w(e)$ on

an element $M \in \mathcal{M}$. Such a weight function is said to be *isolating* for $\mathcal{M}$ if there is at most one element $M_0 \in \mathcal{M}$ with the minimum weight. The notion of isolation can be extended to a collection of families of graphs such as the collection of families of s,t-shortest paths for all $s,t \in V$. The weight function $w(e) = 2^{(n+1)u+v}$ for $e = \{u,v\}$, $u < v$, is a trivial isolating function – instead we want to give weights polynomially bounded in the size of the universe. A randomized construction of such a weight function is known for arbitrary set systems [27].

A function $w : \vec{E} \to \mathbb{Z}$ is called *skew-symmetric* if for all $e \in \vec{E}$, $w(e) = -w(e^r)$ (where e^r represent the edge with its direction reversed). The *circulation* of a directed cycle under a skew symmetric weight function is the absolute value of the sum of weights of the directed edges in the cycle. The skew-symmetric weight function w induces a *non-zero circulation* on the graph if every directed cycle in the graph gets a non-zero circulation under w.

We know from [3] that if w assigns non-zero circulation to every cycle that consists of edges of $\vec{E}$, then it isolates a directed path between each pair of vertices in $G = (V, \vec{E})$. Also, if G is a bipartite graph, then the weight function w can be used to construct a weight function $w' : E \to \mathbb{Z}$ that isolates a perfect matching in G [29]. For planar [29], bounded genus [10], bounded treewidth [11], and for any single crossing minor-free graph [4] non-zero circulation weights can be computed *deterministically* in Logspace, which is a subclass of NC.

Our convention represents by $\langle w_1, \ldots, w_k \rangle$ the weight function that on edge e takes weight $\sum_{i=1}^{k} w_i(e) B^{k-i}$, where $w_1, \ldots, w_k$ are weight functions such that $\max_{i=1}^{k} (n \cdot w_i(e)) \leq B$.

Maintaining Witnesses. The proof of [27] for the construction of a perfect matching witness carries over to the dynamic setting also and allow us to maintain a witness to the solution in $\mathsf{DynFO}[\oplus](\leq, +, \times)$. Since the perfect matching is isolated, from its weight one can infer the edges in the matching by deleting the edges one a time in parallel and see if the weight remains unchanged and accordingly place the edge in the matching. This is doable in $\mathsf{FO}[\oplus](\leq, +, \times)$. The extraction procedure for shortest path and maximum cardinality matching is similar.

3 Dynamic Isolation from Static Non-Zero Circulation

We know (from Section 2) that non-zero circulation weights are isolating weights. Thus, statically (when the given graph does not change over time) we can use them to obtain efficient parallel algorithm for distance and matching. However maintaining these weights seems to be hard in an evolving graph. This is because even for planar graphs where static non-zero circulation weights are easy to construct [3, 9, 29] maintaining them dynamically seems to need a dynamic planar embedding algorithm. And that alone doesn't seem to suffice since even small changes in the input can lead to large changes in the embedding and that will require us to change the weights of many edges (since weight of the edges are determined by the embedding [3, 29]). This induces us to side-step maintaining non-zero circulation weights.

In this paper, we circumvent this problem by modifying the approach of [15] to convert the given static nonzero-circulation weights to dynamic isolating weights. Notice that [15] yields a black box recipe to produce isolating weights of quasipolynomial magnitude for bipartite graphs, which we label as *FGT-weights* in the following way. Let $e_1, e_2, \ldots$ be the edges of a bipartite graph. Consider a non-zero circulation of exponential magnitude viz. $w_0(e_i) = 2^i$. Next, consider a list of $\ell = O(\log n)$ primes $\vec{p} = (p_1, \ldots, p_\ell)$, each of $O(\log n)$ bits, which yield a weight function $w_{\vec{p}}(e_i)$. This is defined by taking the ℓ weight functions $w_0 \bmod p_j$ for $j \in \{1, \ldots, \ell\}$ and concatenating them with *shifting* the weights to the higher-order bits appropriately, that is: $w_{\vec{p}}(e) = \langle w_0(e) \bmod p_1, \ldots, w_0(e) \bmod p_\ell \rangle$. This

is so that there is no overflow from the j-th to the $(j-1)$-th weight for any $j \in \{2, \ldots, \ell\}$. In [15], it was proved that for every graph there exist some set of ℓ primes such that the respective weight function $w_{\vec{p}}$ isolates a perfect matching in the graph.

Suppose we start with a graph with static polynomially bounded weights ensuring non-zero circulation. In a step, some edges are inserted or deleted. The graph after deletion is a subgraph of the original graph; hence the non-zero circulation remains non-zero after a deletion[1], but we have to do more in the case of insertions. We aim to give the newly inserted edges FGT-weights in the higher-order bits while giving weight 0 to all the original edges in G again in the higher-order bits. Thus the weight of all perfect matchings that survive the deletions in a step remains unchanged. Moreover, if none such survive but new perfect matchings are introduced (due to insertion of edges) the lightest of them is determined solely by the weights of the newly introduced edges. In this case, our modification of the existential proof from [15] ensures that the minimum weight perfect matching is unique.

In order to handle bulk insertion of $N = \log^{O(1)} n$ edges, we need to apply the FGT-recipe described above to a set system with a universe of N elements. This yields quasipolynomial ($N^{\log^{O(1)} N}$) weights in N which are therefore still subpolynomial in n ($2^{(\log\log n)^{O(1)}} = 2^{o(\log n)} = n^{o(1)}$). Further, the number of primes is polyloglog ($\log^{O(1)} N = (\log\log n)^{O(1)}$) and so sublogarithmic ($\log^{o(1)} n$). Hence, the number of possible different weights is subpolynomial, which allows us to derandomize our algorithm by going over all possible FGT-weight functions defined above. We point out that in [11] a similar scheme is used for reachability and bears the same relation to [24] as this section does to [15]. We have the following lemma, which we prove in Section 3.1. Our proof of the lemma is crucially based on [15] but our proof is self contained except for Lemma 9 which we assume as a black box.

▶ **Lemma 5.** *Let G be a bipartite graph with non-zero circulation w^{old}. Suppose $N = \log^{O(1)} n$ edges are inserted into G to yield G^{new}. Then we can compute polynomially many weight functions in $\mathsf{FO}(\leq, +, \times)$ that have $O(\log n)$ bit weights, and at least one of them, w^{new} is isolating. Furthermore, the weights of the original edges remain unchanged under w^{new}.*

3.1 Details of Maintaining Dynamic Isolating Weights

We divide the edges of the graph into *real* and *fictitious*, where the former represents the newly inserted edges and the latter original undeleted edges[2].

Next, we follow the proof idea of [15] but focus on assigning weights to only real edges which are $N = \log^{O(1)} n$ in number. We do this in $\log N$ stages starting with a graph $G_0 = G$ and ending with an acyclic graph G_ℓ that contains a unique perfect matching if and only if G contains a perfect matching, where $\ell = \log N$. At each step, we maintain the following.

▶ **Invariant 6.** For $i \geq 1$, G_i contains no cycles with at most 2^{i+1} real edges.

Assuming this invariant we complete the proof of Lemma 5:

Proof of Lemma 5. From the invariant above G_ℓ does not contain any cycle that consists of real edges. From the construction of G_ℓ, if G has a perfect matching, then so does G_ℓ and hence it is a perfect matching. Notice that W_ℓ is obtained from $p_1, \ldots, p_\ell$ that include $O((\log\log n)^2) = o(\log n)$ many bits. Thus there are (sub)polynomially many such weighting functions W_ℓ, depending on the primes $\vec{p}$. Let $w = B \cdot W_\ell + w^{old}$ where we recall that $W_\ell(e)$

[1] If we merely had isolating weights, this would not necessarily preserve isolation.
[2] We use the terms old $\leftrightarrow$ fictitious and new $\leftrightarrow$ real interchangeably in this section.

is non-zero only for the new (real) edges and w^{old} is non-zero only for the old (fictitious) edges. Thus, any perfect matching that consists of only old edges is lighter than any perfect matching containing at least one new edge. Moreover, if the real edges in two matchings differ, then, from the construction of W_ℓ (for some choice of $\vec{p}$) both matchings cannot be the lightest as W_ℓ real isolates a matching. Thus the only remaining case is that we have two distinct lightest perfect matchings, which differ only in the old edges. But the symmetric difference of any two such perfect matchings is a collection of cycles consisting of old edges. But each cycle has a non-zero circulation in the old graph and so we can obtain a matching of even lesser weight by replacing the edges of one of the matchings in one cycle by the edges of the other one. This contradicts that both matchings were of least weight. ◀

Next we show how Invariant 6 is maintained. Notice that the case $i = 0$ follows from the above discussion and the induction starts at $i > 0$.

We first show how to construct G_{i+1} from G_i such that if G_i satisfies the inductive invariant 6, then so does G_{i+1}. In the i-th step, let $\mathcal{C}_{i+1}$ be the set of cycles that contain at most 2^{i+2} real edges, for $i > 1$. For each such cycle $C = f_0, f_1, \ldots$ containing $k \leq 2^{i+2}$ real edges (with f_0 being the least numbered real edge in the cycle), edge-partition the cycle into 4 consecutive paths $P_j(C)$ for $j \in \{0, 1, 2, 3\}$ such that the first three paths contain exactly $\lfloor \frac{k}{4} \rfloor$ real edges and the last path contains the rest. In addition ensure that the first edge in each path is a real edge. Let the first edge of the 4-paths be respectively $f'_0 (= f_0), f'_1, f'_2, f'_3$. We identify each cycle in $\mathcal{C}_{i+1}$ with these 4-tuples $\langle f'_0, f'_1, f'_2, f'_3 \rangle$.

For a cycle $C \in \mathcal{C}_{i+1}$, we define a set C' which consists of only real edges of C. Similarly, $\mathcal{C}'_{i+1} = \{C' \mid C \in \mathcal{C}_{i+1} \wedge C' \neq \emptyset\}$. Note that there can be many cycles in $\mathcal{C}_{i+1}$ corresponding to a single set in $\mathcal{C}'_{i+1}$ (i.e., those cycles that contain the same set of real edges). We fix a particular $C \in \mathcal{C}_{i+1}$ for every $C' \in \mathcal{C}'_{i+1}$ with which it is associated. The 4-tuple associated with the cycle C is also associated with the corresponding set C'. We have the following which shows that the associated 4-tuples $\langle f'_0, f'_1, f'_2, f'_3 \rangle$ uniquely characterise sets in $\mathcal{C}'_{i+1}$.

▷ Claim 7. There is at most one set in $\mathcal{C}'_{i+1}$ that has a given 4-tuple $\langle f'_0, f'_1, f'_2, f'_3 \rangle$ associated with it.

Proof. Suppose two distinct sets $C'_1, C'_2 \in \mathcal{C}'_{i+1}$ have a common 4-tuple $\langle f'_0, f'_1, f'_2, f'_3 \rangle$ associated with them. Let C_1, C_2 be two cycles corresponding to C'_1 and C'_2, respectively. Then for at least one $j \in \{0, 1, 2, 3\}$, $P_j(C_1) \neq P_j(C_2)$. Hence, $P_j(C_1) \cup P_j(C_2)$ is a closed walk in G_i containing at most $2 \times \lceil \frac{2^{i+2}}{4} \rceil = 2^{i+1}$ real edges, contradicting the assumption on G_i. ◁

This shows that there are at most N^4 elements in $\mathcal{C}'_{i+1}$ because that is the maximum number of distinct 4-tuples of real edges. We define circulation for the sets in $\mathcal{C}'_{i+1}$ via the circulation for those in $\mathcal{C}_{i+1}$. We know that for every $C' \in \mathcal{C}'_{i+1}$ there is at least one $C \in \mathcal{C}_{i+1}$ corresponding to it. Let w be a weight function that gives non-zero weights to only real edges of the graph. Circulation of $C' \in \mathcal{C}'_{i+1}$ with respect to w is defined as $c_w(C') = c_w(C) = |w(e_1) - w(e_2) + w(e_3) - \ldots|$, where $e_i \in C$. This is well-defined since in bipartite graphs the parity of the length of any two paths between the same pair of vertices is the same. Thus, all C such that they have C' associated with it, have the same circulation since the sign associated with a real edge does not change for any such C. Now we will use the following lemma to ensure non-zero circulations to sets in $\mathcal{C}'_{i+1}$.

▶ **Lemma 8** (Based on Lemma 2 in [16]). *For every constant $k > 0$ there is a constant $k_0 > 0$ such that for every set S of m-bit integers with $|S| \leq m^k$, the following holds: There is a $k_0 \log m$-bit prime number p such that for any $x, y \in S$, if $x \neq y$ then $x \not\equiv y \bmod p$.*

We apply the above lemma to the set $c_{w_0}(\mathcal{C}'_{i+1}) = \{c_{w_0}(C') : C' \in \mathcal{C}'_{i+1}\}$. Here, the weight function w_0 assigns weights $w_0(e_j) = 2^j$ to the real edges which are $e_1, e_2, \ldots, e_N$ in an arbitrary but fixed order. Notice that from the above claim, the size of this set is $|\mathcal{C}'_{i+1}| \leq N^4$. Since $w_0(e_j)$ is j-bits long hence $c_{w_0}(C)$ for any cycle $C \in \mathcal{C}_i$ that has less than 2^{i+2} real edges is at most $i + j + 2 < 4N$-bits long. Thus, we obtain a prime p_{i+1} of length at most $k_0 \log(4N)$ by picking $k = 4$. We define $w_{i+1}(e_j) = w_0(e_j) \bmod p_{i+1}$. By Lemma 8 we know that circulation of each set in $\mathcal{C}'_{i+1}$ is non-zero with respect to w_{i+1}. Therefore, circulation of all the cycles in $\mathcal{C}_{i+1}$ is nonzero with respect to w_{i+1} (remember that w_{i+1} assign nonzero weights to only real edges).

Now consider the following crucial lemma from [15]:

▶ **Lemma 9** ([15]). *Let $G = (V, E)$ be a bipartite graph with a weight function w. Let C be a cycle in G such that $c_w(C) \neq 0$. Let E_1 be the union of all minimum weight perfect matchings in G. Then the graph $G_1 = (V, E_1)$ does not contain the cycle C. Moreover, all the perfect matchings in G_1 have the same weight.*

Let B be a large enough constant (though bounded by a polynomial in N) to be specified later. We shift the original accumulated weight function W_i and add the new weight function w_{i+1} to obtain: $W_{i+1}(e) = W_i(e)B + w_{i+1}(e)$. Apply W_{i+1} on the graph G_i to obtain the graph G_{i+1}. Inductively suppose we have the invariant 6 that the graph G_i did not have any cycles containing at most 2^{i+1} real edges. This property is preserved when we take all the perfect matchings in G_i and apply W_{i+1} yielding G_{i+1}. Moreover, from Lemma 9 and the construction of w_{i+1}, the cycles of $\mathcal{C}_i$ disappear from G_{i+1} restoring the invariant. This yields a weight function W_ℓ using that $\ell = \log n$ (see the discussion before Invariant 6).

Notice that it suffices to take B greater than the number of real edges times the maximum of $w_i(e)$ over i, e. Showing that G_1 contains no cycle with at most 4 real edges mimics the above more general proof, and we skip it here. We say that a weight function that gives non-zero weights to the real edges, *real isolates* $\mathcal{M}$ for a set system $\mathcal{M}$ if the minimum weight set in $\mathcal{M}$ is unique with respect to that weight function. In our context, $\mathcal{M}$ will refer to the set of perfect/maximum matchings.

4 Maximum Cardinality Matching Search in DynFO$[\oplus](\leq, +, \times)$

In this section, we convert the static algorithm for maximum matching search in bipartite graphs into a dynamic algorithm with the help of the isolating weights from the previous section. In the static setting [6] the problem reduces to determining non-singularity of an associated matrix given a non-zero circulation for the graph.

The algorithm extracts what is called a min-weight *generalized* perfect matching (min-weight GPM), that is, a matching along with some self-loops. The construction proceeds by adding a distinct edge (v, t_v) on every vertex $v \in V(G)$ with a self-loop on the new vertex t_v to yield the graph G'. The idea is to match as many vertices as possible in G' using the actual edges of G while reserving the pendant edges (v, t_v) to match vertices that are unmatched by the maximum matching. If a vertex v is matched in a maximum matching of G then the vertex t_v is "matched" using a self-loop.

Given a non-zero circulation weight w''' for G the weight function for G' is $w = \langle w', w'', w''' \rangle$. Here we represent by w' the function that is identically 0 for all the self loops and is 1 for all the other edges. $w''(e)$ is zero except for pendant edges $e = (v, t_v)$, for $v \in V(G)$, which have $w''(e) = v$ (where v is interpreted as a number in $\{1, \ldots, |V(G)|\}$) such that all vertices get distinct numbers. The paper [6] considers the weighted Tutte matrix

T where for an edge (u, v) the entry $T(u, v) = \pm x^{w(u,v)}$ (say, with a positive sign iff $u < v$) and is zero otherwise. It shows that in the univariate determinant polynomial $det(T)$ the least degree term x^W with a non-zero coefficient must have this coefficient equal to ± 1 and the exponent W is the weight of the minimum weight generalized perfect matching in G'. Further this min-weight GPM consists of a maximum cardinality matching in G along with the edges (v, t_v) for all the vertices v unmatched in the maximum matching. The edges in the maximum matching can then be obtained by checking if, on removing the edge (u, v), the weight of the min-weight maximum matching increases.

The idea behind the proof is as follows:

1. The most significant weight function w' ensures that the cardinality of the actual edges (i.e. edges from G) picked in the min-wight GPM in G' equals the cardinality of the maximum matching in G. This is because the GPM would cover as many of the t_v vertices with self-loops as possible to minimize the weight that ensures the corresponding v must be covered by an actual edge.
2. The next most significant weight function w'' is used to ensure that all the min-weight GPMs use the same set of pendant edges. This is because, otherwise, there is an alternating path in the symmetric difference of the two GPMs that starts and ends at self-loops t_u, t_v. Then, the difference in the weights w'' of the two matchings restricted to the path is $u - v \neq 0$ and we can find a GPM of strictly smaller weight by replacing the edges of one matching with the edges of the other matching restricted to the path, contradicting that both matchings were the lightest GPMs.
3. The least significant weight function w''' then isolates the GPM since all min-weight GPMs are essentially perfect matchings restricted to the same set S of vertices, namely those that are not matched by the corresponding pendant edges and the non-zero circulation weights on G ensures that these are isolating weights on the induced graph $G[S]$.

We claim that we just need *isolating* weights w''' instead of non-zero circulation weights to ensure that the above technique works. Replacing non-zero circulations with isolating weights does not affect the first two steps. It would seem, the third step does not work since isolating weights for G might not be isolating weights for the subgraph $G[S]$. However, Lemma 5 can be applied to the graph $G[S]$ directly – notice that in the above proof sketch S is determined by the first two weight functions w', w'' and does not depend on the third w'''.

As described above, we need to maintain the determinant of a certain matrix A related to the Tutte matrix in order to find the size of the maximum cardinality matching. For a small change matrix B, the Matrix Determinant Lemma [30, 17]

$$\det(A + UBV) = \det(I + BVA^{-1}U)\det(A)$$

allows us to maintain the determinant by reducing it to maintaining the inverse of the matrix. To maintain the inverse, the Sherman-Morrison-Woodbury formula [19]

$$(A + UBV)^{-1} = A^{-1} - A^{-1}U(I + BVA^{-1}U)^{-1}BVA^{-1}$$

tells us how the task of recomputing the inverse of a non-singular matrix A under small changes B reduces to that of computing the inverse of a small matrix $(I+BVA^{-1}U)$ statically. So we need to ensure that the matrix remains invertible throughout which is what we achieve below by tinkering with the definition of the Tutte matrix. We have the following definition:

▶ **Definition 10.** *The generalized Tutte matrix is a matrix with rows and columns indexed by $V(G') = V(G) \cup \{t_v : v \in V(G)\}$ and with the following weights on edges: $T(t_v, t_v) = 1$, $T(v, v) = x^{w_\infty}$ and $T(a, b) = \pm x^{\langle w', w'', w''' \rangle (a,b)}$ whenever $a, b \in V(G'), a \neq b$. It is ensured in the above that $T(a, b) = -T(b, a)$. Here w_∞ is a polynomially bounded number larger than the largest of the weights $\langle w', w'', w''' \rangle (a, b)$.*

Note that the generalized Tutte matrix is not unique. We now have the following:

▶ **Lemma 11.** *Let T be a generalized Tutte matrix defined above, then the highest exponent w such that x^w divides $det(T)$ is twice the weight of the min-weight GPM in G'. Further, the matrix T is invertible.*

Proof. From the properties of the weight function $\langle w', w'', w''' \rangle$ we see that the minimum weight generalized perfect matching is unique see [6, Lemma 9]. The exponent of the least degree monomial is twice the weight of the min-weight GPM when we take superposition of the unique min-weight GPM with itself [6, Observation 14]. The self loops on the vertices t_v appear only once in a monomial and twice in the superposition, but because their weight is zero under $\langle w', w'', w''' \rangle$ it will not affect the exponent.

In order to guarantee invertibility of T we just need to prove that the product of the diagonal terms yields a monomial of much higher degree than any other monomial and of coefficient one, since this implies that the matrix is non-singular because this monomial cannot be canceled out by the rest of the monomials. Consider the product of the diagonal entries viz. $x^{\frac{|V(G')|}{2} w_\infty} = x^{|V(G)| w_\infty}$. Any monomial with less than $|V(G)|$ diagonal entries $T(v, v)$ is bound to be of much smaller exponent. Now the monomials which use an off-diagonal entry $T(v, u)$ or $T(v, t_v)$ must miss out on the diagonal entry in the v-th row, making the exponent much smaller. ◀

4.1 Maintaining the Determinant and Inverse of a Matrix

We need the following definitions and results about univariate polynomials, matrices of univariate polynomials and operations therein over a finite field of characteristic 2. Let $\mathbb{F}_2$ be the field of characteristic 2 containing 2 elements. For (potentially infinite) power series $f, g \in \mathbb{F}_2[[x]]$, we say f *m-approximates* g (denoted by $f \approx_m g$) if the first m terms of f and g are the same. We will extend this notation to matrices and write $F \approx_m G$ where $F, G \in \mathbb{F}_2[[x]]^{\ell \times \ell}$ are matrices of power series. We will have occasion to use this notation only when one of F, G is a matrix of polynomials, that is, a matrix of finite power series.

Notice that if $A \in \mathbb{F}_2[x]^{n \times n}$ with the degree of entries bounded by w_∞, then there exists $A^{-1} \in \mathbb{F}_2[[x]]^{n \times n}$. For us, only the monomials with degrees at most w_∞ are relevant. Thus we will assume that we truncate A^{-1} at w_∞ many terms to yield matrix $A' \approx_{w_\infty} A^{-1}$. Then we have the following:

▶ **Lemma 12** (Lemma 10 in [11]). *Suppose $A \in \mathbb{F}_2[x]^{n \times n}$ is invertible over $\mathbb{F}_2[[x]]$, and $C \in \mathbb{F}_2[x]^{n \times n}$ is an m-approximation of A^{-1}. If $A + UBV$ is invertible over $\mathbb{F}_2[[x]]$ with $U \in \mathbb{F}_2[x]^{n \times \ell}, B \in \mathbb{F}_2[x]^{\ell \times \ell}$, and $V \in \mathbb{F}_2[x]^{\ell \times n}$, then $(A + UBV)^{-1} \approx_m C - CU(I + BVCU)^{-1}BVC$. Furthermore, if $\ell \leq \log^c n$ for some fixed c and all involved polynomials have polynomial degree in n, then the right-hand side can be computed in $\mathsf{FO}[\oplus](\leq, +, \times)$ from C and ΔA.*

Similar to the above (using the closure of m-approximation under product), we get an approximate version of the Matrix Determinant Lemma, that is:

▶ **Proposition 13.** *Suppose $A \in \mathbb{F}_2[x]^{n \times n}$ is invertible over $\mathbb{F}_2[[x]]$, and $C \in \mathbb{F}_2[x]^{n \times n}$ is an m-approximation of A^{-1} and polynomial $d(x) \approx_m det(A)$ then $d \cdot \det(I + BVCU) \approx_m \det(A + UBV)$.*

We can now proceed to prove Theorem 1:

Proof of Theorem 1. By putting $m = w_\infty$ and applying Lemma 12 and Propositions 13 to the generalized Tutte matrix from Lemma 11 and using the (Muddling) Lemma 4, we complete the matching part of Theorem 1 (see Section 6 for the proof involving distance). ◀

5 Maximum Cardinality Matching in DynFO$(\leq, +, \times)$

In this section we prove Theorem 3 by giving a DynFO$(\leq, +, \times)$ algorithm for maintaining the size of a maximum matching under $O(\frac{\log n}{\log\log n})$ changes. Notice that the approach in the previous section has the limitation that it only gives a DynFO$[\oplus](\leq, +, \times)$ bound as we need to maintain polynomials of large (polynomial in n) degree. Instead, the main ingredient here is a new algorithm for maintaining the rank of a matrix in DynFO$(\leq, +, \times)$ under $O(\frac{\log n}{\log\log n})$ changes (Section 7). Our matching algorithm follows the basic approach of the non-uniform DynFO algorithm of [8]. Here, since we use deterministic isolation weights (as opposed to the randomized isolation weights of [27]), with some more work, we obtain a *uniform* DynFO$(\leq, +, \times)$ bound under bulk changes.

The algorithm of [8] builds on the well-known correspondence between the size of maximum matching and the rank of the Tutte matrix of the corresponding graph – if a graph contains a maximum matching of size m then the associated Tutte matrix is of rank $2m$ [25]. The dynamic rank algorithm from Section 7 cannot be applied directly since the entries of the Tutte matrix are indeterminates. However, the rank can be determined by replacing each x_{ij} by $2^{w(i,j)}$. Here w assigns a positive integer weight to every edge (i, j) under which the maximum matching gets unique minimal weight, i.e., it is matching-isolating. Using the Isolation Lemma [27], it can be shown that the correspondence between the rank and the size of the maximum matching does not change after such a weight transformation [22, 8].

Our algorithm diverges from [8] as we need to deterministically compute these isolating weights and also, to somehow maintain those. Since we do not know how to maintain such weights directly, as in Section 4, we convert the static non-zero circulation weights to dynamic isolating weights using the Muddling Lemma 4. Given a graph G, let B_w be its weighted Tutte matrix with each x_{ij} replaced by $2^{w(i,j)}$ for an isolating weight function w. Initially, the static non-zero circulation weights provide such weights. Since we are only interested in computing the rank of B_w, we do not need to make the initial modifications of adding pendant edges or self-loops to G as before. So the weight function w is just the non-zero circulation weight $\langle w''' \rangle$ here. In the dynamic process, similar to Section 4, we use the FGT-weights w_{new} on top for the newly inserted edges. We have the following:

▶ **Lemma 14.** *Given a dynamic algorithm for maintaining the rank of an integer matrix under $k = O(\log^c n)$ changes at each step for some fixed constant c, we can maintain the size of the maximum matching in the same complexity class under $O(k)$ changes for the class of graphs where non-zero circulation weights can be computed in* NC.

Proof. Given a graph G, assume we have an algorithm for computing the non-zero circulation weight function w in $\mathsf{NC}^i \subseteq \mathsf{AC}[\frac{\log^i n}{\log\log n}]$ for some fixed integer i. Once these weights w are available, rank(B_w) can be found in NC^2 [1] which is contained in $\mathsf{AC}[\frac{\log^2 n}{\log\log n}]$. Since $O(k)$ changes can occur at each step, during this time, total of $O(k \cdot (\frac{\log^i n}{\log\log n} + \frac{\log^2 n}{\log\log n}))$ many new changes accumulate. As w assigns non-zero circulation weights to the edges of G, we can assign weight 0 to the deleted edges and the weights remain isolating. For the newly inserted edges, which are only polylog(n) many, we compute the polynomially bounded FGT-weights in AC^0 using Lemma 9. Thanks to Lemma 4, in $O(\frac{\log^i n}{\log\log n} + \frac{\log^2 n}{\log\log n}))$ many steps we can take care of all the insertions by adding k new edges at each step along with k old ones in double

the speed using our rank algorithm. Note that, during the static rank computation phase, we do not restart the static algorithm for computing the weight w. Instead, we recompute these weights once the rank computation using them finishes. More precisely, we can think of a combined static procedure that computes the non-zero circulation weights followed by the rank of the weighted Tutte matrix B_w in NC^b for b=max$(i, 2)$. And on this combined procedure, we apply our Muddling Lemma 4. ◀

We can now prove Theorem 3:

Proof of Theorem 3. Similar to [8, Theorem 16] this implies a *uniform* bounded expansion first-order truth-table (bfo-tt) reduction from maximum matching to rank in this special case.[3] Since $\mathsf{DynFO}(\leq, +, \times)$ is closed under bfo-tt reductions [8, Proposition 4] and dynamic rank maintenance is in $\mathsf{DynFO}(\leq, +, \times)$ under $O(\frac{\log n}{\log \log n})$ changes (Theorem 2), in classes of graphs where non-zero circulation weights can be computed in NC we have the result. ◀

6 Maintaining Distance under Bulk Changes

In this section, extending the reachability result of [11], we show that distances can be maintained in $\mathsf{DynFO}[\oplus](\leq, +, \times)$ under polylog(n) changes in classes of graphs where non-zero circulation can be computed in NC. We start with describing the reachability algorithm of [11] followed by the necessary modifications needed for maintaining distance information.

6.1 Outline of the Approach for Reachability

Let $G = (V, E)$ be the given n-node graph with an isolating weight assignment w. For a formal variable x, let the corresponding weighted adjacency matrix $A = A_{(G,w)}(x)$ be defined as follows: if $(u, v) \in E$, then $A[u, v] = x^{w(u,v)}$, and 0 otherwise. Consider the matrix $D = (I - A)^{-1}$, where I is the identity matrix. Notice that the matrix $(I - A)$ is invertible over the ring of formal power series (see [13]). Here $D = \sum_{i=0}^{\infty}(A)^i$ is a matrix of formal power series in x and in the (s, t)-entry, the coefficient of the i-th terms gives the number of walks from s to t of weight i.

As w isolates the minimal weight paths in G, it is enough to compute these coefficients modulo 2 for all i up to some polynomial in n since there is a unique path with the minimal weight if one exists. So, it is enough to compute and update the inverse of the matrix $I - A$. Though to do it effectively, we compute the *n-approximation* C of D, which is a matrix of formal polynomials that agrees with the entries of D up to degree $i \leq n$ terms. This precision is preserved by the matrix operations we use, see [13, Proposition 14].

When applying a change ΔG to G that affects k nodes, the associated matrix A is updated by adding a suitable change matrix ΔA with at most k non-zero rows and columns, and can therefore be decomposed into a product UBV of suitable matrices U, B, and V, where B is a $k \times k$ matrix. To update the inverse, we employ the Sherman-Morrison-Woodbury identity (cf. [19]), which gives a way to update the inverse when A is changed to $A + \Delta A$ as follows:

$$(A + \Delta A)^{-1} = (A + UBV)^{-1} = A^{-1} - A^{-1}U(I + BVA^{-1}U)^{-1}BVA^{-1}.$$

[3] Intuitively, bounded expansion first-order (bfo) reductions are first-order reductions such that each tuple in a relation and each constant of the input structure affects at most a constant number of tuples and constants in the output structure. A bfo-tt reduction bears the same relation to a bfo-reduction as a truth-table reduction bears to a many-one reduction. For a formal definition see [8, Section 3.2] where it was first formalized.

The right-hand side can be computed in $\mathsf{FO}[\oplus](\leq, +, \times)$ for $k = \log^{O(1)} n$ since modulo 2 computation of (1) multiplication and iterated addition of polynomials over $\mathbb{Z}$ and (2) computation of the inverse of $I + BVA^{-1}U$ which is also a $k \times k$ matrix is possible in $\mathsf{FO}[\oplus](\leq, +, \times)$ for (matrices of) polynomials with polynomial degree [18]. Finally, we need to assign weights to the changed edges as well so that the resulting weight assignment remains isolating. We show how to achieve this starting with non-zero circulation weights. Using [11, Theorem 5] we can assume that such a weight assignment is given, and that we only need to update the weights once.

Let u be skew-symmetric non-zero circulation weights for G and let n^k be the polynomial bound on the weights. Further, let w be the isolating weight assignment that gives weight $n^{k+2} + u(e)$ to each edge $e \in E$. During the AC^d initialization, we compute the weights u and w and an n^b-approximation matrix C of $(I - A_{(G,w)}(x))^{-1}$ mod 2 for some constant b.

When changing G via a change ΔE with deletions E^- and insertions E^+, the algorithm proceeds as follows: To compute the isolating weights w^-, the non-zero circulation weights u^- for G^- are obtained from u by setting the weight of deleted edges $e \in E^-$ to 0. As u^- gives the same weight to all simple cycles in G^- as u gives to these cycles in G, it has non-zero circulation. To handle E^+ it can be shown that [11, Lemma 11] there is a FO-computable (from w, E^+ and the reachability information in G) family W' of polynomially many weight assignments such that $\exists w' \in W'$ isolating for $(V, E \setminus E^- \cup E^+)$.

Hence we need to maintain polynomially many different instances of the graph with different weight functions from W' such that in at least one of them the paths are isolated. The idea is that if there is an s-t-path using at least one inserted edge from E^+, then there is a unique minimal path among all s-t-paths that use at least one such edge, while ignoring the weight of the paths that is contributed by edges from E. The edge weights from E^+ are multiplied by a large polynomial to ensure that the combined weight assignment with the existing weights for edges in E remains isolating. Since the weights are constructed only for a graph with $N = \log^{O(1)} n$ many nodes, and although they are not polynomially bounded in N, they *are* in n. Please refer to [11, Section 6] for more details.

From the above discussion, to prove a similar bound for distances, it suffices to show that (1) after every polylog(n) changes, we can ensure the edge weights remain "shortest path-isolating" and (2) under such weights the distance can be updated in $\mathsf{FO}[\oplus](\leq, +, \times)$.

6.2 Dynamic Isolation of Shortest Paths

In the following, we first describe how the isolating weights for reachability can be modified to give weights for isolating shortest paths. Similar to maintaining reachability, our algorithm handles deletions and insertions differently. In case of deletion, we set the weight of the deleted edges $e \in E^-$ to 0 and due to the non-zero circulation weights, the weights remain isolating. For insertions, the idea is to do a *weight refinement* by shifting the original edge weights $w(e)$ (1 in case of unweighted graphs) to the highest order bits in the bit-representation, in the presence of other newly assigned weights to the edges.

We define a new weight functions $w^* = \langle w, w', u \rangle$ and assign these weights to the inserted edges $e \in E^+$. The existing edges E are not assigned any w' weight and all those bits remain zeroes. So we get a family of weight functions W^*. Here w is the polynomially-bounded original edge weights, w' is one of the (polynomially many) isolating weights from the family W' assigned to the newly added edges E^+ during the dynamic process, and u are the non-zero circulation weights that are computed statically. The combined weights w^* is FO-constructible from the weights w, w' and u as all involving numbers are $O(\log n)$ bits long (see [21, Theorem 5.1]). The correctness of the fact that these weights are indeed shortest

path isolating follows from [11, Lemma 11] with the observation that since the original edge weights are shifted to the highest-order bits, the minimum weight path with these combined weights corresponds to the shortest path in the original graph.

The update algorithm for maintaining reachability can be extended to maintaining distances also [13]. Here, instead of checking only the non-zeroness of the (s,t)-entry in the polynomial matrix C, we compute the minimum degree term as well (with coefficient 1), which can be done in $\mathsf{FO}[\oplus](\leq,+,\times)$. By construction, the degree of a term in this polynomial is same as the weight of the corresponding walk under the dynamic isolating weights and applying an easy transformation gives us back the original weights, that is, the weight of the shortest path from s to t in G'. This proves the distance part of Theorem 1.

7 Maintaining Rank under Bulk Changes

In this section we prove Theorem 2. For ease of exposition, we build upon the algorithm as described in [7, Section 3.1]. Before going into the details of the proof, we start with defining some important notation, followed by our overall proof strategy. Let A be a $n \times n$ matrix over $\mathbb{Z}_p$, where $p = O(n^3)$ is a prime. Let K be the kernel of A. For a vector $v \in \mathbb{Z}_p^n$, we define $S(v) = \{i \in n \mid (Av)_i \neq 0\}$, where $(Av)_i$ denotes the ith coordinate of the vector Av. Let B be a basis of Z_p^n. A vector $v \in B$ is called *i-unique* with respect to A and B if $(Av)_i \neq 0$ and $(Aw)_i = 0$ for all other $w \in B$. A basis B is called *A-good* if all the vectors in $B - K$ are i-unique with respect to A and B. For a vector $v \in B - K$, the minimum i for which it is i-unique is called the principal component of v, denoted as $\mathrm{pc}(v)$.

Starting with an A-good basis B, and introducing a small number of changes to yield A' may lead to B losing its A-goodness. To restore this, we alter the matrix B in two phases to obtain an A'-good basis B'. The first phase involves identifying a full rank submatrix in A' corresponding to the changed entries, inverting it, and restoring the pc's of the columns of that full rank submatrix. In the second phase, we restore the pc's of the rest of vectors, which had lost the pc's either because of the changes or during Phase 1. The rough outline is similar to that of [7] but in order to handle non-constant changes we have to make non-trivial alterations and use efficient small matrix inversion from [13]. We have Theorem 2 from the following lemma, whose proof is provided in Section 7.1:

▶ **Lemma 15.** *Let $A, A' \in \mathbb{Z}_p^{n\times n}$ be two matrices such that A' differs from A in $O(\frac{\log n}{\log\log n})$ places. If B is an A-good basis then we can compute an A'-good basis B' in $\mathsf{FO}(\leq,+,\times)$.*

▶ **Proposition 16** ([8])**.** *Let $A \in \mathbb{Z}_p^{n\times n}$ and B an A-good basis of $\mathbb{Z}_p^n$. Then $rank(A) = n - |B \cap K|$ is the number of vectors in the basis that have a pc.*

▷ Claim 17. If the rank of an $n \times n$ matrix A is r then there exists a prime $p = O(max(n, \log N)^3)$ such that the rank of A over $\mathbb{Z}_p$ is also r, where N is the maximum absolute value the entries of the matrix A contain.

Proof. We know that if the rank of A is r then there exists a $r \times r$ submatrix A_s of A such that its determinant is nonzero. The value of this determinant is at most $n!N^n$, which can be represented by $O(n(\log n + \log N))$ many bits. Therefore, this determinant is divisible by at most $O(n(\log n + \log N))$ many primes. Thus by the prime number theorem, we can say that for a large enough n there exists a prime p of magnitude $O(max(n, \log N)^3)$ such that determinant of A_s is not divisible by p. ◁

Hence, to compute the rank of A, it is sufficient to compute the rank of the matrices (A mod p) for all primes p of size $O(max(n, \log N)^3)$ and take the maximum among them. Below we show how to maintain the rank of the matrix A mod p for a fixed prime p. We replicate the same procedure for all the primes in parallel.

A' is the matrix that is obtained by changing k many entries of A. Notice that if B is not A'-good basis, that means there are some vectors in $B - K'$ which are not i-unique with respect to B and A', where K' is the kernel of A'. A vector $w \in B - K'$ which was i-unique ($i \in [n]$) with respect to A and B may no longer be i-unique with respect to A' and B for the following two reasons, (i) $i \notin S'(w)$, (ii) there may be more than one vector w' such that $i \in S'(w')$. For a vector v, $S'(v)$ denotes the set of non-zero coordinates of the vector $A'v$. Below we give an AC^0 algorithm to construct an A'-good basis.

In several places we make use of the fact that sum and product of polylog(n) many numbers each with of polylog(n) bits can be computed in AC^0 [21, Theorem 5.1].

7.1 Construction of an A'-good basis

Let $k = O(\frac{\log n}{\log \log n})$ and $M = (A'B)_{R,C}$ be the $n \times n$ matrix where R is the set of rows and C is the set of columns of M. We know that $A'B$ differs from AB in a set R_0 of at most k many rows. Let $M_{R_0,*}$ be the matrix M restricted to the rows in the set R_0.

▷ Claim 18. There exists a prime $q = O(\log^3 n)$ such that the rank of $(M_{R_0,*} \bmod q)$ is equal to the rank of $(M_{R_0,*} \bmod p)$.

Proof. The proof follows from the proof of Claim 17. ◁

From the above claim, it follows that a row basis of $(M_{R_0,*} \bmod p)$ remains a row basis of $(M_{R_0,*} \bmod q)$ for some $O(\log \log n)$-bit prime q. Next, we have two constructive claims:

▷ Claim 19. A row basis R_1 of $(M_{R_0,*} \bmod q)$ can be found in AC^0.

Proof. Note that number of rows in R_0 are $O(\frac{\log n}{\log \log n})$; thus, the number of the subsets of the rows of R_1 are polynomially many (in n), and each row in the set R_0 can be indexed by $O(\log \log n)$ many bits. An element of $\mathbb{Z}_q$ can also be represented by $O(\log \log n)$ many bits. Therefore, for a fixed subset S of R_0, all the linear combinations of the rows of S can be represented by $O(\log n)$ many bits. We try all the linear combinations in parallel. Also, we do this for all the subsets in parallel. The subset with the maximum cardinality in which all the linear combinations result in non-zero values will be the maximum set of linearly independent rows in $(M_{R_0,*} \bmod q)$. ◁

▷ Claim 20. A column basis C_1 of $(M_{R_1,*} \bmod q)$ can be found in AC^0.

Proof. To find the maximum set of linearly independent columns in the matrix $M_{R_1,*}$ we just check in parallel if the rank of $M_{R_1,i}$ is greater than the rank of $M_{R_1,i-1}$ for all $i \in [m]$. Let $c_1, c_2 \dots c_m$ be the columns in the matrix $M_{R_1,*}$. Note that the set of columns c_i such that the rank of $M_{R_1,i}$ is more than the rank of $M_{R_1,i-1}$, form a maximum set of linearly independent columns in $M_{R_1,i}$. We can check this in AC^0. ◁

We are going to construct four matrices $D^{(1)}, E^{(1)}, D^{(2)}, E^{(2)}$ successively such that the product $B' = B \times D^{(1)} \times E^{(1)} \times D^{(2)} \times E^{(2)}$ is an A'-good basis. For this, we need to show that each column c_i of $A'B'$ is either an all zero-column or there exists a unique j such that the j-th entry of the column is non-zero. In other words, each column of B' is either i-unique or it is in the kernel of A'. We will show how to obtain each of the four matrices above as well as take their product in AC^0. We need a technical lemma before we start.

Combining Matrices. Here we state a lemma about constructing matrices from smaller matrices that we we will use several times. Let $X \in \mathbb{Z}_p^{n\times n}$ be a matrix and let $X^{1,1} \in \mathbb{Z}_p^{\ell\times\ell}$, $X^{1,2} \in \mathbb{Z}_p^{\ell\times(n-\ell)}$, $X^{2,1} \in \mathbb{Z}_p^{(n-\ell)\times\ell}$, $X^{2,2} \in \mathbb{Z}_p^{(n-\ell)\times(n-\ell)}$ be 4 matrices and let $R, C \subseteq [n]$ be two subsets of indices of cardinality ℓ each. Let $\bar{R} = [n] \setminus R, \bar{C} = [n] \setminus C$. Then we have:

▶ **Lemma 21.** *Given the matrices $X^{i,j}$ for $I, J \in [2]$ and the sets R, C explicitly for $|R| = |C| = \ell = (\log n)^{O(1)}$, we can construct, in AC^0, the matrix Y such that $Y_{R,C} = X^{1,1}$, $Y_{R,\bar{C}} = X^{1,2}$, $Y_{\bar{R},C} = X^{2,1}$ and $Y_{\bar{R},\bar{C}} = X^{2,2}$.*

Proof. Notice that the sets R, C can be sorted in AC^0 because computing the position $pos_R(r)$ of an element $r \in R$ (i.e., the number of elements not larger than r) is equivalent to finding the sum of at most ℓ bits (which are zero for elements of R larger than r and one otherwise).

The position $pos_{\bar{R}}(r')$ of an element $r' \in \bar{R}$ (i.e. the number of elements of $\bar{R}$ not larger than r') can also be found in AC^0. This is because we can first find the set $R(r') = \{r_i \in R : r_i < r'\}$ in AC^0. Then $pos_{\bar{R}}(r') = r' - |R(r')|$ because there are r' rows with indices at most r' and out of these all but $|R(r')|$ are in $\bar{R}$ and thus can be computed in AC^0. We can similarly compute $pos_C(c), pos_{\bar{C}}(c')$ for $c \in C$ and $c' \in \bar{C}$.

Finally given $i, j \in [n]$ the element $Y_{i,j}$ is $X^{1,1}_{pos_R(i),pos_C(j)}$ if $i \in R, j \in C$. Similarly if $i \in \bar{R}, j \in C$ then it is $X^{2,1}_{pos_{\bar{R}}(i),pos_C(j)}$, if $i \in R, j \in \bar{C}$ then it is $X^{1,2}_{pos_R(i),pos_{\bar{C}}(j)}$ and if $i \in \bar{R}, j \in \bar{C}$ then it is $X^{2,2}_{pos_{\bar{R}}(i),pos_{\bar{C}}(j)}$. This completes the proof. ◀

Phase 1. First, we restore the i-uniqueness of the columns indexed by the set C_1. Let R_{C_1} be the set of rows in R indexed by the same set of indices as C_1 in C. We right multiply M with another matrix $D^{(1)} \in \mathbb{Z}_p^{n\times n}$ such that $D^{(1)}_{R_{C_1},C_1}$ is the inverse of M_{R_1,C_1} and $D^{(1)}_{R-R_{C_1},C-C_1}$ is the identity matrix and all the other entries of $D^{(1)}$ are zero. Since the inverse of a $k \times k$ matrix can be computed in AC^0 [11], matrix $D^{(1)}$ can be obtained in AC^0 via Lemma 21.

Let $M^{(1)} = M \times D^{(1)}$, note that ${M^{(1)}}_{R_1,C_1}$ is an identity matrix. Since $M = A' \times B$, we have $M^{(1)} = M \times D^{(1)} = A' \times B \times D^{(1)}$. Note that since ${M^{(1)}}_{R_1,C_1}$ is an identity matrix, the vectors corresponding to the columns in C_1 in the matrix $B \times D^{(1)}$ can now easily be made i-unique. Since ${M^{(1)}}_{R_1,C_1}$ is an identity matrix, all columns in the matrix ${M^{(1)}}_{R_1,C-C_1}$ can be written as the linear combinations of columns of ${M^{(1)}}_{R_1,C_1}$. Let $\tilde{M}^{(1)}$ be a matrix defined as $\tilde{M}^{(1)} = M^{(1)} \times E^{(1)}$, where $E^{(1)} \in \mathbb{Z}_p^{n\times n}$ is constructed as follows. (i) $E^{(1)}_{R_{C_1},*}$ is same as $M^{(1)}_{R_1,*}$. (ii) $E^{(1)}_{R-R_{C_1},C_1}$ is the zero matrix. (iii) $E^{(1)}_{R-R_{C_1},C-C_1}$ is the negative identity matrix. Using Lemma 21, we can construct $E^{(1)}$ in AC^0. Note that $\tilde{M}^{(1)}_{R_1,C_1}$ is an identity matrix and $\tilde{M}^{(1)}_{R_1,C-C_1}$ is a zero matrix. Thus we can say that vectors corresponding to columns in C_1 in the matrix $B \times D^{(1)} \times E^{(1)}$ are i-unique for some $i \in R_1$.

Next, we perform a procedure similar to Phase 1 for those vectors which lost their pc's when we changed the matrix from A to A', i.e. those vectors w which were i-unique for some i, but $i \notin S'(w)$.

Phase 2. There can be at most k vectors which lost their pc's while changing the matrix from A to A'. Some of these vectors might get their pc's set in Phase 1. Let $\tilde{C}_2$ be the remaining set of vectors in B. Notice $C_1 \cap \tilde{C}_2$ is empty. To set the pc's of these vectors, we repeat the above procedure for the matrix $\tilde{M}^{(1)}_{*,\tilde{C}_2}$ as follows.

Find the column basis C_2 of $\tilde{M}^{(1)}_{*,\tilde{C}_2}$ in AC^0 recalling that $|\tilde{C}_2| \leq k$ and using Claim 19 on the transpose of $\tilde{M}^{(1)}_{*,\tilde{C}_2}$. By considering the transpose of $\tilde{M}^{(1)}_{*,C_2}$ and applying Claim 20 we can get a row basis R_2 of $\tilde{M}^{(1)}_{*,C_2}$. Notice that $R_1 \cap R_2$ is empty. We construct a matrix $D^{(2)} \in \mathbb{Z}_p^{n\times n}$ in AC^0 using Lemma 21 such that $D^{(2)}_{R_{C_2},C_2}$ contains the inverse of $\tilde{M}^{(1)}_{R_2,C_2}$, $D^{(2)}_{R-R_{C_2},C-C_2}$ is an identity matrix and the rest of the entries of $D^{(2)}$ are zero. Here R_{C_2} is the set of rows indexed by the same indices as in the set C_2 of columns. Let $\tilde{M}^{(2)} = \tilde{M}^{(1)} \times D^{(2)} \times E^{(2)}$, where $E^{(2)}$ is constructed in AC^0 (using Lemma 21) so that: (i) $\tilde{M}^{(2)}_{R_1\cup R_2,C_1\cup C_2}$ is the identity matrix. (ii) $\tilde{M}^{(2)}_{R_1,C-C_1}$ and $\tilde{M}^{(2)}_{R-R_1,\tilde{C}_2-C_2}$ are zero matrices. Finally, we have $B' = B \times D^{(1)} \times E^{(1)} \times D^{(2)} \times E^{(2)}$. Since each column of the newly constructed matrices contains at most $(k+1)$ non-zero entries, we can obtain B' in AC^0.

▷ Claim 22. B' is an A'-good basis.

Proof. First, we prove that the vectors which lost their pc's, either get new pc's or they are modified to be in the kernel of A'. Let $w \in B$ be a vector that lost its pc and it is not captured in both Phase 1 or Phase 2. Assume it is not captured in Phase 1, i.e. the vector $A'w$ does not belong to the column set indexed by C_1. Then it will be captured in Phase 2. If it is not captured in Phase 2 as well, then we can say that $A'w$ does not belong to the columns indexed by the set C_2. Therefore it can be written as a linear combination of the vectors in C_2. In Phase 2, we modify such vectors in a way that $A'w$ becomes a zero vector, i.e. w goes into the kernel of A'. Also, note that the vector which did not lose their pc's and are not captured in Phase 1 and Phase 2, do not lose their pc's in the procedure. ◁

We prove that we can maintain the number of pc's in B in AC^0 using the next claim. However, we need to set up some notation first. Let $P^{\text{old}}, P^{\text{new}}$ be respectively, the number of pc's before and after the phases. Let $V^{\text{new}}_{R_1}$ and $V^{\text{new}}_{R_2}$ denote the set of vectors that have their pc's in the rows R_1 and R_2, after the phases. Let $V^{\text{old}}_{R_0}$ denote the set of vectors that have their pc's in the rows R_0 before starting of Phase 1 and V_1 denotes the set of vectors which have their pc's in the rows $R - R_0$ before the Phase 1 and attain pc's in the rows R_1 after Phase 2. Note that all the cardinalities of all the sets of vectors mentioned above are $O(\frac{\log n}{\log\log n})$. Therefore, we can compute their cardinalities in AC^0.

▷ Claim 23. $P^{\text{new}} = P^{\text{old}} - |V^{\text{old}}_{R_0}| + |V^{\text{new}}_{R_1}| + |V^{\text{new}}_{R_2}| - |V_1|$.

Proof. First, we assume that all the vectors in the set $V^{\text{old}}_{R_0}$ lose their pc's after the phases therefore we subtract $|V^{\text{old}}_{R_0}|$ from P. But some of these vectors get their pc's in Phase 1 and Phase 2. Therefore, we add $|V^{\text{new}}_{R_1}|$ and $|V^{\text{new}}_{R_2}|$ back to the sum. Notice that $V^{\text{new}}_{R_1}$ may contain those vectors as well that had their pc's in the rows indexed by $R - R_0$ before Phase 1. That means these vectors had a pc before and after the two phases, but we added their number $|V^{\text{new}}_{R_1}|$. Therefore, we subtract the number of such vectors by subtracting $|V_1|$ from the total sum. ◁

This brings us to the proof of Lemma 15.

Proof of Lemma 15. The proof is complete from the above claims because the number of pc's is precisely the rank of the matrix as a consequence of Proposition 16. ◀

8 Conclusion

In this work, we prove two meta-theorems for distance and maximum matching, which provide the best known dynamic bounds in graphs where non-zero circulation weights can be computed in parallel. This includes important graph classes like planar, bounded genus, bounded treewidth graphs. We show how to non-trivially modify two known techniques and combine them with existing tools to yield the best known dynamic bounds for more general classes of graphs, and at the same time allow for bulk updates of larger cardinality. While for bipartite matching we are able to show a $\mathsf{DynFO}(\leq, +, \times)$ bound it would be interesting to achieve this also for maintaining distances, even in planar graphs under single edge changes.

References

1 Eric Allender, Robert Beals, and Mitsunori Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Comput. Complex.*, 8(2):99–126, 1999.

2 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

3 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1):1–17, 2009. doi:10.1145/1490270.1490274.

4 Samir Datta, Chetan Gupta, Rahul Jain, Anish Mukherjee, Vimal Raj Sharma, and Raghunath Tewari. Reachability and matching in single crossing minor free graphs. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

5 Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 356–367. Springer, 2014.

6 Samir Datta, Raghav Kulkarni, Ashish Kumar, and Anish Mukherjee. Planar maximum matching: Towards a parallel algorithm. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 21:1–21:13, 2018.

7 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2015.

8 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018.

9 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory Comput. Syst.*, 47(3):737–757, 2010.

10 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N.V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *Journal of Computer and System Sciences*, 78(3):765–779, 2012. In Commemoration of Amir Pnueli. doi:10.1016/j.jcss.2011.11.002.

11 Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 122:1–122:19, 2020.

12 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *Log. Methods Comput. Sci.*, 15(2), 2019.

13 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 120:1–120:14, 2018.

14 Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995.

15 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in Quasi-NC. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763, 2016.

16 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 165–169, 1982.

17 D. A. Harville. *Matrix Algebra From a Statistician's Perspective*, volume 8 of *Cambridge international series on parallel computation*. New York: Springer- Verlag, 2008.

18 Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, pages 672–683, 2006.

19 Harold V Henderson and Shayle R Searle. On deriving the inverse of a sum of matrices. *Siam Review*, 23(1):53–60, 1981.

20 William Hesse. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.*, 296(3):473–485, 2003.

21 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002.

22 Thanh Minh Hoang. On the matching problem for special graph classes. In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity, CCC 2010, Cambridge, Massachusetts, USA, June 9-12, 2010*, pages 139–150. IEEE Computer Society, 2010.

23 Neil Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989.

24 Vivek Anand T. Kallampally and Raghunath Tewari. Trading determinism for time in space bounded computations. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 10:1–10:13, 2016.

25 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

26 Anish Mukherjee. *Static and Dynamic Complexity of Reachability, Matching and Related Problems*. PhD thesis, CMI, 2019.

27 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 345–354, 1987.

28 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997.

29 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

30 Wikipedia contributors. Matrix determinant lemma — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Matrix_determinant_lemma`, 2021. [Online; accessed 30-September-2021].

Circuit Extraction for ZX-Diagrams Can Be #P-Hard

Niel de Beaudrap
University of Sussex, UK

Aleks Kissinger
University of Oxford, UK

John van de Wetering
Radboud University Nijmegen, The Netherlands
University of Oxford, UK

Abstract

The ZX-calculus is a graphical language for reasoning about quantum computation using ZX-diagrams, a certain flexible generalisation of quantum circuits that can be used to represent linear maps from m to n qubits for any $m, n \geq 0$. Some applications for the ZX-calculus, such as quantum circuit optimisation and synthesis, rely on being able to efficiently translate a ZX-diagram back into a quantum circuit of comparable size. While several sufficient conditions are known for describing families of ZX-diagrams that can be efficiently transformed back into circuits, it has previously been conjectured that the general problem of *circuit extraction* is hard. That is, that it should not be possible to efficiently convert an arbitrary ZX-diagram describing a unitary linear map into an equivalent quantum circuit. In this paper we prove this conjecture by showing that the circuit extraction problem is **#P**-hard, and so is itself at least as hard as strong simulation of quantum circuits. In addition to our main hardness result, which relies specifically on the circuit representation, we give a representation-agnostic hardness result. Namely, we show that any oracle that takes as input a ZX-diagram description of a unitary and produces samples of the output of the associated quantum computation enables efficient probabilistic solutions to NP-complete problems.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases ZX-calculus, circuit extraction, quantum circuits, #P

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.119

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Previous Version*: https://arxiv.org/abs/2202.09194

Funding *Aleks Kissinger*: Acknowledges support from AFOSR grant FA2386-18-1-4028.
John van de Wetering: Acknowledges support from an NWO Rubicon Personal Grant.

Acknowledgements The authors wish to thank the anonymous reviewers for their valuable suggestions regarding the presentation of this paper.

1 Introduction

Quantum circuit notation is widely used in the field of quantum computing to denote computations to be executed on a quantum computer. While quantum circuits are a useful tool for representing computations on a quantum computer, they are somewhat inconvenient for reasoning about computations (such as proving equalities or doing simplifications); and for representing computations in alternative models like the one-way model of measurement-based quantum computation (MBQC) [40], or surface code lattice surgery [31].

ZX-diagrams are an alternative, more general representation of quantum computations, which allow complex operations to be described using a few simple generating operators. ZX-diagrams come with an equational theory, called the *ZX-calculus* [11], which allows one to perform many useful calculations graphically, without resorting to concrete matrix computations. While ZX-diagrams can be seen as an extension of circuits [12], they also readily admit encodings of the one-way model [23] and lattice surgery [20], and allow one to reason more easily about such procedures. There are several known *complete* axiomatisations of the ZX-calculus [37, 49], where any true equality of linear maps can be proved graphically. For a review on the ZX-calculus we refer to [47].

The ZX-calculus has been used in a variety of areas. It was used to optimise T-count [34, 18], braided circuits [29] and MBQC [4]; to find a new normal form for Clifford circuits [22]; to do more effective classical simulation using stabiliser decompositions [35]; and to reason about surface codes [26, 27], mixed-state quantum computations [9], natural language processing [10], condensed matter systems [14], counting problems [21, 44] and spin-networks [24].

As a strict extension of quantum circuit language, ZX-diagrams may express operations in a form that do not correspond directly to a quantum circuit. This added flexibility makes it easier to find novel strategies to simplify quantum circuits, but it comes at a cost: given a ZX-diagram representing a unitary linear map, it might be non-trivial to transform it back into a circuit of comparable size. Such a translation might however be necessary if, for instance, we want to run the computation described by a ZX-diagram on a gate-based quantum computer.

We refer to the above problem, as the *circuit extraction* problem: given a ZX-diagram which denotes a unitary operator U, find a unitary circuit (*i.e.*, a quantum circuit without measurements) that implements U. In recent years, some progress has been made on this problem [22, 34, 4, 41, 33, 19]. However, all known methods for efficient extraction of circuits from ZX-diagrams rely on additional conditions, in particular requiring there to be some kind of *flow* on the diagram, a concept imported from MBQC [6]. Such conditions allow the diagram to be rewritten incrementally into a unitary circuit. Since many ZX-calculus rewrites preserve these conditions, it is possible to perform optimisation of quantum circuits using ZX-calculus rules and still recover circuits efficiently.

However, it is worth trying to generalise these conditions as much as possible, or even remove them. For instance, it was noted in [34] that a certain transformation of ZX-diagrams would decrease the T-count (an important metric for quantum circuit optimisation), but in the process broke the invariant (the existence of a gflow), preventing a circuit from being extracted efficiently using known techniques. Given all this it is then natural to wonder about the following question:

Is there some efficient procedure to translate any unitary ZX-diagram into a quantum circuit?

In this paper we present strong evidence that there is no such efficient procedure, by showing that the circuit extraction problem is **#P**-hard in the worst case. The complexity class **#P** contains for instance the problem of strong simulation of quantum circuits, and counting the number of satisfying solutions to a Boolean formula, so **#P**-hard problems are expected to be intractable. We prove **#P**-hardness by giving an encoding of Boolean formulae into unitary ZX-diagrams in such a way that extracting a polysize circuit provides a solution to the associated **#SAT** instance. A consequence of our result is that if there were a polynomial time algorithm for circuit extraction, then $\mathbf{P} = \mathbf{NP}$.

Alternatively, since there is an evident translation from a ZX-diagram into a quantum circuit with postselection, this result can equivalently be seen as expressing the hardness of translating a postselected circuit that is promised to be proportional to a unitary into a

circuit without postselection. While intuitively this seems likely to be hard, particularly in light of Aaronson's landmark result that **PostBQP** = **PP** [1], our hardness result seems to be quite different in nature due to the unitarity promise. In particular, the postselection does not seem to be the "source of power" in our proof: the measurement outcomes corresponding to the post-selections in our circuits occur with some bounded probability, independent of the problem size.

One could ask how much our hardness result is tied to the fact that we require a procedure that produces quantum circuits from ZX-diagrams. Especially, when considering that in most cases we are not interested in the circuit itself, but instead we simply want to sample the output of the quantum computation. Perhaps one could find some other procedure to "program" a quantum computer using a ZX-diagram describing a unitary and obtain samples of measurement outcomes. We show that an efficient such procedure is unlikely to exist for arbitrary ZX-diagrams, by finding that such a procedure allows you to probabilistically solve **NP**-hard problems. So if there were some way to generically translate unitary ZX-diagrams into procedures which could be realised in polynomial time on a quantum computer, it would follow that the entire polynomial hierarchy is in **BQP**, and in particular that **NP** $\subseteq$ **BQP**.

The paper is structured as follows. We start by covering preliminaries on quantum circuits, ZX-diagrams and the necessary complexity theory in Section 2. Then in Section 3 we formally define the circuit extraction problem and prove it is hard. Section 4 considers several variations on circuit extraction, and in Section 5 we find some upper bounds on the hardness of circuit extraction. We end with some concluding remarks in Section 6.

2 Preliminaries

2.1 Quantum circuits

Since we wish to extract "a circuit" from a ZX-diagram, it will be helpful to first consider what we actually mean by a circuit.

In quantum computational theory, a "circuit" is a description of a computational process consisting of operations which may be decomposed as a sequence of primitive "gates" and "measurements", which act on one or more qubits to change the states of those qubits. The state-space of a qubit is identified with unit vectors of the finite-dimensional Hilbert space $\mathcal{H}_2 \cong \mathbb{C}^2$; the state of k qubits in parallel is described by the tensor product $\mathcal{H}_2^{\otimes k}$. A "gate" is an operation which is applied to one or more qubits and implements a unitary transformation $U : \mathcal{H}_2^{\otimes k} \to \mathcal{H}_2^{\otimes k}$ on the associated state space. A "measurement" is an operation which transforms a state $|\psi\rangle \in \mathcal{H}_2^{\otimes k}$ to some state $p_j^{-1/2}\,\Pi_j|\psi\rangle$ where $\{\Pi_1, \Pi_2, \ldots\}$ is a set of projections that sum up to the identity operator I, the p_j gives the probability of observing that particular measurement outcome and is given by $p_j = \langle\psi|\Pi_j|\psi\rangle$, and the index j provides the classical "outcome" indicating which transformation occurred. A gate or measurement acting on a small number of qubits can be applied to a larger set of qubits by taking the tensor product with an appropriate number of identity operators. A "circuit" is then a composition of such gates and measurements on some number of qubits, acting in sequence or in parallel, to describe more complex (and in general, non-deterministic and irreversible) transformations of a quantum state-space. To define a reasonable model of computational complexity using quantum circuits, one usually elaborates the above with a description of how one would specify a circuit as part of a family of unitary operators, acting on inputs of various sizes. For our purposes, it will suffice to require that the coefficients of the gates be efficiently computable, and in particular provided explicitly in some representation which suffices to approximate them to $O(\text{poly}(n))$ bits of precision in time $O(\text{poly}(n))$ for an n qubit circuit.

It will be convenient to refer to one specific such gate-set – an infinite set $\mathcal{B}$ of gates, consisting of the single-qubit gates Z_α for arbitrary angles α, the single-qubit Hadamard gate H and the two-qubit gate CNOT:

$$Z_\alpha = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1}$$

This gate set forms a *universal* gate set, meaning that a unitary acting on any number of qubits can be written as a circuit consisting of these gates [38]. Other universal gate-sets exist, but so long as one considers gate sets whose parameters are efficiently computable from some input parameters and which act only on a bounded numbers of qubits (*e.g.*, at most two or three qubits), the size of a circuit to represent a given unitary operator can only vary by a constant factor, so that for the purpose of complexity theory, the details of the specific gate set chosen are not important.

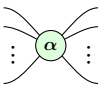
A circuit which contains no measurements, and therefore consists entirely of unitary gates, is called a "unitary circuit". A unitary circuit is reversible, and "deterministic" in the sense that an idealised realisation of such a circuit will transform the state-space in the same way each time. As this is a convenient feature for the design and analysis of quantum algorithms, much of the literature on quantum algorithms concerns itself with unitary circuits, and much of the design of quantum computers is concerned with how to reliably implement unitary circuits.

2.2 ZX-diagrams

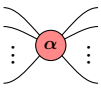
We provide a brief overview of ZX-diagrams. For a review see [47], and for a book-length introduction see Ref. [13].

ZX-diagrams form a diagrammatic language similar to the familiar quantum circuit notation. A *ZX-diagram* (or simply *diagram*) consists of *wires* and *spiders*. Wires entering the diagram from the left are *inputs*; wires exiting to the right are *outputs*. Given two diagrams we can compose them by joining the outputs of the first to the inputs of the second, or form their tensor product by simply stacking the two diagrams [11, 12].

Spiders are linear operations which can have any number of input or output wires. There are two varieties: Z-spiders depicted as green dots and X-spiders depicted as red dots:



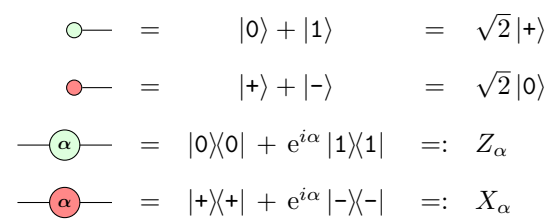
$$:= |0\cdots0\rangle\langle 0\cdots0| + e^{i\alpha}|1\cdots1\rangle\langle 1\cdots1|$$



$$:= |+\cdots+\rangle\langle +\cdots+| + e^{i\alpha}|-\cdots-\rangle\langle -\cdots-| \tag{2}$$

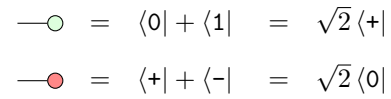
Here $|0\rangle$ and $|1\rangle$ represent the standard basis vectors of $\mathbb{C}^2$ which are the eigenvectors of the Pauli Z matrix; the states $|\pm\rangle := \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$ are sometimes referred to as the "Hadamard basis", and are eigenvectors of the Pauli X matrix. If you are reading this document in monochrome or otherwise have difficulty distinguishing green and red, Z spiders will appear lightly-shaded and X darkly-shaded. Note that here the number of inputs and outputs do not have to match. When $\alpha = 0$, we will not write the phase on the spider.

▶ **Example 2.1.** We can immediately write down some simple state preparations and unitaries in the ZX-calculus:



(3)

We can also represent the effects that are dual to the states above using spiders:



(4)

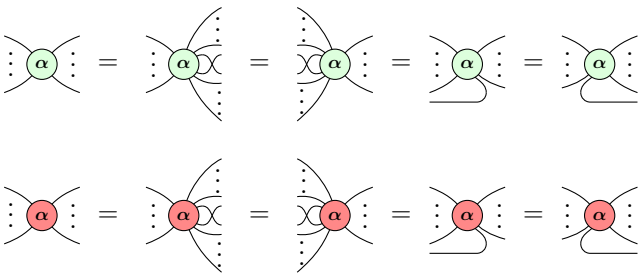
In the diagrams above we write explicit scalars to represent a proportionality constant. In this paper (non-zero) scalar factors will not be important. However, do note it is always possible to represent any scalar as an explicit ZX-diagram (of constant size). For this reason, our results will also apply to other proposed normalisations of the ZX generators, such as those in Refs. [17, 14, 24].

We can compose ZX-diagrams in two ways, either horizontally by connecting the wires together, which corresponds to the regular composition of linear maps, or vertically, which corresponds to the tensor product of linear maps. Any ZX-diagram is built by composing spiders (and permutations of wires) together in these ways.

On a formal level we consider a ZX-diagram D with n inputs and m outputs as a morphism $D : n \to m$ in a category. This category is the compact-closed *PROP* (symmetric monoidal category where the objects correspond to natural numbers and the tensor is addition) freely generated by the Z- and X-spider generators. The interpretation as a linear map is then a strongly monoidal functor into the category of Hilbert spaces, which is fully specified by the interpretation of the spiders (2). This level of formality won't be needed in this paper. The interested reader can look at for instance Refs. [11, 9, 8].

A more intuitive way to view ZX-diagrams is as tensor networks [39]: the spiders are the tensors, and a connection between spiders denotes a contracted index.

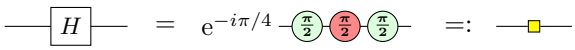
The Z- and X-spiders satisfy the following symmetries:



(5)

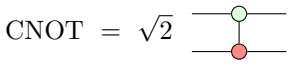
Here we are writing an equality of ZX-diagrams. This is to be understood as saying that the linear maps these ZX-diagrams represent are equal. In particular: in each equality the number of open wires extending to the left is the same, and similarly for the number of open wires extending to the right. Because of these symmetries we can treat ZX-diagrams as (labeled) undirected graphs: if we attach labeled nodes to the open wires at the input and output to distinguish their roles, arbitrary topological deformations of the diagram do not affect its interpretation as a linear map.

It is often convenient to introduce a symbol – a yellow square – for the Hadamard gate. This is defined by the equation:



(6)

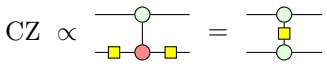
The CNOT gate also has a straightforward representation as a ZX-diagram:



(7)

Here we are allowed to draw a horizontal wire as per the symmetries (5) whether this wire is an input or an output is irrelevant.

Seeing as we can represent Z_α, H and CNOT gates as ZX-diagrams, we can represent the gate set $\mathcal{B}$ of Eq. (1), and hence we can in fact represent any unitary as a ZX-diagram. The above demonstrates that ZX-diagrams can be used as an alternative representation for quantum circuits. However, ZX-diagrams are also more versatile than unitary circuits. Consider for example the following construction of the CZ gate as a ZX-diagram:



(8)

The right-hand-side demonstrates a different diagrammatic construction for CZ, that does not immediately look circuit-like, with the Hadamard-box representing some sort of interaction of two qubits rather than the evolution of a single qubit.

In fact, this versatility is reflected in the property that ZX-diagrams are universal for *all* linear maps between any number of qubits [11]. To see this, note that we can represent states as in Eq. (3). By composing tensor products of these states with some unitary we can write down any quantum state. By the map-state duality of quantum theory (i.e. the Choi-Jamiołkowski isomorphism), we can then also write every linear map, see for instance [47] for the details.

The universality of the gate-set $\mathcal{B}$ and of the ZX-calculus means that any unitary operator on some fixed number of qubits may be represented by some "gadget" in the ZX calculus, consisting of some fixed diagram of finite size – though as the example of CZ in Eq. (8) shows, there may also be "gadgets" which represent a unitary operator which do *not* consist of sequential and parallel composition of gates. Indeed, even the representation of the CNOT is by a simple "gadget" of two nodes, which is not describable as a composition of the other single-node "gadgets". In this respect, ZX-diagrams represent a more versatile notation than a conventional circuit notation. This raises the question of how, given a representation of some unitary U as a ZX-diagram, one might find another representation of U which consists of just compositions from the universal gate-set $\mathcal{B}$. This is the problem that this paper is concerned with.

ZX-diagrams are more than just a notation for unitary circuits (and non-unitary operators more generally): they may be used to perform computations. Specifically, ZX-diagrams come with a set of graphical rewrite rules, which may be used to find equivalent diagrams which represent the same state or operator, just as one might manipulate an algebraic expression. This rewrite system is *complete* [37, 49]: unlike other circuit diagrams, one may show that two equivalent ZX-diagrams are equivalent though transformations of diagrams alone. The possible advantage of this is that ZX-diagrams can often concisely represent operators which have a very large number of non-zero coefficients, and so that this reasoning can be done efficiently while it could not be done using the matrices directly. For instance, one of the rewrite rules we will use in this paper is *spider fusion*:

(9)

These rules say that we can fuse together adjacent spiders of the same colour.

While these rewrite rules are not immediately relevant to our results, the fact that it is possible to compute with ZX-diagrams is the motivation for considering this particular representation of unitary circuits, and also motivates the concept of considering different ZX-diagrams which represent the same unitary transformation. We refer the interested reader to [47] for an overview.

2.3 Circuit extraction

In the above section we saw that we can get ZX-diagrams directly from quantum circuits. We can also get ZX-diagrams from considering measurement patterns in the *one-way model* [40]. In the one-way model of quantum computation we start with a large *graph state*, on which we then do subsequent measurements, where the choice of measurement angle and axis may depend on previous measurement outcomes. This leads to another universal model of quantum computation. The one-way model can be straightforwardly represented in the ZX-calculus [23, 4].

An important property of a one-way computation is that we can perform a computation deterministically, so that we perform the same overall computation regardless of individual measurement outcomes. A sufficient property for ensuring that deterministic processes are possible on a given resource state is that its underlying graph has a property known as *gflow* [6]. This is an efficiently verifiable combinatorial condition on the entangled resource.

When we represent a one-way computation with gflow as a ZX-diagram, the gflow ensures that certain "local" parts of the diagram correspond to individual unitary gates, in a way which can be iteratively translated into an actual unitary circuit. In this case we can hence *extract* a unitary quantum circuit from the ZX-diagram that represents the one-way computation. See for instance [22, 4, 41] for several variations on this idea.

Measurement-based quantum computation like the one-way model is a type of non-unitary quantum computation. Another type of non-unitary model is given by doing *lattice surgery* in the surface code [30, 20]. A lattice surgery procedure can also be represented as a ZX-diagram [20]. Just as in the one-way model, there is a flow condition that ensures such a calculation is deterministic, and that the resulting ZX-diagram can be step-by-step rewritten into a unitary circuit [19].

We see that there are several quantum computational models that can be written in terms of ZX-diagrams, which can be rewritten into a unitary quantum circuit efficiently when they satisfy some condition. The type of flow condition required for these procedures ensures that the diagram can't get "too wild" in the middle, so that we can stepwise rewrite the diagram into something that looks more like a circuit. A natural question to ask then is how much we can weaken such additional conditions, and in particular if we can transform a ZX-diagram into a circuit efficiently in the most general setting, where the only condition we require of the ZX-diagram is that it is proportional to a unitary. The main result of this paper is that such a general efficient procedure most likely does not exist.

2.4 Background on computational complexity

Finally, we provide some background on computational complexity. We assume knowledge of **P**, the boolean satisfiability problem **SAT**, oracle machines, **NP** and nondeterministic Turing machines (NTMs) in general. Our results concern *Cook reductions* (in fact, usually Cook[1] reductions). A Cook reduction from a problem **X** to another problem **Z** is an algorithm for solving **X** using a deterministic Turing machine which halts in polynomial time, but which may query an oracle (in the case of a Cook[1] reduction, exactly once) for **Z**. This implies that, modulo some polynomial-time computation, the problem **Z** is at least as hard as **X**; and that if $\mathbf{Z} \in \mathbf{P}$, we also have $\mathbf{X} \in \mathbf{P}$. In symbols we may write $\mathbf{X} \in \mathbf{P}^{\mathbf{Z}}$. Our results will generally concern problems **Z** related to ZX-diagrams and problems **X** which are at least **NP**-hard (*i.e.*, they suffice to solve **SAT**).

Quantum circuits form a model of computation, which may be considered to generate random outcomes through measurement operations. Note that, just as with the study of boolean circuits as a model of computation, one often considers a quantum circuit to be described by some polynomial-time computable procedure (a sort of "effective blueprint"), which for a given $n \geq 0$ requires time $\mathrm{poly}(\log n)$ to produce a circuit taking inputs of size n. While this constraint is not essential when considering a single circuit on its own (the description of the circuit itself is a finite specification), this constraint prevents us from considering what might otherwise seem like "quantum algorithms" for uncomputable problems (in the same way that one must for boolean circuits). Additionally, to prevent unbounded computational power from being hidden elsewhere in the description of a quantum circuit, one often imposes constraints on the gates and measurements allowed in a circuit. For instance, requiring that gates only act on a small constant number of qubits, and that for parametrised gates like Z_α the parameter α is efficiently computable. The class **BQP** consists of decision problems which can be decided with bounded error (with error probability less than, say, $\frac{1}{3}$) by quantum circuit families satisfying these reasonable constraints. This class represents the decision problems that can be practically solved by an (idealised) quantum computer.

It is not expected that either of **NP** or **BQP** contain the other. So if we can reduce in polynomial time (by many-to-one or oracle reductions) an **NP**-complete problem to some problem **X**, then we expect **X** to be intractable for quantum computers. Certain modifications of the quantum computational model do allow for more difficult problems to be solved, however. For instance, **PostBQP** is the class of problems which may be solved with bounded error by a uniform quantum circuit family, *conditioned on some other measurement* yielding a specific outcome (which occurs with non-zero probability). This "conditioning" restriction is known as *postselection*, and appears to be operationally very powerful, as **PostBQP** coincides with the class **PP**, of decision problems for which a "yes" instance is accepted on more than half of the branches of some NTM halting in polynomial time.

The class **P** is closed under oracles: a deterministic Turing machine equipped with an oracle for some problem in **P** cannot decide more problems in polynomial time than a normal Turing machine, so that $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$. The same is true for **BQP**: any decision problem solvable (with bounded error) by a uniform family of quantum circuits, can also be solved (with bounded error) by some other family of quantum circuits without oracle access, so that $\mathbf{BQP}^{\mathbf{BQP}} = \mathbf{BQP}$. The same is not true, however, for **NP**: it is not known whether $\mathbf{NP}^{\mathbf{NP}}$ (the class of decision problems, for which there is an NTM with an oracle for a problem in **NP**, halts in polynomial time and accepts in some branch precisely for "yes" instances) is equal to **NP**. It is widely conjectured that $\Sigma_2^{\mathrm{p}} := \mathbf{NP}^{\mathbf{NP}} \neq \mathbf{NP}$, and indeed that $\Sigma_3^{\mathrm{p}} := \mathbf{NP}^{\Sigma_2^{\mathrm{p}}} = \mathbf{NP}^{\mathbf{NP}^{\mathbf{NP}}} \neq \mathbf{NP}^{\mathbf{NP}}$, and so forth. The union of $\Sigma_n^{\mathrm{p}} := \mathbf{NP}^{\Sigma_{n-1}^{\mathrm{p}}}$ for all $n > 1$, defines the class **PH**, called the *polynomial hierarchy* [42].

The hardness results which we are most concerned with involve problems in **#P**: the class of problems which may be reduced to counting the number of accepting branches of some NTM on a given input. In particular, we are interested in the problem **#SAT**, of counting the number of "solutions" $x \in \{0,1\}^n$ to an instance of **SAT**, presented as a formula for a function $f : \{0,1\}^n \to \{0,1\}$, where a "solution" satisfies $f(x) = 1$. The problem **#SAT** is **#P**-complete [46], as is tensor contraction over the natural numbers [15], and "strong simulation" (*i.e.*, precise estimation of explicit measurement probabilities) of uniform quantum circuit families [48]. The **#P**-completeness here means that a Cook reduction from any of these problems to some problem **Z**, establishes that there is a Cook reduction from *any* problem $\mathbf{X} \in \mathbf{\#P}$ to **Z**. in this case we say then that **Z** is "**#P**-hard". The computational power of **#P** is considered to be significantly greater than that of **NP**. In particular, Toda [43] showed that $\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{\#P}}$.

3 Proof of hardness of Circuit extraction

We now present the central problem of our work.

CircuitExtraction
Input: A ZX-diagram D with n inputs and outputs and at most k wires and/or spiders, and a set $\mathcal{G}$ of unitary gates (each acting on at most $O(1)$ qubits).
Promise: The operator denoted by D is proportional to a unitary.
Output: Either **(a)** a poly(n,k)-size circuit C, expressed as a sequence of gates from $\mathcal{G}$ and expressing an n-qubit unitary that is proportional to the operator denoted by D, if such a circuit exists; or **(b)** a message that no such circuit exists, if that is the case.

Note that here we make no assumptions on the specific gate set $\mathcal{G}$, apart from the computability of the coefficients as described in Section 2.1, and that the number of qubits which is bounded by some constant. One might object to the requirement that the output list of gates must be polynomially related to the size of the input ZX-diagram: however, as we are interested in whether the extraction problem can be solved efficiently, the restriction on the size of C follows from the time required to represent it as a list of gates.

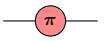
▶ Remark 3.1. For a finite gate set we can consider the gate set $\mathcal{G}$ as being supplied as concrete matrices, while for an infinite set we could consider it as being supplied as an efficient procedure for translating a "gate label" into a matrix. For concreteness sake we can take $\mathcal{G} = \mathcal{B}$, the gate set (1). This gate set contains the parametrised gate Z_α. Formally a circuit will then specify these phases α via some efficiently computable procedure.

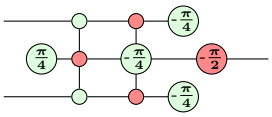
The above problem can of course also be stated for any related graphical language for quantum operations, such as the ZH-calculus [3] or the ZW-calculus [28]. Since such diagrams can be efficiently translated into one another, these problems are of equivalent hardness. There are some other reasonable variations we can consider of **CircuitExtraction** that we will discuss in the next section.

We will now show that **CircuitExtraction** is **#P**-hard. We do this by building a diagram that is proportional to a unitary based on a **SAT** instance, and showing that the resulting matrix the diagram represents is uniquely determined by the number of solutions of the instance.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean formula with poly(n) terms. We say a bit string $x \in \{0,1\}^n$ is a *solution* to f when $f(x) = 1$. The first step will be to build a ZX-diagram that implements the linear map L_f that takes n qubits to 1 qubit by $L_f|x\rangle = |f(x)\rangle$. We can of course represent f as a tree of AND and NOT operations so that to construct L_f it suffices to find linear maps that implement AND and NOT on $|x\rangle$.

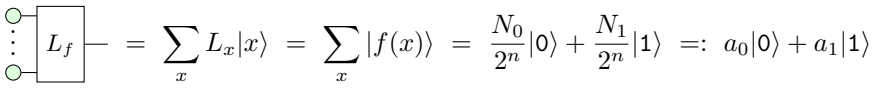
We may consider ZX diagrams for "quantum" versions of the boolean logical AND gate and NOT gate, *i.e.*, linear operators such that $\mathrm{NOT}|0\rangle = |1\rangle$, $\mathrm{NOT}|1\rangle = |0\rangle$, and $\mathrm{AND}|x, y\rangle \mapsto |x \cdot y\rangle$. We could just appeal to the universality of ZX-diagrams to establish that there are diagrams that represent these operations, but for completeness sake let us give some concrete diagrams to realise these operations (up to a constant factor):

NOT $=$ 

AND $\propto$ 

The NOT gate is just an X_π gate, but the AND is more complicated. It is based on the representation of the CCZ gate from [35] that uses 4 T gates. Its correctness can be verified by inputting $|0\rangle$ and $|1\rangle$ on the inputs and seeing that it has the correct action on all these inputs.

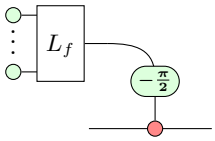
By combining these diagrammatic gadgets for NOT and AND we can build the operation L_f as a ZX-diagram using poly(n) spiders. Now, note that:



(10)

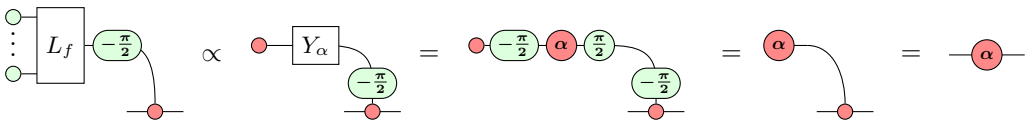
where N_1 is the number of solutions of f, $N_0 = 2^n - N_1$ is the number of "non-solutions" of f, and we set $a_0 = N_0/N$ and $a_1 = N_1/N$ for $N := 2^n = N_0 + N_1$. The resulting state is not normalised: to normalise it we should multiply both sides by $(a_0^2 + a_1^2)^{-1/2}$.

We use the "state" described in Eq. (10) as the input of a controlled operation. By choosing the controlled operation appropriately, we will be left with something proportional to a unitary. We may for instance consider the following diagram:

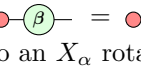
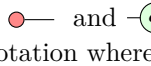


(11)

To see this is unitary first recall that a Y rotation over an angle α applied to $|0\rangle$ gives $Y_\alpha|0\rangle = \cos(\frac{\alpha}{2})|0\rangle + \sin(\frac{\alpha}{2})|1\rangle$. Hence the state of Eq. (10), when properly normalised, can be written as $Y_\alpha|0\rangle$ for $\alpha = 2\sin^{-1}\left(\frac{a_1}{\sqrt{a_0^2+a_1^2}}\right)$. We can then calculate:



(12)

In the above, we use the relation $Y_\alpha = Z_{-\frac{\pi}{2}} X_\alpha Z_{\frac{\pi}{2}}$, and some simple ZX-calculus rewrites (namely that  and ). Hence, the diagram of Eq. (11) is proportional to an X_α rotation where α is uniquely determined by the number of solutions to f. Note that this operation can be easily represented (with at most three gates) using a gate-set such as $\{H, Z_\alpha, \mathrm{CNOT}\}$, in which the set of values of allowed angles α include those that may arise in the diagram of Eq. (12) for some number of solutions N_1 to the formula f; such an operation will be representable using other gate-sets as well.[1]

[1] Note that the gate-set described here cannot be a single, finite gate set for all values of n. However, the angles α arising out of instances of satisfiability in this way can be specified in $O(n)$ bits, precisely by

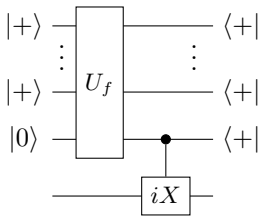
▶ **Theorem 3.2.** **CircuitExtraction** *is* **#P**-*hard.*

Proof. **#SAT** is a **#P**-complete problem, so it suffices to show that we can count the number of solutions to a Boolean formula using a call to an oracle which solves **CircuitExtraction**. Given a Boolean formula $f : \{0,1\}^n \to \{0,1\}$ with $\text{poly}(n)$ terms, construct the diagram of Eq. (11). The diagram here for L_f uses $\text{poly}(n)$ of the diagrammatic gadgets for NOT and AND, and hence the complete diagram consists of $\text{poly}(n)$ spiders, each of which may be restricted to having at most 3 wires. We may apply the **CircuitExtraction** oracle on this diagram subject to a suitable gate set that can exactly generate the possible X-rotations X_α which may arise. As C is a single-qubit circuit with at most $\text{poly}(n)$ gates, we can calculate the unitary it implements, up to any required precision $2^{-O(\text{poly}(n))}$, in polynomial time. We know that the operation realised is of the form X_α, so to determine the value of N_1, it suffices to estimate the entries of the resulting X_α to within an error of $\frac{1}{2\sqrt{2}}$. Determining the value of $a_1/\sqrt{a_0^2 + a_1^2}$ to $2n$ bits of precision is sufficient to do this. ◀

▶ **Corollary 3.3.** *If there is a polynomial time algorithm for* **CircuitExtraction***, then* $\mathbf{P} = \mathbf{P}^{\#\mathbf{P}}$*. In particular, the polynomial hierarchy collapses to the first level:* $\mathbf{P} = \mathbf{NP} = \mathbf{PH}$.

Proof. If **CircuitExtraction** can be done in polynomial time, then the above shows that we can solve **#SAT** in polynomial time, and hence $\mathbf{NP} \subseteq \mathbf{P}^{\#\mathbf{P}} = \mathbf{P}$. ◀

▶ Remark 3.4. Our construction of the diagram we use to prove our result might seem somewhat arbitrary. To motivate it some more, first realise that instead of the function L_f, we could have used the standard unitary quantum oracle for a Boolean function U_f which acts on $n+1$ qubits via $U_f|x,b\rangle = |x, b \oplus f(x)\rangle$. We can get L_f out of U_f by post-selecting the top n qubits to $\langle +|$. Using the language of post-selection, we may then present a circuit version of Eq. (11):



(13)

The top part is calculating the number of solutions, while the bottom part ensures that this information is fed into a qubit in such a way that the overall operation is proportional to a unitary. The choice of iX is for the sake of simplicity: any unitary U that satisfies $U = -U^\dagger$ would also suffice, such as iY or iZ.

▶ Remark 3.5. Even though we can view the diagram as a post-selected circuit, this does not seem to be where the power of the procedure comes from, as it is for instance in Aaronson's characterisation $\mathbf{PostBQP} = \mathbf{PP}$ [1]. In our setting the probability of observing the "correct" outcome is bounded from below by a constant, and does not depend on n. This means in particular that by doing repeat-until-success we could with high probability implement the circuit Eq. (13) on a quantum computer. However, this does not allow you to solve **#SAT**, as adjacent possibilities of the rotation angle α are exponentially close. So rather, the power of the procedure comes from getting an explicit description of the circuit which allows us to exactly calculate the rotation angle.

characterising them in the way we have, by relating some integer ranging in $\{0, 1, \ldots, 2^n\}$ via inverse trigonometric functions. For remarks on what can be achieved with finite gate-sets, the reader may be interested in the related problem **ApproxCircuitExtraction**, in Section 4.

4 Variations on extraction

There are several variations on circuit extraction which we can consider, all of which also turn out to be hard.

The essential trick we used in our proof is that our resulting circuit has just *one* qubit, and hence a description of a unitary on it can easily be transformed into the actual unitary it implements by just multiplying all the resulting matrices. But of course the same statement remains true if we have slightly more than one qubit, say a logarithmic amount in the size of the **SAT** instance. We also see that it then doesn't matter if our circuit contains auxiliary qubits, measurements, or classically-controlled corrections. All of these can be efficiently calculated as long as the number of qubits is small enough. Therefore, let's define the following variant of circuit extraction.

AuxCircuitExtraction
Input: A ZX-diagram D with n inputs and outputs and at most k wires and/or spiders, and a set $\mathcal{G}$ of unitary gates (each acting on at most $O(1)$ qubits).
Promise: The operator denoted by D is proportional to a unitary.
Output: Either **(a)** a deterministic n-qubit circuit implementing the unitary of the input ZX-diagram, described as a $\mathrm{poly}(n,k)$ length list of gates, auxiliary qubit preparations, measurements, and classical corrections, with at most $O(\log k)$ auxiliary qubits; or **(b)** a message that no such circuit exists, if that is the case.

▶ **Theorem 4.1. AuxCircuitExtraction** *is* **#P***-hard.*

Proof. We construct the same diagram as in the proof of Theorem 3.2 to solve a **#SAT** instance, except that we can no longer assume that the final circuit will act only on a single qubit: instead it may act on up to $O(\log k)$ qubits, including the operations on the auxiliary qubits. The size of the matrices involved when trying to calculate the resulting unitary is $O(2^{\log \mathrm{poly}(k)}) = O(\mathrm{poly}(k))$, where here k is the size of the input diagram. We may then still multiply the matrices together in polynomial time to obtain sufficiently precise estimates of the coefficients. ◀

One might also object that requiring the output unitary to *exactly* represent the ZX-diagram is too strong – in particular, impossible in general even with an approximately universal, finite gate set – and wish for an approximate output instead. We say that a unitary operator $\tilde{U}$ is an ε*-approximation* of another unitary U for some $\varepsilon > 0$, if $\|\tilde{U} - e^{i\alpha}U\| < \varepsilon$ for some global phase α. Here, $\|M\|$ denotes the operator norm of M: the largest singular value of M.

ApproxCircuitExtraction
Input: A ZX-diagram D with n inputs and outputs and at most k wires and/or spiders, a set $\mathcal{G}$ of unitary gates (each acting on at most $O(1)$ qubits), and a precision parameter $\varepsilon > 0$.
Promise: The operator denoted by D is proportional to a unitary.
Output: Either **(a)** a $\mathrm{poly}(n, k, \log(1/\varepsilon))$-size circuit C, expressed as a sequence of gates from $\mathcal{G}$ and expressing an n-qubit unitary $\tilde{U}$ which is an ε-approximation to either the operator denoted by D, or some operator proportional to it; or **(b)** a message that no such circuit exists, if that is the case.

▶ **Theorem 4.2. ApproxCircuitExtraction** *is* **#P***-hard.*

Proof. For a given **SAT** instance $f : \{0,1\}^n \to \{0,1\}$ we again construct the same diagram as in the proof of Theorem 3.2 which denotes a unitary X_α, where α allows us to determine the number of solutions to f. This diagram has $\mathrm{poly}(n)$ spiders. Set $\varepsilon = 2^{-cn}$ for some large enough constant c. Then applying **ApproxCircuitExtraction** gives rise to a circuit, which has $\mathrm{poly}(\mathrm{poly}(n), \log(1/2^{-cn})) = \mathrm{poly}(n)$ gates. We can hence just multiply out the matrices in order to determine the unitary U it implements. This unitary U approximates X_α to degree 2^{-cn}. Since the top left entry of X_α is real, we can first multiply U by the appropriate global phase to ensure it is also real. If we have picked c large enough then the entries of U are then within $\frac{1}{2}2^{-n}$ of that of X_α so that we can determine α by rounding to the nearest allowed value. ◀

Note that, even for exponentially small angles α as might arise when f has few solutions, circuits of polynomial size do *exist* for X_α when $\mathcal{G}$ is an approximately universal gate-set: using the Solovay–Kitaev algorithm [36, 16] or any of its many refinements (see *e.g.* Ref. [5] and references therein), we may synthesise circuits approximating X_α to any precision ε in time scaling polynomially in $\log(1/\varepsilon)$. The difficulty of **ApproxCircuitExtraction** stems from determining *which* angle α to approximate. One might nevertheless want to consider a variation on **ApproxCircuitExtraction** with a polynomial dependence on $1/\varepsilon$ instead of $\log 1/\varepsilon$. We believe this variant will still be hard: see Remark 4.4.

Let us consider one final variation on extraction. One could argue that the reason that we end up with a hard problem in these instances, is because requiring the output to be some kind of circuit is too restrictive. The ultimate goal of circuit extraction is that we wish for the ZX-diagram to be run on a quantum computer in order to obtain some probability distribution over outcomes; but the complexity of **CircuitExtraction** and its variations seems to arise from the complexity of finding a precise description of the procedure to do so. Cutting out the middle-man, we may consider *any* process which takes as input a unitary ZX-diagram, and produces bit strings as output whose distribution conforms with the one we expect from the unitary.

UnitaryZXSampling
Input: A ZX-diagram D with n inputs and outputs and at most k wires and/or spiders.
Promise: The operator denoted by D is proportional to some unitary U.
Output: A sample $x \in \{0,1\}^n$ from a probability distribution, given by (or sufficiently close to) $|\langle x|U|0\cdots 0\rangle|^2$.

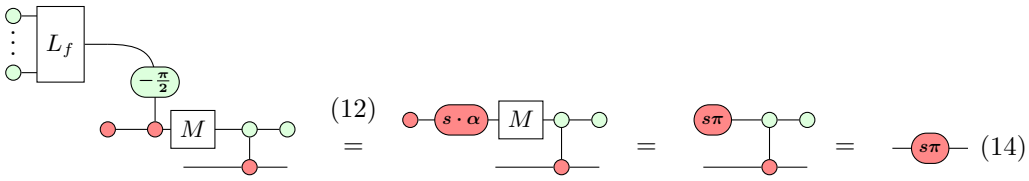
It is clear that **UnitaryZXSampling** is at least as hard as **BQP**: we could just input a ZX-diagram that directly represents a quantum circuit, in which case this problem is equivalent to simulating that circuit. The reason we write here that the probabilities just have to be "sufficiently close" is because the exact number doesn't matter for the following theorem. (For instance: we could allow the probability to additively deviate by 1/3 from the true value.)

▶ **Theorem 4.3.** *There is a randomised polynomial reduction from* **NP** *to* **UnitaryZX-Sampling**. *In other words: with access to a* **PromiseUnitaryZXSampling** *oracle – which produces the expected output if the input diagram is unitary and arbitrary output otherwise – we can with high probability solve* **NP***-complete problems.*

Proof. **SAT** is an **NP**-complete problem. To randomly reduce **NP** it however suffices to consider the problem **USAT** by the Valiant–Vazirani theorem [45]. **USAT** asks us to determine whether a Boolean formula is satisfiable, given the promise that it has at most one solution. Using the randomised reduction from **SAT** to **USAT**, we consider how to solve **USAT** using a **PromiseUnitaryZXSampling** oracle.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean formula that has at most one solution. Construct the diagram Eq. (11) as in the previous proofs: as a unitary this implements the identity iff f is not satisfiable, and X_α for some fixed angle $\alpha > 0$ when f is satisfiable. In the latter case, the value of α is exponentially small, but known precisely, as f has exactly one solution in this case. So we can say the circuit implements $X_{s\cdot\alpha}$ where $s \in \{0,1\}$ encodes whether f is satisfiable or not.

Let M be the one-qubit (non-unitary) matrix that maps $|0\rangle \mapsto |0\rangle$ and $X_\alpha|0\rangle \mapsto |1\rangle$, so that in particular $MX_{s\cdot\alpha}|0\rangle = |s\rangle$. By universality of ZX-diagrams we can find some (constant sized) diagram to represent M. We can then calculate:



Here the last step is just spider fusion (9). We see then that the ZX-diagram on the left in Eq. (14) implements either the identity, or an X_π operation (that is to say, a NOT operation), depending on whether f is satisfiable. When we feed this ZX-diagram to an oracle for **PromiseUnitaryZXSampling**, we get either the output `0` or `1`, where a `0` indicates with high probability that the circuit is the identity, and a `1` indicates that the circuit is a NOT operation. We can repeatedly call the oracle to get additional samples to increase our confidence in the result.

Now suppose f is a general instance of **SAT**, which may have more than one solution. Using the Valiant–Vazirani reduction multiple times we probabilistically produce different Boolean formulae $f_1, \ldots, f_m$. If f is not satisfiable, then none of the f_j will be satisfiable either and this is what the **PromiseUnitaryZXSampling** will tell us as well. If f *is* satisfiable, then a significant fraction of the f_j will have a unique solution, so that our oracle tells us they are satisfiable. For the other f_j the oracle will return some arbitrary output. So by picking m large enough there will with high probability be some f_j that will be uniquely satisfiable, and so we can conclude that f is satisfiable as well.

Hence, we can determine with arbitrary high probability whether a **SAT** instance is satisfiable using enough calls to **PromiseUnitaryZXSampling**. ◀

▶ Remark 4.4. For **ApproxCircuitExtraction** we allowed a polynomial dependence on $\log 1/\varepsilon$ for the circuit size. We believe this is reasonable as the Solovay–Kitaev algorithm allows you to find a $\mathrm{poly}(\log 1/\varepsilon)$-sized circuit when approximating a unitary. However, one can also consider the hardness of the problem when we allow the circuit size to depend on $\mathrm{poly}(1/\varepsilon)$, or even when the circuit size does not depend at all on the error. One might suspect that this could change the hardness of the problem. (As an analogy: one may compute the permanent of an $n \times n$ matrix with positive entries to within a multiplicative error of ε in time $\mathrm{poly}(n, 1/\varepsilon)$ [32], despite the exact problem being **#P**-complete.) However, note that if we can approximately do circuit extraction up to some *constant* error (say $\varepsilon = 1/10$), we can feed the resulting circuit to a quantum computer in order to solve **UnitaryZXSampling**. Hence, even if we were to relax **ApproxCircuitExtraction** to allow an output circuit of size $\mathrm{poly}(1/\varepsilon)$, an efficient algorithm for **ApproxCircuitExtraction** would imply that **NP** $\subseteq$ **BQP** by Theorem 4.3.

▶ Remark 4.5. If we knew that the number of solutions to the **SAT** instance was some other fixed number, then we could pick a different matrix M' to boost the state up to X_π gate as in the proof of Theorem 4.3. If we pick M' "slightly wrong", then the resulting diagram will just be close to X_π. One might think that we could use such a procedure to try and determine the number of solutions to f by doing binary search on the number of solutions, and so boost the power of **UnitaryZXSampling** to **#P**. However, the problem with this is that the resulting diagrams are not proportional to a unitary most of the time. There might be some way around this issue, so that **UnitaryZXSampling** is still **#P**-hard: we leave this as an open problem.

▶ Remark 4.6. Note that if we were to consider a version of **UnitaryZXSampling**, without the promise of unitarity, such an oracle would be as powerful as **PostBQP**, since we can represent any ZX-diagram as a post-selected quantum circuit (and conversely). In our case, the power again comes not so much from postselection, as being able to take advantage of the versatility of ZX-diagrams to gain access, in some way, to extract very precise information regarding a **#P** problem.

5 Upper bounding the complexity of CircuitExtraction

Given that **CircuitExtraction** is **#P**-hard, one might ask whether or not the problem is **#P**-complete (or more precisely: $\mathbf{FP}^{\mathbf{\#P}}$-complete since we are not making a decision but rather outputting a circuit), in the sense that a Turing machine with access to a **#P** oracle would be able to solve it, for some given polynomial upper bound on circuit size and some given gate-set (perhaps with suitable restrictions), in polynomial time. We have not managed to prove such a completeness result. We will however present the following upper bounds on decision problem versions of circuit extraction, relying on techniques from [2] that relate calculating amplitudes of quantum circuits to counting complexity problems.

First, consider the following decision problem: given a ZX-diagram, and a circuit, determine whether the circuit implements a unitary which is proportional to that represented by the ZX-diagram (whether by a factor of $\mathrm{e}^{i\theta}$ for some angle θ, or a more general complex number). This problem is in $\mathbf{coNP}^{\mathbf{\#P}}$. To sketch why this is, consider a circuit C representing a unitary U, and a ZX diagram D representing an operator V. If a_0 and a_1 are two non-zero coefficients from U, and b_0 and b_1 are the corresponding (non-zero) pair of coefficients from the matrix V represented by D, then $U \propto V$ only if $a_0/b_0 = a_1/b_1$ for all possible such pairs. We also require that for any coefficient a in U which is zero, the corresponding coefficient b of V is also zero. Taken together, this implies that for all corresponding pairs of coefficients of U and V we should have $a_0 b_1 = a_1 b_0$. This is also sufficient for $U \propto V$ to hold. Now, a **#P** oracle allows one to calculate coefficients[2] of ZX-diagrams and circuits. Hence, if $U \not\propto V$, an NTM with access to a **#P** oracle can non-deterministically find a witness that these two operators are not in fact proportional to one another. Thus, determining whether a circuit does *not* represent a unitary which is denoted (up to scalar factors) by a ZX-diagram, is in $\mathbf{NP}^{\mathbf{\#P}}$.

The above result has a simple corollary: determining whether a ZX-diagram is proportional to a unitary itself belongs to $\mathbf{coNP}^{\mathbf{\#P}}$. We may see this by the fact that a ZX-diagram denoting an operator V, which is proportional to a unitary, satisfies $VV^\dagger \propto I$. We may

[2] In this case, it is not necessary to compute complete information about a_0, a_1, b_0, and b_1: it suffices to compute information about individual components of the products $a_0 b_1$ and $a_1 b_0$ when considering these as numbers in some number field over $\mathbb{Q}$. See Ref. [2] for the details.

represent $VV^\dagger$ by composing the diagram D with its adjoint (which is the left-to-right mirror image of D, with all phase angles negated). This composite diagram may easily be computed, at which point we may ask whether the operator it represents is proportional to the identity by a non-zero scalar factor. As we note above, this problem is in $\mathbf{coNP}^{\#\mathbf{P}}$.

Finally, using these ideas, we may consider the decision problem of determining for a ZX-diagram D denoting an operator V and some gate set $\mathcal{G}$ and polynomial length bound N, whether *there exists* a circuit of at most N gates over $\mathcal{G}$ which implements V (up to a scalar). This problem is in $\mathbf{NP}^{\mathbf{NP}^{\#\mathbf{P}}}$: for an NTM with access to an $\mathbf{NP}^{\#\mathbf{P}}$ oracle, it suffices to make a nondeterministic guess at a circuit of length N (where each gate may be the identity operator, or some gate $G \in \mathcal{G}$ acting on a non-deterministically chosen set of qubits) and then query the oracle to determine whether the circuit realises V. A deterministic Turing machine, with access to an oracle for this problem, could then solve **CircuitExtraction** in polynomial time using standard techniques, using the oracle to facilitate a search for a circuit to realise D.

These observations represent the most straightforward approach to determining an upper bound for the circuit extraction problem, and seem to place it at a level of complexity significantly higher than $\mathbf{P}^{\#\mathbf{P}}$. If we conceive of **#P** as broadly representing the complexity of evaluating a tensor network, a superficial analogy between **CircuitExtraction** and boolean circuit minimisation [25, 7] would seem to suggest that **CircuitExtraction** is likely to be hard for some complexity class higher than $\mathbf{P}^{\#\mathbf{P}}$ (barring some collapse of complexity classes).

6 Conclusion

In this paper we studied the problem of extracting a quantum circuit description from a unitary ZX-diagram. We've shown that this problem is **#P**-hard by reducing **#SAT** to an application of circuit extraction. We've also studied some variations where we allow auxiliary qubits, classical control, and/or approximate synthesis of the desired unitary, and have shown that these problems are also **#P**-hard. In addition, we studied the hardness of a machine that takes in a unitary ZX-diagram and outputs measurement samples from that ZX-diagram, and have shown that such a machine allows one to probabilistically solve **NP**-hard problems.

A conclusion to be drawn from our results is that if we want some efficient procedure to transform a unitary ZX-diagram into a quantum circuit, then we will have to have some additional information about the structure of the ZX-diagram. In the known procedures for efficient circuit extraction [4, 41, 19], this additional information takes the form of a kind of "flow" on the diagram that prevents parts of the diagram from becoming too unwieldy. An immediate question then is if there are other types of, more general, promises on the structure of the diagram which then allow you to extract a circuit from it.

Aaronson showed that sampling from a post-selected quantum circuit is hard [1]. Our results imply that some other tasks surrounding *unitary* post-selected circuits (that is, circuits which perform a unitary transformation conditioned on some post-selection) are hard. However, this hardness seems to stem not from the post-selection itself, as the post-selections can be simulated with high probability in our case. Rather, the hardness seems to stem from a hypothetical ability to find an equivalent, deterministic way to realise the same operation – which implies an ability to extract difficult-to-access information about the input diagram.

A question related to circuit extraction from ZX-diagrams is circuit extraction from deterministic measurement patterns (in for instance the one-way model or lattice surgery). When we have a deterministic measurement pattern, we can represent each branch of the computation by a ZX-diagram denoting a unitary. Our hardness proof does however not

immediately translate to this setting, as it might be that the fact that all of these ZX-diagrams are branches of the same measurement pattern forces some kind of structure on the diagrams that might make it easier to rewrite them into circuits. The diagrams we used to show hardness of circuit extraction are as far as we are aware not representable as branches of some deterministic measurement pattern, so that we can't use the same proof. We leave it for future work to determine the hardness of extracting unitary circuits from deterministic measurement patterns.

References

1 Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 461(2063):3473–3482, 2005. doi:10.1098/rspa.2005.1546.

2 Leornard M. Adleman, Jonathan DeMarrais, and Ming-Deh A. Huang. Quantum computability. *SIAM Journal on Computing*, 26:1524–1540, 1997. doi:10.1137/S0097539795293639.

3 Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. In Peter Selinger and Giulio Chiribella, editors, *Proceedings of the 15th International Conference on Quantum Physics and Logic, Halifax, Canada, 3-7th June 2018*, volume 287 of *Electronic Proceedings in Theoretical Computer Science*, pages 18–34. Open Publishing Association, 2019. doi:10.4204/EPTCS.287.2.

4 Miriam Backens, Hector Miller-Bakewell, Giovanni de Felice, Leo Lobski, and John van de Wetering. There and back again: A circuit extraction tale. *Quantum*, 5:421, March 2021. doi:10.22331/q-2021-03-25-421.

5 Adam Bouland and Tudor Giurgica-Tiron. Efficient universal quantum compilation: An inverse-free Solovay–Kitaev algorithm. *arXiv preprint*, 2021. arXiv:2112.02040.

6 Daniel E. Browne, Elham Kashefi, Mehdi Mhalla, and Simon Perdrix. Generalized flow and determinism in measurement-based quantum computation. *New Journal of Physics*, 9(8):250, 2007. doi:10.1088/1367-2630/9/8/250.

7 David Buchfuhrer and Christopher Umans. The complexity of Boolean formula minimization. *Journal of Computer and System Sciences*, 77(1):142–153, 2011. Celebrating Karp's Kyoto Prize. doi:10.1016/j.jcss.2010.06.011.

8 Titouan Carette. *Wielding the ZX-calculus, Flexsymmetry, Mixed States, and Scalable Notations*. PhD thesis, Loria, Université de Lorraine, 2021. URL: https://hal.archives-ouvertes.fr/tel-03468027/document.

9 Titouan Carette, Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of Graphical Languages for Mixed States Quantum Mechanics. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 108:1–108:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.108.

10 Bob Coecke, Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Foundations for Near-Term Quantum Natural Language Processing. *arXiv preprint*, 2020. arXiv:2012.03755.

11 Bob Coecke and Ross Duncan. Interacting quantum observables. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, 2008. doi:10.1007/978-3-540-70583-3_25.

12 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13:043016, 2011. doi:10.1088/1367-2630/13/4/043016.

13 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes*. Cambridge University Press, 2017. doi:10.1017/9781316219317.

14 Richard D. P. East, John van de Wetering, Nicholas Chancellor, and Adolfo G. Grushin. AKLT-states as ZX-diagrams: diagrammatic reasoning for quantum states. *PRX Quantum*, 3:010302, January 2022. doi:10.1103/PRXQuantum.3.010302.

15 Carsten Damm, Markus Holzer, and Pierre McKenzie. The complexity of tensor calculus. *Computational Complexity*, 11(1):54–89, 2002. doi:10.1007/s00037-000-0170-4.

16 Christopher M. Dawson and Michael A. Nielsen. The solovay-kitaev algorithm. *arXiv preprint*, 2005. arXiv:0505030.

17 Niel de Beaudrap. Well-tempered ZX and ZH Calculi. In Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden, editors, *Proceedings 17th International Conference on Quantum Physics and Logic, Paris, France, June 2 - 6, 2020*, volume 340 of *Electronic Proceedings in Theoretical Computer Science*, pages 13–45. Open Publishing Association, 2021. doi:10.4204/EPTCS.340.2.

18 Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. Techniques to Reduce $\pi/4$-Parity-Phase Circuits, Motivated by the ZX Calculus. In Bob Coecke and Matthew Leifer, editors, *Proceedings 16th International Conference on Quantum Physics and Logic, Chapman University, Orange, CA, USA., 10-14 June 2019*, volume 318 of *Electronic Proceedings in Theoretical Computer Science*, pages 131–149. Open Publishing Association, 2020. doi:10.4204/EPTCS.318.9.

19 Niel de Beaudrap, Ross Duncan, Dominic Horsman, and Simon Perdrix. Pauli Fusion: a Computational Model to Realise Quantum Transformations from ZX Terms. In Bob Coecke and Matthew Leifer, editors, *Proceedings 16th International Conference on Quantum Physics and Logic, Chapman University, Orange, CA, USA., 10-14 June 2019*, volume 318 of *Electronic Proceedings in Theoretical Computer Science*, pages 85–105. Open Publishing Association, 2020. doi:10.4204/EPTCS.318.6.

20 Niel de Beaudrap and Dominic Horsman. The ZX calculus is a language for surface code lattice surgery. *Quantum*, 4, 2020. doi:10.22331/q-2020-01-09-218.

21 Niel de Beaudrap, Aleks Kissinger, and Konstantinos Meichanetzidis. Tensor Network Rewriting Strategies for Satisfiability and Counting. In Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden, editors, *Proceedings 17th International Conference on Quantum Physics and Logic, Paris, France, June 2 - 6, 2020*, volume 340 of *Electronic Proceedings in Theoretical Computer Science*, pages 46–59. Open Publishing Association, 2021. doi:10.4204/EPTCS.340.3.

22 Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum*, 4:279, June 2020. doi:10.22331/q-2020-06-04-279.

23 Ross Duncan and Simon Perdrix. Rewriting Measurement-Based Quantum Computations with Generalised Flow. In *Proceedings of ICALP*, Lecture Notes in Computer Science, pages 285–296. Springer, 2010. doi:10.1007/978-3-642-14162-1_24.

24 Richard D. P. East, Pierre Martin-Dussaud, and John van de Wetering. Spin-networks in the ZX-calculus. *arXiv preprint*, 2021. arXiv:2111.03114.

25 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

26 Craig Gidney and Austin G. Fowler. Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation. *Quantum*, 3:135, April 2019. doi:10.22331/q-2019-04-30-135.

27 Craig Gidney and Austin G. Fowler. Flexible layout of surface code computations using AutoCCZ states. *arXiv preprint*, 2019. arXiv:1905.08916.

28 Amar Hadzihasanovic. A diagrammatic axiomatisation for qubit entanglement. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 573–584. IEEE, 2015. doi:10.1109/LICS.2015.59.

29 Michael Hanks, Marta P. Estarellas, William J. Munro, and Kae Nemoto. Effective Compression of Quantum Braided Circuits Aided by ZX-Calculus. *Physical Review X*, 10:041030, 2020. doi:10.1103/PhysRevX.10.041030.

30 C. Horsman. Quantum picturalism for topological cluster-state computing. *New Journal of Physics*, 13(9):095011, 2011. doi:10.1088/1367-2630/13/9/095011.

31 C. Horsman, Austin G. Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14:123011, 2012. doi:10.1088/1367-2630/14/12/123011.

32 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Nonnegative Entries. *J. ACM*, 51(4):671–697, July 2004. doi:10.1145/1008731.1008738.

33 Aleks Kissinger and Arianne Meijer-van de Griend. CNOT circuit extraction for topologically-constrained quantum memories. *Quantum Information and Computation*, 20:581–596, 2020. doi:10.26421/QIC20.7-8.

34 Aleks Kissinger and John van de Wetering. Reducing the number of non-Clifford gates in quantum circuits. *Physical Review A*, 102:022406, August 2020. doi:10.1103/PhysRevA.102.022406.

35 Aleks Kissinger and John van de Wetering. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions. *Quantum Science and Technology*, 2022. doi:10.1088/2058-9565/ac5d20.

36 Alexei Y. Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, 1997.

37 Kang Feng Ng and Quanlong Wang. A universal completion of the ZX-calculus. Preprint, 2017. arXiv:1706.09877.

38 M. A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010. doi:10.1119/1.1463744.

39 Roger Penrose. Applications of negative dimensional tensors. In *Combinatorial Mathematics and its Applications*, pages 221–244. Academic Press, 1971.

40 Robert Raussendorf and Hans J. Briegel. A One-Way Quantum Computer. *Physical Review Letters*, 86:5188–5191, May 2001. doi:10.1103/PhysRevLett.86.5188.

41 Will Simmons. Relating Measurement Patterns to Circuits via Pauli Flow. In Chris Heunen and Miriam Backens, editors, *Proceedings 18th International Conference on Quantum Physics and Logic, Gdansk, Poland, and online, 7-11 June 2021*, volume 343 of *Electronic Proceedings in Theoretical Computer Science*, pages 50–101. Open Publishing Association, 2021. doi:10.4204/EPTCS.343.4.

42 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977. doi:10.1016/0304-3975(76)90061-X.

43 Seinosuke Toda. PP is as hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, pages 865–877, October 1991. doi:10.1137/0220053.

44 Alex Townsend-Teague and Konstantinos Meichanetzidis. Classifying Complexity with the ZX-Calculus: Jones Polynomials and Potts Partition Functions. *arXiv preprint*, 2021. arXiv:2103.06914.

45 L G Valiant and V V Vazirani. NP is as Easy as Detecting Unique Solutions. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 458–463, New York, NY, USA, 1985. Association for Computing Machinery. doi:10.1145/22145.22196.

46 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. doi:10.1137/0208032.

47 John van de Wetering. ZX-calculus for the working quantum computer scientist. *arXiv preprint*, 2020. arXiv:2012.13966.

48 Maarten Van Den Nest. Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond. *Quantum Information & Computation*, 10(3):258–271, 2010. doi:10.5555/2011350.2011356.

49 Renaud Vilmart. A Near-Minimal Axiomatisation of ZX-Calculus for Pure Qubit Quantum Mechanics. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, 2019. doi:10.1109/LICS.2019.8785765.

Hiding Pebbles When the Output Alphabet Is Unary

Gaëtan Douéneau-Tabot
IRIF, Université Paris Cité & Direction générale de l'armement – Ingénierie de projets, France

Abstract

Pebble transducers are nested two-way transducers which can drop marks (named "pebbles") on their input word. Blind transducers have been introduced by Nguyên et al. as a subclass of pebble transducers, which can nest two-way transducers but cannot drop pebbles on their input.

In this paper, we study the classes of functions computed by pebble and blind transducers, when the output alphabet is unary. Our main result shows how to decide if a function computed by a pebble transducer can be computed by a blind transducer. We also provide characterizations of these classes in terms of Cauchy and Hadamard products, in the spirit of rational series. Furthermore, pumping-like characterizations of the functions computed by blind transducers are given.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases polyregular functions, pebble transducers, rational series, factorization forests, Cauchy product, Hadamard product

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.120

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2112.10212`

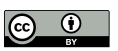
Acknowledgements The author is grateful to Olivier Carton, Lê Thành Dũng Nguyên and Pierre Pradic for discussing about this work.

1 Introduction

Transducers are finite-state machines obtained by adding outputs to finite automata. In this paper, we assume that these machines are always deterministic and have finite inputs, hence they compute functions from finite words to finite words. In particular, a *deterministic two-way transducer* consists of a two-way automaton which can produce outputs. This model computes the class of *regular functions*, which is often considered as one of the functional counterparts of *regular languages*. It has been largely studied for its numerous regular-like properties: closure under composition [4], equivalence with logical transductions [9] or regular transducer expressions [5], decidable equivalence problem [11], etc.

Pebble transducers and blind transducers. Two-way transducers can only describe functions whose output size is at most linear in the input size. A possible solution to overcome this limitation is to consider nested two-way transducers. In particular, the nested model of *pebble transducers* has been studied for a long time (see e.g. [10, 8]).

A *k-pebble transducer* is built by induction on $k \geq 1$. For $k = 1$, a 1-pebble transducer is just a two-way transducer. For $k \geq 2$, a k-pebble transducer is a two-way transducer that, when on any position i of its input word, can launch a $(k-1)$-pebble transducer. This submachine works on the original input where position i is marked by a "pebble". The original two-way transducer then outputs the concatenation of all the outputs returned by the submachines that it has launched along its computation. The intuitive behavior of a 3-pebble transducer is depicted in Figure 2. It can be seen as program with 3 nested loops. The class of word-to-word functions computed by k-pebble transducers for some $k \geq 1$ is



49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 120; pp. 120:1–120:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

known as *polyregular functions*. It has been quite intensively studied over the past few years due to its regular-like properties such as closure under composition [8], equivalence with logical interpretations [3] or other transducer models [2], etc.

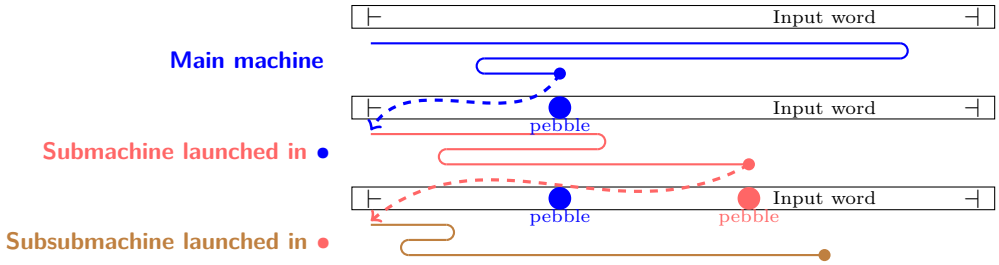


Figure 1 Behavior of a 3-pebble transducer.

A subclass of pebble transducers named *blind transducers* was recently introduced in [13]. For $k = 1$, a 1-blind transducer is just a two-way transducer. For $k \geq 2$, a k-blind transducer is a two-way transducer that can launch a $(k-1)$-blind transducer like a k-pebble transducer. However, there is no pebble marking the input of the submachine (i.e. it cannot see the position i from which it was called). The behavior of a 3-blind transducer is depicted in Figure 2. It can be seen as a program with 3 nested loops which cannot see the upper loop indexes. We call *polyblind functions* the class of functions computed by blind transducers. It is closed under composition and deeply related to lambda-calculus [12].

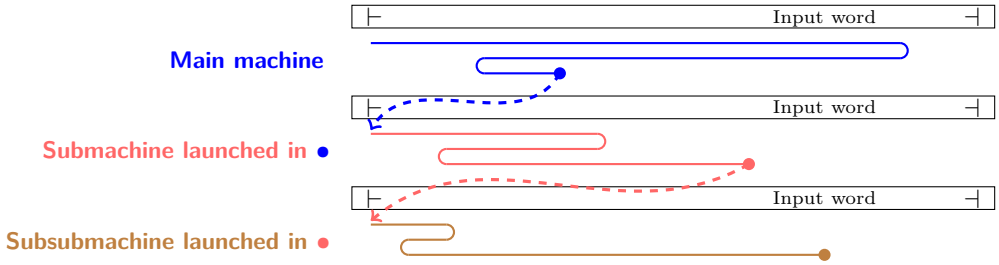


Figure 2 Behavior of a 3-blind transducer.

We study here polyregular and polyblind functions whose output alphabet is unary. Up to identifying a word with its length, we thus consider functions from finite words to $\mathbb{N}$. With this restriction, we show that one can decide if a polyregular function is polyblind, and connect these classes of functions to *rational series*.

Relationship with rational series. Rational series over the semiring $(\mathbb{N}, +, \times)$. are a well-studied class of functions from finite words to $\mathbb{N}$. They can be defined as the closure of (unary output) regular functions under sum $+$, Cauchy product $\copyright$ (product for formal power series) and Kleene star $*$ (iteration of Cauchy products). It is also well known that rational series are closed under Hadamard product $\circledH$ (component-wise product) [1].

The first result of this paper states that polyregular functions are exactly the subclass of rational series "without star", that is the closure of regular functions under $+$ and $\copyright$ ($\circledH$ can also be used but it is not necessary). This theorem is obtained by combining several former works. Our second result establishes that polyblind functions are exactly the closure of regular functions under $+$ and $\circledH$. It is shown in a self-contained way.

The aforementioned classes are depicted in Figure 3. All the inclusions are strict and this paper provides a few separating examples (some of them were already known in [6]).

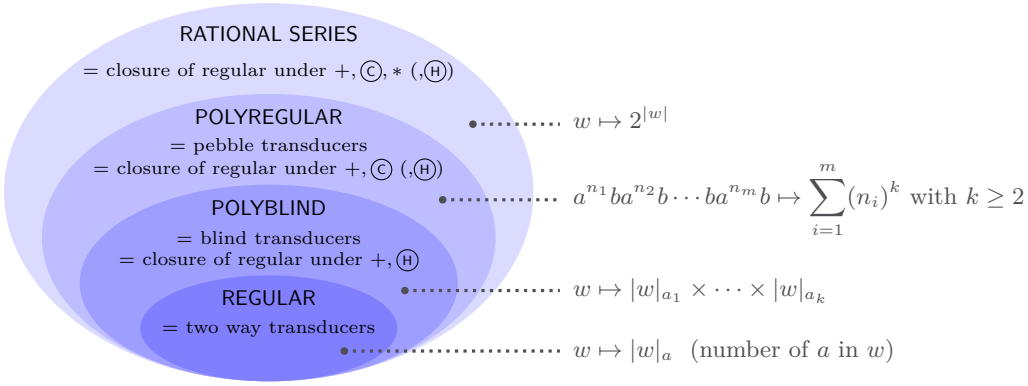


Figure 3 Classes of functions from finite words to $\mathbb{N}$ studied in this paper.

Class membership problems. We finally show how to decide whether a polyregular function with unary output is polyblind. It is by far the most involved and technical result of this paper. Furthermore, the construction of a blind transducer is effective, hence this result can be viewed as program optimization. Indeed, given a program with nested loops, our algorithm is able to build an equivalent program using "blind" loops if it exists.

In general, decision problems for transductions are quite difficult to solve, since contrary to regular languages, there are no known "canonical" objects (such as a minimal automaton) to represent (poly)regular functions. It is thus complex to decide an intrinsic property of a function, since it can be described in several seemingly unrelated manners. Nevertheless, the membership problem from rational series to polyregular (resp. regular) functions was shown to be decidable in [7, 6]. It is in fact equivalent to checking if the output of the rational series is bounded by a polynomial (resp. a linear function) in its input's length.

However, both polyregular and polyblind functions can have polynomial growth rates. To discriminate between them, we thus introduce the new notion of *repetitiveness* (which is a pumping-like property for functions) and show that it exactly captures the polyregular functions that are polyblind. The proof is a rather complex induction on the depth $k \geq 1$ of the k-pebble transducer representing the function. We show at the same time that repetitiveness is decidable and that a blind transducer can effectively be built whenever this property holds. Partial results were obtained in [6] to decide "blindness" of the functions computed by 2-pebble transducers. Some of our tools are inspired by this paper, such as the use of bimachines and factorization forests. Nevertheless, our general result requires new proof techniques (e.g. the induction techniques which insulate the term of "highest growth rate" in the function) and concepts (e.g. repetitiveness).

Outline. We first describe in Section 2 the notions of pebble and blind transducers. In the case of unary outputs, we recall the equivalent models of pebble, marble and blind bimachines introduced in [6]. These bimachines are easier to handle in the proofs, since they manipulate a monoid morphism instead of having two-way moves. In Section 3 we recall the definitions of sum $+$, Cauchy product Ⓒ, Hadamard product Ⓗ and Kleene star $*$ for rational series. We then show how to describe polyregular and polyblind functions with these operations. Finally, we claim in Section 4 that the membership problem from polyregular to polyblind functions is decidable. The proof of this technical result is sketched in sections 5 and 6. Due to space constraints we focus on the most significant lemmas.

2 Preliminaries

$\mathbb{N}$ is the set of nonnegative integers. If $i \leq j$, the set $[i{:}j]$ is $\{i, i{+}1, \dots, j\} \subseteq \mathbb{N}$ (empty if $j < i$). The capital letter A denotes an alphabet, i.e. a finite set of letters. The empty word is denoted by ε. If $w \in A^*$, let $|w| \in \mathbb{N}$ be its length, and for $1 \leq i \leq |w|$ let $w[i]$ be its i-th letter. If $I = \{i_1 < \cdots < i_\ell\} \subseteq [1{:}|w|]$, let $w[I] \coloneqq w[i_1] \cdots w[i_\ell]$. If $a \in A$, let $|w|_a$ be the number of letters a occurring in w. We assume that the reader is familiar with the basics of automata theory, in particular one-way and two-way automata, and monoid morphisms.

Two-way transducers. A deterministic two-way transducer is a deterministic two-way automaton (with input in A^*) enhanced with the ability to produce outputs (from B^*) when performing a transition. The output of the transducer is defined as the concatenation of these productions along the unique accepting run on the input word (if it exists): it thus describes a (partial) function $A^* \to B^*$. Its behavior is depicted in Figure 4. A formal definition can be found e.g. in [6]. These machines compute the class of *regular functions*.

Figure 4 Behavior of a two-way transducer.

▶ **Example 2.1.** The function $a_1 \cdots a_n \mapsto a_1 \cdots a_n \# a_n \cdots a_1$ can be computed by a two-way transducer which reads its input word from left to right and then from right to left.

From now on, the output alphabet B of our machines will always be a singleton. By identifying B^* and $\mathbb{N}$, we assume that the functions computed have type $A^* \to \mathbb{N}$.

▶ **Example 2.2.** Given $a \in A$, the function $\mathsf{nb}_a : A^* \to \mathbb{N}, w \mapsto |w|_a$ is regular.

Blind and pebble transducers. Blind and pebble transducers extend two-way transducers by allowing to "nest" such machines. A 1-blind (resp. 1-pebble) transducer is just a two-way transducer. For $k \geq 2$, a k-blind (resp. k-pebble) transducer is a two-way transducer which, when performing a transition from a position $1 \leq i \leq |w|$ of its input $w \in A^*$, can launch a $(k{-}1)$-blind (resp. $(k{-}1)$-pebble) transducer with input w (resp. $w[1{:}i{-}1]\underline{w[i]}w[i{+}1{:}|w|]$ i.e. w where position i is marked). The two-way transducer then uses the output of this submachine as if it was the output produced along its transition. The intuitive behaviors are depicted for $k = 3$ in figures 1 and 2. Formal definitions can be found e.g. in [13, 6].

▶ **Example 2.3.** Let $a_1, \dots, a_k \in A$, then $\mathsf{nb}_{a_1,\dots,a_k} : w \mapsto |w|_{a_1} \times \cdots \times |w|_{a_k}$ can be computed by a k-blind transducer. The main transducer processes its input from left to right, and it calls inductively a $(k{-}1)$-blind transducer for $\mathsf{nb}_{a_1,\dots,a_{k-1}}$ each time it it sees an a_k.

▶ **Example 2.4.** The function $2\text{-}\mathsf{powers} : a^{n_1} b \cdots a^{n_m} b \mapsto \sum_{i=1}^{m} n_i^2$ can be computed by a 2-pebble transducer. Its main transducer ranges over all the a of the input, and calls a 1-pebble (= two-way) transducer for each a, which produces n_i if the a is in the i-th block (it uses the pebble to detect which block is concerned). Similarily, the function $k\text{-}\mathsf{powers} : a^{n_1} b \cdots a^{n_m} b \mapsto \sum_{i=1}^{m} n_i^k$ for $k \geq 1$ can be computed by a k-pebble transducer.

▶ **Definition 2.5.** *We define the class of* polyregular functions *(resp.* polyblind functions*) as the class of functions computed by a k-pebble (resp. k-blind) transducer for some $k \geq 1$.*

It is not hard to see that polyblind functions are a subclass of polyregular functions. Indeed, a blind transducer is just a pebble transducer "without pebbles".

Bimachines. In this paper, we shall describe formally the regular, polyregular and polyblind functions with another computation model. A *bimachine* is a transducer which makes a single left-to-right pass on its input, but it can use a morphism into a finite monoid to check regular properties of the prefix (resp. suffix) ending (resp. starting) in the current position. This notion of bimachines enable us to easily use algebraic techniques in the proofs, and in particular *factorization forests* over finite monoids.

▶ **Definition 2.6.** *A bimachine* $\mathcal{M} \coloneqq (A, M, \mu, \lambda)$ *is:*

- *an input alphabet* A*, a finite monoid* M *and a monoid morphism* $\mu : A^* \to M$*;*
- *an output function* $\lambda : M \times A \times M \to \mathbb{N}$*.*

$\mathcal{M}$ computes $f : A^* \to \mathbb{N}$ defined by $f(w) \coloneqq \sum_{i=1}^{|w|} \lambda(\mu(w[1{:}i{-}1]), w[i], \mu(w[i{+}1{:}|w|]))$ for $w \in A^*$. The production of each term of this sum is depicted intuitively in Figure 5.

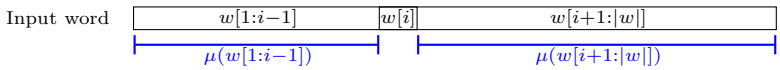


Figure 5 Behavior of a bimachine when producing $\lambda(\mu(w[1{:}i{-}1]), w[i], \mu(w[i{+}1{:}|w|]))$.

When dealing with bimachines, we consider without loss of generality total functions such that $f(\varepsilon) = 0$ (the domains of two-way transducers are regular languages [14], and the particular image of ε does not matter). In this context, it is well known that regular functions of type $A^* \to \mathbb{N}$ are exactly those computed by bimachines (in other words, it means that regular functions and rational functions are the same). Now we recall how [6] has generalized this result to k-blind and k-pebble transducers. Intuitively, a *bimachine with external functions* is a bimachine enriched with the possibility to launch a submachine for each letter of the input (it outputs the sum of all the outputs returned by these submachines).

▶ **Definition 2.7** ([6])**.** *A* bimachine with external pebble (*resp.* external blind*, resp.* external marble) functions $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ *consists of:*

- *an input alphabet* A*;*
- *a finite monoid* M *and a morphism* $\mu : A^* \to M$*;*
- *a finite set* $\mathfrak{H}$ *of external functions* $\mathfrak{h} : (A \uplus \underline{A})^* \to \mathbb{N}$ *(resp.* $A^* \to \mathbb{N}$*, resp.* $A^* \to \mathbb{N}$*);*
- *an output function* $\lambda : M \times A \times M \to \mathfrak{H}$*.*

Given $1 \leq i \leq |w|$ a position of $w \in A^*$, let $\mathfrak{h}_i \coloneqq \lambda(\mu(w[1{:}i{-}1]), w[i], \mu(w[i{+}1{:}|w|])) \in \mathfrak{H}$. A bimachine $\mathcal{M}$ with external pebble functions computes a function $f : A^* \to \mathbb{N}$ defined by $f(w) \coloneqq \sum_{1 \leq i \leq |w|} \mathfrak{h}_i(w[1{:}i{-}1]\underline{w[i]}w[i{+}1{:}|w|])$. Intuitively, it means that for each position $1 \leq i \leq |w|$, $\mathcal{M}$ calls an external function $\mathfrak{h}_i$ (depending on a regular property of $w[1{:}i{-}1]$,$w[i]$ and $w[i{+}1{:}|w|]$) with input $w[1{:}i{-}1]\underline{w[i]}w[i{+}1{:}|w|]$ (that is "w with a pebble on position i"), and uses the result $\mathfrak{h}_i(w[1{:}i{-}1]\underline{w[i]}w[i{+}1{:}|w|])$ of this function in its own output.

For a bimachine with external blind (resp. marble) functions, the output is defined by $f(w) \coloneqq \sum_{1 \leq i \leq |w|} \mathfrak{h}_i(w)$ (resp. $f(w) \coloneqq \sum_{1 \leq i \leq |w|} \mathfrak{h}_i(w[1{:}i])$). In this case $\mathcal{M}$ calls $\mathfrak{h}_i$ with argument w (resp. the prefix of w ending in position i).

▶ **Definition 2.8** ([6])**.** *Given* $k \geq 1$*, a* k*-pebble (resp.* k*-blind, resp.* k*-marble) bimachine is:*

- *for* $k = 1$*, a bimachine (withtout external functions, see Definition 2.6);*
- *for* $k \geq 2$*, it a bimachine with external pebble (resp. external blind, resp. external marble) functions (see Definition 2.7) which are computed by* $(k{-}1)$*-pebble (resp.* $(k{-}1)$*-blind, resp.* $(k{-}1)$*-marble) bimachines. These* $(k{-}1)$*-bimachines are implicitly fixed and given by the external functions of the main bimachine.*

▶ Remark 2.9. For pebble bimachines, a natural question is whether the inner bimachines can ask which pebble was dropped by which ancestor, or whether they can only see that "there is a pebble". Both models are in fact equivalent, since the number of pebbles is bounded.

Now we recall in Theorem 2.10 that pebble and blind bimachines are respectively equivalent to the aforementioned pebble and blind transducers. More interestingly, the marble bimachines (which call "prefixes") also correspond to pebble transducers. In our proofs, we shall preferentially use the marble model to represent polyregular functions.

▶ **Theorem 2.10** ([6]). *For all $k \geq 1$, k-pebble transducers, k-pebble bimachines and k-marble bimachines compute the same class of functions. Furthermore, k-blind transducers and k-blind bimachines compute the same class of functions.*

3 From pebbles to rational series

The class of *rational series* (which are total functions $A^* \to \mathbb{N}$) is the closure of regular functions under sum, Cauchy product and Kleene star. It is well known that it can be described by *weighted automata*, furthermore this class is closed under Hadamard product (see e.g. [1, Theorem 5.5]). Let us recall the definition of these operations for $f, g : A^* \to \mathbb{N}$:

- the sum $f + g : w \mapsto f(w) + g(w)$;
- the Cauchy product $f \copyright g : w \mapsto \sum_{i=0}^{|w|} f(w[1{:}i])g(w[i{+}1{:}|w|])$;
- the Hadamard product $f \hadamard g : w \mapsto f(w)g(w)$;
- if and only if $f(\varepsilon) = 0$, the Kleene star $f^* := \sum_{n \geq 0} f^n$ where $f^0 : \varepsilon \mapsto 1, u \neq \varepsilon \mapsto 0$ is neutral for Cauchy product and $f^{n+1} := f \copyright f^n$.

▶ **Example 3.1.** In Example 2.3 we have $\mathsf{nb}_{a_1,\dots,a_k} = \mathsf{nb}_{a_1} \hadamard \cdots \hadamard \mathsf{nb}_{a_k}$.

▶ **Example 3.2.** Let $f, g : \{a, b\}^* \to \mathbb{N}$ defined by $f(wa) = 2$ for $w \in A^*$ and $f(w) = 0$ otherwise; $g(a^n bw) = n$ for $w \in A^*$ and $g(w) = 0$ otherwise. It is easy to see that $f \copyright g(a^{n_1} b \cdots a^{n_m} b) = \sum_{i=1}^{m} n_i(n_i{-}1)$. Hence $2\text{-}\mathsf{powers} = \mathsf{nb}_a + f \copyright g$ (see Example 2.4).

We are now ready to state the first main results of this paper. The first one shows that polyregular functions correspond to the subclass of rational series where the use of star is disallowed (and it also corresponds to rational series whose "growth" is polynomial).

▶ **Theorem 3.3.** *Let $f : A^* \to \mathbb{N}$, the following conditions are (effectively) equivalent:*

1. *f is polyregular;*
2. *f is a rational series with polynomial growth, i.e. $f(w) = \mathcal{O}(|w|^k)$ for some $k \geq 1$;*
3. *f belongs to the closure of regular functions under sum and Cauchy product;*
4. *f belongs to the closure of regular functions under sum, Cauchy and Hadamard products.*

Proof sketch. $1 \Leftrightarrow 2$ follows from [7, 6]. For $2 \Rightarrow 3$, it is shown in [1, Exercise 1.2 of Chapter 9] that the rational series of polynomial growth can be obtained by sum and Cauchy products from the characteristic series of rational languages (which are regular functions). Having $3 \Rightarrow 4$ is obvious. Finally for $4 \Rightarrow 2$, it suffices to note that regular functions have polynomial growth, and this property is preserved by $+, \copyright$ and $\hadamard$. ◀

Note that Hadamard products do not increase the expressive power of this class. However, removing Cauchy products gives polyblind functions, as shown in Theorem 3.4.

▶ **Theorem 3.4.** *Let $f : A^* \to \mathbb{N}$, the following conditions are equivalent:*

1. *f is polyblind;*
2. *f belongs to the closure of regular functions under sum and Hadamard product.*

Proof sketch. We first show $1 \Rightarrow 2$ by induction on $k \geq 1$ when f is computed by a k-blind bimachine. For $k = 1$, it is obvious. Let us describe the induction step from $k \geq 1$ to $k+1$. Let f be computed by a bimachine $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$, with external blind functions computed by k-blind bimachines. Given $\mathfrak{h} \in \mathfrak{H}$, let $f'_{\mathfrak{h}}$ be the function which maps $w \in A^*$ to the cardinal $|\{1 \leq i \leq |w| : \lambda(\mu(w[1{:}i{-}1]), w[i], \mu(w[i{+}1{:}|w|])) = \mathfrak{h}\}|$. Then $f'_{\mathfrak{h}}$ is a regular function and $f = \sum_{\mathfrak{h} \in \mathfrak{H}} f'_{\mathfrak{h}} \circledH \mathfrak{h}$. The result follows by induction hypothesis.

For $2 \Rightarrow 1$, we show that functions computed by blind bimachines are closed under sum and Hadamard product. For the sum, we use the blind transducer model to simulate successively the computation of the two terms of a sum. For the Hadamard product, we show by induction on $k \geq 1$ that if f is computed by a k-blind bimachine and g is polyblind, then $f \circledH g$ is polyblind. If $k = 1$, we transform the bimachine for f in a blind bimachine computing $f \circledH g$ by replacing each output $n \in \mathbb{N}$ by a call to a machine computing $n \times g$. If $k \geq 2$, using the notations of the previous paragraph we have $f = \sum_{\mathfrak{h} \in \mathfrak{H}} f'_{\mathfrak{h}} \circledH \mathfrak{h}$, thus $f \circledH g = \sum_{\mathfrak{h} \in \mathfrak{H}} f'_{\mathfrak{h}} \circledH (\mathfrak{h} \circledH g)$. By induction hypothesis, each $\mathfrak{h} \circledH g$ is polyblind. Hence we compute $f \circledH g$ by replacing in $\mathcal{M}$ each external function $\mathfrak{h}$ by the function $\mathfrak{h} \circledH g$. ◀

We conclude this section by recalling that polyregular (resp. polyblind) functions with unary output enjoy a "pebble minimization" property, which allows to reduce the number of nested layers depending on the growth rate of the output.

▶ **Definition 3.5.** *We say that a function $f : A^* \to \mathbb{N}$ has* growth at most k *if $f(w) = \mathcal{O}(|w|^k)$.*

▶ **Theorem 3.6** ([7, 6, 13])**.** *A a polyregular (resp. polyblind) function f can be computed by a k-pebble (resp. k-blind) transducer if and only it has growth at most k. Furthermore this property can be decided and the construction is effective.*

▶ Remark 3.7. The result also holds for blind transducers with non-unary outputs [13]. However, it turns out to be false for pebble transducers with non-unary outputs (unpublished work of the author of [2]).

In the next section, we complete the decidability picture by solving the membership problem from polyregular to polyblind functions.

4 Membership problem from polyregular to polyblind

In this section, we state the most technical and interesting result of this paper, which consists in deciding if a polyregular function is polyblind. We also give in Theorem 4.6 a semantical characterization of polyregular functions which are polyblind, using the notion of *repetitiveness*. Intuitively, a k-repetitive function is a function which, when given two places in a word where the same factor is repeated, cannot distinguish between the first and the second place. Hence its output only depends on the total number of repetitions of the factor.

▶ **Definition 4.1** (Repetitive function)**.** *Let $k \geq 1$. We say that $f : A^* \to \mathbb{N}$ is* k-repetitive *if there exists $\eta \geq 1$, such that the following holds for all $\alpha, \alpha_0, u_1, \alpha_1, \ldots, u_k, \alpha_k, \beta \in A^*$ and $\omega \geq 1$ multiple of η. Let $W : \mathbb{N}^k \to A^*$ defined by:*

$$W : X_1, \ldots, X_k \mapsto \left(\alpha_0 \prod_{i=1}^{k} {u_i}^{\omega X_i} \alpha_i \right).$$

and let $w \coloneqq W(1, \ldots, 1)$. Then there exists a function $F : \mathbb{N}^k \to \mathbb{N}$ such that for all $\overline{X} \coloneqq X_1, \ldots, X_k \geq 3$ and $\overline{Y} \coloneqq Y_1, \ldots, Y_k \geq 3$, we have:

$$f(\alpha w^{2\omega-1} W(\overline{X}) w^{\omega-1} W(\overline{Y}) w^{\omega} \beta) = F(X_1 + Y_1, \ldots, X_k + Y_k).$$

▶ Remark 4.2. If f is k-repetitive, f is also $(k-1)$-repetitive, by considering an empty u_k.

Let us now give a few examples in order so see when this criterion holds, or not.

▶ **Example 4.3.** The function $\mathsf{nb}_{a_1,\dots,a_k}$ (see Example 2.3) is k-repetitive for all $k \geq 1$.

▶ **Example 4.4.** If $f : A^* \to \mathbb{N}$ where $A = \{a\}$ is a singleton, then f is k-repetitive for all $k \geq 1$. Indeed, $\alpha w^{2\omega-1} W(\overline{X}) w^{\omega-1} W(\overline{Y}) w^\omega \beta \in A^*$ is itself a function of $X_1+Y_1, \dots, X_k+Y_k$, hence so is its image $f(\alpha w^{2\omega-1} W(\overline{X}) w^{\omega-1} W(\overline{Y}) w^\omega)$.

▶ **Example 4.5.** For all $k \geq 2$, the function $k\text{-powers} : a^{n_1} b \cdots a^{n_m} b \mapsto \sum_{i=1}^m n_i^k$ is not 1-repetitive. Let us choose any $\omega \geq 1$ and fix $\alpha = \beta := \varepsilon$, $u_1 := a$ and $\alpha_0 = \alpha_1 := b$, then:

$$\begin{aligned} k\text{-powers}(W(X_1, Y_1)) &= k\text{-powers}\left((ba^\omega b)^{2\omega-1} b a^{\omega X_1} b (ba^\omega b)^{\omega-1} b a^{\omega Y_1} b (ba^\omega b)^\omega\right) \\ &= \omega^k(4\omega - 2) + \omega^k X_1^k + \omega^k Y_1^k \end{aligned}$$

which is *not* a function of $X_1 + Y_1$ for $k \geq 2$.

We are now ready to state our main result of this paper.

▶ **Theorem 4.6.** *Let $k \geq 1$ and $f : A^* \to \mathbb{N}$ be computed by a k-pebble transducer. Then f is polyblind if and only if it is k-repetitive. Furthermore, this property can be decided and one can effectively build a blind transducer for f when it exists.*

▶ Remark 4.7. By Theorem 3.6, we can even build a k-blind transducer.

Theorem 4.6 provides a tool to show that some polyregular functions are not polyblind:

▶ **Example 4.8.** By Example 4.5, the polyregular function $k\text{-powers}$ is not k-repetitive for $k \geq 2$. Therefore it is not polyblind. Also note that it is computable by a k-pebble transducer, but not by an ℓ-pebble transducer for $\ell < k$ (Theorem 3.6).

An immediate consequence of Theorem 4.6 is obtained below for functions which have both a unary output alphabet and a unary input alphabet. This result was already obtained by the authors of [13] using other techniques, in an unpublished work.

▶ **Corollary 4.9.** *Polyblind and polyregular functions over a unary input alphabet coincide.*

Proof. Polyregular functions with unary input are k-repetitive by Example 4.4. ◀

5 Repetitive functions and permutable bimachines

The rest of this paper is devoted to the proof of Theorem 4.6. Since k-pebble transducers and k-marble bimachines compute the same functions (Theorem 2.10), it follows directly from Theorem 5.1 below (the notions used are defined in the next sections).

From now on, we assume that a k-marble bimachine uses the same morphism $\mu : A^* \to M$ in its main bimachine, and inductively in all its submachines computing the external functions. We do not lose any generality here, since it is always possible to get this situation by taking the product of all the morphisms used. We also assume that μ is surjective (we do no lose any generality, since it is possible to replace M by the image $\mu(M)$).

▶ **Theorem 5.1.** *Let $k \geq 2$ and $f : A^* \to \mathbb{N}$ be computed by $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ a k-marble bimachine. The following conditions are equivalent:*

1. *f is polyblind;*
2. *f is k-repetitive;*
3. *$\mathcal{M}$ is $2^{3|M|}$-permutable (see Definition 5.11) and the function f'' (built from Proposition 6.1) is polyblind.*

Furthermore this property is decidable and the construction is effective.

Let us now give the skeleton of the proof of Theorem 5.1, which is rather long and involved. The propositions on which it relies are shown in the two following sections.

Proof of Theorem 5.1. $1 \Rightarrow 2$ follows from Proposition 5.2. For $2 \Rightarrow 3$, if f is k-repetitive then $\mathcal{M}$ is $2^{3|M|}$-permutable by Proposition 5.12. Then Proposition 6.1 gives $f = f' + f''$ where f' is polyblind and f'' is computable by a $(k-1)$-marble bimachine. By Proposition 5.2, f' is $(k-1)$-repetitive. Since f is also $(k-1)$-repetitive, then $f'' = f - f'$ is also $(k-1)$-repetitive by Claim 5.3. Thus by induction hypothesis f'' is polyblind. For $3 \Rightarrow 1$, since in Proposition 6.1 we have $f = f' + f''$ where f' and f'' are polyblind, then f is (effectively) polyblind. The decidability is also obtained by induction: one has to check that $\mathcal{M}$ is $2^{3|M|}$-permutable (which is decidable) and inductively that f'' is polyblind. ◀

Now we present the tools used in this proof. We first show that polyblind functions are repetitive. Then we introduce the notion of *permutability*, and show that repetitive functions are computed by permutable marble bimachines.

5.1 Polyblind functions are repetitive

Intuitively, a blind transducer cannot distinguish between two "similar" iterations of a given factor in a word, since it cannot drop a pebble for doing so. We get the following using technical but conceptually easy pumping arguments.

▶ **Proposition 5.2.** *A polyblind function is k-repetitive for all $k \geq 1$.*

Proof idea. We first show that a regular function (computed by a bimachine without external functions) is k-repetitive for all $k \geq 1$. In this case, η is chosen as the idempotent index of the monoid used by the bimachine. Then, it is easy to conclude by noting that k-repetitiveness is preserved under sums and Hadamard products. ◀

The induction which proves Theorem 5.1 also requires the following result. Its proof is obvious since k-repetitiveness is clearly preserved under subtractions.

▷ Claim 5.3. If f, g are k-repetitive and $f - g \geq 0$, then it is k-repetitive.

5.2 Repetitive functions are computed by permutable machines

In this subsection, we show that k-marble bimachines which compute k-repetitive functions have a specific property named *permutability* (which turns out to be decidable).

Productions. We first introduce the notion of *production*. In the rest of this paper, the notation $\{\!\{\cdots\}\!\}$ represents a multiset (i.e. a set with multiplicities). If S is a set and m is a multiset, we write $m \subseteq S$ to say that each element of m belongs to S (however, there can be multiplicities in m but not in S). For instance $\{\!\{1, 1, 2, 3, 3\}\!\} \subseteq \mathbb{N}$.

▶ **Definition 5.4** (Production). *Let $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ be a k-marble bimachine, $w \in A^*$. We define the* production *of $\mathcal{M}$ on $\{\!\{i_1, \dots, i_k\}\!\} \subseteq [1{:}|w|]$ as follows if $i_1 \leq \cdots \leq i_k$:*

- *if $k = 1$, $\mathsf{prod}^w_{\mathcal{M}}(\{\!\{i_1\}\!\}) \coloneqq \lambda(\mu(w[1{:}i_1{-}1]), w[i_1], \mu(w[i_1{+}1{:}|w|])) \in \mathbb{N}$;*
- *if $k \geq 2$, let $\mathfrak{h} \coloneqq \lambda(\mu(w[1{:}i_k{-}1]), w[i_k], \mu(w[i_k{+}1{:}|w|])) \in \mathfrak{H}$ and $\mathcal{M}_{\mathfrak{h}}$ be the $(k{-}1)$-marble bimachine computing $\mathfrak{h}$. Then $\mathsf{prod}^w_{\mathcal{M}}(\{\!\{i_1, \cdots, i_k\}\!\}) \coloneqq \mathsf{prod}^{w[1:i_k]}_{\mathcal{M}_{\mathfrak{h}}}(\{\!\{i_1, \dots, i_{k-1}\}\!\})$.*

The value $\mathsf{prod}^w_{\mathcal{M}}(\{\!\{i_1, \cdots, i_k\}\!\})$ with $i_1 \leq \cdots \leq i_k$ thus corresponds to the value output when doing a call on position i_k, then on i_{k-1}, etc. Now let $\{\!\{I_1, \dots, I_k\}\!\}$ be a multiset of sets of positions of w (i.e. for all $1 \leq i \leq k$ we have $I_i \subseteq [1{:}|w|]$). We define the production of $\mathcal{M}$ on $\{\!\{I_1, \dots, I_k\}\!\}$ as the combination of all possible productions on positions among these sets:

$$\mathsf{prod}^w_{\mathcal{M}}(\{\!\{I_1, \dots, I_k\}\!\}) \coloneqq \sum_{\substack{\{\!\{i_1,\dots,i_k\}\!\} \subseteq [1:|w|] \\ \text{with } i_j \in I_j \text{ for } 1 \leq j \leq k}} \mathsf{prod}^w_{\mathcal{M}}(\{\!\{i_1, \dots, i_k\}\!\}).$$

▶ Remark 5.5. Note that we no longer have $i_1 \leq \cdots \leq i_k$.

By rewriting the sum which defines the function f from $\mathcal{M}$, we get the following.

▶ **Lemma 5.6.** *Let $\mathcal{M}$ be a k-marble bimachine computing a function $f : A^* \to \mathbb{N}$. Let $w \in A^*$ and $J_1, \dots J_n$ be a partition of $[1{:}|w|]$:*

$$f(w) = \sum_{\{\!\{I_1,\dots,I_k\}\!\} \subseteq \{J_1,\dots,J_n\}} \mathsf{prod}^w_{\mathcal{M}}(\{\!\{I_1, \dots, I_k\}\!\}).$$

Following the definition of bimachines, the production $\mathsf{prod}^w_{\mathcal{M}}(\{\!\{i_1, \cdots, i_k\}\!\})$ should only depend on $w[i_1], \dots, w[i_k]$ and of the image under μ of the factors between these positions. Now we formalize this intuition in a more general setting.

▶ **Definition 5.7** (Multicontext). *Given $x \geq 0$, an x-*multicontext *consists of two sequences of words $v_0, \cdots, v_x \in A^*$ and $u_1, \cdots, u_x \in A^*$. It is denoted $v_0[\![u_1]\!]v_1 \cdots [\![u_x]\!]v_x$.*

Let $w \coloneqq v_0u_1 \cdots u_kv_k \in A^*$. For $1 \leq i \leq k$, let $I_i \subseteq [1{:}|w|]$ be the set of positions corresponding to u_i. We define the production on the multicontext $v_0[\![u_1]\!]v_1 \cdots [\![u_k]\!]v_k$ by:

$$\mathsf{prod}_{\mathcal{M}}(v_0[\![u_1]\!]v_1 \cdots [\![u_k]\!]v_k) \coloneqq \mathsf{prod}^w_{\mathcal{M}}(\{\!\{I_1, \dots, I_k\}\!\}). \tag{1}$$

As stated in Proposition-Definition 5.8, this quantity only depends on the u_i and the images of the v_i under the morphism μ of $\mathcal{M}$, which leads to a new notion of productions.

▶ **Proposition-Definition 5.8.** *Let $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ be a k-marble bimachine. Let $v_0u_1 \cdots u_kv_k \in A^*$ and $v'_0u_1 \cdots u_kv'_k \in A^*$ be such that $\mu(v_i) = \mu(v'_i)$ for all $0 \leq i \leq k$. Then: $\mathsf{prod}_{\mathcal{M}}(v_0[\![u_1]\!]v_1 \cdots [\![u_k]\!]v_k) = \mathsf{prod}_{\mathcal{M}}(v'_0[\![u_1]\!]v'_1 \cdots [\![u_k]\!]v'_k)$. Let $m_i \coloneqq \mu(v_i) = \mu(v'_i)$, we define $\mathsf{prod}_{\mathcal{M}}(m_0[\![u_1]\!]m_1 \cdots [\![u_k]\!]m_k)$ as the previous value.*

▶ Remark 5.9. In the following, we shall directly manipulate multicontexts of the form $m_0[\![u_1]\!]m_1 \cdots [\![u_x]\!]m_x$ with $m_i \in M$ and $u_i \in A^*$. Note that an x-multicontext and a y-multicontext can be concatenated to obtain an $(x + y)$-multicontext.

We finally introduce the notion of (x, K)-*iterator*, which corresponds to an x-multicontext in which the words u_i have length at most $K \geq 0$ and have idempotent images under μ. Recall that $e \in M$ is *idempotent* if and only if $ee = e$.

▶ **Definition 5.10** (Iterator). *Let $x, K \geq 0$ and $\mu : A^* \to M$. An (x, K)-*iterator *for μ is an x-multicontext of the form $m_0 (\prod_{i=1}^{x} e_i[\![u_i]\!]e_im_i)$ such that $m_0, \dots, m_x \in M$ and $u_1, \dots, u_x \in A^*$ are such that for all $1 \leq i \leq x$, $|u_i| \leq K$ and $e_i = \mu(u_i) \in M$ is idempotent.*

Permutable k-marble bimachines. We are now ready to state the definition of *permutability* for marble bimachines. Intuitively, this property means that, under some idempotency conditions, $\mathsf{prod}_{\mathcal{M}}(m_0[\![u_1]\!]m_1 \cdots [\![u_k]\!]m_k)$ only depends on the 1-multicontexts of each $[\![u_i]\!]$, which are $m_0\mu(u_1)\cdots m_i[\![u_i]\!]m_{i+1}\mu(u_{i+1})\cdots m_k \in M$ for $1 \leq i \leq k$. In particular, it does not depend on the relative position of the u_i nor on the m_i which separate them. Hence, it will be possible to simulate a permutable k-marble bimachine by a k-blind bimachine which can only see the 1-multicontext of one position at each time.

▶ **Definition 5.11.** *Let $\mathcal{M}$ be a k-marble bimachine using a surjective morphism $\mu : A^* \to M$. Let $K \geq 0$, we say that $\mathcal{M}$ is* K-permutable *if the following holds whenever $\ell + x + r = k$:*

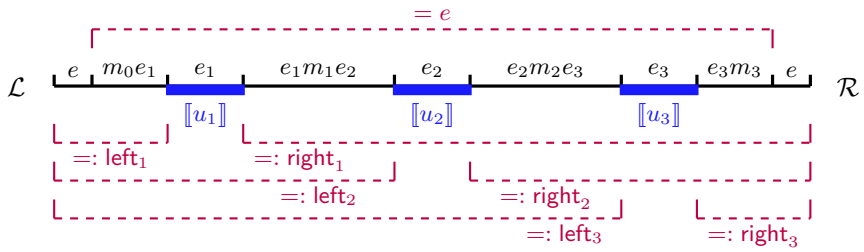
- *for all (ℓ, K)-iterator $\mathcal{L}$ and (r, K)-iterator $\mathcal{R}$;*
- *for all (x, K)-iterator $m_0\left(\prod_{i=1}^{x} e_i[\![u_i]\!]e_i m_i\right)$ such that $e := m_0\left(\prod_{i=1}^{x} e_i m_i\right)$ idempotent;*
- *for all $1 \leq j \leq x$, define the left and right contexts:*

$$\mathsf{left}_j := e\left(\prod_{i=1}^{j} m_{i-1}e_i\right) \quad \text{and} \quad \mathsf{right}_j := \left(\prod_{i=j}^{x} e_i m_i\right)e.$$

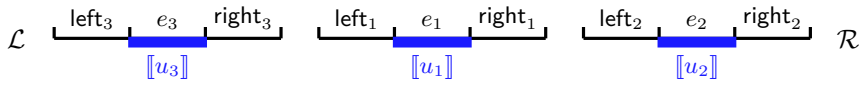
Then if σ is a permutation of $[1{:}x]$, we have:

$$\mathsf{prod}_{\mathcal{M}}\left(\mathcal{L}\; em_0\left(\prod_{i=1}^{x} e_i[\![u_i]\!]e_i m_i\right)e\;\mathcal{R}\right) = \mathsf{prod}_{\mathcal{M}}\left(\mathcal{L}\left(\prod_{i=1}^{x} \mathsf{left}_{\sigma(i)}\,[\![u_{\sigma(i)}]\!]\,\mathsf{right}_{\sigma(i)}\right)\mathcal{R}\right).$$

An visual description of permutability is depicted in Figure 6.



(a) Initial multicontext and definition of the left_j / right_j for $1 \leq j \leq 3$.



(b) Multicontext which separates the factors using the left_j / right_j and σ.

Figure 6 Productions which must be equal in Definition 5.11, with $x = 3$ and $\sigma = (3, 1, 2)$.

The next result follows from a technical proof based on iteration techniques (it can be understood as a kind of pumping lemma on iterators).

▶ **Proposition 5.12.** *Let $\mathcal{M}$ be a k-marble bimachine computing a k-repetitive function. Then $\mathcal{M}$ is K-permutable for all $K \geq 0$.*

Let us finally note that being K-permutable (for a fixed K) is a decidable property. Indeed, it suffices to range over all $\ell + x + r = k$ and (ℓ, K)-, (x, K)- and (r, K)-iterators (there are finitely many of them, since they correspond to bounded sequences which alternate between monoid elements and words of bounded lengths), and compute their productions.

6 From permutable bimachines to polyblind functions

The purpose of this section is to show Proposition 6.1, which allows us to perform the induction step in the proof of Theorem 5.1, by going from k to $k-1$ marbles.

▶ **Proposition 6.1.** *Let $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ be a $2^{3|M|}$-permutable k-marble bimachine computing a function $f : A^* \to \mathbb{N}$. One can build a polyblind function $f' : A^* \to \mathbb{N}$ and a function $f''A^* \to \mathbb{N}$ computed by a $(k-1)$-marble bimachine such that $f = f' + f''$.*

Proof overview. It follows from Equation 2 and Lemma 6.14 that:

$$\begin{aligned} f &= (\mathsf{sum\text{-}dep}_{\mathcal{M}} + \mathsf{sum\text{-}ind}_{\mathcal{M}}) \circ \mathsf{forest}_{\mu} \quad \text{(see Theorem 6.5 for the definition of } \mathsf{forest}_{\mu}\text{)} \\ &= \underbrace{\mathsf{sum\text{-}ind}'_{\mathcal{M}} \circ \mathsf{forest}_{\mu}}_{=:f'} + \underbrace{(\mathsf{sum\text{-}dep}_{\mathcal{M}} + \mathsf{sum\text{-}ind}''_{\mathcal{M}}) \circ \mathsf{forest}_{\mu}}_{=:f''} \end{aligned}$$

By Lemma 6.14 one has $f'' \geq 0$. Furthermore, $\mathsf{sum\text{-}ind}'_{\mathcal{M}}$ is polyblind, hence f' is polyblind since the class of polyblind functions is closed under pre-composition by regular functions (even when the outputs are not unary, see [13]). Similarly, it follows from lemmas 6.12 and 6.14 that $\mathsf{sum\text{-}dep}_{\mathcal{M}}$ and $\mathsf{sum\text{-}ind}''_{\mathcal{M}}$ are polyregular with growth at most $k-1$ (see Definition 3.5), hence so is their sum and its pre-composition by a regular function [2]. Thus f'' is polyregular and has growth at most $k-1$. By theorems 2.10 and 3.6, it can be computed by a $(k-1)$-marble bimachine. ◀

Now we give the statements and the proofs of lemmas 6.12 and 6.14. An essential tool is the notion of *factorization forest*, which is recalled below.

6.1 Factorization forests

If $\mu : A^* \to M$ is a morphism into a finite monoid and $w \in A^*$, a *factorization forest* (called *μ-forest* in the following) of w is an unranked tree structure defined as follows.

▶ **Definition 6.2** (Factorization forest [15])**.** *Given a morphism $\mu : A^* \to M$ and $w \in A^*$, we say that $\mathcal{F}$ is a* μ-forest *of w if:*

- *either $\mathcal{F} = a$ and $w = a \in A$;*
- *or $\mathcal{F} = \langle \mathcal{F}_1 \rangle \cdots \langle \mathcal{F}_n \rangle$, $w = w_1 \cdots w_n$ and for all $1 \leq i \leq n$, $\mathcal{F}_i$ is a μ-forest of $w_i \in A^+$. Furthermore, if $n \geq 3$ then $\mu(w_1) = \cdots = \mu(w_n)$ is an idempotent of M.*

▶ Remark 6.3. If $\langle \mathcal{F}_1 \rangle \cdots \langle \mathcal{F}_n \rangle$ is a μ-forest, then so is $\langle \mathcal{F}_i \rangle \langle \mathcal{F}_{i+1} \rangle \ldots \langle \mathcal{F}_j \rangle$ for $1 \leq i \leq j \leq n$. The empty forest ε is the unique μ-forest of the empty word ε.

We shall use the standard tree vocabulary of "height" (a leaf is a tree of height 1), "child", "sibling", "descendant" and "ancestor" (both defined in a non-strict way: a node is itself one of its ancestors/descendants), etc. We denote by $\mathsf{Nodes}(\mathcal{F})$ the set of nodes of $\mathcal{F}$. In order to simplify the statements, we identify a node $\mathfrak{t} \in \mathsf{Nodes}(\mathcal{F})$ with the subtree rooted in this node. Thus $\mathsf{Nodes}(\mathcal{F})$ can also be seen as the set of subtrees of $\mathcal{F}$, and $\mathcal{F} \in \mathsf{Nodes}(\mathcal{F})$. We say that a node is *idempotent* if is has at least 3 children (see Definition 6.2).

Given $\mu : A^* \to M$, we denote by $\mathsf{Forests}_{\mu}(w)$ the set of μ-forests of $w \in A^*$. If $K \geq 0$, let $\mathsf{Forests}^K_{\mu}(w)$ be the μ-forests of w of height at most K. Note that $\mathsf{Forests}_{\mu}(w)$ is a set of tree structures which can also be seen as a language over $\widehat{A} := A \uplus \{\langle, \rangle\}$. Indeed, a forest of w can also be seen as "the word w with brackets" in Definition 6.2.

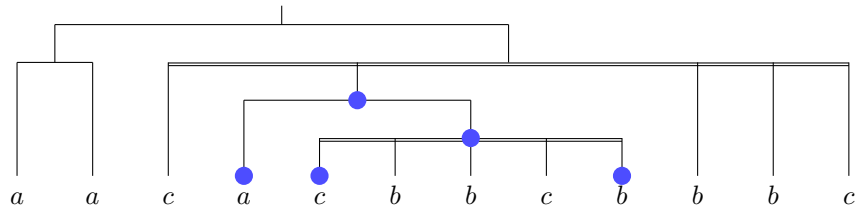


Figure 7 The μ-forest $\langle aa\rangle\langle c\langle a\langle cbbcb\rangle\rangle bbc\rangle$ on $aacacbbcbbbc$.

▶ **Example 6.4.** Let $A = \{a, b, c\}$, $M{=}(\{1, -1, 0\}, \times)$ with $\mu(a) \coloneqq -1$ and $\mu(b) \coloneqq \mu(c) \coloneqq 0$. Then $\mathcal{F} \coloneqq \langle aa\rangle\langle c\langle a\langle cbbcb\rangle\rangle bbc\rangle \in \mathsf{Forests}^5_\mu(aacacbbcbbbc)$ (we dropped the brackets around single letters for more readability) is depicted in Figure 7. Double lines are used to denote idempotent nodes (i.e. with more than 3 children).

A celebrated result states that for any word, a forest of bounded height always exist and it can be computed by a bimachine (with non-unary output alphabet), or a two-way transducer. The following theorem can also be found in [2, Lemma 6.5].

▶ **Theorem 6.5** ([15]). *Given a morphism $\mu : A^* \to M$, we have $\mathsf{Forests}^{3|M|}_\mu(w) \neq \varnothing$ for all $w \in A^*$. Furthermore, one can build a two-way transducer (with a non-unary output alphabet) which computes a total function $\mathsf{forest}_\mu : A^* \to (\widehat{A})^*, w \mapsto \mathcal{F} \in \mathsf{Forests}^{3|M|}_\mu(w)$.*

6.2 Iterable nodes and productions

We define the iterable nodes $\mathsf{Iters}(\mathcal{F}) \subseteq \mathsf{Nodes}(\mathcal{F})$ as the set of nodes which have both a left and a right sibling. Such nodes are thus exactly the middle children of idempotent nodes.

▶ **Definition 6.6.** *Let $\mathcal{F} \in \mathsf{Forests}_\mu(w)$, we define inductively the* iterable nodes *of $\mathcal{F}$:*
- *if $\mathcal{F} = a \in A$ is a leaf, $\mathsf{Iters}(\mathcal{F}) \coloneqq \varnothing$;*
- *otherwise if $\mathcal{F} = \langle\mathcal{F}_1\rangle \cdots \langle\mathcal{F}_n\rangle$, then:*

$$\mathsf{Iters}(\mathcal{F}) \coloneqq \{\mathcal{F}_i : 2 \leq i \leq n{-}1\} \cup \bigcup_{1\leq i\leq n} \mathsf{Iters}(\mathcal{F}_i).$$

Now we define a notion of *skeleton* which selects the right-most and left-most children.

▶ **Definition 6.7.** *Let $\mathcal{F} \in \mathsf{Forests}_\mu(w)$ and $\mathfrak{t} \in \mathsf{Nodes}(\mathcal{F})$, we define the* skeleton *of $\mathfrak{t}$ by:*
- *if $\mathfrak{t} = a \in A$ is a leaf, then $\mathsf{Skel}(\mathfrak{t}) \coloneqq \{\mathfrak{t}\}$;*
- *otherwise if $\mathfrak{t} = \langle\mathcal{F}_1\rangle \cdots \langle\mathcal{F}_n\rangle$, then $\mathsf{Skel}(\mathfrak{t}) \coloneqq \{\mathfrak{t}\} \cup \mathsf{Skel}(\mathcal{F}_1) \cup \mathsf{Skel}(\mathcal{F}_n)$.*

Intuitively, $\mathsf{Skel}(\mathfrak{t}) \subseteq \mathsf{Nodes}(\mathcal{F})$ contains all the descendants of $\mathfrak{t}$ except those which are descendant of a middle child. We then define the frontier of $\mathfrak{t}$, denoted $\mathsf{Fr}_\mathcal{F}(\mathfrak{t}) \subseteq \{1, \ldots, |w|\}$ as the set of positions of w which belong to $\mathsf{Skel}(\mathfrak{t})$ (when seen as leaves of $\mathcal{F}$).

▶ **Example 6.8.** In Figure 7, the top-most blue node $\mathfrak{t}$ is iterable. Furthermore $\mathsf{Skel}(\mathfrak{t})$ is the set of blue nodes, $\mathsf{Fr}_\mathcal{F}(\mathfrak{t}) = \{4, 5, 9\}$ and $w[\mathsf{Fr}_\mathcal{F}(\mathfrak{t})] = acb$.

Using the frontiers, we can naturally lift the notion of productions of a k-marble bimachine from multisets of sets of positions to multisets of nodes in a forest.

▶ **Definition 6.9.** *Let $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ be a k-marble bimachine, $w \in A^*$, $\mathcal{F} \in \mathsf{Forests}_\mu(w)$ and $\mathfrak{t}_1, \ldots, \mathfrak{t}_k \in \mathsf{Nodes}(\mathcal{F})$. We let $\mathsf{prod}^\mathcal{F}_\mathcal{M}(\{\!\{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}\!\}) \coloneqq \mathsf{prod}^w_\mathcal{M}(\{\!\{\mathsf{Fr}_\mathcal{F}(\mathfrak{t}_1), \ldots, \mathsf{Fr}_\mathcal{F}(\mathfrak{t}_k)\}\!\})$.*

Using Lemma 5.6, we can recover the function f from the productions over the nodes. Lemma 6.10 roughly ranges over all possible tuples of calling positions of $\mathcal{M}$.

▶ **Lemma 6.10.** *Let $f : A^* \to \mathbb{N}$ be computed by a k-marble bimachine $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$. If $w \in A^*$ and $\mathcal{F} \in \mathsf{Forests}_\mu(w)$, we have:*

$$f(w) = \sum_{\{\!\{\mathfrak{t}_1,\dots,\mathfrak{t}_k\}\!\} \subseteq \mathsf{Iters}(\mathcal{F}) \cup \{\mathcal{F}\}} \mathsf{prod}^{\mathcal{F}}_{\mathcal{T}}\left(\{\!\{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}\!\}\right).$$

Proof. It follows from [6] that for all $w \in A^*$ and $\mathcal{F} \in \mathsf{Forests}_\mu(w)$, the set of frontiers $\{\mathsf{Fr}_{\mathcal{F}}(\mathfrak{t}) : \mathfrak{t} \in \mathsf{Iters}(\mathcal{F}) \cup \{\mathcal{F}\}\}$ is a partition of $[1{:}|w|]$. We then apply Lemma 5.6. ◀

6.3 Dependent multisets of nodes

The multisets $\{\!\{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}\!\} \subseteq \mathsf{Iters}(\mathcal{F}) \cup \{\mathcal{F}\}$ of Lemma 6.10 will be put into two categories. The *independent* multisets are those whose nodes are distinct and "far enough" in $\mathcal{F}$. The remaining ones are said *dependent*; their number is bounded by a polynomial of degree $k-1$.

▶ **Definition 6.11** (Independent multiset)**.** *Let $\mu : A^* \to M$, $w \in A^*$ and $\mathcal{F} \in \mathsf{Forests}_\mu(w)$, we say that a multiset $\mathfrak{T} \coloneqq \{\!\{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}\!\} \subseteq \mathsf{Iters}(\mathcal{F})$ is* independent *if for all $1 \leq i \neq j \leq k$:*

- *$\mathfrak{t}_i$ is not an ancestor of $\mathfrak{t}_j$;*
- *$\mathfrak{t}_i$ is not the immediate left sibling of an ancestor of $\mathfrak{t}_j$;*
- *$\mathfrak{t}_i$ is not the immediate right sibling of an ancestor of $\mathfrak{t}_j$.*

Note that if $\mathfrak{T}$ is independent, then $\mathcal{F} \notin \mathfrak{T}$ since it is not an iterable node. We denote by $\mathsf{Ind}^k(\mathcal{F})$ the set of independent multisets. Conversely, let $\mathsf{Dep}^k(\mathcal{F})$ be the set of multisets $\{\!\{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}\!\} \subseteq \mathsf{Iters}(\mathcal{F}) \cup \{\mathcal{F}\}$ which are *dependent* (i.e. not independent). By Lemma 6.10, if $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ computes $f : A^* \to \mathbb{N}$ and $\mathcal{F} \in \mathsf{Forests}_\mu(w)$, then:

$$f(w) = \sum_{\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F})} \mathsf{prod}^{\mathcal{F}}_{\mathcal{M}}(\mathfrak{T}) + \sum_{\mathfrak{T} \in \mathsf{Dep}^k(\mathcal{F})} \mathsf{prod}^{\mathcal{F}}_{\mathcal{M}}(\mathfrak{T}) \tag{2}$$

The idea is now to compute these two sums separately. We begin with the second one.

▶ **Lemma 6.12.** *Given a k-marble bimachine $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$, the following function is (effectively) polyregular and has growth at most $k-1$:*

$$\mathsf{sum\text{-}dep}_{\mathcal{M}} : (\widehat{A})^* \to \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \displaystyle\sum_{\mathfrak{T} \in \mathsf{Dep}^k(\mathcal{F})} \mathsf{prod}^{\mathcal{F}}_{\mathcal{M}}(\mathfrak{T}) & \text{if } \mathcal{F} \in \mathsf{Forests}^{3|M|}_\mu(w) \text{ for some } w \in A^*; \\ 0 & \text{otherwise.} \end{cases}$$

▶ Remark 6.13. For this result, we do not need to assume that $\mathcal{M}$ is permutable.

6.4 Independent multisets of nodes

In order to complete the description of f from Equation 2, it remains to treat the productions over independent multisets of nodes. When $\{\!\{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}\!\} \in \mathsf{Ind}^k(\mathcal{F})$, all the $\mathfrak{t}_i$ must be distinct, hence we shall denote it by a set $\{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}$. We define the counterpart of $\mathsf{sum\text{-}dep}_{\mathcal{M}}$:

$$\mathsf{sum\text{-}ind}_{\mathcal{M}} : (\widehat{A})^* \to \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \displaystyle\sum_{\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F})} \mathsf{prod}^{\mathcal{F}}_{\mathcal{M}}(\mathfrak{T}) & \text{if } \mathcal{F} \in \mathsf{Forests}^{3|M|}_\mu(w) \text{ for some } w \in A^*; \\ 0 & \text{otherwise.} \end{cases}$$

▶ **Lemma 6.14.** *Given a k-marble bimachine $\mathcal{M}$ which is $2^{3|M|}$-permutable, one can build a polyblind function $\mathsf{sum\text{-}ind}'_{\mathcal{M}} : (\widehat{A})^* \to \mathbb{N}$ and a polyregular function $\mathsf{sum\text{-}ind}''_{\mathcal{M}} : (\widehat{A})^* \to \mathbb{N}$ with growth at most $k-1$, such that $\mathsf{sum\text{-}ind}_{\mathcal{M}} = \mathsf{sum\text{-}ind}'_{\mathcal{M}} + \mathsf{sum\text{-}ind}''_{\mathcal{M}}$.*

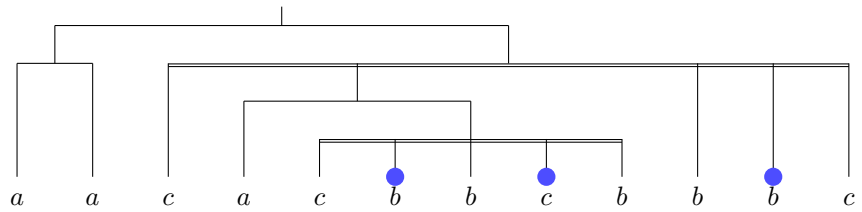
If $\mathsf{sum\text{-}ind}_{\mathcal{M}}$ was a polynomial, then $\mathsf{sum\text{-}ind}'_{\mathcal{M}}$ should roughly be its term of highest degree and $\mathsf{sum\text{-}ind}''_{\mathcal{M}}$ corresponds to a corrective term.

The rest of this section is devoted to the proof of Lemma 6.14. In order to simplify the notations, we extend a morphism μ to its μ-forests by $\mu(\mathcal{F}) \coloneqq \mu(w)$ when $\mathcal{F} \in \mathsf{Forests}_\mu(w)$. Given $\mathsf{t} \in \mathsf{Nodes}(\mathcal{F})$, we denote by $\mathsf{depth}^{\mathcal{F}}(\mathsf{t})$ its depth in the tree structure $\mathcal{F}$ (the root has depth 1, and it is defined inductively as usual). Now we introduce the notion of *linearization* of $\mathsf{t} \in \mathsf{Nodes}(\mathcal{F})$, which is used to describe $w[\mathsf{Fr}_{\mathcal{F}}(\mathsf{t})]$ as a 1-multicontext.

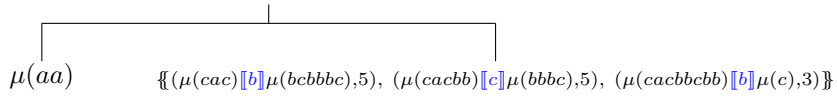
▶ **Definition 6.15** (Linearization). *Let $\mu : A^* \to M$, $w \in A^*$ and $\mathcal{F} \in \mathsf{Forests}_\mu(w)$. The* linearization *of $\mathsf{t} \in \mathsf{Nodes}(\mathcal{F})$ is a 1-multicontext $m[\![u]\!]m'$ defined by induction:*

- *if $\mathsf{t} = \mathcal{F}$ then $\mathsf{lin}^{\mathcal{F}}(\mathsf{t}) \coloneqq [\![w[\mathsf{Fr}_{\mathcal{F}}(\mathcal{F})]]\!]$;*
- *otherwise $\mathcal{F} = \langle \mathcal{F}_1 \rangle \cdots \langle \mathcal{F}_n \rangle$, and $\mathsf{t} \in \mathsf{Nodes}(\mathcal{F}_i)$ for some $1 \leq i \leq n$. We define:* $\mathsf{lin}^{\mathcal{F}}(\mathsf{t}) \coloneqq \mu(\mathcal{F}_1) \cdots \mu(\mathcal{F}_{i-1})\, \mathsf{lin}^{\mathcal{F}_i}(\mathsf{t}) \mu(\mathcal{F}_{i+1}) \cdots \mu(\mathcal{F}_n)$.

We finally introduce the notion of *architecture*. Intuitively, it is a simple tree which describes the positions of a set of nodes $\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F})$ in its forest $\mathcal{F}$. We build it inductively on the example depicted in Figure 8. At the root, we see that there is no node of $\mathfrak{T}$ in the left subtree, hence we replace it by its image under μ. The right subtree is an idempotent node whose leftmost and rightmost subtrees have no node in $\mathfrak{T}$. We thus replace this idempotent node by a leaf containing the multiset of the linearizations and depths of the $\mathsf{t} \in \mathfrak{T}$. Since our machine $\mathcal{M}$ is permutable, this simple information will be enough to recover $\mathsf{prod}^{\mathcal{F}}_{\mathcal{M}}(\mathfrak{T})$.



(a) In blue, a set $\mathfrak{T}$ of 3 independent nodes in the forest from Figure 7.



(b) The corresponding architecture.

Figure 8 A set of independent nodes and its architecture.

▶ **Definition 6.16** (Architecture). *Let $w \in A^*$, $\mathcal{F} \in \mathsf{Forests}_\mu(w)$ and $\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F})$. We define the* architecture *of $\mathfrak{T}$ in $\mathcal{F}$ by induction as follows:*

- *if $\mathcal{F} = \varepsilon$, then $k = 0$. We define $\mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) \coloneqq \varepsilon$;*
- *if $\mathcal{F} = a$, then $k = 0$. We define $\mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) \coloneqq \mu(a)$;*
- *otherwise $\mathcal{F} = \langle \mathcal{F}_1 \rangle \cdots \langle \mathcal{F}_n \rangle$ with $n \geq 1$:*
  - *if $k = 0$, we set $\mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) = \langle \mu(\mathcal{F}) \rangle$;*
  - *else if $\mathfrak{T}_1 \coloneqq \mathfrak{T} \cap \mathsf{Nodes}(\mathcal{F}_1) \neq \varnothing$, then $\mathfrak{T}_1 \in \mathsf{Ind}^{|\mathfrak{T}_1|}(\mathcal{F}_1)$ (since $\mathcal{F}_1 \notin \mathfrak{T}$ by iterability) and $\mathfrak{T} \setminus \mathfrak{T}_1 \in \mathsf{Ind}^{k-|\mathfrak{T}_1|}(\langle \mathcal{F}_2 \rangle \cdots \langle \mathcal{F}_n \rangle)$ (since $\mathcal{F}_2 \notin \mathfrak{T}$ by independence). We set:* $\mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) \coloneqq \langle \mathsf{arc}^{\mathcal{F}_1}(\mathfrak{T}_1) \rangle\, \mathsf{arc}^{\langle \mathcal{F}_2 \rangle \cdots \langle \mathcal{F}_n \rangle}(\mathfrak{T} \setminus \mathfrak{T}_1)$.

- *else if* $\mathfrak{T}_n \coloneqq \mathfrak{T} \cap \mathsf{Nodes}(\mathcal{F}_n) \neq \varnothing$*, we define symmetrically:* $\mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) \coloneqq \mathsf{arc}^{\langle \mathcal{F}_1 \rangle \cdots \langle \mathcal{F}_{n-1} \rangle}(\mathfrak{T} \setminus \mathfrak{T}_n) \langle \mathsf{arc}^{\mathcal{F}_n}(\mathfrak{T}_n) \rangle$
- *else* $\mathfrak{T}_1 = \mathfrak{T}_n = \varnothing$ *but* $k > 0$*, thus* $n \geq 3$ *and* $\mu(\mathcal{F})$ *is idempotent. We define:* $\mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) \coloneqq \langle \{\!\{ (\mathsf{lin}^{\mathcal{F}}(\mathfrak{t}), \mathsf{depth}^{\mathcal{F}}(\mathfrak{t})) : \mathfrak{t} \in \mathfrak{T} \}\!\} \rangle$

Given a morphism, the set of architectures over bounded-height forests is finite.

▷ Claim 6.17. The set $\mathsf{Arcs}_\mu^{3|M|} \coloneqq \{\mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) : \mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F}), \mathcal{F} \in \mathsf{Forests}_\mu^{3|M|}(w), w \in A^*\}$ is finite, given $k \geq 0$ and a morphism $\mu : A^* \to M$.

Proof. The architectures from $\mathsf{Arcs}_\mu^{3|M|}$ are tree structures of height at most $3|M|$. Furthermore they have a branching bounded by $k+3$ and their leaves belong to a finite set (they are either idempotents $e \in M$, or multisets of at most k elements of the form $(m[\![u]\!]m', d)$ with $m, m' \in M$, $|u| \leq 2^{3|M|}$ and $1 \leq d \leq 3|M|$). ◁

Using the permutability of the k-marble bimachine, we show that the production over a set of independent nodes only depends on its architecture. This result enables us to define the notion of production over an architecture.

▶ **Proposition-Definition 6.18.** *Let* $\mathcal{M} = (A, M, \mu, \mathfrak{H}, \lambda)$ *be a* $2^{3|M|}$*-permutable* k*-marble bimachine. Let* $w, w' \in A^*$, $\mathcal{F} \in \mathsf{Forests}_\mu^{3|M|}(w)$ *and* $\mathcal{F}' \in \mathsf{Forests}_\mu^{3|M|}(w')$, $\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F})$ *and* $\mathfrak{T}' \in \mathsf{Ind}^k(\mathcal{F}')$ *such that* $\mathbb{A} \coloneqq \mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) = \mathsf{arc}^{\mathcal{F}'}(\mathfrak{T}')$*. Then* $\mathsf{prod}_{\mathcal{M}}^{\mathcal{F}}(\mathfrak{T}) = \mathsf{prod}_{\mathcal{M}}^{\mathcal{F}'}(\mathfrak{T}')$.
We define $\mathsf{prod}_{\mathcal{M}}(\mathbb{A})$ *as the above value.*

By using the previous statements, we get for all $w \in A^*$ and $\mathcal{F} \in \mathsf{Forests}_\mu^{3|M|}(w)$:

$$\begin{aligned}\mathsf{sum\text{-}ind}_{\mathcal{M}}(\mathcal{F}) &= \sum_{\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F})} \mathsf{prod}_{\mathcal{M}}^{\mathcal{F}}(\mathfrak{T}) = \sum_{\mathbb{A} \in \mathsf{Arcs}_\mu^{3|M|}} \sum_{\substack{\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F}) \\ \mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) = \mathbb{A}}} \mathsf{prod}_{\mathcal{M}}^{\mathcal{F}}(\mathfrak{T}) \\ &= \sum_{\mathbb{A} \in \mathsf{Arcs}_\mu^{3|M|}} \mathsf{prod}_{\mathcal{M}}(\mathbb{A}) \times \mathsf{count}_{\mathbb{A}}(\mathcal{F})\end{aligned}$$

where $\mathsf{count}_{\mathbb{A}}(\mathcal{F}) \coloneqq |\{\mathfrak{T} \in \mathsf{Ind}^k(\mathcal{F}) : \mathsf{arc}^{\mathcal{F}}(\mathfrak{T}) = \mathbb{A}\}|$. It describes the number of multisets of independent nodes which have a given architecture. Now we show how to compute this function as a sum of a polyblind function and a polyregular function with lower growth.

▶ **Lemma 6.19.** *Let* $\mu : A^* \to M$*. Given an architecture* $\mathbb{A} \in \mathsf{Arcs}_\mu^{3|M|}$*, one can build:*

- *a polyblind function* $\mathsf{count}'_{\mathbb{A}} : (\widehat{A})^* \to \mathbb{N}$*;*
- *a polyregular function* $\mathsf{count}''_{\mathbb{A}} : (\widehat{A})^* \to \mathbb{N}$ *with growth at most* $k-1$*;*

such that $\mathsf{count}_{\mathbb{A}}(\mathcal{F}) = \mathsf{count}'_{\mathbb{A}}(\mathcal{F}) + \mathsf{count}''_{\mathbb{A}}(\mathcal{F})$ *for all* $\mathcal{F} \in \mathsf{Forests}_\mu^{3|M|}(w)$ *and* $w \in A^*$.

To conclude the proof of Lemma 6.14, we define the function (which is polyblind):

$$\mathsf{sum\text{-}ind}'_{\mathcal{M}} \coloneqq \sum_{\mathbb{A} \in \mathsf{Arcs}_\mu^{3|M|}} \mathsf{prod}_{\mathcal{M}}(\mathbb{A}) \times \mathsf{count}'_{\mathbb{A}}.$$

We define similarly the following function which is polyregular and has growth at most $k-1$:

$$\mathsf{sum\text{-}ind}''_{\mathcal{M}} = \sum_{\mathbb{A} \in \mathsf{Arcs}_\mu^{3|M|}} \mathsf{prod}_{\mathcal{M}}(\mathbb{A}) \times \mathsf{count}''_{\mathbb{A}}.$$

7 Outlook

This paper provides a technical solution to a seemingly difficult membership problem. This result can be interpreted both in terms of nested transducers (i.e. programs with visible or blind recursive calls) and in terms of rational series. We conjecture that the new techniques introduced here (especially the induction techniques), and the concepts of productions on words and forests, give an interesting toolbox to tackle other decision problem for transducers such as equivalence or membership issues. It could also be interesting to characterize polyblind functions as the series computed by specific weighted automata over $(\mathbb{N}, +, \times)$.

References

1 Jean Berstel and Christophe Reutenauer. *Noncommutative rational series with applications*, volume 137. Cambridge University Press, 2011.

2 Mikolaj Bojańczyk. Polyregular functions. *arXiv preprint*, 2018. arXiv:1810.08760.

3 Mikolaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-string interpretations with polynomial-size output. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, pages 106:1–106:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.106.

4 Michal P Chytil and Vojtěch Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In *4th International Colloquium on Automata, Languages, and Programming, ICALP 1977*, pages 135–147. Springer, 1977.

5 Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular transducer expressions for regular transformations. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 315–324. ACM, 2018.

6 Gaëtan Douéneau-Tabot. Pebble transducers with unary output. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, 2021.

7 Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register transducers are marble transducers. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, 2020.

8 Joost Engelfriet. Two-way pebble transducers for partial functions and their composition. *Acta Informatica*, 52(7-8):559–571, 2015.

9 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001.

10 Noa Globerman and David Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169(2):161–184, 1996.

11 Eitan M Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM Journal on Computing*, 11(3):448–452, 1982.

12 Lê Thành Dung Nguyên. *Implicit automata in linear logic and categorical transducer theory*. PhD thesis, Université Paris 13, 2021.

13 Lê Thành Dung Nguyên, Camille Noûs, and Pierre Pradic. Comparison-free polyregular functions. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, 2021.

14 John C Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

15 Imre Simon. Factorization forests of finite height. *Theor. Comput. Sci.*, 72(1):65–94, 1990. doi:10.1016/0304-3975(90)90047-L.

Regular Expressions for Tree-Width 2 Graphs

Amina Doumane
CNRS, LIP, ENS Lyon, France

Abstract

We propose a syntax of regular expressions, which describes languages of tree-width 2 graphs. We show that these languages correspond exactly to those languages of tree-width 2 graphs, definable in the counting monadic second-order logic (CMSO).

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Tree width, MSO, Regular expressions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.121

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Acknowledgements I want to thank Denis Kuperberg for helpful discussions on the content and presentation.

1 Introduction

Regular word languages form a robust class of languages. One of the witnesses for this robustness is the variety of equivalent formalisms defining them. They can be described by finite automata, by monadic second-order (MSO) formulas, by regular expressions or by finite monoids [3, 6, 10]. Each of these formalisms has some advantages, depending on the context where it is used. For example, MSO is close to natural language, regular expressions define regular languages via their closure properties, automata have good algorithmic properties and can be used as actual algorithms to decide membership in a language, etc. Similarly, regular tree languages have equivalent formalisms, for various kinds of trees [11, 13, 9].

We will here further generalize the structures considered, by moving to graphs of bounded tree-width. Intuitively, they can be thought of as "graphs which resemble trees". In this framework, we already know that counting MSO (CMSO), an extension of MSO with counting predicates, and recognizability by algebra are equivalent [1, 2], yielding a notion that could be called "regular languages of graphs of tree-width k". Engelfriet [7] also proposes a regular expressions formalism matching this class, but by his own admission, these expressions closely mimic the behavior of CMSO. The main feature missing in Engelfriet's regular expressions is a mechanism for iteration, which is the central operator for word regular expressions: the Kleene star.

In this paper, we propose a syntax of regular expressions for languages of tree-width 2 graphs, that follow more closely the spirit of regular expressions on words, using Kleene-like iterations. This constitutes a first step towards the long term objective of obtaining such expressions for languages of graphs of tree-width k. We believe the case of tree-width 2 is already a significant step in itself. Graphs of tree-width 2 form a robust class of graphs with several interesting characterizations. One of them is the characterization via the forbidden minor K_4, the complete graph with four vertices. By the Robertson-Seymour theorem [12], it is known that for every $k \in \mathbb{N}$, the class of tree-width k graphs is characterized by a finite set of excluded minors. However, this result is not constructive, and only the forbidden minors for $k \leq 3$ are known.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 121; pp. 121:1–121:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let us now give more intuition about our expressions for graphs of tree-width 2. Our Kleene-like iteration is defined in terms of least fixed points $\mu x.\, e$. However without restriction, such an operator is too powerful and takes us outside of the CMSO-definable graphs. This phenomenon actually already happens on words: with an arbitrary fixed point, we can write $\mu x.\, (axb)$, defining the non-regular language $\{a^n x b^n \mid n \in \mathbb{N}\}$. The Kleene star on words can be seen as a restriction of the least fixed point operator: only fixed points of the form $\mu x.\, ex$ are allowed, where x does not appear in e. Here the idea is the same, but our restriction will be more involved, and will require a fine understanding of the structure of tree-width 2 graphs.

This work was inspired by the work of Gazdag and Németh [8] on regular expressions for bisemigroups and binoids. One of the main difference with our work is that their operators are only associative, while the operations generating our graphs satisfy more properties.

The paper is structured as follows. Sec. 2 is a preliminary section where we introduce graphs of tree-width 2, the logic CMSO and recognizability by algebra, which are known to be equivalent. In Sec 3, we introduce regular expressions, explain the condition that the iteration should satisfy, and give some examples to illustrate it. At the end of this section, we state our main result, which says that this formalism is equivalent to the two introduced in the preliminary section. We introduce in Sec. 4 the logic CMSO^r, an extension of CMSO with a very restricted form of quantification over relations, and show that it is equivalent to CMSO. Based on this, we show in section 5 that regularity implies CMSO-definability. Finally, we show in section 6 that recognizability implies regularity, proving our main result.

2 Preliminaries

Let Σ_1 and Σ_2 be two disjoint sets of *unary* and *binary* letters respectively. Throughout the paper, we work with the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$.

2.1 Tree-width 2 graphs

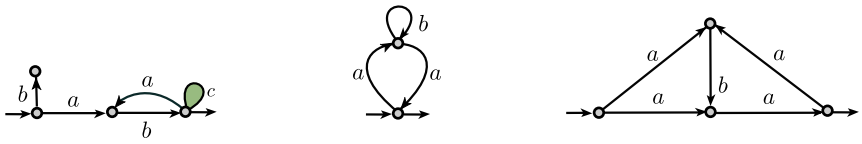
▶ **Definition 1** (Graphs). *A* graph G *is a tuple* $(V, E_1, E_2, s, t, l_1, l_2, \iota, o)$*, where* V *is a set of* vertices*,* E_1 *and* E_2 *are two disjoint sets of* unary *and* binary edges*,* $s : E_1 \uplus E_2 \to V$ *and* $t : E_1 \to V$ *are a* source *and a* target *functions specifying the source and the target of each edge*[1]*,* $l_1 : E_1 \to \Sigma_1$ *and* $l_2 : E_2 \to \Sigma_2$ *are* labeling functions *indicating the label of each edge,* ι *is the* input *vertex and* o *is the* output *vertex,* ι *and* o *are the* interface vertices *of* G*. All the vertices of* G *which are not interface vertices are called* inner vertices*. The* interface of G *is the pair* (ι, o) *if* $\iota \neq o$*, or the vertex* ι *otherwise. An* a-edge *is an edge labeled by the letter* a*. We say that* G *is* unary *if* $\iota = o$*, and* binary *otherwise. The* interface *of a binary edge* e *is* $(s(e), t(e))$*, the interface of a unary edge* e *is* $s(e)$*. An* interface in G *is a list of vertices of length* 1 *or* 2*. A graph is* empty *if it has no edges, and if all its vertices are interface vertices.*

▶ Remark 2. What we call here a graph is what is usually called a hypergraph (because of the unary edges) with interface. We depict graphs with unlabeled ingoing and outgoing arrows to denote the input and the output, respectively.

▶ **Definition 3** (Paths). *A* path p *of* G *is a non-repeating list* $(v_0, e_1, v_1, \ldots, e_n, v_n)$ *where* v_i *is a vertex of* G *and* e_i *is an edge of* G*, such that the interface of* e_i *is either* (v_{i-1}, v_i) *or* (v_i, v_{i-1})*, for every* $i \in [1, n]$*. The path* p *is* directed *if the interface of* e_i *is* (v_{i-1}, v_i) *for every* $i \in [1, n]$*. The vertex* v_0 *is the* input *of* p*,* v_n *is its* output *and* (v_0, v_n) *its* interface*. The path* p *is* safe *if it does not contain an interface vertex of* G *as an inner vertex.*

[1] For unary edges, we specify only their source.

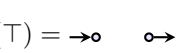
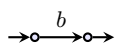
▶ **Example 4.** Here are some examples of graphs. The c-edge in the graph G a unary edge.



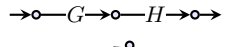
▶ **Definition 5** (tw_2 graphs). *Consider the signature σ containing the binary operations $\cdot$ and $\|$, the unary operations $^\circ$ and* dom*, and the nullary operations* 1 *and* $\top$*. We define* tw_2 terms *as the terms generated by the signature σ and the alphabet Σ:*

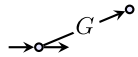
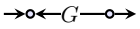
$$t, u := \ a \ \mid \ t{\cdot}u \ \mid \ (t \parallel u) \ \mid \ t^\circ \ \mid \ \mathsf{dom}(t) \ \mid \ 1 \ \mid \ \top \qquad\qquad a \in \Sigma$$

We define the graph of a term t, $\mathcal{G}(t)$, by induction on t, by letting:

$\mathcal{G}(1) =$ $\mathcal{G}(\top) =$  $\mathcal{G}(a) =$  $\mathcal{G}(b) =$ 

and interpreting the operations of the syntax as follows:

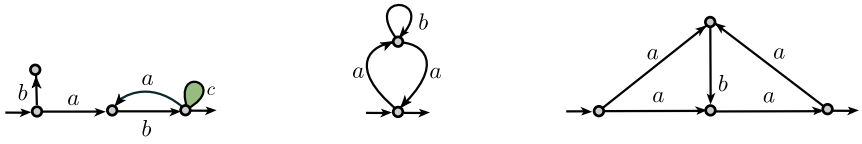
$G{\cdot}H =$  $G \parallel H =$ 

$\mathsf{dom}(G) =$  $G^\circ =$ 

In the picture above, we represent the graph G by an arrow from its input to its output. For example, the graph dom*(G) is obtained from G by relocating the output to the input. We usually write tu instead of $t{\cdot}u$ and give priorities to the symbols of σ so that $ab \| c$ parses to $(a{\cdot}b) \| c$. We define the set of* tw_2 graphs *as the graphs of the terms above. The graphs of a and $a \parallel 1$, where $a \in \Sigma$, are called* atomic*.*

We will sometimes identify terms with the graphs they generate. For example we may say that $(a \parallel b)$ is binary or connected to say that its graph is so.

▶ **Example 6.** Below, from left to right, two tw_2 graphs and a graph which is not tw_2.



$\mathsf{dom}(b)a(a^\circ \parallel b)c$ $\qquad$ $a(b \parallel 1)a$ $\qquad$ No corresponding term

▶ Remark 7. The tw_2 graphs are exactly those graphs whose skeleton[2] has tree-width 2 [4].

▶ **Definition 8** (Graph languages). *Sets of graphs are called* graph languages*. A graph language is* unary *or binary if all its graphs have this arity.*

2.2 Counting monadic second-order logic

We introduce CMSO, the *counting monadic second-order logic*, which is used to describe graph languages.

[2] The skeleton of a graph is the graph obtained by forgetting the labels and the orientation of the edges, and by adding an edge between the input and the output.

▶ **Definition 9** (The logic CMSO). *Let $\mathcal{V}$ be the relational signature which contains two binary symbols* source *and* target*, two unary symbols* input *and* output *and a unary symbol a for each (unary or binary) letter $a \in \Sigma$.*

Let $\mathbb{X}_1$ be a countable set of first-order variables *and $\mathbb{X}_2$ a countable set of* set variables *The formulas of* CMSO *are defined as follows:*

$$\varphi, \psi := \ r(x_1, \ldots, x_n) \ \mid \ x \in X \ \mid \ x = y \ \mid \ \exists x.\varphi \ \mid \ \exists X.\varphi \ \mid \ \varphi \vee \psi \ \mid \ \neg\varphi \ \mid \ (|X| \equiv k) \ [m]$$

where r is an n-ary symbol of $\mathcal{V}$, $x_1, \ldots, x_n, x \in \mathbb{X}_1$, $X \in \mathbb{X}_2$ and $k, m \in \mathbb{N}$. Free and bound variables *are defined as usual. A* sentence *is a formula without free variables. We use the usual syntactic sugar, for example $\varphi \Rightarrow \psi$ as a shortcut for $\neg\varphi \vee \psi$.*

We define the semantics of CMSO formulas. To handle free variables, CMSO formulas are interpreted over *pointed graphs*.

▶ **Definition 10** (Semantics of CMSO). *Let G be a graph and Γ be a set of variables. An* interpretation *of Γ in G is a function mapping each first-order variable of Γ to an edge or vertex of G, and each set variables to a a set of edges and vertices of G. A* pointed graph *is a pair $\langle G, I\rangle$ where G is a graph and I is an interpretation of a set of variables Γ in G. If Γ is empty, we denote it simply as G. Let φ be a* CMSO *formula whose free variables are Γ and let $\langle G, I\rangle$ be a pointed graph such that I is an interpretation of Γ. We define the* satisfiability relation $\langle G, I\rangle \models \varphi$ *as usual, by induction on the formula φ. Here is an example of the semantics of some* CMSO *formulas:*

$\mathsf{source}(v, e)$:	*the source of the edge e is the vertex v.*	$\mathsf{input}(v)$:	*v is the input of G.*
$\mathsf{target}(e, v)$:	*the target of the edge e is the vertex v.*	$\mathsf{output}(v)$:	*v is the output of G.*
$(\lvert X\rvert = k)[m]$:	*the size of X is congruent to k modulo m.*	$a(e)$:	*e is an a-edge.*

If φ is a sentence, we define $\mathcal{L}(\varphi)$, the graph language of φ *as follows:*

$$\mathcal{L}(\varphi) = \{G \ \mid \ G \text{ is a graph and } G \models \varphi\}.$$

▶ **Definition 11** (CMSO definability). *A graph language is* CMSO definable *if it is the graph language of a* CMSO *sentence.*

▶ **Example 12.** The language of graphs having an a-edge from the input to the output is definable in CMSO, by the following formula for instance:

$$\varphi := \ \exists e.\ \exists i.\ \exists o.\ \mathsf{input}(i) \wedge \mathsf{output}(o) \wedge a(e) \wedge \mathsf{source}(i, e) \wedge \mathsf{target}(e, o)$$

Note that the graphs of this language may not be tw_2 graphs.

▶ **Example 13.** The set of tw_2 graphs is a CMSO definable language. Indeed, tw_2 graphs are those graphs which exclude K_4, the complete graph with four vertices, as minor. The set of graphs which exclude a fixed set of minors can be easily defined in CMSO [5].

The set of tw_2 graphs having an a-edge from the input to the output is definable in CMSO, by the conjunction of the formula φ of Ex. 12 and the formula defining tw_2 graphs.

We state below a *localization result*, which allows us to transform a CMSO sentence into another one which talks only about a part of the original graph.

▶ **Proposition 14.** *Let φ be a* CMSO *sentence, $x, y \in \mathbb{X}_1$ and $X \in \mathbb{X}_2$. There is a* CMSO *formula $\varphi_{|(x,X,y)}$ such that, for every graph G and interpretation $I : (x \mapsto s, X \mapsto H, y \mapsto t)$, such that (s, H, t) is a subgraph of G, we have:*

$$\langle G, I\rangle \models \varphi|_{(x,X,y)} \qquad \Leftrightarrow \qquad (s, H, t) \models \varphi$$

▶ Remark 15. There is another presentation of the syntax of CMSO, where we remove first-order variables and the formulas including them, and add the following formulas:

$$X \subseteq Y \quad \text{and} \quad r(X_1 \dots, X_n) \quad \text{where } r \text{ is an } n\text{-ary symbol of } \mathcal{V}.$$

The formula $X \subseteq Y$ is interpreted as "X is a subset of Y" and $r(X_1 \dots, X_n)$ as "for each i, X_i is a singleton containing x_i and $r(x_1 \dots, x_n)$". This presentation is more convenient in proofs by induction as there are less cases to analyze.

2.3 Recognizability

We can specify languages of graphs by means of σ-algebras, generalizing to graphs the notion of recognizability by monoids. A *σ-algebra* $\mathcal{A}$ is the collection of a set D called its *domain*, and for each n-ary operation o of σ, a function $o^{\mathcal{A}} : D^n \to D$. A homomorphism $h : \mathcal{A} \to \mathcal{B}$ between two σ-algebras $\mathcal{A}$ and $\mathcal{B}$ is a function from the domain of $\mathcal{A}$ to the domain of $\mathcal{B}$ which preserves the operations of σ. Note that the set of tw_2 graphs, where the operations of σ are interpreted as in Def. 5, forms a σ-algebra which we denote by $\mathcal{G}_{\mathsf{tw}_2}$.

▶ **Definition 16** (Recognizability). *We say that a language L of tw_2 graphs is* recognizable *if there is a σ-algebra $\mathcal{A}$ with finite domain, a homomorphism $h : \mathcal{G}_{\mathsf{tw}_2} \to \mathcal{A}$ and a subset P of the domain of $\mathcal{A}$ such that $L = h^{-1}(P)$.*

▶ **Theorem 17.** *If a language of tw_2 graphs is CMSO definable, then it is recognizable.*

2.4 Operations on graph languages

The operations of σ can be lifted from graphs to graph languages in the natural way. We say that an operation on graph languages is CMSO *compatible* if, whenever its arguments are CMSO definable, then so is its result.

▶ **Proposition 18.** *Union and the operations of σ are CMSO compatible.*

We define two additional operations: *substitution* and *iteration*.

▶ **Definition 19** (Substitution and iteration). *Let x be a letter, L and M be tw_2 graph languages and let be G a tw_2 graph. We define the set of graphs $G[L/x]$ by induction on G as follows:*

$$x[L/x] = L, \quad a[L/x] = a \ (a \neq x) \quad \text{and} \quad o(G_1 \dots, G_n)[L/x] = o(G_1[L/x], \dots, G_n[L/x])$$

where o is an n-ary operation of σ. We define $M[L/x]$ as:

$$M[L/x] = \bigcup_{G \in M} G[L/x]$$

We define similarly the simultaneous substitution $M[\vec{L}/\vec{x}]$, where $\vec{L}$ and $\vec{x}$ are respectively a list of tw_2 graph languages and a list of letters of the same length.

For every $n \geq 1$, we define the language $L^{n,x}$ and the iteration $\mu x.L$ as follows:

$$L^{1,x} := L, \qquad L^{n+1,x} =: L[L^{n,x}/x] \cup L^{n,x}, \qquad \mu x.L := \bigcup_{n \geq 1} L^{n,x}.$$

▶ Remark 20. Substitution and iteration are not CMSO compatible in general. For instance, the iteration of the CMSO language $\{axb\}$, which is the set $\{a^n x b^n \mid n \in \mathbb{N}\}$, is not CMSO definable. However, under a *guard condition* that we introduce later, we recover CMSO compatibility.

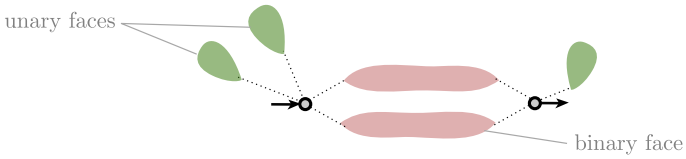
We finally consider two restricted forms of iteration called *Kleene* and *parallel iteration*.

▶ **Definition 21** (Kleene and parallel iteration). *We define the* Kleene iteration L^+ *and the* parallel iteration $L^{\|}$ *of a language L as follows, where x is a letter not appearing in L:*

$$L^+ = (\mu x.\ L{\cdot}x)[L/x], \qquad L^{\|} = (\mu x.\ L \,\|\, x)[L/x].$$

2.5 Pure graphs and modules

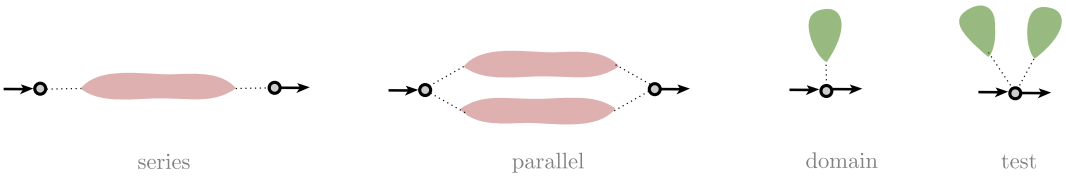
▶ **Definition 22** (Pure graphs.). *Let G be a graph. If we remove the interface vertices of G we obtain one or several connected components which we call the* faces *of G. The* arity of a face *is the number of interface vertices of G it is incident to.*



We say that G is pure *if it has at least one face and all it faces have the same arity as itself. We say that G is* prime *if it has exactly one face, and* composite *if it has at least two faces.*

▶ Remark 23. Pure graphs are connected and non-empty. Not all graphs are pure.

▶ **Definition 24** (Type of a pure graph). *The* type *of a pure graph is a pair specifying its arity and whether it is prime or composite. We say that a graph is* series *if it is binary and prime,* parallel *if it is binary and composite,* domain *if it is unary and prime and* test *if it is unary and composite. We denote by* $\mathsf{s}, \mathsf{p}, \mathsf{d}$ *and* t *the type series, parallel, domain and test respectively. Series, parallel domain and test graphs look like this:*



A graph language is (of type) series, parallel, domain *or* test *if* ***all*** *its graphs have this type.*

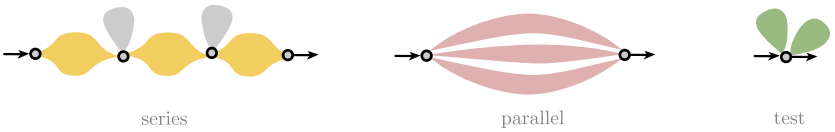
There is a canonical way to decompose pure graphs of type series, parallel and test.

▶ **Proposition 25** ([4]). *Let G be a pure graph. The graph G has the following shape:*

$$\begin{aligned} G &:= P_0 \cdot U_1 \cdot P_1 \ldots U_n \cdot P_n && \text{if } G \text{ is series,} \\ G &:= S_0 \,\|\, \cdots \,\|\, S_n && \text{if } G \text{ is parallel,} \\ G &:= D_0 \,\|\, \cdots \,\|\, D_n && \text{if } G \text{ is test,} \end{aligned}$$

P_j being parallel or atomic, U_i unary, S_i series and D_i domain, for all $j \in [0, n], i \in [1, n]$.

Here is a picture illustrating this proposition:

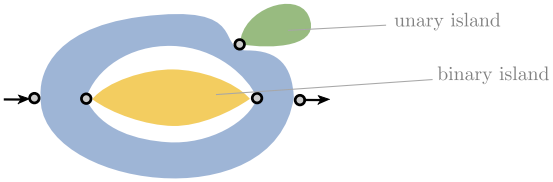


▶ **Definition 26** (Contexts). *Let* $\mathbb{S}$ *be a set of* special (unary and binary) letters*, and let* $n \geq 1$*. An* n-context *is a graph such that* n *of its edges, called* holes*, are numbered from* 1 *to* n*, and labeled by* n *distinct special letters. We call* 1*-contexts simply* contexts*.*

Let C *be an* n*-context whose holes are* $h_1 \dots, h_n$ *and let* $H_1, \dots, H_n$ *be graphs such that* h_i *and the* H_i *have the same arity, for all* $i \in [1, n]$*. We define* $C[H_1, \dots, H_n]$ *as the graph obtained from the disjoint union of* C *and* $H_1, \dots, H_n$*, by removing the holes of* G*, and for every* $i \in [1, n]$ *identifying the input of* h_i *with the input of* H_i*, the output of* h_i *with the output of* H_i*, and by letting its interface of to be that of* C*.*

▶ **Definition 27** (Islands and modules). *An* island *of a graph* G *is a graph* H *such that there is a context* C *satisfying* $G = C[H]$*. A* module *is a island which is pure. Two islands (or modules) of a graph are* parallel *if they have the same interface.*
Since modules are pure, we can speak of series, parallel, domain and test modules of a graph.

The following picture illustrates a unary and binary island of a graph.



▶ Remark 28. Our notion of modules is different from the one usually used in graph theory, more precisely in the setting of *modular decompositions*.

▶ Remark 29. The parallel composition of two islands of a graph G with the same interface is also an island of G with the same interface. Similarly, the parallel composition of two modules of a graph G with the same interface is also a module of G with the same interface. This justifies the following definition.

▶ **Definition 30** (Maximal islands and modules). *Let* G *be a graph and* I *an interface in* G*. The* maximal island at I *is the parallel composition of all the islands of* G *whose interface is* I*, we denote it by* $\mathsf{max\text{-}island}_G(I)$*. The* maximal module at I *is the parallel composition of all the modules of* G *whose interface is* I*, we denote it by* $\mathsf{max\text{-}module}_G(I)$*.*

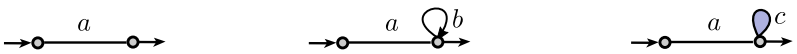
▶ Remark 31. The maximal module at a given interface does not always exist.

▶ **Proposition 32.** *Being series, parallel, domain, test, an island, a module, a maximal island, a maximal module are* CMSO *definable properties.*

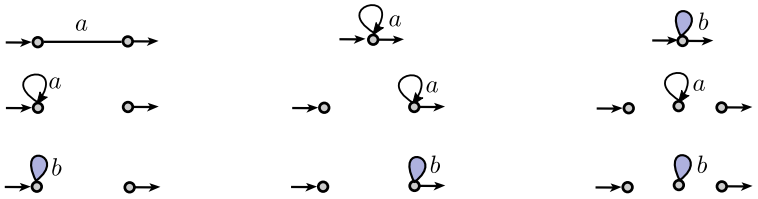
3 Regular expressions for tw_2 graphs

3.1 Regular expressions for word and multiset graphs

▶ **Definition 33** (Word and multiset alphabets). *Let* Σ_{w} *be the set of terms whose graphs have the following form, where* $a, b \in \Sigma_2$ *and* $c \in \Sigma_1$*:*



Let Σ_{m} *be the set of terms whose graphs have the following form, where* $a \in \Sigma_2$ *and* $b \in \Sigma_1$*:*



Word graphs *are the graphs generated from those of* Σ_{w} *by series composition, and* multiset graphs *are the graphs generated from those of* Σ_{m} *by parallel composition.*

▶ **Example 34.** Below, from left to right, a word graph and two multiset graphs.

▶ **Definition 35** (Word an multiset expressions)**.** Word expressions *are defined as follows:*

$$e, f := a \mid e \cdot f \mid e \cup f \mid e^{+} \qquad (a \in \Sigma_{\mathsf{w}})$$

Multiset pre-expressions *are defined as follows:*

$$e, f := a \mid (e \parallel f) \mid e \cup f \mid e^{\parallel} \qquad (a \in \Sigma_{\mathsf{m}})$$

Multiset expressions *are those pre-expressions, where each sub-term appearing under a parallel iteration, is built using a single element* $a \in \Sigma_{\mathsf{m}}$ *(all the other operations are allowed). The graph language of an expressions is defined as usual.*

▶ Remark 36. To see why the condition on multiset regular expressions is useful, consider the expression $e = (a \parallel b)$. The language of its parallel iteration is the set of multiset graphs which have the same number of a-edges and b-edges, and this is not a CMSO definable language.

3.2 Context-free expressions

▶ **Definition 37** (Context-free expressions)**.** *We define* context-free expressions *as the set of terms generated by the following syntax:*

$$\begin{aligned} e, f := \quad & e_{\mathsf{w}} \mid e_{\mathsf{m}} \\ & \mid e \cdot f \mid (e \parallel f) \mid e^{\circ} \mid \mathsf{dom}(e) \mid 1 \mid \top \\ & \mid e \cup f \mid e[f/x] \mid \mu x.e \end{aligned}$$

where e_{w} *and* e_{m} *are respectively word and multiset regular expressions. We define the* language *of a context-free expression* e*, denoted* $\mathcal{L}(e)$*, by induction on* e*, interpreting the operations of the syntax as described in Sec. 2.4.*

Regular expressions for tw_2 *graphs* will be defined as a restriction of context-free expressions, where substitution and iteration are allowed only under a *guard condition* that we shall explain in the following.

3.3 The guard condition

▶ **Definition 38** (Guarded letters)**.** *Let* G *be a graph and* x *a letter. We say that:*

- x *is* s-guarded *in* G *if* x *is binary and every* x*-labeled edge of* G *is parallel to a module.*
- x *is* p-guarded *in* G *if* x *is binary and no* x*-labeled edge of* G *is parallel to a module.*

- *x is* d*-guarded in G if x is unary.*
- *x is* t*-guarded in G if x is unary and no x-labeled edge of G is parallel to a module.*

Let $\tau \in \{\mathsf{s}, \mathsf{p}, \mathsf{d}, \mathsf{t}\}$ be a type and L a graph language. We say that x is τ-guarded *in L if it is τ-guarded in every graph of L.*

▶ **Definition 39** (Guard condition)**.** *Let x be a letter, M a tw_2 graph language and L a pure language of type τ. The substitution $M[L/x]$ is guarded if x is τ-guarded in M. The iteration $\mu x.L$ is guarded if x is τ-guarded in L.*

We say that the iteration $\mu x.L$ is of type τ *if L is of type τ.*

▶ **Definition 40** (Regular expression)**.** *A* regular expression *is a context-free expression where every substitution and iteration is guarded. A language of graphs is* regular *if it is the language of some regular expression.*

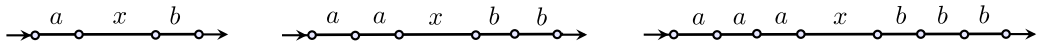
▶ Remark 41. When L is test and x is a unary letter, then $\mu x.L$ is always guarded.

▶ **Proposition 42.** *We can decide if a context-free expression is regular.*

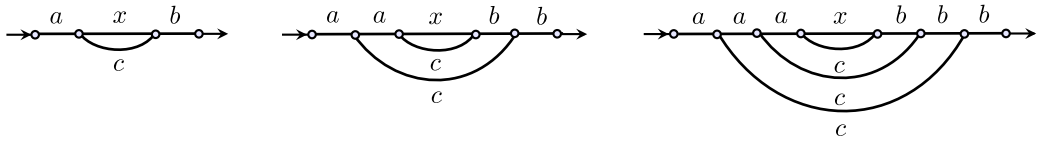
▶ Remark 43. Be aware that prop. 42 is about deciding a syntactic property of e, namely that the iterations and substitutions are guarded. However, the problem of determining if a context-free expression *defines a* CMSO *language* is undecidable. This apparent contradiction comes from the fact that some context-free expressions, which are not guarded, define CMSO languages, as we shall see in the upcoming examples.

3.4 Examples

▶ **Example 44.** The iteration $\mu x.axb$ is not guarded. Indeed, the language of axb is series, as it contains a single series graph G. However, the letter x is not s-guarded in G, because it is not parallel to any module of G. The graph of this iteration look like this:

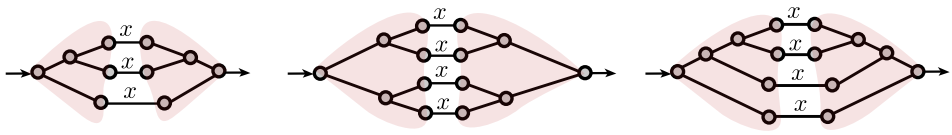


▶ **Example 45.** The iteration $\mu x.a(x \parallel c)b$ is guarded. Indeed the language of $a(x \parallel c)b$ is series, actually it contains a single graph G, depicted below left, which is series. The letter x is s-guarded in G, because it is parallel to a module, namely the c-edge. The graph of this iteration look like this:



Note the similarity between the graph language of $\mu x.axb$ and that of $\mu x.a(x \parallel c)b$: the former is obtained by forgetting the c-edges of the latter. Yet, the latter is CMSO definable, while the former is not. In the case of $\mu x.a(x \parallel c)b$, the c-edges will guide a CMSO formula to relate the a-edges and the b-edges of the same iteration depth. This is the main intuition behind the guard condition for series languages.

▶ **Example 46.** The iteration $\mu x.(axa \parallel axa)$ is guarded. Indeed, the language of $(axa \parallel axa)$ is parallel, as it contains a unique graph G (the left graph below) which is parallel. The letter x is p-guarded because all the occurrences of x are not parallel to any module of G. Note that the graphs of this expression have the following shape: they all start with a binary tree whose edges are labeled by a, end ends with the mirror image of this tree, while the corresponding leafs are connected by an x-edge. Those trees are colored in red below.



At first glance, this expression dos not seem to be CMSO definable, as it seems that we need to test whether a graph starts and ends with the same tree. We will see however that the language of this expression, as those of all regular expressions, is CMSO definable.

The guard condition is not "perfect", in the sense that some non-guarded context-free expressions might generate CMSO definable languages, as shown in the following example.

▶ **Example 47.** The context-free expression $(\mu x.axb)[1/a, 1/x, 1/b]$ is not regular because the iteration $\mu x.axb$ is not guarded. However its language, the graph of 1, is CMSO definable.

▶ Remark 48. Intuitively, the guard condition allows only those graphs where series and parallel operations alternate. This why we add the word and multiset expressions: to allow graphs where we can iterate only series or parallel operations respectively.

3.5 Main result

The main result of this paper is the following theorem:

▶ **Theorem 49.** *Let L be a language of* tw_2 *graphs. We have:*

$$L \text{ is recognizable} \quad \Leftrightarrow \quad L \text{ is CMSO definable} \quad \Leftrightarrow \quad L \text{ is regular}$$

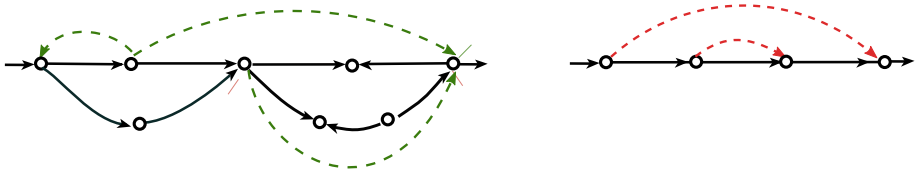
Thanks to Thm. 17, CMSO definability implies recognizablity. We show that regularity implies CMSO definability in Sec. 5 and that recognizabilty implies regularity in Sec. 6.

4 Companion relations

▶ **Definition 50** (Companion relation). *Let G be a graph. Two paths of G are* orthogonal *if they do not share any edge, and whenever they share a vertex, it is necessarily an interface vertex of one of them. A set of paths is a* set of orthogonal paths *if its paths are pairwise orthogonal.*

A relation R on the vertices of G is a companion relation *if there is a set of orthogonal paths P such that $(v, w) \in R$ iff (v, w) is the interface of a path $p \in P$. We say that p is a* witness *for (v, w), and that P is a* witness *for the relation R.*

▶ **Example 51.** The relation indicated by the green dotted arrows below is a companion relation. This is not the case for the one indicated by the red dotted arrows.



We introduce CMSO^r, an extension of CMSO where quantification over companion relations is possible.

▶ **Definition 52** (The logic CMSO^r). *Let $\mathbb{X}_r$ be a set of* relation variables*, whose elements are denoted $R, S, \ldots$. The formulas of CMSO^r are of the following form:*

$$\varphi := \mathsf{CMSO} \mid \exists R.\ \varphi \mid (x, y) \in R \qquad (R \in \mathbb{X}_r,\ x, y \in \mathbb{X}_1).$$

As for CMSO, we need to define the semantics of a formula over pointed graphs to handle free variables.

▶ **Definition 53** (Semantics of CMSO^r). *Let G be a graph and Γ be a set of variables. An interpretation of Γ is as usual, but here every relation variable is mapped to a binary relation on the vertices of G. We define the* satisfiability relation $\langle G, I\rangle \models \varphi$ *as usual, by induction on the formula φ. The only new cases are the quantification $\exists R$ which is interpreted as "there exists a companion relation R on the vertices of the graph", and the formulas $(x, y) \in R$ which are interpreted as "there is a pair of vertices (x, y) in R".*

4.1 The logic CMSO^r have the same expressive power as CMSO

To guess a companion relation in CMSO, we show how to encode a set of guarded paths by a collection of sets called a *footprint*.

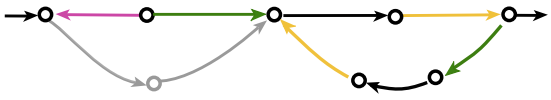
▶ **Definition 54** (Frontier edges of a path). *Let $p = (v_0, e_1, v_1, \ldots, e_n, v_n)$ be a path. If $n > 1$, we call e_1 the* opening edge *of p and e_n its* closing edge*. If $n = 0$, we call e_0 its* single edge*. Opening, closing and single edges are called the* frontier edges *of p, the other edges are called its* inner edges*.*

▶ **Definition 55** (Footprint). *A* footprint *in a graph G is the following collection of data: a partition of the vertices of G into* non-path *and* path *vertices, a partition of edges into* non-path *and* path *edges, a partition of path edges into* frontier *and* inner *edges, a partition of frontier edges into* opening, closing and single *edges and a partition of path edges into* direct *and* inverse *edges.*

The partition of path edges into direct and inverse ones provides them with a new orientation: they conserve their original orientation if they are direct, or get reversed (we swap the source and target) if they are inverse edges.

Let $\mathbb{F}$ be a footprint. A path p is encoded by $\mathbb{F}$ if its edges and vertices are path edges and path vertices of $\mathbb{F}$, if its inner, frontier, opening, closing and single edges are edges of the corresponding sets in $\mathbb{F}$. Moreover, p must form a directed path with the new orientation dictated by $\mathbb{F}$.

▶ **Example 56.** We represent below a footprint in the left graph of Ex. 51. Non-path edges and vertices are grey, path vertices are black, opening edges are green, closing edges are yellow, single edges are pink and all the other inner edges are black. For path edges, we display the new orientation induced by the footprint instead of the original one. The set of paths encoded by this footprint are a witness that the green relation of Ex. 51 is a companion relation.



▶ **Proposition 57.** *Let G be a graph and P a set of orthogonal paths of G. There is a footprint $\mathbb{F}$ such that P is the set of paths encoded by $\mathbb{F}$.*

▶ **Corollary 58.** *If a language is CMSO^r definable then it is* CMSO *definable.*

5 Regular implies CMSO definable

▶ **Theorem 59.** *If a language is regular, then it is* CMSO *definable.*

To prove Thm. 59, we proceed by induction on regular expressions. The cases of word and multiset regular expressions follow from the similar result for words and commutative words. The cases of union and the operations of the signature σ follow from Prop. 18. We are left with the cases of substitution and iteration; the rest of this section is dedicated to proving the following proposition.

▶ **Proposition 60.** *Let x be a letter and L and M be languages of* tw_2 *graphs. We have:*

$$M[L/x] \text{ is guarded and } L \text{ and } M \text{ are } \mathsf{CMSO}\text{-definable} \Rightarrow M[L/x] \text{ is } \mathsf{CMSO}\text{-definable.}$$
$$\mu x.L \text{ is guarded and } L \text{ is } \mathsf{CMSO}\text{-definable} \Rightarrow \mu x.L \text{ is } \mathsf{CMSO}\text{-definable.}$$

We handle the case of iteration, the case of substitution being similar. We show first that the iteration of a CMSO definable language, without any guard condition, is definable in an extension of CMSO where we are allowed to quantify existentially over sets of subgraphs of the input graph, which we call CMSO^d. This logic is obviously strictly more expressive then CMSO, because it amounts to quantify over sets of sets. Based on this, we show that the *guarded iteration* of a CMSO definable language is definable in CMSO^r, the extension of CMSO with companion relations defined in the previous section. This concludes the proof, the logic CMSO^r being equivalent to CMSO.

5.1 Iteration of CMSO formulas is CMSO^d definable

5.1.1 Decompositions

When a graph is in the iteration $\mu x.L$ of some language L, it is possible to structure it into a tree shaped decomposition, such that each part of this decomposition "comes from L". In the following, we define such decompositions.

▶ **Definition 61** (Independent graphs)**.** *Let G be a graph and H, K be subgraphs of G. We say that H and K are* independent *if they do not share any edge; and whenever they share a vertex, it is necessarily an interface vertex of both H and K.*

▶ **Definition 62** (Decompositions)**.** *A* decomposition *of G is a set $\mathcal{D}$ of modules of G such that $G \in \mathcal{D}$ and for every pair of graphs in $\mathcal{D}$, they are either independent, or module one of the other. We call the graphs of a decomposition its* components*. We call* the interfaces of $\mathcal{D}$ *the set of interfaces of its components.*

Let H and K be components of a decomposition $\mathcal{D}$. We say that H is a child *of K, if H is a module of K, and if there is no component C of $\mathcal{D}$, distinct from H and K, such that H is a module of C and C is a module of K.*

The graph G is called the head *of $\mathcal{D}$. A component of $\mathcal{D}$ is a* leave *if it does not contain another component of $\mathcal{D}$ as a module.*

▶ **Definition 63** (Body of a component)**.** *Let G be a graph, $\mathcal{D}$ a decomposition of G and C a component of $\mathcal{D}$.*

The body *of C is the subgraph of G whose vertices are those of C minus the* ***inner*** *vertices of its children; and whose edges are those of C minus those of its children.*

*The x-*body of C *is the graph whose interface is the interface of C, whose vertices are the vertices of the body of C, and whose edges are the edges of the body of C plus, for each child F of C, an x-edge whose interface is the interface of F. We denote it by x-$\mathsf{body}_{\mathcal{D}}(C)$.*

▶ **Definition 64** (L-decompositions). *Let L be a graph language. An L-decomposition of a graph G is a decomposition of G such that the x-body of each of its components is in L.*

▶ Remark 65. The body of a component is a subgraph of G, but its x-body is not a subgraph of G in general, because of the added x-edges.

▶ **Proposition 66.** *Let L be a graph language. We have:*

$$G \in \mu x.L \qquad \Leftrightarrow \qquad \exists \mathcal{D}. \quad \mathcal{D} \text{ is an } L\text{-decomposition of } G.$$

5.1.2 The logic CMSO^d

Let φ be a CMSO formula defining a graph language L. Using Prop 66, we can express that a graph G is in the iteration $\mu x.L$ by guessing a decomposition $\mathcal{D}$ of G, and ensuring that the x-body of each component satisfies φ. But guessing a set of subgraphs is not expressible in CMSO. This is why we introduce CMSO^d, an extension of CMSO where this is allowed.

▶ **Definition 67** (CMSO^d logic). *Let $\mathbb{X}_d$ be a set of* graph set variables*, whose elements are denoted $\mathcal{X}, \mathcal{Y} \ldots$. The formulas of* CMSO^d *are of the following form:*

$$\varphi := \mathsf{CMSO} \mid \exists \mathcal{X}.\ \varphi \mid (s, Z, t) \in \mathcal{X} \qquad\qquad (\mathcal{X} \in \mathbb{X}_d, \quad Z \in \mathbb{X}_2, \quad s, t \in \mathbb{X}_1).$$

Free and bound variables are defined as usual. As for CMSO, we need to define the semantics of a formula over pointed graphs to handle free variables.

▶ **Definition 68** (Semantics of CMSO^d). *Let G be a graph and Γ be a set of variables.*

An interpretation of Γ *is a function mapping every first-order variable of Γ to an edge or vertex of G, every set variable to a set of edges and vertices of G, and every graph set variable to a set of subgraphs of G.*

We define the satisfiability relation $\langle G, I\rangle \models \varphi$ *as usual, by induction on φ. The only new cases compared to* CMSO *are the quantification $\exists \mathcal{X}$ which is interpreted as "there exists a set of subgraphs $\mathcal{X}$", and the formulas $(s, Z, t) \in \mathcal{X}$ which are interpreted as "the graph whose input is s, whose output is t and whose set of edges and vertices is Z, is an element of $\mathcal{X}$".*

▶ **Proposition 69.** *There is a* CMSO^d *formula* $\mathsf{decomp}(\mathcal{X})$*,* without graph set quantification*, such that for every graph G and every set of subgraphs $\mathcal{D}$ of G, we have:*

$$\langle G, \mathcal{X} \mapsto \mathcal{D}\rangle \models \mathsf{decomp}(\mathcal{X}) \quad \Leftrightarrow \quad \mathcal{D} \text{ is a decomposition of } G.$$

5.1.3 Iteration is expressible in CMSO^d

Given a CMSO formula φ, we construct a formula $[\![\varphi]\!]$ having $\mathcal{X}$ as unique free variable, which expresses the fact that the x-body the head of the decomposition $\mathcal{X}$ satisfies φ. To construct $[\![\varphi]\!]$, we need the following definition.

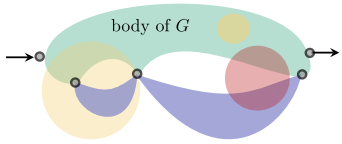
▶ **Definition 70** (Complete sets). *Let $\mathcal{D}$ be a decomposition of a graph G.*

Let H be a set of edges and vertices of G. We say that H is complete *if, whenever it contains an edge or an inner vertex of a child C of G (seen as a component of $\mathcal{D}$), then it contains all the edges and inner vertices of C.*

Let K be a set of edges and vertices of the x-body of G. We denote by $\mathsf{complete}_{\mathcal{D}}(K)$ the set of edges and vertices of G, obtained from K by replacing every x-edge coming from a child C of G by the set of edges and inner vertices C.

▶ Remark 71. Note that if H is complete, there is a set S such that $H = \mathsf{complete}_{\mathcal{D}}(S)$.

Here is a picture illustrating complete sets. The green part is the body of G and the purple modules are its children. The yellow sets are complete, but the pink one is not.



▶ **Proposition 72.** *The following formulas are* CMSO^d *definable:*

$\mathsf{child}_{\mathcal{X}}(Y)$	:	*Y is the set of edges and inner vertices of a child of the input graph w.r.t. the decomposition $\mathcal{X}$.*
$\mathsf{is\text{-}complete}_{\mathcal{X}}(Y)$	:	*Y is complete wrt $\mathcal{X}$.*
$\mathsf{body\text{-}edge}_{\mathcal{X}}(Y)$	:	*Y is a singleton containing an edge from the body of the input graph wrt $\mathcal{X}$.*
$\mathsf{source}_{\mathcal{X}}(Y,Z)$	:	$\mathsf{child}_{\mathcal{X}}(Z)$ *and Y is a singleton containing the input of the corresponding child.*
$\mathsf{target}_{\mathcal{X}}(Y,Z)$	:	*the same as above, where input should be replaced by output.*
$\mathsf{choice}_{\mathcal{X}}(Y,Z)$	:	*Z contains all the body elements of Y, and for every child contained in Y, Z contains exactly one element of this child.*

We construct the formula $[\![\varphi]\!]$ by induction on the structure of φ. We suppose that φ is build using the syntax of CMSO where only set variables are allowed.

▶ **Definition 73.** *Let φ be a* CMSO *formula whose free variables are Γ. We define the* CMSO^d *formula $[\![\varphi]\!]$, whose free variables are $\Gamma \cup \{\mathcal{X}\}$, by induction as follows:*

$$
\begin{array}{lcl}
[\![\varphi \vee \psi]\!] & = & [\![\varphi]\!] \vee [\![\psi]\!] \\
[\![\neg\varphi]\!] & = & \neg\, [\![\varphi]\!] \\
[\![(|Y| \equiv k)[m]]\!] & = & \exists Z.\ \mathsf{choice}_{\mathcal{X}}(Y,Z) \wedge (|Z| \equiv k)[m] \\
[\![Y \subseteq Z]\!] & = & Y \subseteq Z \\
[\![a(Y)]\!] & = & a(Y) \qquad (a \neq x) \\
[\![x(Y)]\!] & = & \mathsf{child}_{\mathcal{X}}(Y) \vee (\mathsf{body\text{-}edge}_{\mathcal{X}}(Y) \wedge x(Y)) \\
[\![\exists Y.\ \varphi]\!] & = & \exists Y.\ \mathsf{is\text{-}complete}_{\mathcal{X}}(Y) \wedge [\![\varphi]\!] \\
[\![\mathsf{source}(Y,Z)]\!] & = & (\mathsf{body\text{-}edge}_{\mathcal{X}}(Z) \wedge \mathsf{source}(Y,Z)) \vee (\mathsf{child}_{\mathcal{X}}(Z) \wedge \mathsf{source}_{\mathcal{X}}(Y,Z)) \\
[\![\mathsf{target}(Y,Z)]\!] & = & (\mathsf{body\text{-}edge}_{\mathcal{X}}(Z) \wedge \mathsf{target}(Y,Z)) \vee (\mathsf{child}_{\mathcal{X}}(Z) \wedge \mathsf{target}_{\mathcal{X}}(Y,Z))
\end{array}
$$

Transfer results are results of this form: to check that a transformation $\mathsf{f}(G)$ of a structure G satisfies a formula φ, construct a formula $\mathsf{f}^{-1}(\varphi)$ that G should satisfy. The proposition below is a transfer result, where the transformation is the x-body.

▶ **Proposition 74.** *Given a* CMSO *sentence φ defining, there is a* CMSO^d *formula $[\![\varphi]\!]$ having $\mathcal{X}$ as unique free variable, such that for every graph G and every decomposition $\mathcal{D}$ of G* whose components are non-empty*:*

$$\langle G, \mathcal{X} \mapsto \mathcal{D} \rangle \models [\![\varphi]\!] \qquad \Leftrightarrow \qquad x\text{-}\mathsf{body}_{\mathcal{D}}(G) \models \varphi.$$

The formula $[\![\varphi]\!]$ expresses the fact that the x-body of the head of a decomposition satisfies φ. Using this formula and the localization construction of Prop. 14, we construct a formula $\mu x.L$ saying that the x-body of **all** the components of a decomposition satisfy φ.

▶ **Definition 75.** *If φ is a* CMSO *formula, we let $\mu x.\varphi$ be the following* CMSO^d *formula:*

$$\mu x.\varphi := \exists \mathcal{X}.\ \mathsf{decomp}(\mathcal{X}) \quad \wedge \quad \forall s.\forall Z.\ \forall t.\ (s,Z,t) \in \mathcal{X} \ \Rightarrow\ [\![\varphi]\!]\,|_{(s,Z,t)}$$

The following proposition says that the language of $\mu x.\varphi$ is the iteration of that of φ.

▶ **Proposition 76.** *If φ is a* CMSO *formula defining a language of non-empty graphs, then:*

$$\mathcal{L}(\mu x.\varphi) = \mu x.\mathcal{L}(\varphi).$$

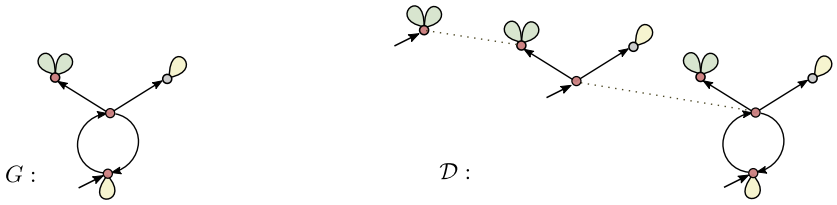
▶ **Corollary 77.** *If L is* CMSO *definable then $\mu x.L$ is* CMSO^d *definable.*

5.2 Guarded iteration of CMSO languages is CMSO^r

The idea here is that when the iteration $\mu x.L$ is guarded, L-decompositions can be encoded by sets of edges and vertices and by companion relations.

5.2.1 The case of test languages

Let $\mu x.L$ be a guarded iteration of type *test*, $G \in \mu x.L$ and $\mathcal{D}$ an L-decomposition of G. Suppose that G is the left graph below, and that the red vertices are the interfaces[3] of $\mathcal{D}$.



We claim that, thanks to the guard condition, this information is enough to reconstruct the whole decomposition $\mathcal{D}$. More precisely, we claim that the components of $\mathcal{D}$ are exactly the maximal modules of G, whose interfaces are the red vertices, as depicted above.

▶ **Definition 78.** *Let G be a graph and S be a set of vertices of G. We define $\mathcal{D}_{\mathsf{t}}(S)$ as the set of maximal modules of G, whose type is test, and whose interfaces belong to S.*

▶ **Proposition 79.** *Let $\mu x.L$ be a guarded iteration of type test. We have:*

$$G \in \mu x.L \qquad \Leftrightarrow \qquad \exists S.\ \ S \text{ is a set of vertices of } G \text{ and } \mathcal{D}_{\mathsf{t}}(S) \text{ is an } L\text{-decomposition of } G.$$

Proof. ($\Rightarrow$) Follows from Prop. 66. To prove ($\Leftarrow$), we define the property P_n as follows:

$$P_n : \qquad \forall G.\ \ G \in L^{n,x} \ \Rightarrow\ \exists S.\ \ S \text{ is a set of vertices of } G \text{ and } \mathcal{D}_{\mathsf{t}}(S) \text{ is an } L\text{-decomposition of } G.$$

We prove, by induction on n, that P_n is valid for every $n \geq 1$, and this is enough to conclude.

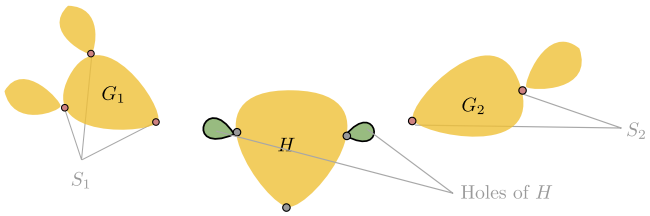
[3] Recall that test graphs are unary, hence all the components of a decomposition of G are unary.

When $n = 1$, take S to be the singleton containing the interface of G. We have that $\mathcal{D}_{\mathsf{t}}(S) = \{G\}$ and since $G \in L$, we have that $\mathcal{D}_{\mathsf{t}}(S)$ is an L-decomposition of G.

Let $G \in L^{n+1,x}$. By definition, there is a k-context H and graphs $G_1, \dots, G_k$ such that:

$$G = H[G_1, \dots, G_k], \quad H[x, \dots, x] \in L \quad \text{and} \quad G_i \in L^{n,x}, \text{ for } i \in [1, k].$$

Thanks to the guard condition, there is no module of H parallel to a hole of H. For every $i \in [1, k]$, let S_i be the set of vertices provided by the induction hypothesis applied to the graph G_i. Here is a picture illustrating these notations:



The set of subgraphs $\mathcal{D}$ defined below is an L-decomposition of G.

$$\mathcal{D} := \mathcal{D}_{\mathsf{t}}(S_1) \cup \cdots \cup \mathcal{D}_{\mathsf{t}}(S_k) \cup \{G\}$$

To conclude we only need to find a set of vertices S of G such that $\mathcal{D}_{\mathsf{t}}(S) = \mathcal{D}$. Let $S = S_1 \cup \cdots \cup S_k \cup \{\iota\}$, where ι is the interface of G. Let us show that $\mathcal{D}_{\mathsf{t}}(S) = \mathcal{D}$. This is a consequence of the following lemma:

▶ **Lemma 80.** *Let C be a context, K a graph and I an interface in H of the same arity as the hole of C. Suppose that the hole of C is not parallel to any module. We have:*

$$\mathsf{max\text{-}module}_{C[K]}(I) = \mathsf{max\text{-}module}_{K}(I)$$ ◀

▶ **Theorem 81.** *Suppose that $\mu x.L$ is a guarded iteration of type test. We have:*

$$L \textit{ is } \mathsf{CMSO} \textit{ definable} \quad \Rightarrow \quad \mu x.L \textit{ is } \mathsf{CMSO} \textit{ definable}$$

Proof. Let φ be a CMSO formula whose language is L. We transform the CMSO^d formula $\mu x.\varphi$ of Def. 75, whose language is $\mu x.L$, into a CMSO formula $\mu x^g.\varphi$ of the same language. The formula $\mu x^g.\varphi$ is obtained by replacing the quantification $\exists \mathcal{X}.$ by the set quantification $\exists S.$, and by replacing every sub-formula of $\mu x.\varphi$ of the form $(s, Z, t) \in \mathcal{X}$ by this formula:

$$(s = t) \ \wedge \ s \in S \ \wedge \ \text{“}(s, Z, t) \text{ is a maximal module”}$$

The last part of this formula is expressible in CMSO thanks to Prop. 32. The language of $\mu x^g.\varphi$ is the set of graphs for which we can find an L-decomposition encoded by a set of vertices S, and this is precisely the language $\mu x.L$ thanks to Prop. 79. ◀

5.2.2 The case of domain languages

Let $\mu x.L$ be a guarded iteration of type *domain*, $G \in \mu x.L$ and $\mathcal{D}$ an L-decomposition of G. Contrarily to the test case, the interfaces of $\mathcal{D}$ are not enough to reconstruct $\mathcal{D}$. Indeed, in this case, a component of $\mathcal{D}$ whose interface is v is not necessarily the maximal module at v, but some domain module of interface v, among possibly many others. A way to determine if a domain module is in the decomposition is to check whether it contains an interface of the decomposition. This works only for the components which are not the leaves of the decomposition. This is why we need to say explicitly which domain modules are the leaves. We do so by *coloring* the edges of the later.

In the following, we show that a set of vertices of a graph (representing the interfaces of a decomposition) together with a coloring of this graph (indicating which modules are leaves), is enough to recover the decomposition.

▶ **Definition 82** (Coloring, active modules.). *A* coloring *of a graph G is a set of its edges called* leaf edges. *A module of G is* active *if it contains a leaf edge.*

▶ **Definition 83** ($\mathcal{D}_{\mathsf{d}}(S, \mathsf{col})$). *Let G be a graph, S a set of vertices and* col *a coloring of G. We let $\mathcal{D}_{\mathsf{d}}(S, \mathsf{col})$ bet the set of active modules of G of type* d*, whose interfaces belong to S.*

▶ **Proposition 84.** *Let $\mu x.L : \mathsf{d}$ be a guarded iteration. We have:*

$$G \in \mu x.L \quad \Leftrightarrow \quad \exists S, \mathsf{col}. \quad \begin{array}{l} S \text{ is a set of vertices and } \mathsf{col} \text{ a coloring of } G \text{ such that} \\ \mathcal{D}_{\mathsf{d}}(S, \mathsf{col}) \text{ is an } L\text{-decomposition of } G. \end{array}$$

Proof. Similar to the proof of prop. 79. ◀

As in the previous section, we use Prop. 84 to get the following theorem.

▶ **Theorem 85.** *Suppose that $\mu x.L : \mathsf{d}$ be a guarded iteration. We have:*

$$L \text{ is } \mathsf{CMSO} \text{ definable} \quad \Rightarrow \quad \mu x.L \text{ is } \mathsf{CMSO} \text{ definable}$$

5.2.3 The case of parallel languages

The case of guarded iterations of type parallel is similar to the test case. Let $\mu x.L$ be a guarded iteration of type parallel, $G \in \mu x.L$ and $\mathcal{D}$ an L-decomposition of G. We show that the interfaces I of $\mathcal{D}$ is enough to recover the whole decomposition $\mathcal{D}$, because its components are the maximal modules of G whose interfaces belong to I. However, in this case, I is no longer a set of vertices, but a set of pairs of vertices, that is a relation on the vertices of G. We will show that this relation is necessarily a companion relation. Using this result and the fact that CMSO and CMSO^r have the same expressive power, we prove that the iteration is CMSO definable.

▶ **Definition 86** ($\mathcal{D}_{\mathsf{p}}(R)$). *Let G be a graph and R a relation on the vertices of G. We define $\mathcal{D}_{\mathsf{p}}(R)$ as the set of maximal modules of G, whose type is parallel, and whose interfaces belong to S.*

▶ **Proposition 87.** *Let $\mu x.L$ be a guarded iteration of type parallel. We have:*

$$G \in \mu x.L \quad \Leftrightarrow \quad \exists R. \quad \begin{array}{l} R \text{ is a set of vertices of } G \text{ and} \\ \mathcal{D}_{\mathsf{p}}(R) \text{ is an } L\text{-decomposition of } G. \end{array}$$

▶ **Proposition 88.** *Let $\mu x.L$ be an iteration of type parallel and let G a graph. The interfaces of every L-decomposition of G form a companion relation.*

Proof. We prove by induction on $n \geq 1$ that the interfaces of every L-decomposition of depth n of some graph G form a companion relation, witnessed by a set of paths P, such that the interface of G is witnessed by two parallel paths of P.

When $n = 1$, the decomposition $\mathcal{D}$ is reduced to the graph G. Since G is parallel, it has two parallel paths whose interface is the interface of G. Take P to be these two paths.

Suppose that $\mathcal{D}$ is a decomposition of depth $n + 1$. Hence it is of the form:

$$\mathcal{D} = \mathcal{D}_1 \cup \ldots \mathcal{D}_k \cup \{G\}$$

where $\mathcal{D}_i$ is an L-decomposition of depth at most n, of a graph G_i, for every $i \in [1,k]$. Let P_i be the set of paths provided by the induction hypothesis for $\mathcal{D}_i$, and let p_i, q_i be the two paths witnessing the interface of G_i, for $i \in [1,k]$.

We set $H := x\text{-}\mathsf{body}_{\mathcal{D}}(G)$. Since H is parallel, it has two parallel paths p and q whose interface is the interface of H. We transform the paths p and q of H into the paths p' and q' of G as follows. The paths p' and q' are obtained from p and q respectively by the following procedure: if e is an x-edge of H which is substituted by some G_i, then replace e by the path of p_i. Let P be the following set of paths:

$$P = (P_1 \setminus \{p_1\}) \cup \ldots (P_k \setminus \{p_k\}) \cup \{p', q'\}\,.$$

The set P is orthogonal and witnesses the interfaces of $\mathcal{D}$. Moreover, the interface of G is witnessed by two parallel paths of P, namely p' and q'. This concludes the proof. ◀

▶ **Theorem 89.** *Suppose that $\mu x.L$ is a guarded iteration of type parallel. We have:*

$$L \text{ is } \mathsf{CMSO} \text{ definable} \quad \Rightarrow \quad \mu x.L \text{ is } \mathsf{CMSO} \text{ definable}$$

5.2.4 The case of series languages

Let $\mu x.L$ be a guarded iteration of type series, G a graph in $\mu x.L$ and $\mathcal{D}$ an L-decomposition of G whose set of interfaces is I. As for the domain case, the set I is not enough to reconstruct the decomposition $\mathcal{D}$, and we need a coloring of the graph to determine which modules are the leaves of the decomposition $\mathcal{D}$. We show also that the set of interfaces I is a companion relation, which will be enough to conclude.

▶ **Definition 90** ($\mathcal{D}_{\mathsf{s}}(R, \mathsf{col})$)**.** *Let G be a graph, R a relation on the vertices of G and* col *a coloring of G. We let $\mathcal{D}_{\mathsf{s}}(R, \mathsf{col})$ bet the set of active modules of G of type series, whose interfaces belong to R.*

▶ **Proposition 91.** *Let $\mu x.L$ be a guarded iteration of type series. We have:*

$$G \in \mu x.L \quad \Leftrightarrow \quad \exists R, \mathsf{col}. \quad \begin{array}{l} R \text{ is a relation on the vertices of } G, \\ \mathsf{col} \text{ is a coloring of } G \text{ and} \\ \mathcal{D}_{\mathsf{s}}(R, \mathsf{col}) \text{ is an } L\text{-decomposition of } G. \end{array}$$

▶ **Proposition 92.** *Let $\mu x.L$ be a guarded iteration of type series and let G be a graph. The interfaces of every L-decomposition of G form a companion relation.*

▶ **Theorem 93.** *Suppose that $\mu x.L$ is a guarded iteration of type series. We have:*

$$L \text{ is } \mathsf{CMSO} \text{ definable} \quad \Rightarrow \quad \mu x.L \text{ is } \mathsf{CMSO} \text{ definable}$$

6 Recognizable implies regular

▶ **Theorem 94.** *If a language of tw_2-graphs is recognizable, then it is regular.*

We proceed gradually, by showing that this result holds for *domain-free graphs*, for *domain-free* graphs, then for tw_2 graphs.

▶ **Definition 95** (Domain-free)**.** *A graph is* domain-free *if all its domain modules are atomic.*

To give an example of how these proofs work, suppose that we have the following property:

▶ **Proposition 96.** *If a language of domain-free graphs is recognizable, then it is regular.*

Using Prop. 96, let us show the following property:

▶ **Proposition 97.** *If a language of domain graphs is recognizable, then it is regular.*

Proof. Let L be a language of domain graphs, $\mathcal{A}$ an algebra whose domain is D, $h : \mathbb{G}_{\mathsf{tw}_2}(\Sigma) \to \mathcal{A}$ a homomorphism and $F \subseteq D$ such that $h^{-1}(F) = L$. Let us show that L_v, the set of graphs over Σ whose image is v, is regular for every $v \in D$.

We associate every $v \in D$ with a new letter x_v and let $\Gamma := \{x_v \mid v \in D\}$. If $Q \subseteq D$, we denote by X_D the letters of Γ corresponding to these elements. We extend the homomorphism h to tw_2-graphs over the alphabet $\Sigma \cup \Gamma$ by letting $h(x_v) = v$ for every $x_v \in \Gamma$.

Let $v \in D$, $Q \subseteq D$ and $X \subseteq \Gamma$. We define the set of graphs $L_v^{Q,X}$ as follows. We let G be in this set if and only if:

- G is a domain graph over the alphabet $\Sigma \cup X$,
- the image of G is v,
- the image of the strict domain modules of G belong to Q.

Let us show that $L_v^{Q,X}$ is regular when $X_Q \cap X = \emptyset$. We proceed by induction on the size of Q. Suppose that $Q = \emptyset$. This case is based on the following lemma, obtained by case analysis on the graph G.

▶ **Lemma 98.** *Let G be a domain graph whose domain modules, distinct from G itself, are all atomic. There is a domain-free graph H such that $G = \mathsf{dom}(H)$.*

For every $w \in D$, let M_w^X be the set of domain-free graphs over the alphabet $\Sigma \cup X$ whose image is w. By Lem. 98, we have the following equation:

$$L_v^{\emptyset,X} = \bigcup_{\substack{w \in D \\ \mathsf{dom}(w)=v}} \mathsf{dom}(M_w)$$

which concludes the base case, thanks to Prop. 96. To handle the inductive case, we notice the following equality:

$$L_v^{Q\cup\{w\},X} = L_v^{Q,X\cup\{x_w\}}[\mu x_w.\ L_w^{Q,X\cup\{x_w\}}/x_w][L_w^{Q,X}/x_w] \qquad \blacktriangleleft$$

7 Conclusion

We are interested in studying the complexity-theoretic properties of our expressions. For instance understanding the complexity of deciding whether an expression is guarded, and what are the costs of translations between different formalisms (expressions, CMSO, algebra). This can help us get a better grasp of what role these expressions can play, and what is the fine interplay between these different formalisms. As stated in the introduction, this work on tree-width 2 graphs is meant to constitute a first step towards the case of tree-width k.

References

1 Mikolaj Bojanczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *LICS*, pages 407–416. ACM, 2016.

2 Mikolaj Bojanczyk and Michal Pilipczuk. Optimizing tree decompositions in MSO. In *STACS*, volume 66 of *LIPIcs*, pages 15:1–15:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

3 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960. doi:0.1002/malq.19600060105.

4 Enric Cosme-Llópez and Damien Pous. K4-free graphs as a free algebra. In *MFCS*, volume 83 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

5 Bruno Courcelle and Joost Engelfriet. Graph structure and monadic second-order logic – A language-theoretic approach. In *Encyclopedia of mathematics and its applications*, 2012.

6 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–51, 1961.

7 Joost Engelfriet. A regular characterization of graph languages definable monadic second-order logic. *Theor. Comput. Sci.*, 88(1):139–150, October 1991. doi:10.1016/0304-3975(91)90078-G.

8 Zsolt Gazdag and Zoltán L. Németh. A kleene theorem for bisemigroup and binoid languages. *Int. J. Found. Comput. Sci.*, 22:427–446, 2011.

9 Ferenc Gécseg and Magnus Steinby. Tree languages. In *Handbook of Formal Languages*, 1997.

10 S.C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–40, 1956. Princeton.

11 Dietrich Kuske and Ingmar Meinecke. Construction of tree automata from regular expressions. *RAIRO – Theoretical Informatics and Applications – Informatique Théorique et Applications*, 45(3):347–370, 2011. doi:10.1051/ita/2011107.

12 Neil Robertson and Paul D. Seymour. Graph minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 48:227–254, 1990.

13 James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2:57–81, 2005.

A Generic Solution to Register-Bounded Synthesis with an Application to Discrete Orders

Léo Exibard
Reykjavik University, Iceland

Emmanuel Filiot
Université libre de Bruxelles, Brussels, Belgium

Ayrat Khalimov
Université libre de Bruxelles, Brussels, Belgium

Abstract

We study synthesis of reactive systems interacting with environments using an infinite data domain. A popular formalism for specifying and modelling such systems is register automata and transducers. They extend finite-state automata by adding registers to store data values and to compare the incoming data values against stored ones. Synthesis from nondeterministic or universal register automata is undecidable in general. However, its register-bounded variant, where additionally a bound on the number of registers in a sought transducer is given, is known to be decidable for universal register automata which can compare data for equality, i.e., for data domain $(\mathbb{N}, =)$. This paper extends the decidability border to the domain $(\mathbb{N}, <)$ of natural numbers with linear order. Our solution is generic: we define a sufficient condition on data domains (regular approximability) for decidability of register-bounded synthesis. The condition is satisfied by natural data domains like $(\mathbb{N}, <)$. It allows one to use simple language-theoretic arguments and avoid technical game-theoretic reasoning. Further, by defining a generic notion of reducibility between data domains, we show the decidability of synthesis in the domain $(\mathbb{N}^d, <^d)$ of tuples of numbers equipped with the component-wise partial order and in the domain $(\Sigma^*, \prec)$ of finite strings with the prefix relation.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects; Theory of computation → Transducers

Keywords and phrases Synthesis, Register Automata, Transducers, Ordered Data Domains

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.122

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2105.09978

Funding This work was supported by the Fonds de la Recherche Scientifique – F.R.S.-FNRS under the MIS project F451019F. Emmanuel Filiot is a senior research associate at F.R.S.-FNRS. Léo Exibard was supported by the project "Mode(l)s of Verification and Monitorability (MoVeMnt)" (no. 217987-051) of the Icelandic Research Fund.

1 Introduction

Synthesis. Reactive synthesis aims at the automatic construction of an interactive system from its specification. A system is usually modelled as a transducer. In each step, it reads an input from the environment and produces an output. In this way, the transducer, reading an infinite sequence of inputs, produces an infinite sequence of outputs. Specifications are modelled as a language of desirable input-output sequences. The synthesis problem then asks to automatically construct a transducer whose input-output sequences belong to a given specification. Traditionally [30, 4], the inputs and outputs have been modelled as letters from a finite alphabet. This, however, limits the application of synthesis. Recently researchers have started investigating synthesis of systems working on data domains [12, 24, 15, 25, 2, 14].

Automata as specifications. In the finite-alphabet setting, specifications are usually given as logical formulas and a synthesiser performs a series of translations: first, from the formula to an automaton, then from the automaton to a game, and finally it searches for a winning strategy in the game. It is the second step, from automata to games, that captures the game-theoretic essence of synthesis, whereas the first step is an orthogonal problem of finding a convenient logical formalism. In the context of synthesis over data domains, this first step is problematic as there is no decidable, and expressive enough, logic having a corresponding automaton model. For that reason, in this paper we focus on the second step and use automata for specifications.

Register automata. A well-studied automata formalism for specifying and modelling data systems are register automata and transducers [22, 28, 23, 33]. Register automata extend classical finite-state automata to infinite alphabets $\mathbb{D}$ by introducing a finite number of *registers*. In each step, the automaton reads a data value $\ell \in \mathbb{D}$, compares it with the values held in its registers, then depending on this comparison it decides to store ℓ into some of its registers, and finally moves to a successor state. This way, it builds a sequence of *configurations* (pairs of state and register values) representing its run on reading a word from $\mathbb{D}^\omega$: it is accepting if the visited states satisfy a certain condition, e.g. parity. Transducers are similar except that in each step they also output the content of one register.

Universal register automata. Unlike classical finite-state automata, the expressive power of register automata depends on whether they are deterministic, nondeterministic, or universal (a.k.a. co-nondeterministic). Among these, universal register automata suit synthesis best. First, they can specify request-grant properties: every requested data shall be eventually outputted. This is a key property in reactive synthesis, and in the data setting it can be expressed by a universal register automaton but not by a nondeterministic one. Furthermore, universal register automata are closed, in linear time, under intersection. Hence they allow for succinct conjunction of properties, which is desirable in synthesis as specifications usually consist of many independent properties. Finally, in the register-free setting universal automata are often used to obtain synthesis methods feasible in practice [26, 31, 17, 4].

Data domains with order. Another factor affecting expressivity of register automata is the data-comparison operators. Originally, register automata compared data for equality only, i.e., operated in data domain $(\mathbb{D}, =)$ [22]. This limits synthesis applications as we cannot specify priority arbiters [8] that should give a resource to a requesting process with the lowest ID. Such properties require data domains with linear order $<$ (in addition to $=$). Further, there are data domains with dense order, like $(\mathbb{Q}, <)$, and those with discrete order, like $(\mathbb{N}, <)$. The domain $(\mathbb{Q}, <)$ is well-suited for abstracting physical phenomena like changing temperature in a room. However, for abstracting hardware, the domain $(\mathbb{N}, <)$ suits better as it excludes Zeno-like behaviours (when a process ID gets infinitely closer to another ID but never reaches it). The domain $(\mathbb{N}, <)$ is also interesting from the theoretical point of view as it demands new proof techniques.

Known synthesis results for register automata. Already for $(\mathbb{D}, =)$, the synthesis problem of register transducers from universal register automata is undecidable [12, 15]. Decidability is recovered in the deterministic case [15, 14], but, as argued above, universal automata are more desirable in synthesis. To circumvent undecidability, the works [24, 15, 25] studied *register-bounded* synthesis: given a universal register automaton *and a bound* k on the number

Table 1 Decidability of register-unconstrainted synthesis.

	$(\mathbb{D}, =)$	$(\mathbb{Q}, <)$	$(\mathbb{N}, <)$
DRA	✓[14]	✓[14]	✓[14]
NRA	✗[15]	✗	✗
URA	✗[15]	✗	✗

Table 2 Decidability of register-bounded synthesis.

	$(\mathbb{D}, =)$	$(\mathbb{Q}, <)$	$(\mathbb{N}, <)$
DRA	✓[24]	✓[13]	✓[THIS PAPER]
NRA	✗[15]	✗	✗
URA	✓[24]	✓[13]	✓[THIS PAPER]

of transducer registers, return a k-register transducer realising the automaton or "No" if no such transducer exists. They showed the decidability of register-bounded for $(\mathbb{D}, =)$, and it is not hard to adapt their techniques to $(\mathbb{Q}, <)$ and other oligomorphic domains [6], however the domain $(\mathbb{N}, <)$ remained elusive. Tables 1 and 2 summarise known and new results, where DRA/NRA/URA stand for deterministic/nondeterministic/universal register automata.

Contributions. We prove that register-bounded synthesis is decidable for $(\mathbb{N}, <)$ in time doubly exponential in the number of registers of the specification automaton and of the sought transducer. Our procedure is effective: it constructs a transducer if one exists. When the total number of registers is fixed, it is ExpTime-c, matching the complexity of classical (register-free) synthesis. This result generalises the works of [15, 25, 24] on $(\mathbb{D}, =)$. We then extend the decidability boundary farther to include the domain $(\mathbb{N}^d, <^d)$ of tuples of naturals with the component-wise partial order, and the domain $(\Sigma^*, \prec)$ of strings with the prefix relation.

Technical contributions. Our proof technique is generic and greatly simplifies the task of proving new synthesis decidability results by removing the need to reason about synthesis alltogether. We now describe the technique in detail.

The key idea of existing approaches [24, 15, 25] is to reduce the register-bounded synthesis problem in a data domain to a two-player Church game with a finite alphabet and an ω-regular winning condition. In such a game, two players alternately play for an infinite number of rounds. Adam, modelling the environment, picks a test over the k registers describing how its input data compares with the current content of the registers of a sought transducer. Eve, modelling the system, picks a subset of the k registers, meant to store the data, and a register whose value is meant for output. No data are manipulated in the game. Infinite plays in the game induce infinite sequences of tests, assignments, and outputs over the k registers, called *action words*; they are over a *finite* alphabet. Action words are meant to abstract data words; an action word is *feasible* if there is at least one data word that satisfies all its tests and assignments. The reduction ensures that any strategy of Eve winning in the game can be converted into a k-register transducer realising the specification, and vice versa. To this end, the game winning condition declares a play to be won by Eve if all data words satisfying the action word induced by the play are accepted by the specification automaton. In particular, a play whose action word is *un*feasible is won by Eve as it does not correspond to any environment-system interaction in the data domain. In the case of $(\mathbb{D}, =)$, such winning conditions are known to be ω-regular [24, 15, 25]. However, in $(\mathbb{N}, <)$ the set of feasible action words is not ω-regular [14], and neither is the winning condition. Such winning conditions could be expressed by nondeterministic ωS automata [5], but games with such objectives are not known to be decidable, to the best of our knowledge.

To overcome the latter obstacle, we introduce the notion of ω-regularly approximable (*regapprox*) data domains. A regapprox data domain has an ω-regular over-approximation of the set of feasible action words that is exact on the lasso-shaped action words (of the form

uv^ω). Thus, in regapprox domains the set of feasible lasso-shaped action words is ω-regular. This allows us to avoid dealing with non-ω-regularity and reduce synthesis to solving classic ω-regular games. Our first technical contribution is the generic decidability result:

For regapprox domains, register-bounded synthesis from URA is decidable.

The procedure is constructive: for realisable specifications it outputs a transducer. Note that all oligomorphic domains [6], e.g. $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$, are regapprox, because their sets of feasible action words are ω-regular, so our result subsumes works [15, 25, 24]. For $(\mathbb{N}, <)$, we construct its over-approximation relying on the result [14], and then instantiate the theorem.

There are many domains with discrete order resembling $(\mathbb{N}, <)$: the domain $(\mathbb{Z}, <)$ of integers, the domain $(\mathbb{N}^d, <^d)$ of tuples of naturals with the component-wise partial order, and even the domain $(\Sigma^*, \prec)$ of strings with the prefix relation. To further simplify decidability proofs on these domains, we define a natural and generic notion of reducibility between data domains. Intuitively, a data domain $\mathbb{D}$ *reduces* to $\mathbb{D}'$ if there is a rational transduction that relates action words in $\mathbb{D}$ and $\mathbb{D}'$ while preserving feasibility. Our second technical contribution is the reduction result:

If $\mathbb{D}$ reduces to $\mathbb{D}'$, and $\mathbb{D}'$ is regapprox, then $\mathbb{D}$ is regapprox.

This implies that a synthesis procedure for $\mathbb{D}'$ can be used to solve synthesis in $\mathbb{D}$. We illustrate the technique by reducing to $(\mathbb{N}, <)$ the domains $(\mathbb{N}^d, <^d)$ and $(\Sigma^*, \prec)$. The reduction for $(\Sigma^*, \prec)$ relies on the work [10]. These reductions entail the decidability of register-bounded synthesis on these domains.

Related works. We already mentioned the works [24, 15, 25, 13] on synthesis of register transducers in domains $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$, and that our result generalises them for the case of URAs. The paper [14] studies *Church's synthesis* for DRA specifications, where a *data* strategy not necessarily with finitely-many states is sought. However, they show that considering register transducers is sufficient, with with the number of registers equal that of the specification automaton. Hence our register-bounded synthesis procedure for URAs can also be used to solve the Church's synthesis problem.

Another formalism for specifications of data systems is that of *variable automata* [20]. The paper [16] studies synthesis of symbolic transducers from specifications given in a fragment of nondeterministic variable automata. They solve synthesis for data domain $(\mathbb{Q}, <)$ and leave the domain $(\mathbb{N}, <)$ for future work. Variable automata are incomparable with register automata, and their particular fragment cannot express request-grant properties of arbiters that we believe is desirable in synthesis.

Our proof techniques resemble those from some works on satisfiability of data logics. Constraint LTL [11] extends Linear Temporal Logic (LTL) by atoms allowing one to compare data values within the horizon or pre-defined length. The satisfiability of this logic is decidable for data domains $(\mathbb{D}, =)$, $(\mathbb{Q}, <)$, $(\mathbb{N}, <)$ [11], and $(\Sigma^*, \prec)$ [10]. Their proof technique relies on the abstraction of data values at different moments by relations between each other. For the data domain $(\mathbb{N}, <)$, they additionally prove that considering lasso-shaped witnesses of satisfiability is sufficient. Our generic synthesis result uses a similar idea by defining regapprox domains. We note that formulas in Constraint LTL can always be translated into universal register automata (which are more expressive) [32]. Hence our approach can be used to solve register-bounded synthesis from Constraint LTL.

The papers [19, 27] suggest a sound/incomplete procedure to synthesis from Temporal Stream Logic. This logic extends LTL by adding the atoms that are either first-order predicate terms or are assignments of variables to a first-order function term. Similarly, transducers

can test data using the predicate terms and update its values by the function terms. A transducer satisfies a specification if it does so under *every* interpretation of predicates and functions. It is possible to model domains like $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$ in their formalism, by encoding the axioms for $>$ and $=$ into specification. This would give a sound/*in*complete synthesis approach. Our approach is less general but retains the completeness.

More generally, our notion of regular approximation echoes a general idea common to verification techniques, for example of programs manipulating data variables (see, e.g., [21]), to abstract concrete behaviours by regular ones. When an over-approximation is used, it is guaranteed that if the abstract program satisfies some safety properties, so does the concrete program. This yields sound algorithm which are not necessarily complete. Here in the context of register automata, instead, we require that the over-approximation is exact on lasso-like executions, and show that this implies completeness (for the synthesis problem).

2 Synthesis Problem

Let $\mathbb{N} = \{0, 1, \dots\}$ denote the set of natural numbers including 0.

Data domain and data words. A *data domain* is a tuple $\mathcal{D} = (\mathbb{D}, P, C, c_0)$ consisting of an infinite countable set $\mathbb{D}$ of *data values*, a finite set P of interpreted *predicates* (predicate names with arities and their interpretations) which must contain the equality predicate $=$, a finite set $C \subset \mathbb{D}$ of *constants*, and a distinguished *initialiser* constant $c_0 \in C$. For example, $(\mathbb{N}, \{<, =\}, \{0\}, 0)$ is the data domain of natural numbers with the usual interpretation of $<$, $=$, and 0. In the tuple notation, we often omit the brackets, as well as the mention of $=$ and of c_0 when the initialiser constant is clear from the context. E.g., we write $(\mathbb{N}, <, 0)$ for $(\mathbb{N}, \{<, =\}, \{0\}, 0)$. Another familiar example is $(\mathbb{Z}, <, 0)$, which is the data domain of integers with the usual $<$, $=$, and 0. Throughout the paper we assume that the satisfiability problem of quantifier-free formulas built on the signature (P, C) is decidable in $\mathcal{D}$, and whenever we state complexity results, the satisfiability problem is additionally assumed to be decidable in PSpace. This is the case for all data domains considered in this paper. Finally, *data words* are infinite sequences $\mathit{d}_0\mathit{d}_1 \dots \in \mathbb{D}^\omega$, and for two sets I and O and a language $L \subseteq (I \cdot O)^\omega$, we call I and O its *input* and *output* alphabets respectively.

Action words. Fix a data domain $\mathcal{D} = (\mathbb{D}, P, C, c_0)$ and a finite set R of elements called *registers*. A register *valuation* (over $\mathcal{D}$) is a mapping $\nu : R \to \mathbb{D}$. Given a valuation ν, a variable x (not necessarily in R), and a data value $\mathit{d} \in \mathbb{D}$, define $\nu[x \leftarrow \mathit{d}]$ to be the valuation $R \cup \{x\} \to \mathbb{D}$ that maps x to d and every $r \in R \setminus \{x\}$ to $\nu(r)$. We extend this notation to any finite set $A = \{a_1, \dots, a_n\}$ by letting $\nu[A \leftarrow \mathit{d}] = \nu[a_1 \leftarrow \mathit{d}] \dots [a_n \leftarrow \mathit{d}]$.

A *test* (over $\mathcal{D}$) is a conjunction ($\wedge$) of distinct literals over predicates P and constants C, encoded as a set of literals $p(x_1, \dots, x_a)$ and $\neg p(x_1, \dots, x_a)$, where $p \in P$, a is the arity of p and $x_1, \dots, x_a \in R \cup C \cup \{\star\}$. The symbol $\star$ is a fresh symbol used as a placeholder for incoming data values. By convention, $\wedge\varnothing = \top$, and the empty set encodes the test that is always true. Depending on the context, we use the formula or set notation. A register valuation $\nu : R \to \mathbb{D}$ and data value $\mathit{d} \in \mathbb{D}$ *satisfy* a test φ, written $\nu, \mathit{d} \models \varphi$, if $\nu[\star \leftarrow \mathit{d}]$ satisfies φ, where predicates and constants are interpreted in the data domain $\mathcal{D}$. A test φ is *maximal* if it specifies the relation between all variables and constants wrt. the predicates, i.e. it is a maximally consistent conjunction of literals: $\varphi = \bigwedge_{\substack{p \in P \\ p \text{ of arity } r}} \bigwedge_{\substack{x_1, \dots, x_r \\ \in R \cup C \cup \{\star\}}} l_{p, x_1, \dots, x_r}$, where $l_{p, x_1, \dots, x_r} \in \{p(x_1, \dots, x_r), \neg p(x_1, \dots, x_r)\}$. Maximal tests are mutually exclusive: a

given valuation cannot satisfy simultaneously two of them. Observe that a test is equivalent to a (possibly exponential) disjunction of maximal ones. Let $\mathsf{Tst}_R^{\mathcal{D}}$ denote the set of all possible tests over registers R in domain $\mathcal{D}$, and $\mathsf{MTst}_R^{\mathcal{D}} \subset \mathsf{Tst}_R^{\mathcal{D}}$ the subset of maximal ones.

▶ Example. Consider domain $(\mathbb{N}, <, 0)$ and $R = \{r\}$. Atomic formulas are $r < \star$, $\star = r$, $r < 0$, $\star = 0$, etc. The test $0 < r \wedge r < \star$ specifies that the content of register r is strictly positive and that the incoming data is greater than it. It is not maximal, since it does not contain the atoms $0 < \star$, $\neg(\star = r)$, $\neg(\star = 0)$, $\neg(r = 0)$. For readability, we write $0 < r < \star$.

An *assignment* is a set $\mathsf{asgn} \subseteq R$ of registers meant to store the current input data value. Let $\mathsf{Asgn}_R = 2^R$ denote the set of all possible assignments. An *action* is a pair $(\mathsf{tst}, \mathsf{asgn}) \in \mathsf{Tst}_R \times \mathsf{Asgn}_R$. We now describe how valuations are updated: given a valuation ν, a data value $\mathscr{d}$, a test tst and an assignment asgn, we say that the valuation ν' is the *successor* of ν following action $(\mathsf{tst}, \mathsf{asgn})$ on reading $\mathscr{d}$, written $\nu \xrightarrow{\mathscr{d},\mathsf{tst},\mathsf{asgn}} \nu'$, if the data value satisfies the test, i.e. $\nu, \mathscr{d} \models \mathsf{tst}$, and $\nu' = \nu[\mathsf{asgn} \leftarrow \mathscr{d}]$.

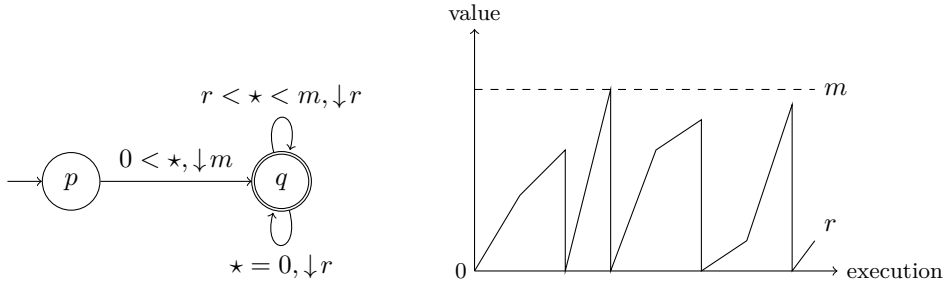
An *automaton action word*, or simply *action word*, is an infinite sequence of actions $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1) \ldots \in (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$. It is *feasible* by a sequence of valuation-data pairs $(\nu_0, \mathscr{d}_0)(\nu_1, \mathscr{d}_1) \ldots$ if $\nu_0 : r \in R \mapsto c_0$, i.e. ν_0 maps every $r \in R$ to c_0, and for all i: $\nu_i \xrightarrow{\mathscr{d}_i,\mathsf{tst}_i,\mathsf{asgn}_i} \nu_{i+1}$. We then say that the data word $\mathscr{d}_0\mathscr{d}_1 \ldots$ is *compatible* with $\bar{a}$. Let $\mathsf{AW}_R^{\mathcal{D}}$ denote the set of action words over R in $\mathcal{D}$, and $\mathsf{FEAS}_R^{\mathcal{D}}$ the subset of feasible ones. We may write either AW_R, or $\mathsf{AW}^{\mathcal{D}}$ or just AW when $\mathcal{D}$, R or both are clear from the context, similarly for FEAS.

▶ Example. Consider domain $(\mathbb{N}, <, 0)$ and $R = \{r\}$. For $r \in R$, the assignment $\{r\}$ is denoted $\downarrow r$. The action word $(0 < \star, \downarrow r)(\star < r, \downarrow r)^\omega$ is unfeasible in $(\mathbb{N}, <, 0)$, because it requires having an infinite chain of strictly decreasing values, which is not possible since $\mathbb{N}$ is well-founded. The same action word can be interpreted in $(\mathbb{Z}, <, 0)$ and in $(\mathbb{Q}, <, 0)$ and there it is feasible, as well as in $(\mathbb{Q}_+, <, 0)$ since $\mathbb{Q}_+$ is dense.

Register automata. A register automaton over data domain $\mathcal{D}$ is a tuple $S = (Q, q_0, R, \delta, \alpha)$, where Q is a finite set of *states* containing the *initial* state q_0, R is a finite set of *registers*, $\delta \subseteq Q \times \mathsf{Tst}_R \times \mathsf{Asgn}_R \times Q$ is a *transition relation*, and $\alpha : Q \to \{1, ..., c\}$ is a *priority function* where c is the priority *index*. A *configuration* of S is a pair $(p, \nu) \in Q \times \mathbb{D}^R$; it is *initial* if $p = q_0$ and $\nu : r \in R \mapsto c_0$. The configuration (q, ν') is a *successor* of (p, ν) on reading data value $\mathscr{d} \in \mathbb{D}$ and taking transition $p' \xrightarrow{\mathsf{tst},\mathsf{asgn}} q' \in \delta$, written $(p, \nu) \xrightarrow[S]{\mathscr{d},\mathsf{tst},\mathsf{asgn}} (q, \nu')$ or simply $(p, \nu) \xrightarrow[S]{\mathscr{d}} (q, \nu')$, if $p = p'$, $q = q'$ and $\nu \xrightarrow{\mathscr{d},\mathsf{tst},\mathsf{asgn}} \nu'$, i.e. $\nu, \mathscr{d} \models \mathsf{tst}$ and $\nu' = \nu[\mathsf{asgn} \leftarrow \mathscr{d}]$.

A *run* of S over a data word $\mathscr{d}_0\mathscr{d}_1 \ldots$ is a sequence of configurations $\rho = (q_0, \nu_0)(q_1, \nu_1) \ldots$ such that (q_0, ν_0) is initial and for every i, (q_{i+1}, ν_{i+1}) is a successor of (q_i, ν_i) on reading $\mathscr{d}_i$, on taking some transition $q_i \xrightarrow{\mathsf{tst}_i,\mathsf{asgn}_i} q_{i+1} \in \delta$. We then say that the automaton action word $(\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1) \ldots$ *labels* ρ; note that it is feasible by $\nu_0\mathscr{d}_0\nu_1\mathscr{d}_1 \ldots$. The run ρ is *accepting* if the maximal priority appearing infinitely often in $\alpha(q_0)\alpha(q_1) \ldots$ is even, otherwise it is *rejecting*. A data word may have several runs of S. For *universal* register automata, abbreviated URA, a word is *accepted* if all its runs are accepting; for *nondeterministic* automata, there should be at least one accepting run. The set of all data words over $\mathcal{D}$ accepted by S is called the *language* of S and denoted $L(S)$. We may write $L_{\mathcal{D}}(S)$ to emphasise that $L(S)$ is defined over $\mathcal{D}$.

A *finite (parity) automaton* (without registers) is a tuple $(\Sigma, Q, q_0, \delta, \alpha)$, where Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$, and the definition of runs, accepted words, and language is standard. Such automata operate on words from Σ^ω.



(a) A register automaton over $(\mathbb{N}, <, 0)$ (state q is accepting). **(b)** An example of the sequence of values taken by registers r and m along the run.

Figure 1 A register automaton whose action words do not form an ω-regular language.

Syntactical language of a register automaton. A register automaton $S = (Q, q_0, R, \delta, \alpha)$ can be treated syntactically, it then induces a universal finite automaton $S_{synt} = (\Sigma, Q, q_0, \delta, \alpha)$ with $\Sigma = \mathsf{Tst}_R \times \mathsf{Asgn}_R$. Note that since S_{synt} is universal, words that have no run are accepted. Notice that the language of S_{synt} may contain action words which are not feasible.

▶ Example. Consider the automaton of Figure 1a. Its syntactical language is

$$(0 < \star, \downarrow m)\big((r < \star < m, \downarrow r) \mid (0 = \star, \downarrow r)\big)^\omega$$

which includes not only feasible but also unfeasible action words, e.g. $(0 < \star, \downarrow m)(r < \star < m, \downarrow r)^\omega$. The feasible accepted action words have the form

$$(0 < \star, \downarrow m) \prod_{i=1}^{\infty} \big((r < \star < m, \downarrow r)^{n_i}(0 = \star, \downarrow r)\big)$$

such that the numbers $(n_i)_i$ are uniformly bounded by some value; the bound corresponds to the first read data value. This language is not ω-regular but an ωB-language [5].

Register transducers. A *k-register transducer* is a tuple $T = (Q, q_0, R, \delta)$, where Q, q, R ($|R| = k$) are as in automata but $\delta : Q \times \mathsf{MTst} \to \mathsf{Asgn} \times R \times Q$. Note that δ is a total function; moreover, since we restrict to maximal tests, exactly one test holds per incoming data value, so the transducers are deterministic and complete. A configuration is a pair $(p, \nu) \in Q \times \mathbb{D}^R$. From configuration (p, ν), on reading $\mathscr{d} \in \mathbb{D}$, the transducer takes the unique transition $p \xrightarrow{\mathsf{tst},\mathsf{asgn}|r} q$ such that $\nu, \mathscr{d} \models \mathsf{tst}$, updates its configuration to (q, ν') where $\nu \xrightarrow{\mathscr{d},\mathsf{tst},\mathsf{asgn}} \nu'$, and outputs the value $\nu'(r)$. Note that the output is produced *after* assignment. We then write $(p, \nu) \xrightarrow[T]{\mathscr{d},\mathsf{tst},\mathsf{asgn}|r,\nu'(r)} (q, \nu')$, or simply $(p, \nu) \xrightarrow[T]{\mathscr{d}|\nu'(r)} (q, \nu')$. A *run* of T on an input data word $\mathscr{d}_0^{\mathsf{i}}\mathscr{d}_1^{\mathsf{i}}\ldots$ is a sequence $(q_0, \nu_0)(q_1, \nu_1)\ldots$ such that (q_0, ν_0) is initial and for all $i \geq 0$, $(q_i, \nu_i) \xrightarrow{\mathscr{d}_i^{\mathsf{i}},\mathsf{tst}_i,\mathsf{asgn}_i|r_i,\mathscr{d}_i^{\mathsf{o}}} (q_{i+1}, \nu_{i+1})$ for some unique $\mathscr{d}_i^{\mathsf{o}} \in \mathbb{D}$. The sequence $\mathscr{d}_0^{\mathsf{o}}\mathscr{d}_1^{\mathsf{o}}\ldots$ is the *output word* of T on reading $\mathscr{d}_0^{\mathsf{i}}\mathscr{d}_1^{\mathsf{i}}\ldots$; since the transducers are deterministic and have a run on every input word, the output word is uniquely defined. The sequence $\mathscr{d}_0^{\mathsf{i}}\mathscr{d}_0^{\mathsf{o}}\mathscr{d}_1^{\mathsf{i}}\mathscr{d}_1^{\mathsf{o}}\ldots$ is called the *input-output word*. We then say that the *transducer action word* $\mathsf{tst}_0(\mathsf{asgn}_0, r_0)\mathsf{tst}_1(\mathsf{asgn}_1, r_1)\ldots \in \big(\mathsf{MTst}\cdot(\mathsf{Asgn} \times R)\big)^\omega$ is *feasible* by $(\nu_0, \mathscr{d}_0^{\mathsf{i}}, \mathscr{d}_0^{\mathsf{o}})(\nu_1, \mathscr{d}_1^{\mathsf{i}}, \mathscr{d}_1^{\mathsf{o}})\ldots$. It is naturally associated with the automaton action word $(\mathsf{tst}_0, \mathsf{asgn}_0)(\star = r_0, \varnothing)(\mathsf{tst}_1, \mathsf{asgn}_1)(\star = r_1, \varnothing)\ldots$, which is then feasible by $\nu_0\mathscr{d}_0^{\mathsf{i}}\nu_1\mathscr{d}_0^{\mathsf{o}}\nu_1\mathscr{d}_1^{\mathsf{i}}\nu_2\mathscr{d}_1^{\mathsf{o}}\ldots$. The set of all transducer action words over R in data domain $\mathcal{D}$ is denoted by $\mathsf{TW}_R^{\mathcal{D}}$. The *language* $L(T)$ consists of all input-output words of T.

A *finite transducer* is a standard Mealy machine: it is a tuple $(\Sigma, \Gamma, Q, q_0, \delta)$, where Σ and Γ are finite input and output alphabets, $\delta : Q \times \Sigma \to \Gamma \times Q$, and the definition of language is standard. Treating a register transducer T syntactically gives a finite transducer denoted T_{synt} of the same structure as T with $\Sigma = \mathsf{MTst}_R$ and $\Gamma = \mathsf{Asgn}_R \times R$.

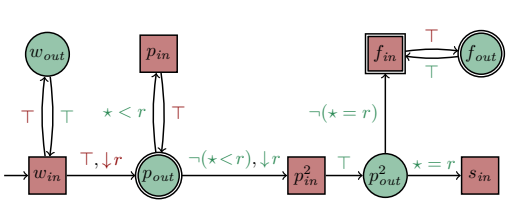
Synthesis problem. Fix a data domain $(\mathbb{D}, P, C, c_0)$. A register transducer T *realises* a register automaton S if $L(T) \subseteq L(S)$. The *register-bounded synthesis problem* is:

- input: $k \in \mathbb{N}$ and a URA S;
- output: yes iff there exists a k-register transducer which realises S.

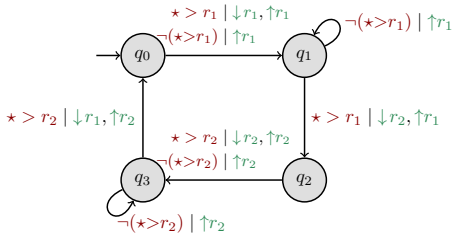
In this paper, when the synthesis problem is decidable, we are able to synthesise, i.e., effectively construct, a transducer realising the specification. We now make two remarks. First, notice that the number of transducer states is finite but unconstrained. Thus, register-bounded synthesis generalises classical register-free synthesis from (data-free) ω-regular specifications. Second, observe that transducers are complete, and therefore produce an ouptput word on every input word. Thus, a specification for which some input words do not have an associated output word is unrealisable. It is known that in the finite-alphabet case, the refined synthesis problem of good-enough synthesis [1], which requires a transducer to react only to inputs that belong to the domain of the specification, is still decidable. However, the good-enough register-bounded synthesis is undecidable [13, Chapter 8].

▶ Example. We illustrate the synthesis problem by describing a specification, its URA, and a register transducer realising it.

Let us start with the specification of priority arbiters. Such an arbiter reads an ID of a process requesting the resource, and outputs an ID of a process to whom the resource is granted. The specification requires that every requesting process is either acknowledged consecutively twice on the output, or this is done for a process of higher ID. We model the specification using the URA over $(\mathbb{N}, <, 0)$ with a single register from Figure 2a.



(a) URA for the priority-arbiter specification.



(b) Register transducer realising the specification.

Figure 2 A URA specification and a transducer implementing it.

The automaton reads words interleaving between arbiter data input and output, so its states are partitioned into box states (for reading input) and circle states (for reading output). The double-circle states are rejecting and can be visited only finitely often. Thus, a run looping in wait states w_{in} and w_{out} is accepting. Branching is universal, hence some run always loops around w_{in} and w_{out}. On reading an ID of a requesting process, a copy of the automaton moves from w_{in} to a pending state p_{out} while storing the ID into register r. It stays in states p_{out} and p_{in} as long as the request is not acknowledged, and such an infinite run is rejecting. If the request is eventually acknowledged (transitions from p_{out} to a sink state s_{in}), the run dies, so it is accepting. If a run reaches the failure state f_{in}, it is rejecting.

Figure 2b depicts a transducer with two registers r_1 and r_2 realising the above specification. On the left of the vertical bar are the tests over the inputs received by the transducer (in red), and on the right is the output action performed by the transducer (in green). For example, from state q_0 to q_1, if the input data $\mathbb{d}$ is larger than the data stored in register r_1, the transducer stores it into r_1, and outputs the content of r_1. The transducer uses one register to store the maximal value seen so far, while outputting the content of the other register, and the roles of these registers interchange as the transducer transits along the states. Thus, the instance of register-bounded synthesis with the described URA and $k = 2$ has a positive answer. However, when $k = 1$ the answer is negative.

3 Sufficient Condition for Decidable Synthesis for URA

In this section, we first show a reduction from register-bounded synthesis to (register-free) finite-alphabet synthesis. In the following, we fix a data domain $\mathcal{D}$. Given a specification S (as a URA over $\mathcal{D}$) and a bound k, we show how to construct a finite-alphabet specification $W^{\mathsf{F}}_{S,k}$ on action words over k registers, which is realisable by a finite-alphabet transducer iff S is realisable by a k-register transducer (Lemma 1). The main idea is to see the actions of the URA and of the sought k-register transducer as finite-alphabet letters. In particular, the specification $W^{\mathsf{F}}_{S,k}$ accepts a transducer action word $\bar{a}_k$ iff every action word $\bar{a}_S$ of the specification S, such that both $\bar{a}_k$ and $\bar{a}_S$ are feasible by the same data word, is accepted by S_{synt}. One can compose automata and transducer action words through a form of parallel product, which allows to talk about their joint feasibility. Then, in general, $W^{\mathsf{F}}_{S,k}$ is not necessarily ω-regular, and in a second step, we provide sufficient conditions on the data domain making synthesis wrt. $W^{\mathsf{F}}_{S,k}$ decidable, namely, that it can be under-approximated by an ω-regular language which coincides with $W^{f}_{S,k}$ over lasso words (Section 3.1). We obtain a general decidability result for data domains having this property (Theorem 4). We then instantiate this result for data domain $(\mathbb{N}, <, 0)$ (Section 3.2).

In the following, we fix a URA S with registers R_S and a disjoint set R_k consisting of k registers, and let $R = R_S \uplus R_k$. Given a transducer action word $\bar{a}_k = \mathsf{tst}^k_0\,(\mathsf{asgn}^k_0, r^k_0)\ldots \in \mathsf{TW}^{\mathcal{D}}_{R_k}$ and an automaton action word $\bar{a}_S = (\mathsf{tst}^{S\mathsf{i}}_0, \mathsf{asgn}^{S\mathsf{i}}_0)(\mathsf{tst}^{S\mathsf{o}}_0, \mathsf{asgn}^{S\mathsf{i}}_0)\ldots \in \mathsf{AW}^{\mathcal{D}}_{R_S}$, the product $\bar{a}_k \otimes \bar{a}_S$ of $\bar{a}_k$ and $\bar{a}_S$ is the automaton action word over registers R defined as $(\mathsf{tst}^k_0 \wedge \mathsf{tst}^{S\mathsf{i}}_0, \mathsf{asgn}^k_0 \cup \mathsf{asgn}^{S\mathsf{i}}_0)((\star = r^k_0) \wedge \mathsf{tst}^{S\mathsf{o}}_0, \mathsf{asgn}^{S\mathsf{i}}_0)\ldots$, which is essentially the parallel product of $\bar{a}_S$ and of the automaton word associated with $\bar{a}_k$.

We now show how to abstract a data specification given as URA S with registers R_S by a *finite-alphabet* specification over k-register transducer action words. Let $\mathsf{FEAS}^{\mathcal{D}}_R$ be the set of automata action words over R feasible in $\mathcal{D}$, then we define

$$W^{\mathsf{F}}_{S,k} = \left\{\bar{a}_k \in \mathsf{TW}_{R_k} \mid \forall \bar{a}_S \in \mathsf{AW}_{R_S} : \bar{a}_k \otimes \bar{a}_S \in \mathsf{FEAS}^{\mathcal{D}}_R \;\Rightarrow\; \bar{a}_S \in L(S_{synt})\right\}.$$

Thus, $W^{\mathsf{F}}_{S,k}$ *rejects* a feasible transducer action word $\bar{a}_k$ iff there is an automaton action word $\bar{a}_S$ feasible by the same data word as $\bar{a}_k$ and rejected by S.

▶ **Lemma 1.** *These two are equivalent:*

- *a URA S is realisable by a k-register transducer,*
- *$W^{\mathsf{F}}_{S,k}$ is realisable (by a finite-alphabet transducer).*

Proof. $\Rightarrow$: Assume that S is realisable by a register transducer T, i.e. $L_{\mathcal{D}}(T) \subseteq L_{\mathcal{D}}(S)$. Let $\bar{a}_k \in L(T_{synt})$, and let $\bar{a}_S \in \mathsf{AW}_{R_S}$ such that $\bar{a}_k \otimes \bar{a}_S \in \mathsf{FEAS}^{\mathcal{D}}_R$. Then, $\bar{a}_k \otimes \bar{a}_S$ is feasible by some input-output data word $w = \mathbb{d}^{\mathsf{i}}_0\mathbb{d}^{\mathsf{o}}_0\mathbb{d}^{\mathsf{i}}_1\mathbb{d}^{\mathsf{o}}_1\ldots$. By definition of the product, both $\bar{a}_k$ and $\bar{a}_S$ are feasible by w. Since $L_{\mathcal{D}}(T) \subseteq L_{\mathcal{D}}(S)$, if $\bar{a}_S$ labels a run of S on w, it means that it is accepting otherwise $w \notin L_{\mathcal{D}}(S)$ since S is a universal automaton. Thus, $\bar{a}_S \in L(S_{synt})$.

$\Leftarrow$: Conversely, assume that $W^{\mathsf{F}}_{S,k}$ is realisable by some finite transducer M, and let T be the associated register transducer, i.e. such that $T_{synt} = M$. Let $w \in L_{\mathscr{D}}(T)$ and let $\bar{a}_k$ be the action word labelling the run of T on w. Let $\bar{a}_S$ be an action word labelling a run of S on w if it exists (it might be that w is accepted by S by having no run on it). Then, $\bar{a}_k \otimes \bar{a}_S$ is feasible by w. By definition of $W^{\mathsf{F}}_{S,k}$, it means that $\bar{a}_S \in L(S_{synt})$, so $\bar{a}_S$ labels an accepting run of S on w. Overall, all runs of S on w are accepting, so $w \in L_{\mathscr{D}}(S)$. Thus, $L_{\mathscr{D}}(T) \subseteq L_{\mathscr{D}}(S)$, i.e. T realises S. ◀

3.1 General Decidability Result

In $(\mathbb{N}, <, 0)$, $W^{\mathsf{F}}_{S,k}$ is not ω-regular in general. To overcome this obstacle, we define the notion of ω-regularly approximable data domains. Such domains have an ω-regular equi-realisable subset of $W^{\mathsf{F}}_{S,k}$.

Let $lasso_R$ be the set of lasso-shaped[1] action words over a given set of registers R; we write *lasso* when R is clear. A data domain $\mathbb{D}$ is *ω-regularly approximable* (*regapprox*) if for every R there exists an ω-regular language $\mathsf{QFEAS}_R \subseteq (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$ satisfying

$$\mathsf{QFEAS}_R \cap lasso_R \;\subseteq\; \mathsf{FEAS}_R \;\subseteq\; \mathsf{QFEAS}_R$$

and recognisable by a nondeterministic Büchi automaton that can be effectively constructed given R. The definition implies that FEAS_R and QFEAS_R coincide on lasso words. Such a set QFEAS_R is called *regular approximation* and written as QFEAS when R is clear.

▶ Example. The data domains $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$ are regapprox because their sets FEAS_R for every R are ω-regular, so there is no need to approximate them. On these domains, to check whether a given action word is feasible, one can track the relations between the registers and check if the read tests are consistent with these relations. For instance, if $r_1 < r_2$ but we read the test $* = r_1 = r_2$, then the action word is unfeasible.

The domain $(\mathbb{N}, <, 0)$ is also regapprox. Here, it is not sufficient to track the relations between the registers. We also need to ensure that between any two stored data values only a bounded number of different values is inserted along the action word. (Recall the example on page 7 with Figure 1a.) However, when an action word is lasso-shaped, it suffices to check the absence of an *infinite* number of such insertions. The latter can be checked by an ω-regular automaton, which allows for proving the regapproximability of $(\mathbb{N}, <, 0)$.

Finally, consider the data domain $(\mathbb{N}, \{S, =\}, \{0\}, 0)$, where S is the successor relation, i.e. $S(a, b)$ holds iff $a = b + 1$. This domain is not regapprox. Intuitively, this is because the domain allows for counting, which enables non ω-regular phenomena even in lasso words. We prove this by contradiction. Consider the following ω-regular language of action words over a single register r:

$$L \;=\; \big\{ \big(S(*, r), {\downarrow}r\big)^n \big(S(r, *), {\downarrow}r\big)^m \big(* = 0 = r, \varnothing\big)^\omega \mid n, m \in \mathbb{N} \big\},$$

i.e. the value in r is incremented n times, then decremented m times, then compared to zero and not updated. L contains feasible as well as unfeasible action words. Every feasible word of L has $n = m$, hence $\mathsf{FEAS} \cap L$ is not ω-regular. Moreover, every word of L is a lasso, thus $L \cap lasso = L$. Let us assume that the data domain is regapprox, witnessed by QFEAS for $R = \{r\}$. Since $\mathsf{QFEAS} \cap lasso = \mathsf{FEAS} \cap lasso$ by definition, we get

$$\mathsf{QFEAS} \cap L = \mathsf{QFEAS} \cap lasso \cap L = \mathsf{FEAS} \cap lasso \cap L = \mathsf{FEAS} \cap L.$$

The language $\mathsf{QFEAS} \cap L$ is ω-regular, but $\mathsf{FEAS} \cap L$ is not. Contradiction. Therefore $(\mathbb{N}, \{S, =\}, \{0\}, 0)$ is not regapprox. ◀

[1] A word w is *lasso-shaped* (or regular, or ultimately periodic) if it is of the form $w = uv^\omega$ for some finite words u and v.

Given a URA S with registers R_S and k, we define

$$W^{\mathsf{QF}}_{S,k} = \{\bar{a}_k \mid \forall \bar{a}_S \colon \bar{a}_k \otimes \bar{a}_S \in \mathsf{QFEAS}_R \;\Rightarrow\; \bar{a}_S \in L(S_{synt})\},$$

where $R = R_S \uplus R_k$. The definition of $W^{\mathsf{QF}}_{S,k}$ differs from $W^{\mathsf{F}}_{S,k}$ only in using QFEAS_R instead of FEAS_R. Since $\mathsf{FEAS}_R \subseteq \mathsf{QFEAS}_R$, we have $W^{\mathsf{QF}}_{S,k} \subseteq W^{\mathsf{F}}_{S,k}$.

We now show that $W^{\mathsf{QF}}_{S,k}$ is ω-regular (which essentially follows from ω-regularity of QFEAS and S_{synt}), and estimate the size of an automaton recognising $W^{\mathsf{QF}}_{S,k}$ and the time needed to construct it. For that we use the following terminology for functions of asymptotic growth: a function is $poly(t)$ if it is $O(t^\kappa)$, $exp(t)$ if it is $O(2^{t^\kappa})$, and $2exp(t)$ if it is $O(2^{2^{t^\kappa}})$, for a constant $\kappa \in \mathbb{N}$. When $poly$, exp, and $2exp$ are used with several arguments, the maximal among them shall be taken for t. The construction and complexity analysis rely on standard automata techniques; see the full version for details.

▶ **Lemma 2.** *Let S be a URA and let $k \geq 1$. Then, $W^{QF}_{S,k}$ is ω-regular. Moreover, $W^{QF}_{S,k}$ is recognisable by a universal co-Büchi automaton with $O(2^k Nnc)$ many states that can be constructed in time $poly(N, n, exp(r, k))$, where n, r, and c are the number of states, registers, and priorities in S, and N is the number of states in a nondeterministic Büchi automaton recognising $QFEAS_{R_S \uplus R_k}$.*

We now prove that $W^{\mathsf{F}}_{S,k}$ and $W^{\mathsf{QF}}_{S,k}$ are equi-realisable. For ω-regular specifications (like $W^{\mathsf{QF}}_{S,k}$) there is no distinction between realisability by finite- and infinite-state transducers [7]. This is not known for $W^{\mathsf{F}}_{S,k}$ specifications over domains such as $(\mathbb{N}, <, 0)$; we leave this question for future work, and in this paper focus on realisability by *finite-state* transducers.

▶ **Lemma 3.** *$W^{F}_{S,k}$ is realisable by a finite-state transducer iff $W^{QF}_{S,k}$ is realisable by a finite-state transducer.*

Proof. Direction $\Leftarrow$ follows from the inclusion $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$, which implies $W^{\mathsf{QF}}_{S,k} \subseteq W^{\mathsf{F}}_{S,k}$. Consider direction $\Rightarrow$. Let T be a finite-state transducer that T does not realise $W^{\mathsf{QF}}_{S,k}$. We show that T does not realise $W^{\mathsf{F}}_{S,k}$ either. First, we have that $L(T) \not\subseteq W^{\mathsf{QF}}_{S,k}$, so the language $\{\bar{a}_k \otimes \bar{a}_S \in \mathsf{AW}^{\mathcal{D}}_R \mid \bar{a}_k \in L(T) \wedge \bar{a}_k \otimes \bar{a}_S \in \mathsf{QFEAS} \wedge \bar{a}_S \notin L(S_{synt})\}$ is nonempty. Since QFEAS and $L(S_{synt})$ are ω-regular, and since T is a finite-state transducer, this language is ω-regular. Thus, it contains a lasso-shaped word $\bar{a}_k \otimes \bar{a}_S$; by definition of the product, both $\bar{a}_k$ and $\bar{a}_S$ are then lasso-shaped. Since $\mathsf{QFEAS} \cap \mathit{lasso} \subseteq \mathsf{FEAS}$, we get that $\bar{a}_S$ is feasible, i.e. $\bar{a}_k \otimes \bar{a}_S \in \{\bar{a}_k \otimes \bar{a}_S \mid \bar{a}_k \in L(T) \wedge \bar{a}_k \otimes \bar{a}_S \in \mathsf{FEAS} \wedge \bar{a}_S \notin L(S_{synt})\}$, which implies that $L(T) \not\subseteq W^{\mathsf{F}}_{S,k}$: T does not realise $W^{\mathsf{F}}_{S,k}$. ◀

We are now able to prove the main result of this paper.

▶ **Theorem 4.** *Let $\mathcal{D}$ be a regapprox data domain such that for every set of registers R, one can construct a nondeterministic Büchi automaton with n_{QF} states recognising $QFEAS_R$ in time $f(|R|)$ for some function f. Then:*

- *register-bounded synthesis for URAs over $\mathcal{D}$ is decidable in time $exp(exp(k, r), n_{QF}, n, c) + f(k + r)$, where n is the number of states of the URA, c its number of priorities, r its number of registers, k is the number of transducer registers. It is ExpTime-c for fixed r and k.*
- *For every positive instance of the register-bounded synthesis problem, one can construct, within the same time complexities, a register transducer realising the specification.*

Proof. Lemmas 1,2,3 reduce register-bounded synthesis to (finite-alphabet) synthesis for the ω-regular specification $W^{\mathsf{QF}}_{S,k}$. Since synthesis wrt. to ω-regular specifications is decidable, we get the decidability part of the theorem. Let us now study the complexity. Let R_S

be the set of r registers of the URA and R_k be a disjoint set of k registers. First, one needs to construct an automaton recognising $\mathsf{QFEAS}_{R_S \cup R_k}$. This is done by assumption in time $f(k+r)$. Then, one can apply Lemma 2 and get that $W^{\mathsf{QF}}_{S,k}$ can be recognised by universal co-Büchi automaton A with $O(2^k n_{qf} nc)$ states, which can be constructed in time $poly(n_{qf}, n, exp(r,k))$. A universal co-Büchi automaton with m states can be determinised into a parity automaton with $exp(m)$ states and $poly(m)$ priorities (see e.g. [29]). Recall that the alphabet of A is $\mathsf{Tst}_k \cup (\mathsf{Asgn}_k \times R_k)$. Hence by determinising A, and seeing it as a two-player game arena, we get a parity game with $exp(k)$ edges (corresponding to the actions of Adam and Eve), $exp(exp(k), n_{\mathsf{QF}}, n, c))$ states, and $poly(exp(k), n_{\mathsf{QF}}, n, c))$ priorities. The latter can be solved in polynomial time in the number of its states, as the number of priorities is logarithmic in the number of states (see e.g. [9]), giving the overall time complexity $exp(exp(k), n_{\mathsf{QF}}, n, c))$ for solving the game. If we sum this to the complexity of constructing an automaton for $W^{\mathsf{QF}}_{S,k}$ plus the complexity for construction an automaton for QFEAS, we get $exp(exp(k), n_{\mathsf{QF}}, n, c)) + poly(n_{\mathsf{QF}}, n, exp(r,k)) + f(r+k)$, which is $exp(exp(k,r), n_{\mathsf{QF}}, n, c)) + f(r+k)$. If both r and k are fixed, then $exp(k,r)$ and $f(r+k)$ are constants, so the complexity is exponential only. It is folklore that the hardness holds in the register-free setting (for $r = k = 0$). See for example [18, Proposition 6] for a proof in the finite word setting over a finite alphabet (which straightforwardly generalises to infinite words). There, the proof is done for nondeterministic finite automata, but by determinacy, hardness also holds for universal automata, as they are dual.

Now, if a URA specification is realisable for some given k, then by Lemmas 1 and 3, $W^{\mathsf{QF}}_{S,k}$ is realisable by a finite-alphabet transducer M. Since $W^{\mathsf{QF}}_{S,k} \subseteq W^{\mathsf{F}}_{S,k}$, M also realises the specification $W^{\mathsf{F}}_{S,k}$. The mapping $\cdot_{synt}$ which turns a register transducer into a finite-alphabet transducer is bijective, and hence there exists a register transducer T such that $T_{synt} = M$. The proof of Lemma 1 exactly shows that T realises S, hence we are done. ◀

3.2 Register-bounded Synthesis over Data Domain $(\mathbb{N}, <, 0)$

We instantiate Theorem 4 for the data domain $(\mathbb{N}, <, 0)$. In [14], though there was no general notion of ω-regular approximability for data domains, it was implicitly used for $(\mathbb{N}, <, 0)$. The following fact follows from [14, Thm.8] after adapting to our notions.[2]

▶ **Fact 5.** For all R, $(\mathbb{N}, <, 0)$ has a witness QFEAS_R of ω-regular approximability expressible by a nondeterministic parity automaton with $exp(|R|)$ states and $poly(|R|)$ priorities, which can be constructed in time $exp(|R|)$.

A parity automaton can be translated to a nondeterministic Büchi automaton with a quadratic number of states, so we can instantiate Theorem 4 on domain $(\mathbb{N}, <, 0)$ and get:

▶ **Theorem 6.** *For a URA in $(\mathbb{N}, <, 0)$ with r registers, n states, and c priorities, k-register-bounded synthesis is solvable in time $exp(exp(r,k), n, c)$: it is singly exponential in n and c, and doubly exponential in r and k. It is ExpTime-c for fixed k and r.*

[2] Strictly speaking, their paper considers maximal tests only. However, using their deterministic automaton for QFEAS_R over action words with maximal tests, we can construct a *non*det. automaton recognising quasi-feasible action words with all tests, incl. partial ones. Our nondet. automaton, on reading a partial test, *guesses* its completion into a maximal test and simulates the original automaton on it.

4 Reducibility Between Data Domains

Theorem 6 relies on the study of feasibility of action words in $(\mathbb{N}, <, 0)$ of [14], which requires some effort. Such a study could in principle be generalised to domains such as $\mathbb{Z}$-tuples, as well as finite strings with the prefix relation, by leveraging the results of [10]. However, this would come at the price of a high level of technicality. We choose a different path, and introduce a notion of reducibility between domains, which allows us to reuse the study of $(\mathbb{N}, <, 0)$ and yields a compositional proof of the decidability of register-bounded synthesis for the quoted domains.

▶ **Definition.** A data domain $\mathcal{D}$ *reduces* to a data domain $\mathcal{D}'$ if for every finite set of registers R, there exists a finite set of registers R' and a rational relation[3] K between R-automata action words in $\mathcal{D}$ and R'-automata action words in $\mathcal{D}'$ that *preserves feasibility*, in the sense that for every R-action word $\bar{a} \in (\mathsf{Tst}_R^{\mathcal{D}}\mathsf{Asgn}_R)^\omega$: $\bar{a}$ is feasible in $\mathcal{D}$ iff there exists an R'-action word in $K(\bar{a}) \in (\mathsf{Tst}_{R'}^{\mathcal{D}'}\mathsf{Asgn}_{R'})^\omega$ feasible in $\mathcal{D}'$.[4]

▶ Remark. Reducibility is a transitive relation, since rational relations are closed under composition [3, Theorem 4.4], and feasibility preservation is transitive.

Since K is rational and preserves feasibility, for all R, $K^{-1}(\mathsf{QFEAS}_{R'})$ is a witness of regapproximability, where R' is as in the above definition (see the proof below for details), thus we get:

▶ **Lemma 7.** *If $\mathcal{D}$ reduces to $\mathcal{D}'$ and $\mathcal{D}'$ is regapprox, then $\mathcal{D}$ is regapprox.*

Proof. Let R be a fixed set of registers, and let R' be a set of registers satisfying the definition of reducibility. Let FEAS (respectively, FEAS') be the set of R-action words feasible in $\mathcal{D}$ (resp., feasible R'-action words in $\mathcal{D}'$).

Our goal is to define an ω-regular set QFEAS (for R) s.t. $\mathsf{QFEAS} \cap \mathit{lasso} \subseteq \mathsf{FEAS} \subseteq \mathsf{QFEAS}$. Since $\mathcal{D}'$ is regapprox, there is an ω-regular set QFEAS' (for R') s.t. $\mathsf{QFEAS}' \cap \mathit{lasso} \subseteq \mathsf{FEAS}' \subseteq \mathsf{QFEAS}'$. Define $\mathsf{QFEAS} = K^{-1}(\mathsf{QFEAS}')$; as the preimage of an ω-regular set by a rational relation, it is (effectively) ω-regular, thus satisfying one of the condition for $\mathcal{D}$ to be regapprox.

We now show that $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$. Before proceeding, notice that $\mathsf{FEAS} = K^{-1}(\mathsf{FEAS}')$, since K preserves feasibility. Since $\mathsf{FEAS}' \subseteq \mathsf{QFEAS}'$, we have $K^{-1}(\mathsf{FEAS}') \subseteq K^{-1}(\mathsf{QFEAS}')$, hence $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$.

It remains to show that $\mathsf{QFEAS} \cap \mathit{lasso} \subseteq \mathsf{FEAS}$. The inclusion $\mathsf{QFEAS}' \cap \mathit{lasso} \subseteq \mathsf{FEAS}'$ implies $K^{-1}(\mathsf{QFEAS}' \cap \mathit{lasso}) \subseteq K^{-1}(\mathsf{FEAS}') = \mathsf{FEAS}$ (the latter equality is because $\mathsf{FEAS} = K^{-1}(\mathsf{FEAS}')$). We prove that $\mathsf{QFEAS} \cap \mathit{lasso} \subseteq K^{-1}(\mathsf{QFEAS}' \cap \mathit{lasso})$, which entails the desired result. Pick an arbitrary $\bar{a} \in \mathsf{QFEAS} \cap \mathit{lasso}$. Since K is rational, $K(\bar{a})$ is ω-regular. Moreover, QFEAS' is ω-regular, which entails that $K(\bar{a}) \cap \mathsf{QFEAS}'$ is ω-regular as well. Since $\bar{a} \in K^{-1}(\mathsf{QFEAS}')$, the intersection $K(\bar{a}) \cap \mathsf{QFEAS}'$ is nonempty. Since $K(\bar{a}) \cap \mathsf{QFEAS}'$ is ω-regular and nonempty, it contains a lasso word $\bar{a}'$. Thus, $\bar{a}' \in K(\bar{a}) \cap \mathsf{QFEAS}' \cap \mathit{lasso}$, hence $\bar{a} \in K^{-1}(\mathsf{QFEAS}' \cap \mathit{lasso})$. ◀

As a direct consequence of Lemma 7 and Theorem 4, we get the following result:

[3] Given two finite alphabets Σ and Γ, a relation $K \subseteq \Sigma^\omega \times \Gamma^\omega$ is rational if there exists an ω-regular language $L \subseteq (\Sigma \cup \Gamma)^\omega$ such that $K = \{(\mathrm{proj}_\Sigma(u), \mathrm{proj}_\Gamma(u)) \mid u \in L\}$. This is equivalent to saying that it can be computed by a nondeterministic asynchronous finite-state transducer over input Σ with output in Γ^*. See, e.g., [3, Section 3].

[4] Note that we do not forbid the existence of unfeasible action words in the image.

▶ **Theorem 8.** *If a data domain $\mathcal{D}$ reduces to a regapprox data domain, then register-bounded synthesis is decidable for $\mathcal{D}$. Moreover, for any positive instance of the register-bounded synthesis problem over $\mathcal{D}$, one can effectively construct a register transducer realising the specification of that instance.*

4.1 Adding Labels to Data Values

As a first application, we show that one can equip data values with labels from a finite alphabet while preserving regapproximability. By Theorem 8, this yields decidability of register-bounded synthesis for such domains.

Formally, given a data domain $\mathcal{D} = (\mathbb{D}, P, C, c_0)$ and a finite alphabet Σ, we define the domain of Σ-labeled data values over $\mathcal{D}$ as $\Sigma \times \mathcal{D} = (\Sigma \times \mathbb{D}, P \cup \{\mathsf{lab}_\sigma \mid \sigma \in \Sigma\}, \Sigma \times C, (\sigma_0, c_0))$, where $\sigma_0 \in \Sigma$ is a fixed but arbitrary element of Σ and, for each $\sigma \in \Sigma$, $\mathsf{lab}_\sigma(\gamma, d)$ holds if and only if $\gamma = \sigma$.

▶ **Lemma 9.** *For all finite alphabet Σ and data domain $\mathcal{D}$, $\Sigma \times \mathcal{D}$ reduces to $\mathcal{D}$.*

Proof. Wlog we assume that the set of constants C is the singleton $\{c_0\}$ (modulo adding new predicates to P). Let $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_n\}$, where σ_0 is such that (σ_0, c_0) is the initialiser of $\Sigma \times \mathcal{D}$. We first define an encoding at the level of data words. Let $\mu : \Sigma \to \mathbb{D}$ be an injective mapping such that $\mu(\sigma_0) = c_0$. A data word u over $\mathcal{D}$ is a μ-encoding of $v = (\sigma_{i_1}, \ell_1)(\sigma_{i_2}, \ell_2)\ldots$ if it is equal to $\mu(\sigma_1)\ldots\mu(\sigma_n)\mu(\sigma_{i_1})\ell_1\mu(\sigma_{i_2})\ell_2\ldots$. The data word u is a *valid encoding* of v if it is a μ-encoding of v for some μ.

Now, the idea is to define a rational relation K from action words $\bar{a}$ over $\Sigma \times \mathcal{D}$ to actions words $\bar{b}$ over $\mathcal{D}$ such that $\bar{a}$ is feasible by some u iff there exists $\bar{b}$ such that $(\bar{a}, \bar{b}) \in K$ and $\bar{b}$ is feasible by a valid encoding of u. Let R be a set of registers and assume $\bar{a}$ is built over R. Let $R' = \{r_\sigma \mid \sigma \in \Sigma\} \uplus R$. Then, any $\bar{b}$ such that $(\bar{a}, \bar{b}) \in K$ should ensure that the n first data values are distinct and store them in $r_{\sigma_1}, \ldots, r_{\sigma_n}$ respectively. So, we require that $\bar{b}$ is of the form $\bar{b} = b_\Sigma \cdot b_{\bar{a}}$ where $b_\Sigma = (\mathsf{tst}_{\sigma_1}, \downarrow r_{\sigma_1}) \ldots (\mathsf{tst}_{\sigma_n}, \downarrow r_{\sigma_n})$ such that for all $1 \leq i \leq n$, $\mathsf{tst}_i = \bigwedge_{1 \leq j \leq i} \star \neq r_{\sigma_j}$. The second part $b_{\bar{a}}$ is an encoding of the tests and assignments of $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1)\ldots$. It is of the form $b_{\bar{a}} = (\mathsf{tst}_0^{lab}, \varnothing)(\mathsf{tst}_0^{data}, \mathsf{asgn}_0)(\mathsf{tst}_1^{lab}, \varnothing)(\mathsf{tst}_1^{data}, \mathsf{asgn}_1)\ldots$, where for all $i \geq 0$:

- for every predicate $p \in P$ of arity n, for every $x_1, \ldots, x_n \in R \cup \{\star\}$: if $(\neg)p(x_1, \ldots, x_n) \in \mathsf{tst}_i$, then $(\neg)p(x_1, \ldots, x_n) \in \mathsf{tst}_i^{data}$, and
- for all $\sigma \in \Sigma$ and $x \in R \cup \{\star\}$: $\mathsf{lab}_\sigma(x) \in \mathsf{tst}_i^{lab}$ iff $(r_\sigma = x) \in \mathsf{tst}_i$.

Correctness follows from the construction; see the extended paper for details. ◀

The latter result combined with Theorem 8 yields:

▶ **Corollary 10.** *Let $\mathcal{D}$ be an regapprox data domain and Σ be a finite alphabet, then register-bounded synthesis is decidable for $\Sigma \times \mathcal{D}$.*

4.2 Quantifier-Free Interpretations

When the relation between valuations over $\mathcal{D}$ and over $\mathcal{D}'$ is local, it is more convenient to operate directly at the level of tests. To that end, we define a notion of quantifier-free interpretation (see [13, Section 12.3.6] for a presentation of the notion in the context of data words), that allows us to encode elements of $\mathcal{D}$ as tuples of elements of $\mathcal{D}'$.

A *quantifier-free interpretation* (or *interpretation* for short) of dimension $l \geq 1$ with signature (P, C) over a data domain $\mathcal{D}' = (\mathbb{D}', P', C')$ is given by quantifier-free formulas over signature (P', C'). The formula $\phi_{\text{domain}}(x_1, \ldots, x_l)$ defines the domain $\mathbb{D} =$

$\{(\mathcal{d}_1, \dots, \mathcal{d}_l) \mid \mathcal{D}' \models \phi_{\text{domain}}(\mathcal{d}_1, \dots, \mathcal{d}_l)\}$. Then, for each constant symbol $c \in C$, the formula $\phi_c(x_1, \dots, x_l)$ defines the encodings[5] of c as the tuples $(\mathcal{d}_1^c, \dots, \mathcal{d}_l^c) \in \mathbb{D}$ that satisfy ϕ_c, i.e. such that $\mathcal{D}' \models \phi_c(\mathcal{d}_1^c, \dots, \mathcal{d}_l^c)$. Finally, for each predicate $p \in P$ of arity a (including $=$), the formula $\phi_p(x_1^1, \dots, x_l^1, \dots, x_1^a, \dots, x_l^a)$ defines the predicate $p^{\mathcal{D}} = \left\{(\mathcal{d}_1^1, \dots, \mathcal{d}_l^1, \dots, \mathcal{d}_1^a, \dots, \mathcal{d}_l^a) \,\middle|\, \mathcal{D}' \models \phi_R(\mathcal{d}_1^1, \dots, \mathcal{d}_l^1, \dots, \mathcal{d}_1^a, \dots, \mathcal{d}_l^a)\right\}$.

▶ **Lemma 11.** $(\mathbb{Z}, <, 0)$ *can be defined as a 2-dimensional interpretation of* $(\mathbb{N}, <, 0)$.

Proof. The encoding consists of two copies of $\mathbb{N}$, one for positive and one for negative integers, whose order is reversed. Formally, $\phi_{domain}(x_1, x_2) \coloneqq x_1 = 0 \vee x_2 = 0$. Then, $\phi_0(x_1, x_2) \coloneqq x_1 = 0 \wedge x_2 = 0$; $\phi_=((x_1, x_2), (y_1, y_2)) \coloneqq x_1 = y_1 \wedge x_2 = y_2$ and $\phi_<((x_1, x_2), (y_1, y_2)) \coloneqq (x_2 = y_2 = 0 \wedge x_1 < y_1) \vee (x_1 = y_1 = 0 \wedge x_2 > y_2) \vee (x_1 = 0 \wedge y_1 > 0)$. Then, $(\mathbb{Z}, <, 0)$ is isomorphic to this structure, through the bijection $n \geq 0 \mapsto (n, 0)$ and $n < 0 \mapsto (0, -n)$. ◀

More generally, d-uples of integers can be easily encoded. In the following, we fix $d \geq 1$. For $(n_1, ..., n_d), (m_1, ..., m_d) \in \mathbb{Z}^d$, define $(n_1, ..., n_d) <^d (m_1, ..., m_d)$ iff for all $i \in \{1, ..., d\}$, $n_i \leq m_i$ and $n_j < m_j$ for some $j \in \{1, \dots, d\}$; it is a partial order on $\mathbb{Z}^d$. The predicate $=^d$ is defined as expected.

▶ **Lemma 12.** $(\mathbb{Z}^d, =^d, <^d, 0^d)$ *can be defined as a d-dimensional interpretation of* $(\mathbb{Z}, <, 0)$.

Proof. Any tuple belongs to the domain, so we let $\phi_{\text{domain}} \coloneqq \top$. Then, $\phi_0(x_1, \dots, x_d) \coloneqq \bigwedge_{1 \leq i \leq d} x_i = 0$, $\phi_=((x_1, \dots, x_d), (y_1, \dots, y_d)) \coloneqq \bigwedge_{1 \leq i \leq d} x_i = y_i$, and similarly for $\phi_<$. ◀

The following theorem allows us to lift our results to the two domains above:

▶ **Theorem 13.** *If $\mathcal{D}$ is a quantifier-free interpretation over $\mathcal{D}'$, then $\mathcal{D}$ reduces to $\mathcal{D}'$.*

Proof (Sketch). We outline the proof, and refer to the extended paper for details. Let $\mathcal{D}' = (\mathbb{D}', P', C')$ be a data domain, and $\mathcal{D}$ be an interpretation over $\mathcal{D}'$ of dimension $l \geq 1$ with signature (P, C). The main idea is, given a set of registers R, to consider l copies of this set, meant to store each dimension of the interpretation. We also add l copies of C to store the encoding of constants, and, since tests are conducted before assignment, l registers to store each component of the input tuple. Overall, an action word $\bar{a}$ over R is sent to one over $(R \cup C \cup \{d\}) \times \{1, \dots, l\}$, where d is a fresh register variable. Then we construct the sought relation K as follows: first, it prefixes its image with a sequence of actions that store the encoding of constants in the corresponding registers, check that they indeed satisfy their respective ϕ_c, and ensure that all registers in $R \times \{1, \dots, l\}$ are initialised with the encoding of c_0. Note that the formulas are not necessarily conjuncts, so we put them in disjunctive normal form and consider all tests that are conjuncts of the DNF. Then, each action is processed separately: an action $(\mathsf{tst}, \mathsf{asgn})$ of $\bar{a}$ is associated with a sequence of $2l + 1$ actions that consist in reading each component of the input data value $\star$, store it in the corresponding copy of d, check that $\star$ indeed belongs to the domain (ϕ_{domain}), and that it satisfies tst (using the $(\phi_p)_{p \in P}$ to encode the predicates). Again, this implies converting the formulas in DNF, so a given action is in general associated with multiple ones. Since K consists in adding a prefix and then processing each action separately, it is rational. Moreover, it preserves feasibility; more precisely for any action word $\bar{a}$, each of its corresponding data word can be associated with its encoding in $K(\bar{a})$. ◀

By Theorems 6, 13 and 8, as well as Lemma 11, we get:

[5] Note that we do not assume the encoding to be unique.

▶ **Corollary 14.** *Register-bounded synthesis is decidable for* $(\mathbb{Z}, <, 0)$.

Then, since $(\mathbb{Z}, <, 0)$ reduces to $(\mathbb{N}, <, 0)$, and reducibility is transitive, we get, by Lemma 12 and Theorems 13 and 8:

▶ **Corollary 15.** *Register-bounded synthesis is decidable for* $(\mathbb{Z}^d, =^d, <^d, 0^d)$.

▶ Remark. One can similarly show that $\mathbb{N}^d$ reduces to $\mathbb{N}$. More generally, the above method allows one to lift decidability of register-bounded synthesis to tuples of data values where predicates are applied component-wise. Besides, note that $\mathbb{N}^d$ also reduces to $\mathbb{Z}^d$, by restricting $\mathbb{Z}^d$ to nonnegative values.

4.3 Finite Strings with the Prefix Relation

In this section, we show that synthesis is decidable over the data domain $(\Sigma^*, =, \prec, \epsilon)$, where Σ is a finite alphabet and $\prec$ denotes the prefix relation, leveraging a result of [10] that encodes prefix constraints as integer ones. This still requires some work, as we cannot use the notion of interpretation: a string valuation is encoded as an integer valuation with a *quadratic* number of registers. In the sequel, Σ is a fixed finite set of size $l \geq 2$.

First, $(\Sigma^*, =, \prec, \epsilon)$ reduces to the richer domain $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$, where, given $u, v \in \Sigma^*$, $\mathit{clen}(u, v)$ denotes the length of the longest common prefix of u and v, and, for $\triangleleft \in \{<, =\}$, $\mathit{clen}_\triangleleft(u, v, u', v')$ holds whenever $\mathit{clen}(u, v) \triangleleft \mathit{clen}(u', v')$. The reduction is direct, and follows the same lines as [10, Lemma 3]: $u \prec v$ is encoded as $(\mathit{clen}(u, u) = \mathit{clen}(u, v)) \wedge (\mathit{clen}(u, u) < \mathit{clen}(v, v))$, and K is a morphism on tests and the identity over assignments.

▶ **Lemma 16.** $(\Sigma^*, =, \prec, \epsilon)$ *reduces to* $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$.

Note also that satisfiability of tests over both domains is decidable, and NP-complete [10, Lemma 7]. It now remains to show that $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$ reduces to $(\mathbb{N}, =, <, 0)$. The proof draws on ideas similar to that of [10, Lemmas 8,9], which mainly relies on [10, Lemmas 5,6]. Here, it remains to lift them to our synthesis framework, and ensure that feasibility is preserved despite the dependencies induced by registers.

▶ **Lemma 17.** $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$ *reduces to* $(\mathbb{N}, =, <, 0)$.

Proof. We describe the main ideas of the proof; a full proof can be found in the extended version. From [10, Lemma 5,6], we know that a string valuation is characterised by the length of the longest common prefixes of all its pairs of values, when prefix constraints are concerned. This allows to encode Σ^* in $\mathbb{N}$: given a set R of registers, we introduce a register $\pi_{r,s}$ for each $(r, s) \in R' = (R \cup \{\mathsf{x}\})^2$, where x is an additional register name that denotes the input data value $\star$ in Σ^*. Along the execution, a register $\pi_{r,s}$ is meant to contain $\mathit{clen}(\nu(r), \nu(s))$. Note that in particular, $\pi_{r,r}$ contains the length of the word stored in r. At each step, we read a sequence of $|R|$ integers that each corresponds to the value of $\mathit{clen}(\star, r)$ for some $r \in R$, that we store in the corresponding register $\pi_{\star,r}$. We then check that they satisfy the *clen* constraints, as well as the properties of [10, Proposition 2]. The latter consist in logical formulas that can be encoded as tests in $(\mathbb{N}, =, <, 0)$, as they only use $=$ and $<$.

Using [10, Lemma 6], from a sequence of integer valuations (called *counter valuations* in [10]) that satisfy those properties, we can reconstruct a sequence of string valuations. As the integer valuations additionally satisfy the *clen* constraints, so does the string valuations. Thus, if an image R'-action word is feasible, the original action word is feasible. The converse direction is easier: given a sequence $\nu_0\nu_1\ldots$ of string valuations that is compatible with the R-action word, at step i one fills each $\pi_{r,s}$ with $\mathit{clen}(\nu_i(r), \nu_i(s))$. ◀

By Theorems 6 and 8, we get:

▶ **Corollary 18.** *Register-bounded synthesis is decidable for* $(\Sigma^*, =, \prec, \epsilon)$.

▶ Remark 19 (Complexity analysis). Note that the data domains in Corollaries 14, 15 and 18 all reduce to $(\mathbb{N}, <, 0)$ (all via some rational relations K depending on a set of registers R). The time complexities of those corollaries depend on the complexities of constructing, given a set of registers R, a nondeterministic Büchi automaton recognising $K^{-1}(\mathsf{QFEAS}_R^{(\mathbb{N},<,0)})$ for all the rational relations K defined in the proofs of those corollaries. It can be seen from those proofs that for any such rational relation K, it is possible to construct a nondeterministic Büchi transducer A_K with polynomially many states in $|R|$ recognising K. By taking the synchronized product of A_K with a nondeterministic automaton recognising $\mathsf{QFEAS}_R^{(\mathbb{N},<,0)}$, say of size n_{qf}, and by projecting it on its inputs, one obtains a nondeterministic Büchi automaton recognising $K^{-1}(\mathsf{QFEAS}_R^{(\mathbb{N},<,0)})$. It can be computed in time $poly(n_{qf})$. By Fact 5 and Theorem 4, one gets that the time complexities of k-register-bounded synthesis for data domains $(\mathbb{Z}, =, <, 0)$, $(\mathbb{Z}^d, =^d, <^d, 0^d)$ (for a fixed d) and $(\Sigma^*, =, \prec, \epsilon)$ is doubly exponential in k and r the number of registers of the specification, and singly exponential in the number of states of the URA and its number of priorities.

5 Conclusion

We have shown that register-bounded synthesis from specifications expressed by universal register-automata over $(\mathbb{N}, <, 0)$ is decidable within the same time complexity class as the case of URA over $(\mathbb{N}, =)$, completing the picture on synthesis from register automata over $(\mathbb{N}, =)$ and $(\mathbb{N}, <, 0)$: (unbounded) synthesis is undecidable for nondeterministic register automata [15], decidable for deterministic register automata over $(\mathbb{N}, =)$ [15] and over $(\mathbb{N}, <)$ [14], and register-bounded synthesis is decidable for URA over $(\mathbb{N}, =)$ [24, 15, 25] and $(\mathbb{N}, <, 0)$ (this paper), and undecidable for nondeterministic register automata [15]. We also get decidability for the data domains of integers, of tuples of integers and of finite words with the prefix relation, by reducing them to $(\mathbb{N}, <, 0)$. A simple complexity analysis (Remark 19) yields a doubly exponential decision procedure for register-bounded synthesis over these domains. Systematising this complexity analysis calls for a notion of polynomial reduction between data domains, that we leave for future work.

There are other challenging future research directions: first, universal automata, as argued in the introduction, are well suited for synthesis, and have been show in the register-free setting to be amenable to synthesis procedures which are feasible in practice [26, 31, 17, 4]. We plan on investigating extensions of these works to the register setting. In particular, our synthesis algorithm first reduces the problem to a synthesis problem over a *finite* alphabet with a specification given by a universal co-Büchi automaton. The latter problem is classically solved by reduction to a parity game obtained by determinising the universal co-Büchi automaton, e.g. by using Safra's determinization procedure. It is an interesting question whether Safraless procedures from [26, 31, 17] could be combined with our game reduction to get more practical algorithms. Another challenging research direction is to consider synthesis problems from logical specifications instead of automata, as the nice correspondences between automata and logics for word languages over finite alphabets do not carry over to data words. Nevertheless, URA encompass Constraint LTL [32], and we believe their expressive power could allow one to target other temporal-like logics with data.

References

1 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020. doi:10.1007/978-3-030-53291-8_28.

2 Béatrice Bérard, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. Parameterized synthesis for fragments of first-order logic over data words. In *FOSSACS*, volume 12077 of *Lecture Notes in Computer Science*, pages 97–118. Springer, 2020.

3 Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979. URL: https://www.worldcat.org/oclc/06364613.

4 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking*, pages 921–962. Springer, 2018.

5 M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 285–296, 2006.

6 Mikołaj Bojańczyk. *Slightly infinite sets*. Mikołaj Bojańczyk, 2019. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.

7 J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

8 Alex Bystrov, David John Kinniment, and Alexandre Yakovlev. Priority arbiters. In *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)(Cat. No. PR00586)*, pages 128–137. IEEE, 2000.

9 C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th ACM Symp. on Theory of Computing*, pages 252–263, 2017.

10 Stéphane Demri and Morgan Deters. Temporal logics on strings with prefix relation. *Journal of Logic and Computation*, 26(3):989–1017, 2016.

11 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint ltl. *Information and Computation*, 205(3):380–415, 2007. doi:10.1016/j.ic.2006.09.006.

12 R. Ehlers, S. Seshia, and H. Kress-Gazit. Synthesis with identifiers. In *Proc. 15th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 8318 of *Lecture Notes in Computer Science*, pages 415–433. Springer, 2014.

13 Léo Exibard. *Automatic Synthesis of Systems with Data*. PhD Thesis, Aix-Marseille Université (AMU); Université libre de Bruxelles (ULB), September 2021. URL: http://www.icetcs.ru.is/leoe/files/Exibard_ASSD_SASD.pdf.

14 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church Synthesis on Register Automata over Linearly Ordered Data Domains. In Markus Bläser and Benjamin Monmege, editors, *STACS 2021*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2021.28.

15 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of Data Word Transducers. *Logical Methods in Computer Science*, Volume 17, Issue 1, March 2021. doi:10.23638/LMCS-17(1:22)2021.

16 Rachel Faran and Orna Kupferman. On synthesis of specifications with arithmetic. In Alexander Chatzigeorgiou, Riccardo Dondi, Herodotos Herodotou, Christos Kapoutsis, Yannis Manolopoulos, George A. Papadopoulos, and Florian Sikora, editors, *SOFSEM 2020: Theory and Practice of Computer Science*, pages 161–173, Cham, 2020. Springer International Publishing.

17 E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643, pages 263–277, 2009.

18 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 125:1–125:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.125.

19 B. Finkbeiner, F. Klein, R. Piskac, and M. Santolucito. Temporal stream logic: Synthesis beyond the bools. In *Proc. 31st Int. Conf. on Computer Aided Verification*, 2019.

20 O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *Proc. 4th Int. Conf. on Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 561–572. Springer, 2010.

21 Ranjit Jhala, Andreas Podelski, and Andrey Rybalchenko. Predicate abstraction for program verification. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 447–491. Springer, 2018. doi:10.1007/978-3-319-10575-8_15.

22 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

23 M. Kaminski and D. Zeitlin. Extending finite-memory automata with non-deterministic reassignment. In *AFL*, pages 195–207, 2008.

24 A. Khalimov, B. Maderbacher, and R. Bloem. Bounded synthesis of register transducers. In *16th Int. Symp. on Automated Technology for Verification and Analysis*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.

25 Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.25.

26 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

27 Benedikt Maderbacher and Roderick Bloem. Reactive synthesis modulo theories using abstraction refinement, 2021. arXiv:2108.00090.

28 F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *26th Int. Symp. on Mathematical Foundations of Computer Science*, pages 560–572. Springer-Verlag, 2001.

29 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.

30 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

31 S. Schewe and B. Finkbeiner. Bounded synthesis. In *5th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2007.

32 Luc Segoufin and Szymon Torunczyk. Automata-based verification over linearly ordered data domains. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

33 N. Tzevelekos. Fresh-register automata. In *Proc. 38th ACM Symp. on Principles of Programming Languages*, pages 295–306, New York, NY, USA, 2011. ACM.

Twin-Width and Types

Jakub Gajarský
University of Warsaw, Poland

Michał Pilipczuk
University of Warsaw, Poland

Wojciech Przybyszewski
University of Warsaw, Poland

Szymon Toruńczyk
University of Warsaw, Poland

Abstract

We study problems connected to first-order logic in graphs of bounded twin-width. Inspired by the approach of Bonnet et al. [FOCS 2020], we introduce a robust methodology of *local types* and describe their behavior in contraction sequences – the decomposition notion underlying twin-width. We showcase the applicability of the methodology by proving the following two algorithmic results. In both statements, we fix a first-order formula $\varphi(x_1, \ldots, x_k)$ and a constant d, and we assume that on input we are given a graph G together with a contraction sequence of width at most d.

- One can in time $\mathcal{O}(n)$ construct a data structure that can answer the following queries in time $\mathcal{O}(\log \log n)$: given $w_1, \ldots, w_k$, decide whether $\varphi(w_1, \ldots, w_k)$ holds in G.
- After $\mathcal{O}(n)$-time preprocessing, one can enumerate all tuples $w_1, \ldots, w_k$ that satisfy $\varphi(x_1, \ldots, x_k)$ in G with $\mathcal{O}(1)$ delay.

In the first case, the query time can be reduced to $\mathcal{O}(1/\varepsilon)$ at the expense of increasing the construction time to $\mathcal{O}(n^{1+\varepsilon})$, for any fixed $\varepsilon > 0$. Finally, we also apply our tools to prove the following statement, which shows optimal bounds on the VC density of set systems that are first-order definable in graphs of bounded twin-width.

- Let G be a graph of twin-width d, A be a subset of vertices of G, and $\varphi(x_1, \ldots, x_k, y_1, \ldots, y_l)$ be a first-order formula. Then the number of different subsets of A^k definable by φ using l-tuples of vertices from G as parameters, is bounded by $O(|A|^l)$.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases twin-width, FO logic, model checking, query answering, enumeration

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.123

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2206.08248

Funding This work is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 948057).

Acknowledgements The authors thank Rose McCarty and Felix Reidl for many initial discussions on the type approach to first-order logic on graphs of bounded twin-width.

1 Introduction

Twin-width is a graph parameter recently introduced by Bonnet et al. [7]. Its definition is based on the concept of a *contraction sequence*: a sequence of partitions of the vertex set of the graph that starts with the partition into singletons, where every subsequent partition is

obtained from the previous one by merging two parts ending with the partition with one part. The main idea lies in measuring the *width* of a contraction sequence: it is the smallest integer d such that at every step, every part of the current partition is *impure* – neither completely adjacent nor completely non-adjacent – towards at most d other parts of that partition. The *twin-width* of a graph G is the smallest possible width of a contraction sequence of G. Thus, one may thinkthat a graph of bounded twin-width can be gradually "folded" into a single part so that at every point, every part has a non-trivial interaction with only a bounded number of other parts.

We remark that while twin-width was originally defined for graphs, the idea can be, and has been, generalized to any classes of binary relational structures, for instance ordered graphs [4] or permutations [8]. In this work we focus on the graph setting for simplicity. However, all our results lift to arbitrary structures over a fixed relational signature in which all relation symbols have arities at most two.

Since its recent introduction, multiple works have investigated combinatorial, algorithmic, and model-theoretic aspects of twin-width. In this work we are mostly interested in the two last ones. As proved by Bonnet et al. [7], provided a graph G is given together with a contraction sequence of width bounded by a constant, every property expressible in first-order logic can be verified in linear time on G; in other words, the model-checking problem for first-order logic can be solved in linear fixed-parameter tractable time. Further, bounded twin-width is preserved under transductions: if a class of graphs $\mathscr{C}$ has bounded twin-width, then any class that can be obtained from $\mathscr{C}$ by a fixed (first-order) transduction also has bounded twin-width [7]. Finally, as proved by Bonnet et al. [4], classes of *ordered* graphs that have bounded twin-width exactly coincide with those that are *monadically NIP*, that is, do not transduce all graphs. All these results witness that twin-width is a model-theoretically important notion and a vital element of the emerging structural theory for graphs based around the notion of a (first-order) transduction. See [8, 13] for further discussion.

In this work we take a closer look at the model-checking algorithm for graphs of bounded twin-width, proposed in [7]. The basic technical notion used there is that of a *morphism tree*. While this is not explicit in [7], it is clear that morphism trees are combinatorial objects representing strategies in a form of an Ehrenfeucht-Fraïsse game, and basic operations on morphism trees correspond to manipulations on strategies. Mirroring the standard approach taken in finite model theory, one should be able to define a notion of a *type* suited for the setting of contraction sequences, as well as a corresponding model of an Ehrenfeucht-Fraïsse game that can be used to argue about properties of types such as compositionality. Providing robust foundations for such a type-based methodology for contraction sequences is the main goal of this work.

We remark that the type/game based perspective of the approach of [7], which we explained above, was recently briefly outlined in [5, Section 5].

Our contribution. We introduce the notion of a *local type* that is suited for describing first-order properties of tuples of vertices in vertex-partitioned graphs. Intuitively speaking, the rank-k local type of a tuple $\mathbf{w}$ in a graph G with vertex partition $\mathcal{P}$ is the set of all quantifier rank k formulas satisfied by $\mathbf{w}$, where we restrict quantification as follows. Whenever a new vertex, say z, is quantified, one has to specify the part $P \in \mathcal{P}$ which contains z, but at depth i of quantification one can quantify only over parts that are at distance at most 2^{k-i} from parts containing already quantified vertices (including vertices of $\mathbf{w}$). Here, we mean the

distance in the *impurity graph*: the graph on parts of $\mathcal{P}$ where two parts of $\mathcal{P}$ are adjacent if and only if they are neither completely adjacent nor completely non-adjacent. This definition mirrors, in logical terms, the morphism trees of Bonnet et al. [7]. In particular, it applies the same idea that the radius of quantification decreases exponentially with the depth.

We prove a set of fundamental lemmas for manipulation of local types upon consecutive steps in a contraction sequence. These reflect the mechanics of morphism trees of [7], but by basing the argumentation essentially on Ehrenfeucht-Fraïssé games, the obtained explanation is arguably simpler and more insightful. Also, contrary to [5, 7], the introduced toolbox applies to tuples of vertices, and not only to single parts in the contraction sequence. This is important in our applications, which we discuss next.

We use the toolbox of local types to give the following algorithmic results on first-order expressible problems in graphs of bounded twin-width. The first one concerns the problem of *query answering*, and the second concerns the problem of *query enumeration*. In both theorems we assume that the graph is *specified through* a contraction sequence; this is explained in Section 2. For a tuple of parameters $\mathbf{p}$, the notation $\mathcal{O}_{\mathbf{p}}(\cdot)$ hides factors depending on $\mathbf{p}$.

▶ **Theorem 1.** *Suppose we are given an n-vertex graph G specified through a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of width d, and a first-order formula $\varphi(\mathbf{x})$, where $\mathbf{x}$ is a set of variables. Then one can construct in time $\mathcal{O}_{d,\varphi}(n)$ a data structure that can answer the following queries in time $\mathcal{O}_{d,\varphi}(\log\log n)$: given $\mathbf{w} \in V(G)^{\mathbf{x}}$, decide whether $G \models \varphi(\mathbf{w})$.*

▶ **Theorem 2.** *Suppose we are given an n-vertex graph G specified through a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of width d, and a first-order formula $\varphi(\mathbf{x})$, where $\mathbf{x}$ is a set of variables. Then after preprocessing in time $\mathcal{O}_{d,\varphi}(n)$, one can enumerate all tuples $\mathbf{w} \in V(G)^{\mathbf{x}}$ such that $G \models \varphi(\mathbf{w})$ with $\mathcal{O}_{d,\varphi}(1)$ delay.*

Note that in Theorem 1 there is a factor of the form $\mathcal{O}_{d,\varphi}(\log\log n)$ appearing in the query time. This is a consequence of using a data structure for orthogonal range queries of Chan [9] that supports queries in time $\mathcal{O}(\log\log n)$. As explained in [19], there is also a simple data structure for orthogonal range queries that, for any fixed $\varepsilon > 0$, offers query time $\mathcal{O}(1/\varepsilon)$ at the expense of increasing the construction time and the space complexity to $\mathcal{O}(n^{1+\varepsilon})$. By replacing the usage of the data structure of Chan with this simple data structure, we may obtain the same tradeoff in Theorem 1: The query time is reduced to $\mathcal{O}(1/\varepsilon)$, while the construction time and the space complexity is increased to $\mathcal{O}(n^{1+\varepsilon})$; this holds for any fixed $\varepsilon > 0$.

Theorems 1 and 2 mirror classic results on evaluation and enumeration of monadic second-order queries on trees [2, 10, 14] (which imply analogous results for graphs of bounded treewidth and of bounded cliquewidth), and of first-order queries on classes of bounded expansion [11, 15] and nowhere dense classes [22]. Therefore, we believe that the applications discussed above witness the robustness of the developed methodology.

As another application, we prove optimal bounds on *VC density* of set systems definable in graphs of bounded twin-width. Suppose $\varphi(\mathbf{x}, \mathbf{y})$ is a first-order formula with free variables partitioned into $\mathbf{x}$ and $\mathbf{y}$. For a graph G and a subset of vertices A, we define the *Stone space*

$$S^{\varphi}(A) \coloneqq \{\, \{\, \mathbf{a} \in A^{\mathbf{x}} \mid G \models \varphi(\mathbf{a}, \mathbf{b}) \,\} \;:\; \mathbf{b} \in V(G)^{\mathbf{y}} \,\}.$$

In other words, every tuple $\mathbf{b} \in V(G)^{\mathbf{y}}$ gives rise to the subset $\varphi(A, \mathbf{b}) \subseteq A^{\mathbf{x}}$ consisting of those tuples $\mathbf{a}$ that together with $\mathbf{b}$ satisfy φ. Then the Stone space $S^{\varphi}(A)$ consists of all sets $\varphi(A, \mathbf{b})$ that can be defined in this way. See for example [18] for a discussion of this notion and its applications.

In general graphs, $S^{\varphi}(A)$ can be as large as the whole powerset of A^x. However, under various structural assumptions, it will be typically much smaller. For instance, suppose that the twin-width of G is bounded by a constant d. Then by combining the results of Bonnet et al. [7] with that of Baldwin and Shelah [3], one can argue that the VC dimension of $S^{\varphi}(A)$, regarded as a set system over universe $A^{\mathbf{x}}$, is bounded by a constant depending only on d and φ. Consequently, by the Sauer-Shelah Lemma [21, 23], the cardinality of $S^{\varphi}(A)$ is bounded polynomially in $|A|$. However, the degree of this polynomial bound, which is known as the *VC density* (studied for example in [1]), still depends on d and φ, and in a quite non-explicit way. We prove that in fact, there is a much sharper upper bound: the VC density is bounded by just the number of variables in $\mathbf{y}$.

▶ **Theorem 3.** *Let G be a graph of twin-width at most d, A be a subset of vertices of G, and $\varphi(\mathbf{x}, \mathbf{y})$ be a first-order formula. Then*

$$|S^{\varphi}(A)| \leqslant \mathcal{O}_{d,\varphi}\left(|A|^{|\mathbf{y}|}\right).$$

It is easy to see (see e.g. [18]) that even in edgeless graphs one cannot hope for a bound better than $|A|^{|\mathbf{y}|}$, and therefore the bound of Theorem 3 is asymptotically optimum.

Theorem 3 mirrors analogous results for monadic second-order formulas on classes of bounded treewidth or cliquewidth [16], and for first-order formulas on classes of bounded expansion and nowhere dense classes [18]. We remark that the case $|\mathbf{x}| = |\mathbf{y}| = 1$ follows from the fact that classes of bounded twin-width are closed under first-order transductions, combined with known linear upper bounds on the *neighborhood complexity*[1] in classes of bounded twin-width [6, 20]. Tackling multiple free variables requires a better understanding of types for tuples of vertices, which is exactly where our methodology of local types comes into play.

Organization. After preliminaries in Section 2, we present the framework of local types in Section 3. Then we prove Theorem 1 in Sections 4. Theorems 2 is proved in Section 5. Theorem 3 is deferred to the full version, due to space constraints. Easy proofs of statements marked with ♠ are also deferred to the full version.

2 Preliminaries

Graphs. In this paper we work with finite, undirected graphs and we use standard graph notation. By $|G|$ we denote the number of vertices of a graph G.

A pair of disjoint vertex subsets $A, B \subseteq V(G)$ is *complete* if every vertex of A is adjacent to every vertex of B, and *anti-complete* if there is no edge with one endpoint in A and the other one in B. The pair A, B is *pure* if it is complete or anti-complete, and *impure* otherwise.

[1] In our notation, bounds on neighborhood complexity exactly correspond to the case when $\mathbf{x} = \{x\}$, $\mathbf{y} = \{y\}$, and $\varphi(x, y)$ just checks that x and y are adjacent.

A *trigraph* is a structure in which there is a vertex set and every pair of distinct vertices is bound by exactly one of the following three symmetric relations: adjacency, non-adjacency, and impurity. Thus, graphs are trigraphs without impurities. Given a partition of $\mathcal{P}$ of the vertex set of a graph G, we define the *quotient trigraph* $G/\mathcal{P}$ as the trigraph on vertex set $\mathcal{P}$ where distinct $A, B \in \mathcal{P}$ are adjacent if the pair A, B is complete in G, non-adjacent if the pair is anti-complete, and impure towards each other if A, B is impure. For a trigraph H, its *impurity graph* $\mathsf{Imp}(H)$ is the graph on vertex set H where two vertices $u, v \in V(H)$ are considered adjacent if they are impure towards each other in H.

Contraction sequences. Let G be a graph on n vertices. A *contraction sequence* for G is a sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of partitions of the vertex set of G such that:

- $\mathcal{P}_1$ is the partition into singletons;
- $\mathcal{P}_n$ is the partition with one part;
- for each $t \in [n]$, $t > 1$, $\mathcal{P}_t$ is obtained from $\mathcal{P}_{t-1}$ by taking some two parts $A, B \in \mathcal{P}_{t-1}$ and *contracting* them: replacing them with a single part $A \cup B \in \mathcal{P}_t$.

Indices $t \in [n]$ will be called *times*. The *width* of the contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ is the maximum degree in graphs $\mathsf{Imp}(G/\mathcal{P}_t)$, at all times $t \in [n]$. The *twin-width* of G is the minimum possible width of a contraction sequence of G.

If G is supplied with a total order $\leqslant$ on $V(G)$, then a subset of vertices A is *convex* if it forms an interval in $\leqslant$, that is, if $a, b \in A$ then also $c \in A$ whenever $a \leqslant c \leqslant b$. A partition is convex if all its parts are convex, and a contraction sequence is convex if all its partitions are convex.

Additional notation for partitions and contraction sequences. Fix a graph G with a partition $\mathcal{P}$ of its vertices. We will use the following notation.

Denote $G_{\mathcal{P}} \coloneqq G/\mathcal{P}$ and $G^{\mathsf{imp}}_{\mathcal{P}} \coloneqq \mathsf{Imp}(G_{\mathcal{P}})$. By $\mathrm{dist}_{\mathcal{P}}(\cdot,\cdot)$ we denote the distance function in $G^{\mathsf{imp}}_{\mathcal{P}}$: for $A, B \in \mathcal{P}$, $\mathrm{dist}_{\mathcal{P}}(A, B)$ is the minimum length of a path in $G^{\mathsf{imp}}_{\mathcal{P}}$ connecting A and B, and $+\infty$ if there is no such path. We extend this notation to subsets, or tuples of elements of $\mathcal{P}$: $\mathrm{dist}_{\mathcal{P}}(X, Y)$ denotes the minimum, over all A occurring in X and B occurring in B, of $\mathrm{dist}_{\mathcal{P}}(A, B)$.

For a set of parts $\mathcal{F} \subseteq \mathcal{P}$ and a radius parameter $r \in \mathbb{N}$, the *r-vicinity* of $\mathcal{F}$, denoted $\mathsf{Vicinity}^r_{\mathcal{P}}(\mathcal{F})$, is the trigraph induced in $G_{\mathcal{P}}$ by all parts at distance at most r from any part belonging to $\mathcal{F}$, that is

$$\mathsf{Vicinity}^r_{\mathcal{P}}(\mathcal{F}) \coloneqq G_{\mathcal{P}}[\{A \in \mathcal{P} \mid \mathrm{dist}_{\mathcal{P}}(A, \mathcal{F}) \leqslant r\}].$$

We may use notation $\mathsf{Vicinity}^r_{\mathcal{P}}(\cdot)$ for single parts or tuples of parts with the obvious meaning.

For brevity, whenever a graph G and its contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are clear from the context, we fix the following notation. First, in all the notation defined above, concerning partitions, we write s in the subscript instead of $\mathcal{P}_s$. So for instance we write G_s to denote $G_{\mathcal{P}_s}$, and G^{imp}_s to denote $G^{\mathsf{imp}}_{\mathcal{P}_s}$, and $\mathrm{dist}_s(\cdot,\cdot)$ to denote $\mathrm{dist}_{\mathcal{P}_s}(\cdot,\cdot)$, etc.

Fix a finite set of variables $\mathbf{x}$. For a pair of times $s, t \in [n]$, $s \leqslant t$, and a tuple of parts $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$, we define the tuple $\mathbf{u}\langle s \to t\rangle \in \mathcal{P}_t^{\mathbf{x}}$ as follows: for each $y \in \mathbf{x}$, $\mathbf{u}\langle s \to t\rangle(y)$ is the unique part of $\mathcal{P}_t$ that contains $\mathbf{u}(y)$. For a tuple $\mathbf{u} \in V(G)^{\mathbf{x}}$ of vertices and $s \in [n]$, by $\mathbf{u}\langle s\rangle \in \mathcal{P}_s^{\mathbf{x}}$ we denote the unique tuple whose y-component, for $y \in \mathbf{x}$, is the part of $\mathcal{P}_s$ containing $\mathbf{u}(y)$.

For $s \in [n-1]$, by B_{s+1} we denote the unique part of $\mathcal{P}_{s+1}$ that is the union of two parts in $\mathcal{P}_s$. For a parameter $r \in \mathbb{N}$, we define the *r-relevant region* in G_s as follows:

$$\mathsf{Relevant}_s^r := G_s[\{C \in \mathcal{P}_s \mid C \subseteq B_{s+1}, \text{ or } C \in \mathcal{P}_{s+1} \text{ and } \mathrm{dist}_{s+1}(C, B_{s+1}) \leqslant r\}].$$

In other words, $\mathsf{Relevant}_s^r$ is the trigraph induced in G_s by the two parts of $\mathcal{P}_s$ that get contracted into B_{s+1} and all parts of $\mathcal{P}_s$ that stay intact in $\mathcal{P}_{s+1}$ and are at distance at most r from B_{s+1} in G_{s+1}^{imp}.

Note that we have $|\mathsf{Relevant}_s^p| \leqslant \mathcal{O}_{d,p}(1)$ for all $s \in [n-1]$. The next lemma shows that the p-relevant regions can be computed efficiently.

▶ **Lemma 4 (♠).** *Suppose a graph G on vertex set $[n]$ is provided through a convex contraction sequence $\mathcal{P}$ of width d. Then for a given $p \in \mathbb{N}$, one can in time $\mathcal{O}_{d,p}(n)$ compute the trigraphs $\mathsf{Relevant}_s^p$ for all $s \in [n-1]$.*

Specifying a graph through its contraction sequence. In all algorithmic statements we will assume that a graph is given by specifying its contraction sequence together with some auxiliary information encoding the edge relation. We now make this precise.

Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be a contraction sequence of a graph G. We assume that every part participating in the partitions $\mathcal{P}_1, \ldots, \mathcal{P}_n$ (that is, every element of the union $\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_n$, where each $\mathcal{P}_i$ is viewed as a set of sets of vertices) is specified through a unique identifier taking a single machine word. For $\mathcal{P}_1$, the identifiers of (singleton) parts coincide with identifiers of the corresponding vertices. Then sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ is represented by providing the following information for every time $s \in [n]$, $s > 1$:

- The identifiers of the two parts $A, A' \in \mathcal{P}_{s-1}$ that get contracted at time s, and the identifier of the obtained part $B = A \cup A' \in \mathcal{P}_s$.
- A list of identifiers of parts $C \in \mathcal{P}_s$ such that the pair B, C is impure in G (that is, the impurities incident to B in $\mathsf{Imp}(G/\mathcal{P}_s)$).
- For each part C on the list above, the relation (completeness, anti-completeness, or impurity) between C and A and between C and B in $G/\mathcal{P}_{s-1}$.

It is easy to see that this representation uniquely defines the graph G. Since the representation takes $\mathcal{O}_d(1)$ machine words at any time s, we can thus represent an n-vertex graph of twin-width d using $\mathcal{O}_d(n)$ machine words.

We now show that, for a graph given through a contraction sequence, one can reindex the vertex set using integers from $[n]$ so that the contraction sequence becomes convex.

▶ **Lemma 5 (♠).** *Suppose a graph G is given by specifying a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of width d. Then one can in time $\mathcal{O}_d(n)$ compute a bijection $\eta \colon V(G) \to [n]$ such that mapping G and $\mathcal{P}_1, \ldots, \mathcal{P}_n$ through η yields an isomorphic graph G' on vertex set $[n]$ and its contraction sequence $\mathcal{P}'_1, \ldots, \mathcal{P}'_n$ such that $\mathcal{P}'_1, \ldots, \mathcal{P}'_n$ is convex in the natural order on integers in $[n]$.*

Note that if a graph is reindexed using Lemma 5, then every part participating in the resulting contraction sequence is an interval in $[n]$. Hence, as the identifier of a part we can simply use a pair of vertices – the left endpoint and the right endpoint – and such identifiers can be computed in time $\mathcal{O}_d(n)$ by scanning the contraction sequence. We will therefore assume that contraction sequences are convex with respect to a fixed ordering of the vertices, and the (convex) parts are identified by their endpoints.

First-order logic. We fix a countable set of variables, together with its enumeration. If Ω is a set and $\mathbf{x}$ is a finite set of variables, then an $\mathbf{x}$-*tuple* with entries in Ω is a function from $\mathbf{x}$ to Ω. Tuples are by convention denoted with boldface small letters, e.g. $\mathbf{u}$ or $\mathbf{v}$. The set of all $\mathbf{x}$-tuples with entries in Ω is denoted by $\Omega^{\mathbf{x}}$. When $\mathbf{a} \in \Omega^{\mathbf{x}}$ is an $\mathbf{x}$-tuple and $b \in \Omega$, then by $\mathbf{a}b$ we denote the $(\mathbf{x} \cup \{y\})$-tuple that extends $\mathbf{a}$ and maps the first variable (according to the fixed enumeration of all variables) y not in $\mathbf{x}$, to b.

We consider standard first-order logic on graphs by modeling them as relational structures where the universe is the vertex set and there is a single binary predicate signifying adjacency. For a graph G, a formula $\varphi(\mathbf{x})$, where $\mathbf{x}$ is the set of free variables of φ, and a tuple of vertices $\mathbf{w} \in V(G)^{\mathbf{x}}$, we write $G \models \varphi(\mathbf{w})$, or $G, \mathbf{w} \models \varphi(\mathbf{x})$, to denote that $\mathbf{w}$ satisfies $\varphi(\mathbf{x})$ in G. We sometimes consider formulas with an explicitly partitioned set of free variables, e.g., $\varphi(\mathbf{x}, \mathbf{y})$. Sentences are formulas with no free variables.

Logical types. While the usual definition of a logical type of quantifier rank k of a tuple $\mathbf{a}$ of vertices of G is the set of all formulas $\varphi(\mathbf{x})$ such that $G \models \varphi(\mathbf{a})$, we will rely on a definition which is more suitable for our purposes and is well known to be equivalent, by the result of Ehrenfeucht and Fraïssé (see for example [12]).

Let $\mathbf{x}$ be a finite set of variables. An *atomic type with variables* $\mathbf{x}$ is a maximal consistent set S of formulas of the form $x = y$, $x \neq y$, $E(x, y)$, $\neg E(x, y)$, where $x, y \in \mathbf{x}$. Here by *consistent* we mean that there is some graph G and a tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$ that satisfies all formulas occurring in the atomic type (this is decidable, as it is sufficient to consider graphs G with $|G| \leqslant |\mathbf{x}|$).

For $\mathbf{a} \in V(G)^{\mathbf{x}}$, the *atomic type of* $\mathbf{a}$ *in* G is the atomic type with variables $\mathbf{x}$ which consists of all formulas of the form $E(x, y)$ or $x = y$, where $x, y \in \mathbf{x}$, such that $G, \mathbf{a} \models x = y$ or $G, \mathbf{a} \models E(x, y)$.

▶ **Definition 6.** *Let G be a graph $\mathbf{x}$ a finite set of variables and $k \in \mathbb{N}$. For every $\mathbf{a} \in V(G)^{\mathbf{x}}$ we define its* type of quantifier rank k*, denoted* $\mathrm{tp}^k(\mathbf{a})$*, as follows.*

- *If $k = 0$, then $\mathrm{tp}^0(\mathbf{a})$ is the atomic type of $\mathbf{a}$ in G.*
- *If $k > 0$, then $\mathrm{tp}^k(\mathbf{a}) = \{\mathrm{tp}^{k-1}(\mathbf{a}b) \mid b \in V(G)\}$.*

For $k \geqslant 1$ we also set $\mathrm{tp}^k(G) = \{\mathrm{tp}^{k-1}(a) \mid a \in V(G)\}$.

This definition is usually intuitively explained in terms of Ehrenfeucht-Fraïssé games. Namely, two $\mathbf{x}$-tuples $\mathbf{a}$ and $\mathbf{b}$ of vertices of two graphs G and H, respectively, have equal types of quantifier rank k if and only if Duplicator wins the k-round game on the graphs G and H, where the initial pebbles in G and H are placed on the vertices occurring in $\mathbf{a}$ and in $\mathbf{b}$, respectively. Indeed, suppose $\mathrm{tp}^k(\mathbf{a}) = \mathrm{tp}^k(\mathbf{b})$, where $k > 0$, and that Spoiler places a pebble on a vertex c of G. Then, since $\mathrm{tp}^{k-1}(\mathbf{a}c) \in \mathrm{tp}^k(\mathbf{a})$ by definition and $\mathrm{tp}^k(\mathbf{a}) = \mathrm{tp}^k(\mathbf{b})$, we have that there is some $d \in \mathrm{tp}^k(\mathbf{b})$ such that $\mathrm{tp}^{k-1}(\mathbf{b}d) \in \mathrm{tp}^k(\mathbf{b})$. Then Duplicator responds by placing the pebble on the vertex d, and we have that $\mathrm{tp}^{k-1}(\mathbf{a}c) = \mathrm{tp}^{k-1}(\mathbf{b}d)$ so, by inductive assumption, Duplicator wins in the $k-1$ round game from the current configuration, which shows that Duplicator has a winning strategy in the k round game starting from $\mathbf{a}$ and $\mathbf{b}$. The implication in the other direction proceeds similarly.

As is well known, the set of types of $\mathbf{x}$-tuples of quantifier rank k that are realized by some tuple $\mathbf{a}$, in some graph, is non-computable, even though this set has size bounded in terms of $\mathbf{x}$ and k. To overcome this problem, the usual solution is to define the set of abstract types (that may not be realized as actual types), which is computable from $\mathbf{x}$ and k, has bounded size, and contains all types that may arise. This is done as follows.

Define $\mathsf{Types}^0_{\mathbf{x}}$ as the set of all atomic types over $\mathbf{x}$ and $\mathsf{Types}^k_{\mathbf{x}} := \{M \mid M \subseteq \mathsf{Types}^{k-1}_{\mathbf{x}y}\}$. Note that for any G and any $\mathbf{a} \in V(G)^{\mathbf{x}}$ it holds that $\mathrm{tp}^k(\mathbf{a}) \in \mathsf{Types}^k_{\mathbf{x}}$, but $\mathsf{Types}^k_{\mathbf{x}}$ can also contain objects which are not realized by any tuple of vertices $\mathbf{a}$ of any graph.

For a graph G we set $\mathsf{Types}^k_{\mathbf{x}}(G) := \{\mathrm{tp}^k(\mathbf{a}) \mid \mathbf{a} \in V(G)^{\mathbf{x}}\}$. Note that we have $\mathrm{tp}^k(G) = \mathsf{Types}^{k-1}_x(G)$.

The following is well known and follows from the fact that our definition of types is equivalent to the usual definition of types using formulas.

▶ **Proposition 7.** *Let G be a graph, $\mathbf{x}$ a set of variables and $k \in \mathbb{N}$.*

- $|\mathsf{Types}^k_{\mathbf{x}}(G)| = \mathcal{O}_{k,\mathbf{x}}(1)$,
- *For any $\mathbf{a} \in V(G)^{\mathbf{x}}$ and any first-order formula $\varphi(\mathbf{x})$ of quantifier rank at most k one can determine whether $G \models \varphi(\mathbf{a})$ from $\mathrm{tp}^k(\mathbf{a})$ in time $\mathcal{O}_{k,\mathbf{x}}(1)$.*
- *For any first-order sentence φ of quantifier rank at most k one can determine whether $G \models \varphi$ from $\mathrm{tp}^k(G)$ in time $\mathcal{O}_k(1)$.*

3 Local types

In this section we define local types of quantifier rank k for partitioned graphs, or *local k-types* for short. They provide a framework for the results proved in the rest of the paper. The key lemmas are Lemma 14 and Lemma 16 and their corollaries Lemma 15 and Lemma 17.

3.1 Local types for partitioned graphs

Let G be a graph and $\mathcal{P}$ be a partition of its vertex set, and let $\mathbf{x}$ be a set of variables. For an $\mathbf{x}$-tuple $\mathbf{a} \in V(G)$ write $\mathbf{a}\langle\mathcal{P}\rangle$ for the $\mathbf{x}$-tuple $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}$ such that $\mathbf{u}(x)$ is the part containing $\mathbf{a}(x)$, for all $x \in \mathbf{x}$.

▶ **Definition 8.** *Let G be a graph, $\mathcal{P}$ be a partition of its vertex set, $\mathbf{x}$ a nonempty set of variables, and $k \in \mathbb{N}$. For any $\mathbf{a} \in V(G)^{\mathbf{x}}$ we define the* local k-type of $\mathbf{a}$*, denoted* $\mathrm{ltp}^k_{\mathcal{P}}(\mathbf{a})$*, as follows:*

- $\mathrm{ltp}^0_{\mathcal{P}}(\mathbf{a})$ *is the atomic type of $\mathbf{a}$ together with the $\mathbf{x}$-tuple $\mathbf{a}\langle\mathcal{P}\rangle \in \mathcal{P}^{\mathbf{x}}$ of parts of $\mathcal{P}$ corresponding to $\mathbf{a}$,*
- *for $k > 0$, let $\mathrm{ltp}^k_{\mathcal{P}}(\mathbf{a}) = \{\mathrm{ltp}^{k-1}_{\mathcal{P}}(\mathbf{a}b) \mid b \in w$ for some $w \in \mathcal{P}$ with $\mathrm{dist}_{\mathcal{P}}(\mathbf{a}\langle\mathcal{P}\rangle, w) \leqslant 2^{k-1}\}$.*

As with usual types of quantifier-rank k defined in the previous section, it is often convenient to think about equality of local types in terms of games. We now briefly describe the corresponding variant of Ehrenfeucht-Fraïssé games game. This game will be played on a single graph G with a fixed partition $\mathcal{P}$ of its vertex set (one can also imagine it being played on two copies of the same graph with the same partition). The starting position of the game is determined by two $\mathbf{x}$-tuples $\mathbf{a}$ and $\mathbf{b}$ of vertices of G (where $\mathbf{x}$ is nonempty) such that for every $y \in \mathbf{x}$ we have that $\mathbf{a}(y)$ is in the same part of $\mathcal{P}$ as $\mathbf{b}(y)$. The game is played for k rounds as the usual Ehrenfeucht-Fraïssé game with the following extra restrictions on the moves of the players: (1) In the ith round, Spoiler picks one of the tuples $\mathbf{a}$ and $\mathbf{b}$, and he will then proceed to extending it. Suppose that he picks $\mathbf{a}$, the other case being symmetric. Spoiler then picks a vertex a in any part $P \in \mathcal{P}$ such that $\mathrm{dist}_{\mathcal{P}}(P, Q) \leqslant 2^{k-1}$, where Q is some part containing a vertex of $\mathbf{a}$. He then appends a to $\mathbf{a}$ to form the tuple $\mathbf{a}a$. (2) Duplicator replies by picking a vertex b in the same part P and extending the other tuple $\mathbf{b}$ to $\mathbf{b}b$. The game then continues to the next round, with $\mathbf{a}a$ and $\mathbf{b}b$ forming the new position. Duplicator wins after k rounds if the two tuples have equal atomic types.

It is not difficult to see that Duplicator wins the k-round game described above, starting from the configuration $\mathbf{a}$ and $\mathbf{b}$, if and only if $\mathrm{ltp}^k_{\mathcal{P}}(\mathbf{a}) = \mathrm{ltp}^k_{\mathcal{P}}(\mathbf{b})$. This is made formal in the following proposition, whose proof is an immediate consequence of Definition 8.

▶ **Proposition 9.** *Let G be a graph and $\mathcal{P}$ be a partition of the vertices of G, and let $\mathbf{x}$ a tuple of variables and $k \in \mathbb{N}$. Then the following holds for any $\mathbf{a}, \mathbf{b} \in V(G)^{\mathbf{x}}$:*

- $\mathrm{ltp}^0_{\mathcal{P}}(\mathbf{a}) = \mathrm{ltp}^0_{\mathcal{P}}(\mathbf{b})$ *if and only if the atomic types of $\mathbf{a}$ and $\mathbf{b}$ are the same, and $\mathbf{a}\langle\mathcal{P}\rangle = \mathbf{b}\langle\mathcal{P}\rangle$;*
- *If $k > 0$ then $\mathrm{ltp}^k_{\mathcal{P}}(\mathbf{a}) = \mathrm{ltp}^k_{\mathcal{P}}(\mathbf{b})$ if and only if for any $P \in \mathcal{P}$ with $\mathrm{dist}_{\mathcal{P}}(\mathbf{a}\langle\mathcal{P}\rangle, P) \leqslant 2^{k-1}$ the following holds: for any $c \in P$ there exists $c' \in P$ such that $\mathrm{ltp}^{k-1}_{\mathcal{P}}(\mathbf{a}c) = \mathrm{ltp}^{k-1}_{\mathcal{P}}(\mathbf{b}c')$, and conversely, for any $c' \in P$ there exists $c \in P$ such that $\mathrm{ltp}^{k-1}_{\mathcal{P}}(\mathbf{a}c) = \mathrm{ltp}^{k-1}_{\mathcal{P}}(\mathbf{b}c')$.*

We will also need to have an abstract set containing all types which could potentially occur for any $k \in \mathbb{N}$ and $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}$. Note that this includes also types which are not realized in G (or even in any graph).

▶ **Definition 10.** *Let $\mathbf{x}$ be a nonempty set of variables and $k \in \mathbb{N}$. Fix a graph G together with a vertex-partition $\mathcal{P}$. For $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}$ we define $\mathsf{Types}^0_{\mathbf{u},\mathcal{P}} := \{(\alpha, \mathbf{u}) \mid \alpha$ is an atomic type with variables $\mathbf{x}\}$. For $k > 0$ let M be the set of all parts w of $\mathcal{P}$ with $\mathrm{dist}_{\mathcal{P}}(\mathbf{u}, w) \leqslant 2^{k-1}$ and let $M' := \bigcup_{w \in M} \mathsf{Types}^{k-1}_{\mathbf{u}w,\mathcal{P}}$. We then define*

$$\mathsf{Types}^k_{\mathbf{u},\mathcal{P}} := \{S \mid S \subseteq M'\}.$$

Define also $\mathsf{Types}^k_{\mathbf{u},\mathcal{P}}(G) := \{\mathrm{ltp}^k(\mathbf{a}) \mid \mathbf{a} \in V(G)^{\mathbf{x}}, \mathbf{u} = \mathbf{a}\langle\mathcal{P}\rangle\}$.

Then $\mathsf{Types}_{\mathbf{u},\mathcal{P}}(G)$ is the set of all local k-types realized in $\mathbf{u}$, and is a subset of $\mathsf{Types}^k_{\mathbf{u},\mathcal{P}}$.

3.2 Properties of local types

In this section we establish the properties of local k-types used in the rest of the paper.

In the rest of this paper, we assume that we have fixed a graph G and a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of G. We write $\mathrm{ltp}^k_s(\cdot)$ to denote $\mathrm{ltp}^k_{\mathcal{P}_s}(\cdot)$, $\mathsf{Types}_{\mathbf{u},s}(\cdot)$ to denote $\mathsf{Types}_{\mathbf{u},\mathcal{P}}(\cdot)$, and $\mathrm{dist}_s(\cdot,\cdot)$ to denote $\mathrm{dist}_{\mathcal{P}_s}(\cdot,\cdot)$.

The following two lemmas establish some basic properties of local k-types. The proof of Lemma 11 follows immediately from the definition.

▶ **Lemma 11.** *The following holds at any time $s \in [n]$ and $k \geqslant 1$.*

- *If $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{b})$, then $\mathrm{ltp}^{k-1}_s(\mathbf{a}) = \mathrm{ltp}^{k-1}_s(\mathbf{b})$.*
- *If $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{b})$, then $\mathbf{a}\langle s\rangle = \mathbf{b}\langle s\rangle$.*

▶ **Lemma 12 (♠).** *Let $s \in [n]$ be a time and $\mathbf{x}$ a tuple of variables, and $k \geqslant 0$. Then $|\mathsf{Types}^k_{\mathbf{u},s}| \leqslant \mathcal{O}_{d,k,\mathbf{x}}(1)$, for all $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}_s$. Moreover, given vicinity $\mathsf{Vicinity}^{2^k}_s(\mathbf{u})$, one can compute $\mathsf{Types}^k_{\mathbf{u},s}$ in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$.*

The following lemma relates local types for partitioned graphs to usual first-order types, as defined in the preliminaries.

▶ **Lemma 13 (♠).** *Let $\mathbf{x}$ be a tuple of variables and $\mathbf{a} \in V(G)^{\mathbf{x}}$. One can compute $\mathrm{tp}^k(\mathbf{a})$ from $\mathrm{ltp}^k_n(\mathbf{a})$ in time $\mathcal{O}_{k,d,\mathbf{x}}(1)$.*

The next lemma is a version of compositionality of local types and plays a key role in computing local types.

▶ **Lemma 14.** *Fix two disjoint sets of variables* $\mathbf{x}$ *and* $\mathbf{y}$*. Let* $\mathbf{a}, \mathbf{a}' \in V^{\mathbf{x}}$ *and* $\mathbf{b}, \mathbf{b}' \in V^{\mathbf{y}}$ *be such that* $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{a}')$ *and* $\mathrm{ltp}^k_s(\mathbf{b}) = \mathrm{ltp}^k_s(\mathbf{b}')$*. Let* $\mathbf{u} = \mathbf{a}\langle s\rangle$ *and* $\mathbf{v} = \mathbf{b}\langle s\rangle$ *and assume that* $\mathrm{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$*. Then* $\mathrm{ltp}^k_s(\mathbf{ab}) = \mathrm{ltp}^k_s(\mathbf{a'b'})$*.*

Proof. We prove the statement by induction on k. For $k = 0$, to prove that $\mathrm{ltp}^0_s(\mathbf{ab}) = \mathrm{ltp}^0_s(\mathbf{a'b'})$, we have to show that the atomic types of $\mathbf{ab}$ and $\mathbf{a'b'}$ are the same. Fix an atomic formula $\varphi(x, y)$, with $x, y \in \mathbf{x} \cup \mathbf{y}$. We show that $G, \mathbf{ab} \models \varphi(x, y)$ if and only if $G, \mathbf{a'b'} \models \varphi(x, y)$. If x and y both belong to $\mathbf{x}$ then the conclusion follows by assumption that $\mathrm{ltp}^0_s(\mathbf{a}) = \mathrm{ltp}^0_s(\mathbf{a}')$. The same holds if x and y both belong to $\mathbf{y}$.

So, by symmetry, it is enough to consider the case when $x \in \mathbf{x}$ and $y \in \mathbf{y}$. Since by our assumption $\mathrm{dist}_s(\mathbf{u}, \mathbf{v}) > 2^0 = 1$, any part of $\mathbf{u}$ is pure to any part $\mathbf{v}$, and so in particular the part $\mathbf{a}(x)\langle s\rangle$ is pure towards $\mathbf{b}(y)\langle s\rangle$. Because $\mathbf{a}(x), \mathbf{a}'(x) \in \mathbf{a}(x)\langle s\rangle$ and $\mathbf{b}(y), \mathbf{b}'(y) \in \mathbf{b}(y)\langle s\rangle$, this implies that $G, \mathbf{ab} \models \varphi(x, y)$ if and only if $G, \mathbf{a'b'} \models \varphi(x, y)$, as required.

For $k > 0$, let c be a vertex in a part $w = c\langle s\rangle$ such that $\mathrm{dist}_s(\mathbf{uv}, w) \leqslant 2^{k-1}$. Our task is to show that there exists $c' \in w$ such that $\mathrm{ltp}^{k-1}_s(\mathbf{ab}c) = \mathrm{ltp}^{k-1}_s(\mathbf{a'b'}c')$. Since $\mathrm{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$, exactly one of $\mathrm{dist}_s(\mathbf{u}, w) \leqslant 2^{k-1}$ and $\mathrm{dist}_s(\mathbf{v}, w) \leqslant 2^{k-1}$ has to hold; without loss of generality assume that $\mathrm{dist}_s(\mathbf{u}, w) \leqslant 2^{k-1}$ holds. Since $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{a}')$, there exists $c' \in w$ such that $\mathrm{ltp}^{k-1}_s(\mathbf{a}c) = \mathrm{ltp}^{k-1}_s(\mathbf{a}'c')$. Because $\mathrm{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$ and $\mathrm{dist}_s(\mathbf{u}, w) \leqslant 2^{k-1}$, we have $\mathrm{dist}_s(\mathbf{u}w, \mathbf{v}) > 2^{k-1}$, so we can apply the induction hypothesis to $\mathbf{a}c$, $\mathbf{a}'c'$ and $\mathbf{b}$, $\mathbf{b}'$, which yields that $\mathrm{ltp}^{k-1}_s(\mathbf{ab}c) = \mathrm{ltp}^{k-1}_s(\mathbf{a'b'}c')$, as desired. ◀

The following lemma follows directly from Lemma 14, except for the part about efficient computation.

▶ **Lemma 15.** *Let* $s \in [n]$ *be a time and* $\mathbf{x}$ *and* $\mathbf{y}$ *are disjoint sets of variables. Suppose* $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}_s$ *and* $\mathbf{v} \in \mathcal{P}^{\mathbf{y}}_s$ *are tuples of parts such that* $\mathrm{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$*. Then there is a function* $f \colon \mathsf{Types}^k_{\mathbf{u},s} \times \mathsf{Types}^k_{\mathbf{v},s} \to \mathsf{Types}^k_{\mathbf{uv},s}$ *such that for every pair of tuples* $\mathbf{a} \in V^{\mathbf{x}}$ *and* $\mathbf{b} \in V^{\mathbf{y}}$ *satisfying* $\mathbf{u} = \mathbf{a}\langle s\rangle$ *and* $\mathbf{v} = \mathbf{b}\langle s\rangle$*, we have*

$$\mathrm{ltp}^k_s(\mathbf{ab}) = f(\mathrm{ltp}^k_s(\mathbf{a}), \mathrm{ltp}^k_s(\mathbf{b})).$$

Moreover, given k*,* $\mathbf{u}$*,* $\mathbf{v}$*, and the vicinity* $\mathsf{Vicinity}^{2^k}_s(\mathbf{uv})$*, one can compute* f *in time* $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$*.*

Regarding the computation of function f in the above lemma, by "computing f in time $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$" we do not mean just evaluating f on any given input in desired time, but constructing the whole input-output table for f. The reason why this can be computed from k, $\mathbf{u}$, $\mathbf{v}$ and $\mathsf{Vicinity}^{2^k}_s(\mathbf{uv})$ in time $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$ is that the input and output sets have size bounded by $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$ and the proof in Lemma 14 uses only information from $\mathsf{Vicinity}^{2^k}_s(\mathbf{uv})$. A concrete approach to implementing this computation similar to that of [7] can be found in the full version.

The next lemma will allow us to determine how the k-type of a tuple $\mathbf{a}$ develops over time.

▶ **Lemma 16.** *Let* $s \in [n]$ *be a time and let* $\mathbf{a} \in V^{\mathbf{x}}$*,* $\mathbf{a}' \in V^{\mathbf{x}}$ *be two tuples of vertices such that* $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{a}')$*. Then* $\mathrm{ltp}^k_{s+1}(\mathbf{a}) = \mathrm{ltp}^k_{s+1}(\mathbf{a}')$*.*

Proof. By induction on k. For $k = 0$ note that $\mathrm{ltp}^0_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{a}')$ implies that atomic types of $\mathbf{a}$ and $\mathbf{a}'$ are the same and $\mathbf{a}\langle s\rangle = \mathbf{a}'\langle s\rangle$. It is easily seen that then also $\mathbf{a}\langle s+1\rangle = \mathbf{a}'\langle s+1\rangle$, as desired.

For $k > 0$, let $\mathbf{u} = \mathbf{a}\langle s+1\rangle = \mathbf{a}'\langle s+1\rangle$. We need to show that for any $w \in \mathcal{P}_{s+1}$ with $\mathrm{dist}_{s+1}(\mathbf{u}, w) \leqslant 2^{k-1}$ and any $b \in w$ there is $b' \in w$ such that $\mathrm{ltp}^{k-1}_{s+1}(\mathbf{a}b) = \mathrm{ltp}^{k-1}_{s+1}(\mathbf{a}'b')$, and symmetrically, that for any $b' \in w$ there is $b \in w$ such that $\mathrm{ltp}^{k-1}_{s+1}(\mathbf{a}b) = \mathrm{ltp}^{k-1}_{s+1}(\mathbf{a}'b')$. We focus on the first option; the proof of the second one is analogous. Let $v = b\langle s\rangle$. We distinguish two possibilities:

- $\mathrm{dist}_s(\mathbf{u}, v) \leqslant 2^{k-1}$: In this case, since $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{a}')$, there exists $b' \in v$ such that $\mathrm{ltp}^{k-1}_s(\mathbf{a}b) = \mathrm{ltp}^{k-1}_s(\mathbf{a}'b')$. Then by induction hypothesis it follows that $\mathrm{ltp}^{k-1}_{s+1}(\mathbf{a}b) = \mathrm{ltp}^{k-1}_{s+1}(\mathbf{a}'b')$, as desired.
- $\mathrm{dist}_s(\mathbf{u}, v) > 2^{k-1}$: In this case we note that $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_s(\mathbf{a}')$ implies that $\mathrm{ltp}^{k-1}_s(\mathbf{a}) = \mathrm{ltp}^{k-1}_s(\mathbf{a}')$, and we set $b' := b$. We can now apply Lemma 14 to $\mathbf{a}$, $\mathbf{a}'$ and b, b' to see that $\mathrm{ltp}^{k-1}_s(\mathbf{a}b) = \mathrm{ltp}^{k-1}_s(\mathbf{a}'b')$, and by induction hypothesis it follows that $\mathrm{ltp}^{k-1}_s(\mathbf{a}b) = \mathrm{ltp}^{k-1}_s(\mathbf{a}'b')$, as desired. ◀

Lemma 16 implies that there exists a function which maps $\mathrm{ltp}^k_s(\mathbf{a})$ to $\mathrm{ltp}^k_{s+1}(\mathbf{a})$, and by induction we get the following lemma.

▶ **Lemma 17.** *Let $s, t \in [n]$ be times with $s \leqslant t$. Suppose $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}_s$ and let $\mathbf{v} = \mathbf{u}\langle s \to t\rangle$. Then there exists a function $f \colon \mathsf{Types}^k_{\mathbf{u},s} \to \mathsf{Types}^k_{\mathbf{v},t}$ such that for every tuple $\mathbf{a} \in V^{\mathbf{x}}$ satisfying $\mathbf{u} = \mathbf{a}\langle s\rangle$, we have*

$$\mathrm{ltp}^k_t(\mathbf{a}) = f(\mathrm{ltp}^k_s(\mathbf{a})).$$

Moreover, if $t = s+1$, then given k, $\mathbf{u}$, $\mathbf{v}$ and the relevant region $\mathsf{Relevant}^{2^k(|\mathbf{x}|+1)}_s$, one can compute f in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$, provided that for every $y \in \mathbf{x}$ we have that $\mathbf{v}(y) \in \mathsf{Relevant}^{2^k|\mathbf{x}|}_s$.

As in the case of Lemma 15, the whole input-output table of function f can be computed in time $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$ from k, $\mathbf{u}$, $\mathbf{v}$ and $\mathsf{Relevant}^{2^k}_s$, since the proof of Lemma 16 uses only information from $\mathsf{Relevant}^{2^k(|\mathbf{x}|+1)}_s$. Again, a concrete approach to implementing this computation similar to that of [7] can be found in the full version.

We will also use the fact that when going from time s to $s+1$ the local k-types of tuples in parts which are not in the trigraph $\mathsf{Relevant}^{2^k}_s$ are not affected.

▶ **Lemma 18 (♠).** *Let $\mathbf{x}$ be a finite set of variables, $s \in [n]$ a time, $k \in N$ and let $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}_s$ be such for every $y \in \mathbf{x}$ it holds that $\mathbf{u}(y) \notin V(\mathsf{Relevant}^{2^k}_s)$. Then $\mathbf{u}\langle s \to s+1\rangle = \mathbf{u}$, and for every $\mathbf{a}$ with $\mathbf{u} = \mathbf{a}\langle s\rangle$ we have that $\mathrm{ltp}^k_s(\mathbf{a}) = \mathrm{ltp}^k_{s+1}(\mathbf{a})$. In particular, $\mathsf{Types}^k_{\mathbf{u},s+1}(G) = \mathsf{Types}^k_{\mathbf{u},s}(G)$.*

Model checking on graphs of bounded twin-width. With the machinery from the previous subsection we can now reprove the result of [7] that the first-order model checking problem on any class $\mathcal{C}$ of graphs of twin-width at most d is solvable in time $\mathcal{O}_{d,\varphi}(n)$, provided that the contraction sequence of the input graph G is provided together with G.

▶ **Theorem 19.** *Let G be a graph on n vertices represented through its contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of width d. Then for any sentence φ of quantifier rank q one can decide whether $G \models \varphi$ in time $\mathcal{O}_{d,\varphi}(n)$.*

Proof. Let q be the quantifier rank of φ and set $k := q-1$ and $r := 2^k$. We will show how to compute the set $\mathsf{Types}^k_x(G)$ in desired time, and since $\mathrm{tp}^q(G) = \mathsf{Types}^{q-1}_x(G)$, the result will follow by Proposition 7.

As a preprocessing step, the algorithm computes in time $\mathcal{O}_{d,k}(n)$ the trigraphs $\mathsf{Relevant}^r_s$ for all $s \in [n-1]$; this can be done due to Lemma 4. For the rest of the proof, let us for any time $s \in [n]$ denote by T_s the set of all sets of realized types at time s, i.e. $T_s := \{\mathsf{Types}^k_{w,s}(G) \mid w \in \mathcal{P}_s\}$.

The algorithm first computes T_1 by computing $\mathsf{Types}^k_{w,1}(G)$ for each $w \in \mathcal{P}_1$; since each such part w contains exactly one vertex, this can be done in time $\mathcal{O}_k(1)$ for any w, and so this takes time $\mathcal{O}_k(n)$ in total. From this point on the algorithm will proceed through times 2 to n and for every time s it will compute T_s from T_{s-1}. By Lemma 18, any part w of $\mathcal{P}_{s-1}$ which is not in $\mathsf{Relevant}^{2^k}_{s-1}$ is the same in $\mathcal{P}_s$ as in $\mathcal{P}_{s-1}$ and we have that $\mathsf{Types}^k_{w,s-1}(G) = \mathsf{Types}^k_{w,s}(G)$, which means that the computation only needs to be performed on parts from $\mathsf{Relevant}^{2^k}_{s-1}$. We distinguish the following two possibilities:

- If v, w are the two parts of $\mathcal{P}_{s-1}$ which get contracted into a part $u \in \mathcal{P}_s$, then the algorithm applies the function from Lemma 17 to all members of $\mathsf{Types}^k_{v,s-1}(G)$ and $\mathsf{Types}^k_{w,s-1}(G)$ and collects the results into $\mathsf{Types}^k_{u,s}(G)$.
- If w is any other part in $\mathsf{Relevant}^r_{s-1}$, then the algorithm applies the function from Lemma 17 to all members of $\mathsf{Types}^k_{w,s-1}$ and collects the results into $\mathsf{Types}^k_{w,s}(G)$.

In each of the above cases the computation can be done in time $\mathcal{O}_{d,k}(1)$, since each application of the function from Lemma 17 can be done in time $\mathcal{O}_{d,k,1}(1)$ and by Lemma 12 we have that $|\mathsf{Types}^k_{w,s-1}(G)| \leqslant \mathcal{O}_{d,k}(1)$. Moreover, since $|\mathsf{Relevant}^r_{s-1}| \leqslant \mathcal{O}_{d,k}(1)$, the computation of T_s from T_{s-1} can be done in time $\mathcal{O}_{d,k}(1)$. There are $n-1$ steps to obtain T_n and so the whole computation takes time $\mathcal{O}_{d,k}(n)$. Now T_n contains only $\mathsf{Types}^k_{w,n}(G)$ where w is the only part of $\mathcal{P}_n$. By Lemma 13, from each local k-type in $\mathsf{Types}^k_{w,n}(G)$ one can compute the corresponding k-type from $\mathsf{Types}^k_x(G)$ in time $\mathcal{O}_k(1)$. This finishes the proof. ◀

4 Query answering

In this section we prove Theorem 1. For the remainder of this section let us fix a graph G and a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of G of width d, where $n = |V(G)|$. In all algorithmic statements that follow, we assume that G and $\mathcal{P}$ are given on input.

Throughout this section our data structures work with the standard word RAM model.

4.1 Proximity oracle

For vertices $u, v \in V(G)$ and $r \in \mathbb{N}$, we define

$$\mathsf{firstClose}_r(u,v) = \min\{t \mid \mathrm{dist}_t(u\langle t\rangle, v\langle t\rangle) \leqslant r\}.$$

In other words, $\mathsf{firstClose}_r(u,v)$ is the first time t such that the parts of $\mathcal{P}_t$ containing u and v are at distance at most r in the impurity graph G^{imp}_t. Note that whenever $u \neq v$, we have $1 < \mathsf{firstClose}_r(u,v) \leqslant n$. The main goal of this section is to construct an auxiliary data structure for answering queries about the values of $\mathsf{firstClose}_r(\cdot,\cdot)$. This is described in the lemma below.

▶ **Lemma 20.** *For a given $r \in \mathbb{N}$, one can in time $\mathcal{O}_{d,r}(n)$ compute a data structure that can answer the following queries in time $\mathcal{O}_{d,r}(\log\log n)$: given $u, v \in V(G)$, output $\mathsf{firstClose}_r(u,v)$.*

By Lemma 5, we may assume that the vertex set $V(G)$ is equal to $[n]$, and $\mathcal{P}$ is a convex contraction sequence for the usual order on $[n]$. In particular, pairs of vertices can be identified with points in a plane, and intuitively, every pair of sets $A, B \subseteq V(G)$ corresponds

to a rectangle $A \times B \subseteq [n] \times [n]$. This correspondence will be important in the proof of Lemma 20, whose key technical component is the data structure for *orthogonal range queries* due to Chan [9], for manipulating rectangles in a plane. (We remark that the applicability of this data structure in the context of twin-width has already been observed in [19].) Let us recall the setting.

A *rectangle* is a set of pairs of integers of the form $\{(x, y) \colon a \leqslant x \leqslant a', b \leqslant y \leqslant b'\}$ for some integers a, a', b, b'. In all algorithmic statements that follow, every rectangle is represented by such a quadruple (a, a', b, b'). In the problem of orthogonal range queries, we are given a list of pairwise disjoint rectangles $\mathcal{R} = \{R_1, \ldots, R_m\}$, all contained in $[n] \times [n]$, and the task is to set up a data structure that can efficiently answer the following queries: given $(x, y) \in [n] \times [n]$, output the index of the rectangle in $\mathcal{R}$ that contains (x, y), or output $\perp$ if there is no such rectangle. Chan proposed the following data structure for this problem.

▶ **Theorem 21** ([9]). *Assuming $|\mathcal{R}| = \mathcal{O}(n)$, there is a data structure for the orthogonal range queries that takes $\mathcal{O}(n)$ space, can be initialized in time $\mathcal{O}(n)$, and can answer every query in time $\mathcal{O}(\log \log n)$.*

We remark that there is also a simple data structure for orthogonal range queries that for any fixed $\varepsilon > 0$, achieves query time $\mathcal{O}(1/\varepsilon)$ at the expense of space complexity and initialization time $\mathcal{O}(n^{1+\varepsilon})$. See the appendix of [19] for details. As we mentioned in Section 1, replacing the usage of the data structure of Chan with this simple data structure results in an analogous tradeoff in Theorem 1.

We reduce the statement of Lemma 20 to the result of Chan using the following lemma.

▶ **Lemma 22** (♠). *One can in time $\mathcal{O}_{d,r}(n)$ compute a list $\mathcal{Q}$ of pairs of the form (R, t), where $R \subseteq [n] \times [n]$ is a rectangle and $t \in [n]$, such that the following holds:*

- *the rectangles in pairs from $\mathcal{Q}$ form a partition of $[n] \times [n]$, and*
- *for each $(u, v) \in [n] \times [n]$, if $(R, t) \in \mathcal{Q}$ is the unique pair satisfying $(u, v) \in R$, then we have $\mathsf{firstClose}_r(u, v) = t$.*

Lemma 20 follows from Lemma 22 as follows. Let $\mathcal{Q}$ be the list provided by Lemma 22; note that $|\mathcal{Q}| \leqslant \mathcal{O}_{d,r}(n)$, because this is an upper bound on the running time of the algorithm computing $\mathcal{Q}$. Let $\mathcal{R}$ be the list of rectangles appearing in the pairs from $\mathcal{Q}$. Set up a data structure of Theorem 21 for $\mathcal{R}$ and, additionally, for each $R \in \mathcal{R}$ remember the unique $t \in [n]$ such that $(R, t) \in \mathcal{Q}$. Then upon query $(u, v) \in [n] \times [n]$, it suffices to use the data structure of Theorem 21 to find the unique $R \in \mathcal{R}$ containing (u, v) and return the associated integer t.

4.2 The tree of r-close $\mathbf{x}$-tuples

In this section we are going to construct an auxiliary data structure for handling local types. Fix a number $k \in \mathbb{N}$; this is the quantifier rank of the types we would like to tackle. Denote $r := 2^k$. Also fix a finite set $\mathbf{x}$ of variables, an n-vertex graph G, together with a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$.

For $s \in [n]$ and a tuple $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$, we call $\mathbf{u}$ *r-close* at the time s if one cannot partition $\mathbf{u}$ into two nonempty tuples $\mathbf{u}', \mathbf{u}''$ such that $\mathrm{dist}_s(\mathbf{u}', \mathbf{u}'') > r$. Equivalently, if one considers an auxiliary graph on vertex set $\mathbf{u}$ where two parts are connected iff they are at distance at most r in G_s^{imp}, then $\mathbf{u}$ is r-close iff this auxiliary graph is connected. Note that if $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$ is r-close at the time s, then for every t with $s \leqslant t \leqslant n$, the tuple $\mathbf{u}\langle s \to t\rangle$ is also r-close at the time t.

For $s \in [n]$ with $s > 1$, by B_s denote the part of $\mathcal{P}_s$ that is the union of two parts in $\mathcal{P}_{s-1}$. Let $T_{r,\mathbf{x}}$ be the set consisting of all pairs of the form $(\mathbf{u}, s)$ such that $s \in [n]$, $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$ is r-close at the time s, and at least one of the following conditions is satisfied:

- $s = 1$; or
- $s > 1$ and $\mathrm{dist}_s(B_s, \mathbf{u}) \leqslant r$; or
- $s < n$ and $\mathrm{dist}_{s+1}(B_{s+1}, \mathbf{u}\langle s \to s+1\rangle) \leqslant r$.

Note that as $\mathbf{u}$ is assumed to be r-close, if the second condition holds then $\mathbf{u} \subseteq \mathsf{Vicinity}_s^{r|\mathbf{x}|}(B_s)$, and if the third condition holds then $\mathbf{u}\langle s \to s+1\rangle \subseteq \mathsf{Vicinity}_{s+1}^{r|\mathbf{x}|}(B_{s+1})$. Since the trigraphs $\mathsf{Vicinity}_s^{r|\mathbf{x}|}(B_s)$ and $\mathsf{Vicinity}_{s+1}^{r|\mathbf{x}|}(B_{s+1})$ are of size $\mathcal{O}_{d,k,\mathbf{x}}(1)$, it follows that $T_{r,\mathbf{x}}$ contains $\mathcal{O}_{d,k,\mathbf{x}}(n)$ elements in total: n elements for $s = 1$ and $\mathcal{O}_{d,k,\mathbf{x}}(1)$ elements for each $1 < s \leqslant n$.

We consider an ancestor relation $\preccurlyeq$ on $T_{r,\mathbf{x}}$ defined as follows:

$$(\mathbf{v}, t) \preccurlyeq (\mathbf{u}, s) \qquad \text{if and only if} \qquad s \leqslant t \text{ and } \mathbf{u}\langle s \to t\rangle = \mathbf{v}.$$

It is easy to see $T_{r,\mathbf{x}}$ together with $\preccurlyeq$ defines a rooted tree whose tree order is $\preccurlyeq$. The root is $(\mathbf{r}, n)$, where $\mathbf{r}$ is the unique tuple of $\mathcal{P}_n^{\mathbf{x}}$, the one that maps all variables of $\mathbf{x}$ to the unique part of $\mathcal{P}_n$. From now on we identify the set $T_{r,\mathbf{x}}$ with the tree it induces. Therefore, we call the elements of $T_{r,\mathbf{x}}$ *nodes* and the child-parent pairs in $T_{r,\mathbf{x}}$ the *edges* of $T_{r,\mathbf{x}}$.

▶ **Definition 23.** *We call $T_{r,\mathbf{x}}$ the* tree of r-close $\mathbf{x}$-tuples *associated with G and the contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$.*

Recall that $r = 2^k$, and $k \in \mathbb{N}$ is fixed. For every node $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$, let $\mathsf{Types}_{\mathbf{u},s}^k$ be the set of all possible k-local types of tuples $\mathbf{w} \in V(G)^{\mathbf{x}}$ satisfying $\mathbf{u} = \mathbf{w}\langle s\rangle$. By Lemma 12, there is a constant $M = \mathcal{O}_{d,k,\mathbf{x}}(1)$ such that $|\mathsf{Types}_{\mathbf{u},s}^k| \leqslant M$ for every node $(\mathbf{u}, s)$, and $\mathsf{Types}_{\mathbf{u},s}^k$ can be computed in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$ given access to G_s and $\mathbf{u}$.

Consider nodes $(\mathbf{u}, s), (\mathbf{v}, t) \in T_{r,\mathbf{x}}$ such that $(\mathbf{v}, t)$ is the parent of $(\mathbf{u}, s)$. Let $e = ((\mathbf{u}, s), (\mathbf{v}, t))$ be the corresponding edge of $T_{r,\mathbf{x}}$. By Lemma 17, there exists a function $f_e \colon \mathsf{Types}_{\mathbf{u},s}^k \to \mathsf{Types}_{\mathbf{v},t}^k$ such that for every tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$ with $\mathbf{u} = \mathbf{w}\langle s\rangle$, we have

$$\mathrm{ltp}_t^k(\mathbf{w}) = f_e(\mathrm{ltp}_s^k(\mathbf{w})). \tag{1}$$

We now verify that all the objects introduced above can be computed efficiently.

▶ **Lemma 24 (♠).** *One can in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ compute the nodes and the edges of $T_{r,\mathbf{x}}$ (where $r = 2^k$) as well as, for every edge e of $T_{r,\mathbf{x}}$, the function f_e.*

We can finally state and prove the main result of this section.

▶ **Lemma 25.** *One can in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ construct a data structure that can answer the following queries in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$: given $\mathbf{w} \in V(G)^{\mathbf{x}}$, two nodes $(\mathbf{v}, t) \preccurlyeq (\mathbf{u}, s)$ of $T_{r,\mathbf{x}}$ such that $\mathbf{u} = \mathbf{w}\langle s\rangle$ and $\mathbf{v} = \mathbf{w}\langle t\rangle$, and the type $\mathrm{ltp}_s^k(\mathbf{w})$, output the type $\mathrm{ltp}_t^k(\mathbf{w})$.*

Proof. Using Lemma 24 construct the tree $T_{r,\mathbf{x}}$ and functions f_e for the edges of $T_{r,\mathbf{x}}$. By Lemma 12, there is a constant $M = \mathcal{O}_{d,k,\mathbf{x}}(1)$ such that $|\mathsf{Types}_{\mathbf{u},s}^k| \leqslant M$ for every node $(\mathbf{u}, s)$. Let $I \coloneqq [M]$ be an indexing set of size M. Since for every node $(\mathbf{u}, s)$ we have $|\mathsf{Types}_{\mathbf{u},s}^k| \leqslant M$, we can set an arbitrary injection $\iota_{\mathbf{u},s} \colon \mathsf{Types}_{\mathbf{u},s}^k \to I$. For an edge $e = ((\mathbf{u}, s), (\mathbf{v}, t))$, we set

$$g_e \coloneqq \iota_{\mathbf{u},s}^{-1} \,;\, f_e \,;\, \iota_{\mathbf{v},t}.$$

Thus, g_e is a function from I to I that, intuitively, is just f_e reindexed using the index set I. Clearly, functions $\iota_{\mathbf{u},s}$ and g_e defined above can be computed in total time $\mathcal{O}_{d,k,\mathbf{x}}(n)$.

We will use the following result proved in [17].

▶ **Theorem 26** (Theorem 5.1 of [17]). *Let S be a semigroup and T be a rooted tree with edges labelled with elements of S. Then one can in time $|S|^{\mathcal{O}(1)} \cdot |T|$ construct a data structure that can answer the following queries in time $|S|^{\mathcal{O}(1)}$: given nodes $u, v \in T$ such that v is an ancestor of u, output the (top-down) product of elements of S associated with the edges on the path from v to u. The data structure uses $|S|^{\mathcal{O}(1)} \cdot |T|$ space.*

Let S be the semigroup of functions from I to I with the product defined as $f \cdot g = g; f$. Thus, the functions g_e form a labelling of edges of $T_{r,\mathbf{x}}$ with elements of S. Apply Theorem 26 to this S-labelled tree, and let $\mathbb{S}$ be the obtained data structure. Now, to answer a query about nodes $(\mathbf{u}, s)$, $(\mathbf{v}, t)$, and type $\alpha = \mathrm{ltp}^k_s(\mathbf{w})$ as in the lemma statement, it suffices to apply the following procedure:

- Compute $\tilde{\alpha} \coloneqq \iota_{\mathbf{u},s}(\alpha)$.
- Query $\mathbb{S}$ to compute the compositions of functions g_e along the path from $(\mathbf{u}, s)$ to $(\mathbf{v}, t)$ in $T_{r,\mathbf{x}}$. Call the resulting function h.
- Compute $\tilde{\beta} \coloneqq h(\tilde{\alpha})$.
- Output $\beta \coloneqq \iota^{-1}_{\mathbf{v},t}(\tilde{\beta})$.

The correctness of the procedure follows from a repeated use of (1), and it is clear that the running time is $\mathcal{O}_{d,k,\mathbf{x}}(1)$. ◀

4.3 Data structure

With all the tools prepared, we can prove Theorem 1.

Let k be the quantifier rank of φ. We set up two auxiliary data structures:

- The data structure of Lemma 20 for radius parameter $r = 2^k$. Call this data structure $\mathbb{P}$.
- For every $\mathbf{z} \subseteq \mathbf{x}$, the data structure of Lemma 25 for parameter k and the set of variables $\mathbf{z}$. Call this data structure $\mathbb{W}_{\mathbf{z}}$.

Moreover, using Lemma 4, we compute for each time $s \in [n-1]$ the trigraph $\mathsf{Relevant}^p_s$, where $p \coloneqq r(|\mathbf{x}| + 1)$. These objects constitute our data structure, so by Lemmas 20, 25, and 4, the construction time is $\mathcal{O}_{d,\varphi}(n)$ as promised. It remains to show how to implement queries.

Suppose we are given a tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$ and we would like to decide whether $G \models \varphi(\mathbf{w})$. By Lemma 13, to answer this it suffices to compute $\mathrm{ltp}^k_n(\mathbf{w})$. In the following, for $\mathbf{z} \subseteq \mathbf{x}$, by $\mathbf{w}_{\mathbf{z}}$ we denote the restriction of $\mathbf{w}$ to the variables of $\mathbf{z}$.

For each time $s \in [n]$, let H_s be the graph on vertex set $\mathbf{x}$ such that $y, y' \in \mathbf{x}$ are adjacent in H_s if and only if $\mathrm{dist}_s(\mathbf{w}\langle s\rangle(y), \mathbf{w}\langle s\rangle(y')) \leqslant r$. The following are immediate:

- For all $1 \leqslant s \leqslant t \leqslant n$, H_t is a supergraph of H_s. That is, if $y, y' \in \mathbf{x}$ are adjacent in H_s, then they are also adjacent in H_t.
- If $\mathbf{z} \subseteq \mathbf{x}$ is such that $H_s[\mathbf{z}]$ is connected for some $s \in [n]$, then $\mathbf{w}_{\mathbf{z}}\langle s\rangle$ is r-close at the time s.

Using the data structure $\mathbb{P}$, we may compute $\mathsf{firstClose}_r(y, y')$ for all $\{y, y'\} \in \binom{\mathbf{x}}{2}$ in total time $\mathcal{O}_{d,\varphi}(\log \log n)$. Let $S \subseteq [n]$ be the set of all those numbers, and include 1 and n in S in addition. Thus $|S| \leqslant 2 + \binom{|\mathbf{x}|}{2} \leqslant \mathcal{O}_\varphi(1)$. We imagine S as ordered by the standard order $\leqslant$, hence we may talk about consecutive elements of S.

Note that the knowledge of the numbers $\mathsf{firstClose}_r(y, y')$ for $\{y, y'\} \in \binom{\mathbf{x}}{2}$ allows us to compute the graphs H_s for all $s \in S$. Further, observe that if $t \in [S]$ is such that $s \leqslant t < s'$ for some $s, s' \in S$ that are consecutive in S, then $H_t = H_s$.

Let L be the set of all pairs of the form $(\mathbf{z}, s)$ where $s \in S$, $\mathbf{z}$ is a connected component of H_s, and either $s = 1$ or $\mathbf{z}$ is not connected in H_{s-1}. Clearly, L has size $\mathcal{O}_{d,\varphi}(1)$ and can be computed in time $\mathcal{O}_{d,\varphi}(1)$. Observe the following.

▷ Claim 27. For each $(\mathbf{z}, s) \in L$, we have $(\mathbf{w}_{\mathbf{z}}\langle s\rangle, s) \in T_{r,\mathbf{z}}$. If moreover $s > 1$, then for every $\mathbf{y} \subseteq \mathbf{z}$ that is a connected component of H_{s-1}, we have $(\mathbf{w}_{\mathbf{y}}\langle s-1\rangle, s-1) \in T_{r,\mathbf{y}}$.

Proof. If $s = 1$, then $\mathbf{z}$ being a connected component of H_1 means that $\mathbf{z}$ is a constant tuple. Hence $\mathbf{w}_{\mathbf{z}}\langle 1\rangle$ is r-close at the time 1, implying that $(\mathbf{w}_{\mathbf{z}}\langle 1\rangle, 1) \in T_{r,\mathbf{z}}$.

Assume then that $s > 1$. As $\mathbf{z}$ is not connected in H_{s-1} and is connected in H_s, it follows that for every connected component $\mathbf{y}$ of H_{s-1} that is contained in $\mathbf{z}$, we have $\mathrm{dist}_s(\mathbf{w}_{\mathbf{y}}\langle s\rangle, B_s) \leqslant r$, and in particular $\mathrm{dist}_s(\mathbf{w}_{\mathbf{z}}\langle s\rangle, B_s) \leqslant r$. The latter statement implies that $(\mathbf{w}_{\mathbf{z}}\langle s\rangle, s) \in T_{r,\mathbf{z}}$ due to fulfilling the second condition in the definition of $T_{r,\mathbf{z}}$. Further, since $\mathbf{y}$ is a connected component of H_{s-1}, $\mathbf{w}_{\mathbf{y}}\langle s-1\rangle$ is r-close at the time $s-1$. So $(\mathbf{w}_{\mathbf{y}}\langle s-1\rangle, s-1) \in T_{r,\mathbf{y}}$ due to fulfilling the third condition in the definition of $T_{r,\mathbf{y}}$. ◁

We now compute the types $\mathrm{ltp}^k_s(\mathbf{w}_{\mathbf{z}})$ for all $(\mathbf{z}, s) \in L$. We do this in any order on L with non-decreasing s, hence when processing $(\mathbf{z}, s)$ we may assume that the corresponding types have already been computed for all $(\mathbf{y}, t) \in L$ with $t < s$.

Assume first that $s = 1$. Then $(\mathbf{z}, 1) \in L$ means that $\mathbf{z}$ is a connected component of H_1, which in turn means that $\mathbf{w}_{\mathbf{z}}$ is a constant tuple. In this case $\mathrm{ltp}^k_1(\mathbf{w}_{\mathbf{z}})$ can be computed trivially.

Assume then that $s > 1$. Since $(\mathbf{z}, s) \in L$, we have that $\mathbf{z}$ is a connected component of H_s, but in H_{s-1}, $\mathbf{z}$ breaks into two or more smaller connected components.

Consider any such component $\mathbf{y}$; that is, $\mathbf{y}$ is a connected component of H_{s-1} that is contained in $\mathbf{z}$. By Claim 27, we have $(\mathbf{w}_{\mathbf{y}}\langle s-1\rangle, s-1) \in T_{r,\mathbf{y}}$. Let then $t \leqslant s-1$ be the smallest time such that $\mathbf{y}$ is a connected component of H_t; clearly we have $t \in S$ and $(\mathbf{y}, t) \in L$. By Claim 27 again, $(\mathbf{w}_{\mathbf{y}}\langle t\rangle, t) \in T_{r,\mathbf{y}}$. Since the type $\mathrm{ltp}^k_t(\mathbf{w}_{\mathbf{y}})$ has been already computed before, we may use one query to $\mathbb{W}_{\mathbf{y}}$ to compute the type $\mathrm{ltp}^k_{s-1}(\mathbf{w}_{\mathbf{y}})$.

Having performed the procedure described above for every connected component $\mathbf{y}$ of H_{s-1} that is contained in $\mathbf{z}$, we may repeatedly use Lemma 15 to compute the type $\mathrm{ltp}^k_{s-1}(\mathbf{w}_{\mathbf{z}})$. Note that for different components $\mathbf{y}, \mathbf{y}'$ as above, we have $\mathrm{dist}_{s-1}(\mathbf{w}_{\mathbf{y}}\langle s-1\rangle, \mathbf{w}_{\mathbf{y}'}\langle s-1\rangle) > r$ due to $\mathbf{y}$ and $\mathbf{y}'$ being non-adjacent in H_{s-1}. Furthermore, all trigraphs required in the applications of Lemma 15 can be easily deduced from the trigraph $\mathsf{Relevant}^p_{s-1}$ and the description of the contraction performed at the time $s-1$; these are stored in our data structure.

Finally, it remains to apply Lemma 17 to compute the type $\mathrm{ltp}^k_s(\mathbf{w}_{\mathbf{z}})$ from $\mathrm{ltp}^k_{s-1}(\mathbf{w}_{\mathbf{z}})$. Again, the trigraphs needed in this application can be easily deduced from $\mathsf{Relevant}^p_{s-1}$ and the contraction performed at the time $s-1$. This finishes the computation of types $\mathrm{ltp}^k_s(\mathbf{w}_{\mathbf{z}})$ for all $(\mathbf{z}, s) \in L$; note that the running time is $\mathcal{O}_{d,\varphi}(1)$.

Finally, let t be the smallest time such that $\mathbf{x}$ is connected in H_t. Such t exists since $\mathbf{x}$ is connected in H_n. Clearly, $t \in S$. By definition we have $(\mathbf{x}, t) \in L$, so the type $\mathrm{ltp}^k_t(\mathbf{w})$ has been computed. By Claim 27, $(\mathbf{w}\langle t\rangle, t) \in T_{r,\mathbf{x}}$. So we can now use the data structure $\mathbb{W}_{\mathbf{x}}$ one last time to compute $\mathrm{ltp}^k_n(\mathbf{w})$. This finishes the proof of Theorem 1.

5 Query enumeration

In this section we prove Theorem 2, which we recall below for convenience.

▶ **Theorem 2.** *Suppose we are given an n-vertex graph G specified through a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of width d, and a first-order formula $\varphi(\mathbf{x})$, where $\mathbf{x}$ is a set of variables. Then after preprocessing in time $\mathcal{O}_{d,\varphi}(n)$, one can enumerate all tuples $\mathbf{w} \in V(G)^{\mathbf{x}}$ such that $G \models \varphi(\mathbf{w})$ with $\mathcal{O}_{d,\varphi}(1)$ delay.*

First, we need to define our notion of enumerators, and prepare some tools for working with them.

Enumerators. Let $x_1, \ldots, x_n$ be a sequence of elements. An *enumerator* of the sequence $x_1, \ldots, x_n$ is a data structure that implements a single method, such that at the ith invocation of the method, it outputs the element x_j of the sequence, where $j = i \mod n$, and reports an "end of sequence" message if $j = 0$. We say that the enumerator has *delay* t if each invocation takes at most t computation steps, including the steps needed to output the element x_j (assuming each element has a fixed representation). An enumerator for a set X is an enumerator for any sequence $x_1, \ldots, x_n$ with $\{x_1, \ldots, x_n\} = X$ and $n = |X|$. Enumerators for Cartesian products and disjoint unions of sets can be obtained in an obvious way:

▶ **Lemma 28.** *Suppose we are given an enumerator for a set X with delay t and an enumerator for a set Y with delay t', where $t, t' \geqslant 1$. Then we can construct in time $\mathcal{O}(1)$ an enumerator with delay $t + t' + \mathcal{O}(1)$ for the set $X \times Y$ and – if X and Y are disjoint – for the set $X \uplus Y$.*

We will also construct enumerators for disjoint unions of families of sets, as follows.

▶ **Lemma 29.** *Suppose $X_1, \ldots, X_n$ are pairwise disjoint, nonempty sets, such that X_i has an enumerator $\mathcal{E}_i$ with delay t. Suppose furthermore we have an enumerator for the sequence $\mathcal{E}_1, \ldots, \mathcal{E}_n$ with delay t'. Then one can construct, in time $\mathcal{O}(1)$ an enumerator for the set $\bigcup_{1 \leqslant i \leqslant n} X_i$ with delay $t + t' + \mathcal{O}(1)$.*

Finally, we will use the following lemma, proved in [17, Lemma 7.15].

▶ **Lemma 30.** *Fix a finite set Q of size q and a set of functions $\mathcal{F} \subseteq Q^Q$. There is a constant c computable from q and an algorithm that, given a rooted tree T, in which each edge vw (v child of w) is labelled by a function $f_{vw} \colon Q \to Q$, computes in time $c \cdot |T|$ a collection $(\mathcal{E}_w)_{w \in V(T)}$ of enumerators, where each $\mathcal{E}_w$ is an enumerator with delay c that enumerates all descendants v of w such that the composition of the functions labeling the edges of the path from v to w, belongs to $\mathcal{F}$.*

This yields the following.

▶ **Corollary 31 (♠).** *Fix a number q. There is a constant c computable from q and an algorithm that, given a rooted tree T, in which each node v is labeled by a set X_v with $|X_v| \leqslant q$ and a set $Y_v \subseteq X_v$, and each edge vw (v child of w) is labelled by a function $f_{wv} \colon X_v \to X_w$, computes in time $c \cdot |T|$ a collection $(\mathcal{E}^\tau_w)_{w \in V(T), \tau \in X_w}$ of enumerators, where each $\mathcal{E}^\tau_w$ is an enumerator with delay c that enumerates all descendants v of w such that there is some $\sigma \in Y_v$ that is mapped to τ by the composition $f \colon X_v \to X_w$ of the functions labeling the edges of the path from v to w.*

Proof of Theorem 2. We now proceed to the proof of Theorem 2.

Fix a number k, a set of variables $\mathbf{x}$, an n-vertex graph G, together with its contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Denote $r := 2^k$.

For every $s \in [n]$ and $\mathbf{x}$-tuple $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$, and local type $\tau \in \mathsf{Types}^k_{\mathbf{u},s}$, denote

$$S^\tau_{\mathbf{u},s} := \{\mathbf{w} \in V(G)^{\mathbf{x}} \mid \mathbf{w}\langle s\rangle = \mathbf{u} \text{ and } \operatorname{ltp}^k_s(\mathbf{w}) = \tau\}.$$

Recall that the root of $T_{r,\mathbf{x}}$ is the pair $(\mathbf{r}, n)$, where $\mathbf{r}$ is the constant $\mathbf{x}$-tuple with all components equal to the unique part of $\mathcal{P}_n$. Then $S^\tau_{\mathbf{r},n}$ is the set of all $\mathbf{x}$-tuples $\mathbf{w} \in V(G)^{\mathbf{x}}$ with $\operatorname{ltp}^k_n(\mathbf{w}) = \tau$. From Lemma 13 and Lemma 7 we get:

▶ **Lemma 32.** *Fix a formula $\varphi(\mathbf{x})$ of quantifier-rank k. Then there is a set $\Gamma \subseteq \mathsf{Types}^n_{\mathbf{r},n}$ such that $\varphi(G) := \{\mathbf{w} \in V(G)^{\mathbf{x}} \mid G \models \varphi(\mathbf{w})\}$ is the disjoint union of the family of sets $\{S^\tau_{\mathbf{r},n} \mid \tau \in \Gamma\}$.*

Therefore, an enumerator for $\varphi(G)$ can be obtained by concatenating enumerators for the sets $S^\tau_{\mathbf{r},n}$, for $\tau \in \Gamma$. Note that here we are concatenating only $\mathcal{O}_{k,d,\mathbf{x}}(1)$ enumerators, by Lemma 12, so, by applying Lemma 28 repeatedly, the resulting enumerator can be obtained in time $\mathcal{O}_{k,d,\mathbf{x}}(1)$ and has delay $\mathcal{O}_{k,d,\mathbf{x}}(1)$. So to prove Theorem 2, it suffices to prove that we can efficiently compute an enumerator for each of the sets $S^\tau_{\mathbf{r},n}$.

Recall that $T_{r,\mathbf{x}}$ is the tree of r-close $\mathbf{x}$-tuples (see Def. 23), and can be computed in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$, by Lemma 24. In the following proposition, we will show how to compute enumerators for all of the sets $S^\tau_{\mathbf{u},s}$, for $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$. All the enumerators jointly will be computed in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$.

▶ **Proposition 33.** *Fix a nonempty set $\mathbf{x}$ of variables and $k \in \mathbb{N}$. Assume G is a graph on n vertices provided on input through a contraction sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of width d. Then one can in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ construct a data structure that associates, to every node $(\mathbf{u}, s)$ of $T_{r,\mathbf{x}}$ and every local type $\tau \in \mathsf{Types}^k_{\mathbf{u},s}$, an enumerator for all tuples in $S^\tau_{\mathbf{u},s}$ with delay $\mathcal{O}_{d,k,\mathbf{x}}(1)$.*

As noted above, Theorem 2 follows from Proposition 33, using Lemma 32. The rest of Section 5 is devoted to proving Proposition 33.

We prove Proposition 33 by induction on $|\mathbf{x}|$. So suppose the statement holds for all strict subsets of $\mathbf{x}$. Recall that we may construct the tree $T_{r,\mathbf{x}}$, in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$, using Lemma 24.

Let v, u be two nodes of $T_{r,\mathbf{x}}$ with $v = (\mathbf{v}, t)$ and $u = (\mathbf{u}, s)$ and $u \preccurlyeq v$. By Lemma 25, there is a function $f_{vu} \colon \mathsf{Types}^k_{\mathbf{v},t} \to \mathsf{Types}^k_{\mathbf{u},s}$ such that for every $\mathbf{w} \in V(G)^{\mathbf{x}}$, with $\mathbf{u} = \mathbf{w}\langle t\rangle$ we have $f_{vu}(\operatorname{ltp}^k_t(\mathbf{w})) = \operatorname{ltp}^k_s(\mathbf{w})$.

For a tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$, let $s \in [n]$ be the first time such that $\mathbf{w}\langle s\rangle$ is r-close at time s, where $r = 2^k$. We then say that $\mathbf{w}$ *registers* at $(\mathbf{u}, s)$, where $\mathbf{u} = \mathbf{w}\langle s\rangle$. By Claim 27, in this case, the pair $(\mathbf{u}, s)$ is a node of $T_{r,\mathbf{x}}$.

For each node $(\mathbf{u}, s)$ of $T_{r,\mathbf{x}}$ and type $\tau \in \mathsf{Types}^k_{\mathbf{u},s}$, denote:

$$R^\tau_{\mathbf{u},s} = \{\mathbf{w} \in V(G)^{\mathbf{x}} \mid \mathbf{w} \text{ registers at } (\mathbf{u}, s) \text{ and } \operatorname{ltp}^k_s(\mathbf{w}) = \tau\}.$$

Fix a node $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and a type $\tau \in \mathsf{Types}^k_{\mathbf{u},s}$. Clearly, every tuple $\mathbf{w} \in S^\tau_{\mathbf{u},s}$ registers at exactly one descendant $v = (\mathbf{v}, t)$ of $u = (\mathbf{u}, s)$ (possibly, $v = u$), and moreover, $f_{vu}(\operatorname{ltp}^k_t(\mathbf{w})) = \tau$. This proves the following.

▶ **Lemma 34.** *For every node $u \in T_{r,\mathbf{x}}$ and type $\tau \in \mathsf{Types}^k_u$, the set S^τ_u is the disjoint union of all the sets R^σ_v, for $v \in T_{r,\mathbf{x}}$ with $v \succcurlyeq u$ and $\sigma \in \mathsf{Types}^k_v$ such that $f_{vu}(\sigma) = \tau$.*

We prove the following two lemmas.

▶ **Lemma 35 (♠).** *For every given node $u = (\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and type $\tau \in \mathsf{Types}^k_{\mathbf{u},s}$, an enumerator for the set $R^\tau_{\mathbf{u},s}$ with delay $\mathcal{O}_{d,k,\mathbf{x}}(1)$ can be constructed in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$.*

▶ **Lemma 36 (♠).** *One can construct in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ a collection of enumerators $\mathcal{E}^\tau_u$, one per each node $u = (\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and type $\tau \in \mathsf{Types}^k_{\mathbf{u},s}$, where $\mathcal{E}^\tau_u$ has delay $\mathcal{O}_{d,k,\mathbf{x}}(1)$ and enumerates all descendants $v = (\mathbf{v}, t)$ of u in $T_{r,\mathbf{x}}$ such that there is some $\sigma \in \mathsf{Types}^k_{\mathbf{v},t}$ with $f_{vu}(\sigma) = \tau$ and $R^\sigma_{\mathbf{v},t} \neq \emptyset$.*

Combining Lemma 34, Lemma 35, Lemma 36 and Lemma 29 yields the required collection of enumerators for each of the sets $S^k_{\mathbf{u},s}$, thus proving Proposition 33 and Theorem 2. The proof of Lemma 36 uses Corollary 31, and is omitted, due to space constraints. We now sketch the proof of Lemma 35.

Proof sketch for Lemma 35. The case when u is a leaf, that is, $s = 1$, is easily solved, since in this case $R^\tau_{\mathbf{u},s}$ is either empty, or consists of a single tuple. In the case when u is an inner node, consider the set

$$\mathcal{V} := \{\mathbf{v} \in \mathcal{P}^{\mathbf{x}}_{s-1} \mid \mathbf{v}\langle s-1 \to s\rangle = \mathbf{u}\}.$$

It is easy to see that $\mathcal{V}$ has size $\mathcal{O}_{k,\mathbf{x},d}(1)$, and can be computed in this time, given $u \in T_{r,\mathbf{x}}$.

Fix $\mathbf{v} \in \mathcal{V}$, and consider the graph $H_{\mathbf{v}}$ with vertices $\mathbf{x}$ where any two distinct $y, y' \in \mathbf{x}$ are adjacent whenever $\mathrm{dist}_s(\mathbf{v}(y), \mathbf{v}(y')) \leqslant r$. Let $C_{\mathbf{v}}$ denote the set of connected components of $H_{\mathbf{v}}$, where each connected component is viewed as a set $\mathbf{y} \subseteq \mathbf{x}$ of vertices of $H_{\mathbf{v}}$. Then each of the sets $C_{\mathbf{v}}$, can be computed in time $\mathcal{O}_{k,\mathbf{x},d}(1)$, given $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and $\mathbf{v} \in \mathcal{V}$. Call a tuple $\mathbf{v} \in \mathcal{V}$ *disconnected* if $H_{\mathbf{v}}$ is such. Note that if $\mathbf{v}$ is disconnected and $\mathbf{y} \in C_{\mathbf{v}}$, then $|\mathbf{y}| < |\mathbf{x}|$.

Fix a disconnected tuple $\mathbf{v} \in \mathcal{V}$ and $\mathbf{y} \in C_{\mathbf{v}}$. Denote by $\mathbf{v}_{\mathbf{y}}$ the restriction of $\mathbf{v}$ to $\mathbf{y}$. Note that the pair $(\mathbf{v}_{\mathbf{y}}, s-1)$ is a node of $T_{r,\mathbf{y}}$. Indeed, by assumption, $\mathrm{dist}_s(B_s, \mathbf{u}) \leqslant r$ holds, so $\mathrm{dist}_s(B_s, \mathbf{v}\langle s \to s+1\rangle) \leqslant r$, and in particular $\mathrm{dist}_s(B_s, \mathbf{v}_{\mathbf{y}}\langle s \to s+1\rangle) \leqslant r$. Moreover, $\mathbf{v}_{\mathbf{y}}$ is r-close, since $\mathbf{y}$ is a connected component of $H_{\mathbf{v}}$. As $|\mathbf{y}| < |\mathbf{x}|$, by inductive assumption, we have already computed enumerators for the sets $S^\sigma_{\mathbf{v}_{\mathbf{y}},s-1}$, for all adequate local types σ.

From Lemma 15 we deduce that the set $R^\tau_{\mathbf{u},s}$ is the disjoint union of sets of the form $S^{\bar\tau}_{\mathbf{v},s-1}$, where:

1. $\mathbf{v} \in \mathcal{V}$ is disconnected;
2. $\bar\tau := (\tau_{\mathbf{y}} : \mathbf{y} \in C_{\mathbf{v}})$ is a tuple of types with $\tau_{\mathbf{y}} \in \mathsf{Types}^k_{\mathbf{v}_y,s-1}$ and which is "merged" into the type τ, using the function from Lemma 15;
3. the set $S^{\bar\tau}_{\mathbf{v},s-1}$ is a Cartesian product of the sets $S^{\tau_{\mathbf{y}}}_{\mathbf{v}_{\mathbf{y}},s-1}$.

Since for the sets $S^{\tau_{\mathbf{y}}}_{\mathbf{v}_{\mathbf{y}},s-1}$ we have enumerators by inductive assumption, by Lemma 28, we can compute in time $\mathcal{O}_{k,d,\mathbf{x}}(1)$ an enumerator for the set $S^{\bar\tau}_{\mathbf{v},s-1}$, with delay $\mathcal{O}_{k,d,\mathbf{x}}(1)$. Finally, we obtain an enumerator for $R^\tau_{\mathbf{u},s}$, by concatenating the $\mathcal{O}_{k,d,\mathbf{x}}(1)$ enumerators for the sets $S^{\bar\tau}_{\mathbf{v},s-1}$. This finishes the proof sketch for Lemma 35. ◀

References

1 Matthias Aschenbrenner, Alf Dolich, Deirdre Haskell, Dugald Macpherson, and Sergei Starchenko. Vapnik-chervonenkis density in some theories without the independence property, i. *Transactions of the American Mathematical Society*, 368, September 2011.

2 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proceedings of the 15th Annual Conference on Computer Science Logic, CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006. doi:10.1007/11874683_11.

3 J. T. Baldwin and S. Shelah. Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic*, 26(3):229–303, 1985.

4 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021. Accepted to STOC 2022. arXiv:2102.03117.

5 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. *CoRR*, abs/2111.00282, 2021. To appear in the proceedings of SODA 2022. arXiv:2111.00282.

6 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. In *Procedings of the 16th International Symposium on Parameterized and Exact Computation, IPEC 2021*, volume 214 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.10.

7 Édouard Bonnet, Eun Jung Kim, Stéphan Thomasse, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *Proceedings of the IEEE 61st Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 601–612. IEEE Computer Society, 2020.

8 Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.

9 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1131–1145. SIAM, 2011.

10 Thomas Colcombet. A combinatorial theorem for trees. In *ICALP 2007*, volume 4596 of *Lecture Notes in Computer Science*, pages 901–912. Springer, 2007.

11 Zdenek Dvorák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013.

12 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

13 Jakub Gajarský, Michał Pilipczuk, and Szymon Toruńczyk. Stable graphs of bounded twin-width. *CoRR*, abs/2107.03711, 2021. arXiv:2107.03711.

14 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25:1–25:12, 2013. doi:10.1145/2528928.

15 Wojciech Kazana and Luc Segoufin. First-order queries on classes of structures with bounded expansion. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:25)2020.

16 Adam Paszke and Michał Pilipczuk. VC density of set systems definable in tree-like graphs. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020*, volume 170 of *LIPIcs*, pages 78:1–78:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.78.

17 Michał Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Toruńczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. *CoRR*, abs/2111.03725, 2021. arXiv:2111.03725.

18 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. On the number of types in sparse graphs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 799–808. ACM, 2018. doi:10.1145/3209108.3209178.

19 Michał Pilipczuk, Marek Sokołowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. *CoRR*, abs/2110.08106, 2021. arXiv:2110.08106.

20 Wojciech Przybyszewski. VC-density and abstract cell decomposition for edge relation in graphs of bounded twin-width. *CoRR*, abs/2202.04006, 2022. arXiv:2202.04006.

21 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

22 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, pages 151–163. ACM, 2018. doi:10.1145/3196959.3196971.

23 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

Reachability in Bidirected Pushdown VASS

Moses Ganardi
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Rupak Majumdar
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Andreas Pavlogiannis
Aarhus University, Denmark

Lia Schütze
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

A pushdown vector addition system with states (PVASS) extends the model of vector addition systems with a pushdown store. A PVASS is said to be *bidirected* if every transition (pushing/popping a symbol or modifying a counter) has an accompanying opposite transition that reverses the effect. Bidirectedness arises naturally in many models; it can also be seen as a overapproximation of reachability. We show that the reachability problem for *bidirected* PVASS is decidable in Ackermann time and primitive recursive for any fixed dimension. For the special case of one-dimensional bidirected PVASS, we show reachability is in PSPACE, and in fact in polynomial time if the stack is polynomially bounded. Our results are in contrast to the *directed* setting, where decidability of reachability is a long-standing open problem already for one dimensional PVASS, and there is a PSPACE-lower bound already for one-dimensional PVASS with bounded stack.

The reachability relation in the bidirected (stateless) case is a congruence over $\mathbb{N}^d$. Our upper bounds exploit saturation techniques over congruences. In particular, we show novel elementary-time constructions of semilinear representations of congruences generated by finitely many vector pairs. In the case of one-dimensional PVASS, we employ a saturation procedure over bounded-size counters.

We complement our upper bound with a TOWER-hardness result for arbitrary dimension and k-EXPSPACE hardness in dimension $2k + 6$ using a technique by Lazić and Totzke to implement iterative exponentiations.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Vector addition systems, Pushdown, Reachability, Decidability, Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.124

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2204.11799` [20]

1 Introduction

The reachability problem for infinite-state systems is one of the most basic and well-studied tasks in verification. Given an infinite-state system and two configurations c_1 and c_2 in the system, it asks: Is there a run from c_1 to c_2? *Pushdown systems* (PDS) and *vector addition systems with states* (VASS) are prominent models for which the reachability problem has been studied extensively. Each of them features a finite set of control states and a storage mechanism that holds an unbounded amount of information. In a PDS, there is a stack where we can push and pop letters. In a VASS, there is a set of counters which can

be *incremented* and *decremented*, but not tested for zero. Reachability in both models is understood in isolation [5, 11, 37, 12, 38], but the reachability problem for their *combination* is a long-standing open problem.

Pushdown VASS. A *pushdown VASS* (PVASS) combines PDS and VASS. A PVASS consists of finitely many control states and has access to both a pushdown stack (as in PDS) and counters (as in VASS). A PVASS is *d-dimensional* if it has d counters. A PVASS is a natural combination of the simple building blocks of PDS and VASS. The reachability problem for PVASS has remained a long-standing open problem [39, 50, 15], even if we combine a pushdown with a single counter.

Bidirectedness. A step toward deciding reachability is to first study natural relaxations of the reachability relation. A relaxation that has recently attracted attention is *bidirectedness*. Bidirectedness assumes that for each transition from state p to q in our infinite-state system, there exists a transition from q to p with opposite effect. For example, in bidirected pushdown systems, for each transition from p to q pushing γ on the stack, there is a transition from q to p that pops γ. Likewise, in bidirected VASS, if there is a transition from p to q that adds some vector $\boldsymbol{v} \in \mathbb{Z}^d$ to counters, then there is a transition from q to p adding $-\boldsymbol{v}$. It turns out that several tasks in program analysis can be formulated or practically approximated as reachability in bidirected pushdown systems [8, 54] or bidirected multi-pushdown systems [51, 52, 55, 40, 41, 31]. Bidirected systems have also been considered in algorithmic group theory as an algorithmic framework to provide simple algorithms for the membership problem in subgroups [42].

Reachability in bidirected systems is usually considerably more efficient than in the general case. In bidirected pushdown systems, reachability can be solved in almost linear time [8] whereas a truly subcubic algorithm for the general case is a long-standing open problem [26, 9]. Reachability in bidirected VASS is equivalent to the uniform word problem in finitely presented commutative semigroups, which is EXPSPACE-complete [43]. A separate polynomial time algorithm for bidirected two-dimensional VASS was given in [41]. Moreover, recent results on reachability in bidirected valence systems shows complexity drops across a large variety of infinite-state systems [21]: For almost every class of systems studied in [21], the complexity of bidirected reachability is lower than in the general case (the only exception being pushdown systems, where the complexity is P-complete in both settings). For example, reachability in bidirected $\mathbb{Z}$-VASS, and even in bidirected $\mathbb{Z}$-PVASS, is in P [21].

However, little is known about bidirected PVASS. They have recently been studied in [31], where decidability of reachability in *dimension one is shown*. However, as in the non-bidirected case, decidability of reachability in bidirected PVASS is hitherto not known.

Contributions. We show that in bidirected PVASS (of arbitrary dimension), reachability is decidable. Moreover, we provide an Ackermann complexity upper bound, and show that in any fixed dimension, reachability is primitive recursive.

▶ **Theorem 1.1.** *Reachability in bidirected pushdown VASS is in* ACKERMANN*, and primitive recursive (in* F_{4d+11}*) if the dimension d is fixed.*

Here, $(\mathsf{F}_\alpha)_\alpha$ is an ordinal-indexed hierarchy of fast-growing complexity classes [48], including $\mathsf{F}_3 = \mathsf{TOWER}$ and $\mathsf{F}_\omega = \mathsf{ACKERMANN}$. The formal definition of the hierarchy can be found in Section 4.3. A recurring theme in our upper bounds is that saturation techniques, the standard method to analyze pushdown systems, combine surprisingly well with counters

in the bidirected setting. Saturation is used in each of our upper bounds. In Section 3, we begin the exposition with a short, self-contained proof that reachability is decidable in bidirected PVASS. It shows that non-reachability is always certified by an inductive invariant of a particular saturation procedure. In Section 4, we show the Ackermann upper bound. Here, we saturate a congruence relation that encodes the reachability relation. The upper bound relies on two key ingredients. First, we use results about Gröbner bases of polynomial ideals to show that in elementary time, one can construct a Presburger formula for the congruence generated by finitely many vector pairs. This construction serves as one step in the saturation. To show termination in Ackermannian time, we rely on a technique from [16] to bound the length of strictly ascending chains of upward closed sets of vectors. Here, the difficulty is to transfer this bound from chains of upward closed sets to chains of congruences.

In Section 5 we present a PSPACE algorithm for bidirected PVASS in dimension one.

▶ **Theorem 1.2.** *Reachability in 1-dimensional bidirected pushdown VASS is in* PSPACE*.*

Here, we rely on an observation from [31] that reachability in bidirected one-dimensional PVASS reduces to (i) coverability in bidirected one-dimensional PVASS and (ii) reachability in one-dimensional bidirected $\mathbb{Z}$-PVASS. Since (ii) is known to be in P [21], we show that (i) can be done in PSPACE. For this, we use saturation to compute, for each state pair (p, q), three bounds on counter values that determine whether coverability holds. We show that these bounds have at most exponential absolute value, which yields a PSPACE procedure.

Finally in Section 6, we show that reachability in bidirected PVASS is TOWER-hard. For this, we adapt a technique from [33] that shows a TOWER lower bound for general PVASS.

▶ **Theorem 1.3.** *Reachability in bidirected PVASS is* TOWER*-hard, and* k*-*EXPSPACE*-hard in dimension* $2k + 6$*.*

Related work. The model of pushdown VASS is surrounded by extensions and restrictions of the storage mechanism for which decidability is understood, the most prominent being the recent Ackermann-completeness for reachability in VASS [11, 37, 12, 38]. If instead of the stack, we have a counter with zero tests, then reachability is still decidable [47, 4]. Here, decidability even holds if we have a zero-testable counter and one additional counter that can be reset [18, 17]. Furthermore, the extension of VASS by *nested zero tests*, where for each $i \in \{1, \ldots, d\}$, we have an instruction that tests all counters $1, \ldots, i$ for zero simultaneously, also allows deciding reachability [47, 3] and can be seen as a special case of pushdown VASS [1]. Another decidability result concerns the *coverability problem*: Here, we are given a configuration c_1 and a control state q and want to know whether from c_1, one can reach some configuration in control state q. It is known that the reachability problem for d-dimensional PVASS reduces to coverability in $(d + 1)$-dimensional PVASS, and that coverability in 1-dimensional PVASS is decidable [39]. According to [15], the latter problem is PSPACE-hard and in EXPSPACE. Furthermore, if the counters in a PVASS are allowed to go negative during a run, then we speak of an *integer PVASS* ($\mathbb{Z}$-PVASS). For these, reachability is known to be decidable [25] and NP-complete [24]. However, if we extend the model of PVASS by allowing resets on the counters, then even coverability is undecidable in dimension one [50].

For VASS, several generalizations of bidirectedness have been studied. It is EXPSPACE-complete whether given two configurations are mutually reachable [35]. Moreover, if two configurations are mutually reachable, then their distance is at most doubly exponential (linear for fixed dimension) in their size [36]. Furthermore, for cyclic VASS (where each transition can be reversed by some execution), it is known that the reachability set has a semilinear representation of at most exponential size [6]. Let us note that in the VASS/Petri net literature, sometimes [6] (but not entirely consistently [35]) the term *reversible* is used to mean bidirected. However, this clashes with the reversibility notion in dynamical systems [30].

2 Preliminaries

Vectors and semilinear sets. We denote integer vectors by bold letters $\boldsymbol{x}$. The maximum norm of $\boldsymbol{x}$ is denoted by $\|\boldsymbol{x}\|$. The i-th unit vector is denoted by $\boldsymbol{e}_i$. The componentwise order $\leq$ on $\mathbb{N}^d$ is a *well-quasi order (wqo)*, i.e. for any infinite sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots$ over $\mathbb{N}^d$ there exist $i < j$ with $\boldsymbol{x}_i \leq \boldsymbol{x}_j$. We write $\boldsymbol{x} < \boldsymbol{y}$ if $\boldsymbol{x} \leq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$. This implies that the set $\min(X)$ of minimal elements in any $X \subseteq \mathbb{N}^d$ is finite. We denote by $X{\uparrow} = \{\boldsymbol{y} \in \mathbb{N}^k \mid \exists \boldsymbol{x} \in X \colon \boldsymbol{x} \leq \boldsymbol{y}\}$ the *upwards closure* of X. We also write $\boldsymbol{x}{\uparrow}$ for $\{\boldsymbol{x}\}{\uparrow}$. A *congruence* on a commutative monoid $(M, +)$, for example $M = \mathbb{N}^d$, is an equivalence relation $\mathcal{Q} \subseteq M \times M$ where $(a, b) \in \mathcal{Q}$ implies $(a + c, b + c) \in \mathcal{Q}$ for all $a, b, c \in M$. We also write $a \sim_{\mathcal{Q}} b$ instead of $(a, b) \in \mathcal{Q}$.

For $X \subseteq \mathbb{N}^k$ we denote by X^* the *submonoid* generated by X. A set $L \subseteq \mathbb{N}^k$ is *linear* if it is of the form $L = \boldsymbol{b} + P^*$ for some base vector $\boldsymbol{b} \in \mathbb{N}^k$ and some finite set $P \subseteq \mathbb{N}^k$ of period vectors. Finite unions of linear sets are called *semilinear*. It is well-known that a set is semilinear if and only if it is definable in *Presburger arithmetic*, i.e. first-order logic over $(\mathbb{N}, +, \leq, 0, 1)$. Furthermore, one can effectively convert between these formats in elementary time: While defining semilinear sets in Presburger arithmetic is straightforward, for the converse we can use Cooper's quantifier elimination [10] running in triply exponential time [44], see also [23] for an excellent overview. We will confuse a semilinear S with its representation, which is either a list of base and period vectors for each linear set or a defining Presburger formula, and denote by $\|S\|$ the size of its representation.

Pushdown VASS. A *d-dimensional pushdown VASS (PVASS)* is a tuple $\mathcal{P} = (Q, \Gamma, T)$ where Q is a finite set of states, Γ is a finite stack alphabet, and $T \subseteq Q \times \mathbb{Z}^d \times \mathsf{Op}(\Gamma) \times Q$ is a finite set of transitions. Here $\mathsf{Op}(\Gamma) = \{a, \bar{a} \mid a \in \Gamma\} \cup \{\varepsilon\}$ is the set of operations on the stack. A configuration over $\mathcal{P}$ is a tuple $(q, \boldsymbol{x}, s) \in Q \times \mathbb{N}^d \times \Gamma^*$. The *one-step relation* $\to$ is the smallest binary relation on configurations such that for all $(p, \boldsymbol{v}, \alpha, q) \in T$ and $\boldsymbol{x} \in \mathbb{N}^d$ with $\boldsymbol{x} + \boldsymbol{v} \geq \boldsymbol{0}$ we have: (i) If $\alpha \in \Gamma \cup \{\varepsilon\}$ then $(p, \boldsymbol{x}, s) \to (q, \boldsymbol{x} + \boldsymbol{v}, s\alpha)$ (ii) if $\alpha = \bar{a}$ then $(p, \boldsymbol{x}, sa) \to (q, \boldsymbol{x} + \boldsymbol{v}, s)$. Its transitive-reflexive closure is denoted by $\xrightarrow{*}$. We say that $\mathcal{P}$ is *bidirected* if $(p, \boldsymbol{v}, \alpha, q) \in T$ implies $(q, -\boldsymbol{v}, \bar{\alpha}, p) \in T$ where we set $\bar{\bar{a}} = a$ for $a \in \Gamma$ and $\bar{\varepsilon} = \varepsilon$. The *reachability problem* for bidirected PVASS asks: Given a bidirected PVASS $\mathcal{P}$ and two states s, t, does $(s, \boldsymbol{0}, \varepsilon) \xrightarrow{*} (t, \boldsymbol{0}, \varepsilon)$ hold?

The counter updates $\boldsymbol{u}$ in a PVASS transition $(p, \boldsymbol{u}, q)$ can be given in either unary or binary encoding since there are logspace translations in both directions: To add a binary encoded number u to a counter we push the binary notation of u to the stack, and repeatedly decrement the stack counter while incrementing u. Since this computation is deterministic, the simulation also works for bidirected PVASS.

For the Ackermann upper bound it is convenient to use pushdown VASS with a single state. A *pushdown VAS (PVAS)* $\mathcal{P} = (\Gamma, T)$ in dimension d consists of a finite stack alphabet Γ and a finite set of transitions $T \subseteq \mathbb{N}^d \times \mathbb{N}^d \times (\Gamma \cup \bar{\Gamma} \cup \{\varepsilon\})$. Here, a configuration is a pair $(\boldsymbol{x}, s) \in \mathbb{N}^d \times \Gamma^*$. The effect of a transition $(\boldsymbol{u}, \boldsymbol{v}, \alpha)$ is subtracting $\boldsymbol{u}$ from the d counters, assuming that the counters stay non-negative, and then adding $\boldsymbol{v}$. A PVAS $\mathcal{P}$ is bidirected if $(\boldsymbol{u}, \boldsymbol{v}, a) \in T$ implies $(\boldsymbol{v}, \boldsymbol{u}, \bar{a}) \in T$. A bidirected PVASS in dimension d can be simulated by a bidirected PVAS in dimension $d + 2$ where the two additional counters add up to the number of states and specify the current state. Hence, one can reduce the reachability problem for bidirected PVASS to the reachability problem for bidirected PVAS: Given a bidirected PVAS $\mathcal{P}$ and two vectors $\boldsymbol{s}, \boldsymbol{t} \in \mathbb{N}^d$, does $(\boldsymbol{s}, \varepsilon) \xrightarrow{*} (\boldsymbol{t}, \varepsilon)$ hold?

3 Decidability

In this section, we present a simple and self-contained proof that reachability is decidable in bidirected PVASS. Consider the reachability relation between configurations with empty stack. For any states p, q, define the set $R_{p,q} \subseteq \mathbb{N}^d \times \mathbb{N}^d$ with

$$R_{p,q} = \{(\boldsymbol{u}, \boldsymbol{v}) \mid \boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^d,\ (p, \boldsymbol{u}, \varepsilon) \xrightarrow{*} (q, \boldsymbol{v}, \varepsilon)\}.$$

We will prove that each $R_{p,q}$ is semilinear, for which we rely on the fact that these sets are slices. A *slice* is a subset $S \subseteq \mathbb{N}^k$ such that if $\boldsymbol{u}, \boldsymbol{u}+\boldsymbol{v}, \boldsymbol{u}+\boldsymbol{w} \in S$ for some $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{N}^k$, then $\boldsymbol{u}+\boldsymbol{v}+\boldsymbol{w} \in S$. Observe that each $R_{p,q} \subseteq \mathbb{N}^{2d}$ is a slice. This is because if $(\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{u}+\boldsymbol{u}_1, \boldsymbol{v}+\boldsymbol{v}_1), (\boldsymbol{u}+\boldsymbol{u}_2, \boldsymbol{v}+\boldsymbol{v}_2) \in R_{p,q}$, then there is a run

$$(p, \boldsymbol{u}+\boldsymbol{u}_1+\boldsymbol{u}_2, \varepsilon) \xrightarrow{*} (q, \boldsymbol{v}+\boldsymbol{v}_1+\boldsymbol{u}_2, \varepsilon) \xrightarrow{*} (p, \boldsymbol{u}+\boldsymbol{v}_1+\boldsymbol{u}_2, \varepsilon) \xrightarrow{*} (q, \boldsymbol{v}+\boldsymbol{v}_1+\boldsymbol{v}_2, \varepsilon),$$

where the middle part exists due to bidirectedness. Thus, the pair $(\boldsymbol{u}+\boldsymbol{u}_1+\boldsymbol{u}_2, \boldsymbol{v}+\boldsymbol{v}_1+\boldsymbol{v}_2)$ belongs to $R_{p,q}$. The following was first shown in [14, Proposition 7.3].

▶ **Theorem 3.1** (Eilenberg & Schützenberger 1969). *Every slice is semilinear.*

This seems to be stronger than the somewhat better-known fact that each congruence on $\mathbb{N}^d$ is semilinear: Observe that every congruence on $\mathbb{N}^d$, seen as a subset of $\mathbb{N}^{2d}$, is a slice. In the case of congruences, a relatively simple proof was obtained by Hirshfeld [27]. We present a proof of Theorem 3.1 that combines ideas from both [14] and [27] and is (in our opinion) simpler than each.

For a set $X \subseteq \mathbb{N}^k$, let $\min X$ be the set of minimal elements of X, with respect to the usual component-wise ordering $\leq$ on $\mathbb{N}^k$. Since this ordering is a well-quasi ordering, $\min X$ is finite for every set X. Suppose $S \subseteq \mathbb{N}^k$ is a slice. For each $\boldsymbol{u} \in S$, let $S - \boldsymbol{u} := \{\boldsymbol{v} \in \mathbb{N}^k \mid \boldsymbol{u}+\boldsymbol{v} \in S\}$. Then $\boldsymbol{u} \leq \boldsymbol{v}$ implies $S - \boldsymbol{u} \subseteq S - \boldsymbol{v}$. Consider for each $\boldsymbol{u} \in S$ the submonoid

$$M_{\boldsymbol{u}} = (\min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\}))^*.$$

In other words, $M_{\boldsymbol{u}}$ is the submonoid of $\mathbb{N}^k$ generated by the non-zero minimal elements of $S - \boldsymbol{u}$. Note that for $\boldsymbol{u}, \boldsymbol{v} \in S$, we have $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$ if and only if $(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\})\uparrow = (S - \boldsymbol{v} \setminus \{\boldsymbol{0}\})\uparrow$. Since S is a slice, we have $\boldsymbol{u} + M_{\boldsymbol{u}} \subseteq S$ for every $\boldsymbol{u} \in S$. Since $\boldsymbol{u} \in \boldsymbol{u} + M_{\boldsymbol{u}}$, we trivially have

$$S = \bigcup_{\boldsymbol{u} \in S} \boldsymbol{u} + M_{\boldsymbol{u}}.$$

Since each $\boldsymbol{u} + M_{\boldsymbol{u}}$ is semilinear, it suffices to show that S is covered by finitely many sets $\boldsymbol{u} + M_{\boldsymbol{u}}$. We first observe that if $\boldsymbol{u} \leq \boldsymbol{v}$ and $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$, then $\boldsymbol{u} + M_{\boldsymbol{u}}$ already covers $\boldsymbol{v} + M_{\boldsymbol{v}}$.

▶ **Lemma 3.2.** *Let $\boldsymbol{u}, \boldsymbol{v} \in S$. If $\boldsymbol{u} \leq \boldsymbol{v}$ and $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$, then $\boldsymbol{v} + M_{\boldsymbol{v}} \subseteq \boldsymbol{u} + M_{\boldsymbol{u}}$.*

Proof. We will use the following claim: For every $\boldsymbol{w} \in S$ with $\boldsymbol{u} \leq \boldsymbol{w} \leq \boldsymbol{v}$, we have $M_{\boldsymbol{u}} = M_{\boldsymbol{w}} = M_{\boldsymbol{v}}$. Indeed, since $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$, we have $\min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\}) = \min(S - \boldsymbol{v} \setminus \{\boldsymbol{0}\})$. Moreover, since S is a slice, we have $S - \boldsymbol{u} \subseteq S - \boldsymbol{w} \subseteq S - \boldsymbol{v}$. Therefore, $\min(S - \boldsymbol{w} \setminus \{\boldsymbol{0}\})$ coincides with $\min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\})$ and $\min(S - \boldsymbol{v} \setminus \{\boldsymbol{0}\})$, which implies $M_{\boldsymbol{w}} = M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$.

Let us prove the lemma. We proceed by induction on $\|\boldsymbol{v} - \boldsymbol{u}\|$. If $\boldsymbol{u} = \boldsymbol{v}$, then we are done. Otherwise, there exists an $\boldsymbol{m} \in \min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\})$ such that $\boldsymbol{u} + \boldsymbol{m} \leq \boldsymbol{v}$. By our claim, we have $M_{\boldsymbol{u}} = M_{\boldsymbol{u}+\boldsymbol{m}} = M_{\boldsymbol{v}}$. Therefore, induction implies $\boldsymbol{v} \in \boldsymbol{u} + \boldsymbol{m} + M_{\boldsymbol{u}+\boldsymbol{m}}$. But since $\boldsymbol{m} \in M_{\boldsymbol{u}}$ and $M_{\boldsymbol{u}+\boldsymbol{m}} \subseteq M_{\boldsymbol{u}}$, this implies $\boldsymbol{v} \in \boldsymbol{u} + M_{\boldsymbol{u}}$. ◀

The following implies semilinearity of S.

▶ **Lemma 3.3.** *There is a finite set $F \subseteq S$ such that $S = \bigcup_{\boldsymbol{u} \in F} \boldsymbol{u} + M_{\boldsymbol{u}}$.*

Proof. Suppose not. Then there is an infinite sequence $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots \in S$ such that each set $\boldsymbol{u}_i + M_{\boldsymbol{u}_i}$ contributes a new element. By Dickson's lemma $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots$ contains a subsequence $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots$ with $\boldsymbol{v}_i \leq \boldsymbol{v}_{i+1}$ for all $i \geq 1$. Since then $S - \boldsymbol{v}_1 \subseteq S - \boldsymbol{v}_2 \subseteq \cdots$, the sequence $(S - \boldsymbol{v}_1 \setminus \{\boldsymbol{0}\})\uparrow \subseteq (S - \boldsymbol{v}_2 \setminus \{\boldsymbol{0}\})\uparrow \subseteq \cdots$ becomes stationary, again by Dickson's lemma, and therefore also the sequence $M_{\boldsymbol{v}_1}, M_{\boldsymbol{v}_2}, \ldots$. By Lemma 3.2, this means that only finitely many terms in the sequence $\boldsymbol{v}_1 + M_{\boldsymbol{v}_1}, \boldsymbol{v}_2 + M_{\boldsymbol{v}_2}, \ldots$ contribute new elements, a contradiction. ◀

Saturation invariants. We have seen that the reachability relations $R_{p,q}$ are all semilinear. However, since the semilinearity proof is non-constructive, this does not explain how to decide reachability. Nevertheless, we shall use semilinearity to show that in case of non-reachability, there exists a certificate. This yields a decision procedure consisting of two semi-algorithms in the style of Leroux's algorithm for reachability in VASS [34]: One semi-algorithm enumerates potential runs, and one enumerates potential certificates for non-reachability.

We assume that we are given a bidirected d-dimensional PVASS with state set Q and stack alphabet Γ. We may assume that all transitions are of the form $p \xrightarrow{\gamma} q$ or $p \xrightarrow{\bar{\gamma}} q$ for $\gamma \in \Gamma$ or $p \xrightarrow{\boldsymbol{v}} q$ for $\boldsymbol{v} \in \mathbb{Z}^d$. Our certificates for non-reachability will be in the form of what we call saturation invariants. Imagine a (non-terminating) naive saturation algorithm that attempts to compute the sets $R_{p,q}$ by adding vector pairs one-by-one to finite sets $F_{p,q}$. It would start with $F_{p,q} = \emptyset$ and then add pairs: For each transition $p \xrightarrow{\boldsymbol{v}} q$ and each vector $\boldsymbol{u} \in \mathbb{N}^d$ with $\boldsymbol{u} + \boldsymbol{v} \in \mathbb{N}^d$, it would add the pair $(\boldsymbol{u}, \boldsymbol{u} + \boldsymbol{v})$ to $F_{p,q}$. Moreover, if $(\boldsymbol{u}, \boldsymbol{v}) \in F_{p,q}$ and $(\boldsymbol{v}, \boldsymbol{w}) \in F_{q,r}$, it would add $(\boldsymbol{u}, \boldsymbol{w})$ to $F_{p,r}$. Finally, if there are transitions $p \xrightarrow{\gamma} p'$ and $q' \xrightarrow{\bar{\gamma}} q$ and there is a $(\boldsymbol{u}, \boldsymbol{v}) \in F_{p',q'}$, then it would add $(\boldsymbol{u}, \boldsymbol{v})$ to $F_{p,q}$.

Intuitively, a saturation invariant is a forward inductive invariant of this naive saturation algorithm. Let us make this precise. For subsets $R_1, R_2 \subseteq \mathbb{N}^d \times \mathbb{N}^d$, we define

$$R_1 \circ R_2 = \{(\boldsymbol{u}, \boldsymbol{w}) \in \mathbb{N}^d \times \mathbb{N}^d \mid \exists \boldsymbol{v} \in \mathbb{N}^d \colon (\boldsymbol{u}, \boldsymbol{v}) \in R_1,\ (\boldsymbol{v}, \boldsymbol{w}) \in R_2\}.$$

A *saturation invariant* consists of a family $(I_{p,q})_{(p,q) \in Q^2}$ of sets $I_{p,q} \subseteq \mathbb{N}^d \times \mathbb{N}^d$ for which

1. For each transition $p \xrightarrow{\boldsymbol{v}} q$, $\boldsymbol{v} \in \mathbb{Z}^d$, each $\boldsymbol{u} \in \mathbb{N}^d$ with $\boldsymbol{u} + \boldsymbol{v} \in \mathbb{N}^d$, we have $(\boldsymbol{u}, \boldsymbol{u} + \boldsymbol{v}) \in I_{p,q}$.
2. For each $p, q, r \in Q$, we have $I_{p,q} \circ I_{q,r} \subseteq I_{p,r}$.
3. For each $p, p', q, q' \in Q$ for which there are transitions $p \xrightarrow{\gamma} p'$, $q' \xrightarrow{\bar{\gamma}} q$ for some $\gamma \in \Gamma$, we have $I_{p',q'} \subseteq I_{p,q}$.

There is a natural ordering of such families $(I_{p,q})_{(p,q) \in Q^2}$ defined by inclusion: We write $(I_{p,q})_{(p,q) \in Q^2} \leq (J_{p,q})_{(p,q) \in Q^2}$, if $I_{p,q} \subseteq J_{p,q}$ for each $p, q \in Q$. In this sense, we can speak of a smallest saturation invariant.

▶ **Lemma 3.4.** *The family $(R_{p,q})_{(p,q) \in Q^2}$ is the smallest saturation invariant.*

Proof. By induction on the length of a run, it follows that $(R_{p,q})_{(p,q) \in Q^2}$ is included in every saturation invariant. Moreover, $(R_{p,q})_{(p,q) \in Q^2}$ is clearly a saturation invariant itself. ◀

Our certificates will consist of saturation invariants defined in Presburger arithmetic. A family $(I_{p,q})_{(p,q) \in Q^2}$ is *Presburger-definable* if for each $(p, q) \in Q^2$, the set $I_{p,q}$ is semilinear. According to Theorem 3.1, the family $(R_{p,q})_{(p,q) \in Q^2}$ is Presburger-definable. Therefore, the following is a direct consequence of Lemma 3.4.

▶ **Theorem 3.5.** *For each $s, t \in Q$, we have $(\boldsymbol{0}, \boldsymbol{0}) \notin R_{s,t}$ if and only if there exists a Presburger-definable saturation invariant $(I_{p,q})_{(p,q) \in Q^2}$ such that $(\boldsymbol{0}, \boldsymbol{0}) \notin I_{s,t}$.*

Algorithm 1 Algorithm for bidirected reachability in PVAS.

Data: Bidirected d-dim. PVAS $\mathcal{P} = (\Gamma, T)$
$R_0 := \mathsf{Cong}(\{(\boldsymbol{u}, \boldsymbol{v}) \mid (\boldsymbol{u}, \boldsymbol{v}, \varepsilon) \in T\})$;
for $i = 1, 2, \ldots$ **do**
  $R_i \leftarrow R_{i-1}$;
  for $(\boldsymbol{u}, \boldsymbol{u}', a) \in T$ *and* $(\boldsymbol{v}', \boldsymbol{v}, \bar{a}) \in T$ **do**
    $R_i \leftarrow R_i \cup \{(\boldsymbol{x} + \boldsymbol{u}, \boldsymbol{y} + \boldsymbol{v}) \mid (\boldsymbol{x} + \boldsymbol{u}', \boldsymbol{y} + \boldsymbol{v}') \in R_{i-1},\ \boldsymbol{x}, \boldsymbol{y} \in \mathbb{N}^d\}$;
  $R_i \leftarrow \mathsf{Cong}(R_i)$;
  if $R_i = R_{i-1}$ **then return** R_i;

This yields our algorithm: One semi-algorithm enumerates transition sequences and terminates if one of them is a run witnessing $(s, \mathbf{0}, \varepsilon) \xrightarrow{*} (t, \mathbf{0}, \varepsilon)$. The other semi-algorithm enumerates Presburger-definable families $(I_{p,q})_{(p,q)\in Q^2}$ in the form of Presburger formulas. Using Presburger arithmetic, it is then easy to check whether (i) $(I_{p,q})_{(p,q)\in Q^2}$ is a saturation invariant and (ii) $(\mathbf{0}, \mathbf{0}) \notin I_{s,t}$. If a saturation invariant is found, the semi-algorithm reports non-reachability. By Theorem 3.5, one of the two semi-algorithms must terminate.

4 Ackermann upper bound

In this section, we show that reachability in bidirected PVASS is solvable in Ackermann time in the general case and in primitive recursive complexity in every fixed dimension.

One way to avoid enumeration in the algorithm of Section 3 would be to start with the semilinear one-step relation described in the first condition of saturation invariants, and then to enlarge it according to the second and third condition. Moreover, one could take the slice closure (the smallest slice that includes the current set) after each enlargement. Since slices satisfy an ascending chain condition [14, Corollary 12.3], this would ensure termination. In fact, computing the slice closure of a semilinear set is possible with an algorithm by Grabowski [22]. Unfortunately, the latter is itself based on enumeration and we are not aware of any complexity bounds for computing slice closures. Therefore, we use an analogous algorithm that uses congruences instead of slices. Since congruences can be encoded in polynomial ideals, we can tap into the rich toolbox of Gröbner bases to compute the congruence generated by a semilinear set.

4.1 The saturation algorithm

In the following we will work with pushdown VAS instead of pushdown VASS. Our decision procedure for bidirected reachability relies on the crucial fact that the reachability relation $R_{\mathcal{P}} = \{(\boldsymbol{s}, \boldsymbol{t}) \in \mathbb{N}^d \times \mathbb{N}^d \mid (\boldsymbol{s}, \varepsilon) \xrightarrow{*} (\boldsymbol{t}, \varepsilon)\}$ of a bidirected pushdown VAS $\mathcal{P}$ is a congruence: It is always reflexive, transitive and additive, even for directed pushdown VAS, and symmetric for bidirected systems. Therefore, whenever we have found an underapproximation $R \subseteq R_{\mathcal{P}}$ we can replace R by the smallest congruence containing R. The smallest congruence containing a set $R \subseteq \mathbb{N}^d \times \mathbb{N}^d$ is denoted by $\mathsf{Cong}(R)$. We also say that R is a *basis of* (or *generates*) $\mathsf{Cong}(R)$. Recall that every congruence on $\mathbb{N}^d$ is a slice. Therefore, congruences are semilinear and ascending chains of congruences stabilize.

Algorithm 1 is a saturation algorithm that computes a semilinear representation for $R_{\mathcal{P}}$. The sets R_i are maintained by semilinear representations or Presburger formulas. Since in this section we only prove elementary complexity bounds, we can use both formats

interchangeably. Observe that the update in line 5 and the equality test in line 7 can be expressed in Presburger arithmetic. The computation of $\mathsf{Cong}(\cdot)$ will be explained in the next subsection. Consider the values of R_i for $i \geq 1$ after line 6 of Algorithm 1. They form an ascending chain of congruences $(R_i)_{i\geq 1}$, which implies that the algorithm must terminate. For the correctness one can prove by induction on i that $(\boldsymbol{x}, \boldsymbol{y}) \in R_i$ if and only if there exists a run between $(\boldsymbol{x}, \varepsilon)$ and $(\boldsymbol{y}, \varepsilon)$ whose stack height does not exceed i. Moreover, if the algorithm terminates after k iterations then $R_k = R_{\mathcal{P}}$.

We will use a primitive recursive algorithm (which is elementary in fixed dimension) to compute $\mathsf{Cong}(R_i)$ from a semilinear representation for R_i. Using the tools from [49] we can then prove upper bounds for the length of the ascending chain.

4.2 Semilinear representations for congruences

In this section, we present an algorithm that, for a given semilinear representation for a set $R \subseteq \mathbb{N}^d \times \mathbb{N}^d$, computes a semilinear representation for $\mathsf{Cong}(R)$. Its run time is bounded by a tower of exponentials in $\|R\|$ of height $O(d)$ (Theorem 4.4). Note that for bidirected VASS, it is known that in exponential space, one can compute a semilinear representation of the reachability set [32, 6]. In other words, one can compute in exponential space a representation of the congruence class of a given vector $\boldsymbol{x} \in \mathbb{N}^d$. In contrast, our algorithm computes a semilinear representation of the entire congruence.

Let the function $\exp^k$ be inductively defined by $\exp^0(x) = x$ and $\exp^{k+1}(x) = \exp^k(2^x)$. In the following we show how to compute a semilinear representation for a congruence $\mathcal{Q}$ given by a semilinear basis $R \subseteq \mathbb{N}^d \times \mathbb{N}^d$ in time $\exp^{O(d)}(\|R\|)$. In fact, we can assume that R is *finite* since a linear set $L = \boldsymbol{b} + P^*$ is contained in $\mathsf{Cong}(F_L)$ where $F_L = \{\boldsymbol{b}, \boldsymbol{b}+\boldsymbol{p} \mid \boldsymbol{p} \in P\}$, and, therefore, a semilinear set $\bigcup_{i=1}^m L_i$ generates the same congruence as $\bigcup_{i=1}^m F_{L_i}$. The semilinear representation of $\mathcal{Q}$ will be obtained by induction on the dimension d via a decomposition of $\mathbb{N}^d$ into smaller regions. A *region* is a linear set $L = \boldsymbol{b} + P^* \subseteq \mathbb{N}^d$ where $P \subseteq \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d\}$. Its *dimension* is $|P|$. In particular all sets $\boldsymbol{b}{\uparrow} = \boldsymbol{b} + \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d\}^*$ are regions. For a region $L = \boldsymbol{b} + P^*$ and a congruence $\mathcal{Q}$, we define the congruence $\mathcal{Q}_L = \{(\boldsymbol{x}, \boldsymbol{y}) \in (P^*)^2 \mid (\boldsymbol{b} + \boldsymbol{x}, \boldsymbol{b} + \boldsymbol{y}) \in \mathcal{Q}\}$ on the submonoid P^*.

A submonoid $S \subseteq \mathbb{N}^k$ is *subtractive* if $\boldsymbol{x}, \boldsymbol{y} \in S$ and $\boldsymbol{x} \leq \boldsymbol{y}$ implies $\boldsymbol{y} - \boldsymbol{x} \in S$. For example, the non-negative restriction $G \cap \mathbb{N}^k$ of a group $G \subseteq \mathbb{Z}^k$ is a subtractive submonoid. The following lemma is well-known, see [14, Proposition 7.1] or the full version [20] for a proof.

▶ **Lemma 4.1.** *Every subtractive submonoid $S \subseteq \mathbb{N}^k$ is of the form $S = M^*$ where M is the finite set of the minimal nonzero elements in S.*

Eilenberg and Schützenberger observed that for every slice S there exists an element $\boldsymbol{s} \in S$ such that $S - \boldsymbol{s}$ is a subtractive submonoid [14, Proposition 7.2]. As a consequence, for every congruence $\mathcal{Q}$ on $\mathbb{N}^d$ there exists $\boldsymbol{b} \in \mathbb{N}^d$ such that $\mathcal{Q}_{\boldsymbol{b}\uparrow}$ is a subtractive submonoid. We provide an elementary bound on $\boldsymbol{b}$.

▶ **Lemma 4.2.** *Given a finite basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$, one can compute in elementary time a vector $\boldsymbol{b} \in \mathbb{N}^d$ and a finite set $M \subseteq \mathbb{N}^{2d}$ such that $\mathcal{Q}_{\boldsymbol{b}\uparrow} = M^*$.*

Gröbner bases. It remains to compute a semilinear representation of $\mathcal{Q}$ on the complement of $\boldsymbol{b}{\uparrow}$. We will decompose $\mathbb{N}^d \setminus \boldsymbol{b}{\uparrow}$ into disjoint $(d-1)$-dimensional regions L_j, compute bases for the restrictions $\mathcal{Q}_{L_j}$, and proceed inductively. To compute the bases for $\mathcal{Q}_{L_j}$, we will exploit the well-studied connection between congruences on $\mathbb{N}^d$ and binomial ideals [43]. Let $\mathbb{Z}[\boldsymbol{x}]$ be the polynomial ring in the variables $\boldsymbol{x} = (x_1, \ldots, x_d)$ over $\mathbb{Z}$. We write $\boldsymbol{x}^{\boldsymbol{u}}$ for

the monomial $x_1^{\boldsymbol{u}(1)} \cdots x_d^{\boldsymbol{u}(d)}$. An *ideal* is a nonempty set $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ such that $f, g \in I$ and $h \in \mathbb{Z}[\boldsymbol{x}]$ implies $f + g, hf \in I$. An ideal I is finitely represented by a *basis* B, i.e. I is the smallest ideal containing B. By Hilbert's basis theorem any ideal $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ has a finite basis. One of the main tools in computer algebra for handling polynomial ideals are *Gröbner bases*, e.g. to solve the ideal membership problem. We defer the reader to [2] for details on Gröbner bases and only mention the properties required for our purposes. A Gröbner basis is defined with respect to an admissible monomial ordering, e.g. a lexicographic ordering on the monomials. Buchberger's algorithm [7] computes for a given basis for an ideal I the *unique reduced* Gröbner basis for I in doubly exponential space [13].

A basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$ can be translated into the polynomial ideal $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ generated by $B_R = \{\boldsymbol{x}^{\boldsymbol{u}} - \boldsymbol{x}^{\boldsymbol{v}} \mid (\boldsymbol{u}, \boldsymbol{v}) \in R\}$. It is known that $\boldsymbol{s} \sim_{\mathcal{Q}} \boldsymbol{t}$ if and only if $\boldsymbol{x}^{\boldsymbol{s}} - \boldsymbol{x}^{\boldsymbol{t}} \in I$ [43, Lemma 1 and 2]. Moreover, the reduced Gröbner basis of I with respect to an admissible monomial order always consists of differences of monomials $\boldsymbol{x}^{\boldsymbol{s}} - \boldsymbol{x}^{\boldsymbol{t}}$ [29, Theorem 2.7].

The following lemma can be reduced to two known applications of Gröbner bases. Let $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ be an ideal. For a subsequence $\boldsymbol{y}$ of $\boldsymbol{x}$ we call $I \cap \mathbb{Z}[\boldsymbol{y}]$ the *elimination ideal*, which is indeed an ideal in $\mathbb{Z}[\boldsymbol{y}]$. For $\boldsymbol{b} \in \mathbb{N}^d$ we define the *ideal quotient* $I : \boldsymbol{x}^{\boldsymbol{b}} = \{p \in \mathbb{Z}[\boldsymbol{x}] \mid p\boldsymbol{x}^{\boldsymbol{b}} \in I\}$, which is also an ideal. It is known that one can compute Gröbner bases for $I \cap \mathbb{Z}[\boldsymbol{y}]$ and $I : \boldsymbol{x}^{\boldsymbol{b}}$ in elementary time [2, Section 6], see the full version [20] for more details.

▶ **Lemma 4.3.** *Given a finite basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$ and a region $L \subseteq \mathbb{N}^d$, one can compute in elementary time a finite basis for $\mathcal{Q}_L$.*

We are ready to compute a semilinear representation of $\mathsf{Cong}(R)$ in $\exp^{O(d)}(n)$ time. We proceed by induction over d. Using Lemma 4.2 we can write $\mathcal{Q}_{\boldsymbol{b}\uparrow} = M^*$. We decompose $\mathbb{N}^d = \bigcup_{i=0}^m L_j$ into regions where $L_0 = \boldsymbol{b}\uparrow$ and $L_1, \ldots, L_m$ are $(d-1)$-dimensional regions. By Lemma 4.3 we can compute bases for $\mathcal{Q}_{L_i}$ for $i \in [1, m]$ and by induction hypothesis semilinear representations for $\mathcal{Q}_{L_i}$. In this way, we obtain semilinear representations for the restrictions $Q_i = \mathcal{Q} \cap L_i^2$ for each $i \in [0, m]$. Finally, we can express $\boldsymbol{s} \sim_{\mathcal{Q}} \boldsymbol{t}$ by a Presburger formula that says that there exists a sequence of intermediate vectors of length $2(m+1)$ where adjacent elements are related by an R-step or are contained in some relation Q_i.

▶ **Theorem 4.4.** *Given a semilinear basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$, one can compute a semilinear representation for $\mathcal{Q}$ in time $\exp^{c_1 d}(n)$ for some absolute constant c_1.*

4.3 Ascending chains of congruences

To bound the length of the chain of congruences R_i in Algorithm 1 we use a *length function theorem* [49, Theorem 3.15], see also [16]. In general, such theorems allow to derive complexity bounds for algorithms whose termination arguments are based on well-quasi orders.

Fast-growing complexity classes. In the following we state a simplified version of [49, Theorem 3.15], which is sufficient for our application. We start by introducing fast-growing functions and complexity classes. Recall that the Cantor normal form of an ordinal $\alpha \leq \omega^\omega$ is the unique representation $\alpha = \omega^{\alpha_1} + \cdots + \omega^{\alpha_p}$ where $\alpha > \alpha_1 \geq \cdots \geq \alpha_p$. In this form α is a limit ordinal if and only if $p > 0$ and $\alpha_p > 0$. A *fundamental sequence* for a limit ordinal λ is a sequence $(\lambda(x))_{x<\omega}$ of ordinals with supremum λ. Given a limit ordinal $\lambda \leq \omega^\omega$ whose Cantor normal form is $\lambda = \beta + \omega^{k+1}$, we use the standard fundamental sequence $(\lambda(x))_{x<\omega}$, defined inductively as $\omega^\omega(x) = \omega^{x+1}$ and $(\beta + \omega^{k+1})(x) = \beta + \omega^k \cdot (x+1)$. Given a function $h \colon \mathbb{N} \to \mathbb{N}$ the *Hardy hierarchy* $(h^\alpha)_{\alpha \leq \omega^\omega}$ relative to h is defined by

$$h^0(x) = x, \qquad h^{\alpha+1}(x) = h^\alpha(h(x)), \qquad h^\lambda(x) = h^{\lambda(x)}(x).$$

Using the Hardy functions $(H^\alpha)_\alpha$ relative to $H(x) = x + 1$ we can define the *fast-growing* complexity classes $(\mathsf{F}_\alpha)_\alpha$ from [48]. We denote by $\mathcal{F}_{<\alpha}$ the class of functions computed by deterministic Turing machines in time $O(H^\beta(n))$ for some $\beta < \omega^\alpha$. By F_α we denote the class of decision problems solved by deterministic Turing machines in time $O(H^{\omega^\alpha}(p(n)))$ for some function $p \in \mathcal{F}_{<\alpha}$. We define $\mathsf{PRIMITIVE\text{-}RECURSIVE} = \bigcup_{k<\omega} \mathsf{F}_k$ and $\mathsf{ACKERMANN} = \mathsf{F}_\omega$.

Controlled bad sequences. By Dickson's lemma, any sequence of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots$ with $x_i \not\leq x_j$ for all $i < j$ must be finite. Such a sequence is also called *bad*. To obtain complexity bounds on the length of bad sequences we need to restrict to sequences that do not grow in an uncontrolled fashion. In the following let $g\colon \mathbb{N} \to \mathbb{N}$ be *monotone*, *strictly inflationary*, i.e. $g(x) > x$ for all x, and *super-homogeneous*, i.e. $g(xy) \geq g(x) \cdot y$ for all $x, y \geq 1$. A sequence of vectors $\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_\ell$ is (g,n)-*controlled* if $\|\boldsymbol{x}_i\| \leq g^i(n)$ for all $i \in [0, \ell]$. The following statement follows from [49, Theorem 3.15] and [49, Eq. (3.13)].

▶ **Theorem 4.5.** *Any (g,n)-controlled bad sequence over $\mathbb{N}^k$ has length at most $g^{\omega^k}(nk)$.*

A *chain* in $\mathbb{N}^k$ is a sequence $S_0 \subsetneq S_1 \subsetneq \cdots \subsetneq S_\ell$ of subsets $S_i \subseteq \mathbb{N}^k$. The chain is (g,n)-*controlled* if for each $i \in [0, \ell-1]$ there exists $\boldsymbol{s}_i \in S_{i+1} \setminus S_i$ with $\|\boldsymbol{s}_i\| \leq g^i(n)$. A set $X \subseteq \mathbb{N}^k$ is *upwards closed* if $X = X{\uparrow}$. Observe that any (g,n)-controlled chain of upwards closed sets in $\mathbb{N}^k$ is bounded by $1 + g^{\omega^k}(nk)$ since we obtain a (g,n)-controlled bad sequence $\boldsymbol{s}_0, \boldsymbol{s}_1, \dots, \boldsymbol{s}_{\ell-1}$ by picking arbitrary $\boldsymbol{s}_i \in S_{i+1} \setminus S_i$ with $\|\boldsymbol{s}_i\| \leq g^i(n)$.

Translating congruences into upwards closed sets. The key trick in our upper bound for ascending chains of congruences is to translate congruences into upwards closed sets. This allows us to translate bounds on the length of ascending chains of upward closed sets into corresponding bounds for congruences. The translation works as follows. To each congruence $\mathcal{Q}$ on $\mathbb{N}^d$, we associate the upwards closed set $U(\mathcal{Q}) \subseteq \mathbb{N}^{4d}$ with

$$U(\mathcal{Q}) = \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}) \mid (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q},\ (\boldsymbol{x} + \boldsymbol{u}, \boldsymbol{y} + \boldsymbol{v}) \in \mathcal{Q},\ (\boldsymbol{u}, \boldsymbol{v}) \neq (\boldsymbol{0}, \boldsymbol{0})\}{\uparrow}.$$

Clearly $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$ implies $U(\mathcal{Q}_1) \subseteq U(\mathcal{Q}_2)$. In fact, strict inclusion is also preserved:

▶ **Lemma 4.6.** *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be congruences with $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$. Then (i) $U(\mathcal{Q}_1) \subseteq U(\mathcal{Q}_2)$ and (ii) for each $\boldsymbol{q} \in \mathcal{Q}_2 \setminus \mathcal{Q}_1$, there is a $\boldsymbol{p} \in U(\mathcal{Q}_2) \setminus U(\mathcal{Q}_1)$ with $\|\boldsymbol{p}\| \leq \|\boldsymbol{q}\|$.*

Proof. Statement (i) is immediate. For statement (ii) let $(\boldsymbol{s}, \boldsymbol{t}) \in \mathcal{Q}_2 \setminus \mathcal{Q}_1$ be minimal. Since $(\boldsymbol{0}, \boldsymbol{0}) \in \mathcal{Q}_1$ we must have $(\boldsymbol{s}, \boldsymbol{t}) \neq (\boldsymbol{0}, \boldsymbol{0})$ and there exists $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q}_1$ with $(\boldsymbol{x}, \boldsymbol{y}) < (\boldsymbol{s}, \boldsymbol{t})$. We choose such a vector $(\boldsymbol{x}, \boldsymbol{y})$ in which $(\boldsymbol{u}, \boldsymbol{v}) := (\boldsymbol{s}, \boldsymbol{t}) - (\boldsymbol{x}, \boldsymbol{y})$ is minimal. Clearly, $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}) \in U(\mathcal{Q}_2)$. We claim that $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}) \notin U(\mathcal{Q}_1)$. Towards a contradiction, suppose that there exists $(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{u}_1, \boldsymbol{v}_1) \leq (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v})$ with $(\boldsymbol{x}_1, \boldsymbol{y}_1) \in \mathcal{Q}_1$, $(\boldsymbol{u}_1, \boldsymbol{v}_1) \neq (\boldsymbol{0}, \boldsymbol{0})$, and $(\boldsymbol{x}_1 + \boldsymbol{u}_1, \boldsymbol{y}_1 + \boldsymbol{v}_1) \in \mathcal{Q}_1$. Since $\mathcal{Q}_1$ is a congruence we have

$$\begin{aligned}\boldsymbol{x} + \boldsymbol{u}_1 = (\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{x}_1 + \boldsymbol{u}_1 &\sim_{\mathcal{Q}_1} (\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{y}_1 + \boldsymbol{v}_1 \\ &\sim_{\mathcal{Q}_1} (\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{x}_1 + \boldsymbol{v}_1 = \boldsymbol{x} + \boldsymbol{v}_1 \sim_{\mathcal{Q}_1} \boldsymbol{y} + \boldsymbol{v}_1.\end{aligned}$$

If $(\boldsymbol{u}_1, \boldsymbol{v}_1) = (\boldsymbol{u}, \boldsymbol{v})$ then this contradicts $(\boldsymbol{x} + \boldsymbol{u}, \boldsymbol{y} + \boldsymbol{v}) = (\boldsymbol{s}, \boldsymbol{t}) \notin \mathcal{Q}_1$. If $(\boldsymbol{u}_1, \boldsymbol{v}_1) < (\boldsymbol{u}, \boldsymbol{v})$ then $(\boldsymbol{s}, \boldsymbol{t}) = (\boldsymbol{x} + \boldsymbol{u}_1, \boldsymbol{y} + \boldsymbol{v}_1) + (\boldsymbol{u} - \boldsymbol{u}_1, \boldsymbol{v} - \boldsymbol{v}_1)$ contradicts the minimality of $(\boldsymbol{u}, \boldsymbol{v})$. ◀

If $(\mathcal{Q}_i)_{i \leq \ell}$ is a (g,n)-controlled chain of congruences in $\mathbb{N}^d$ then by Lemma 4.6, $(U(\mathcal{Q}_i))_{i \leq \ell}$ is a (g,n)-controlled chain of upwards closed subsets of $\mathbb{N}^{4d}$ and thus has length at most $1 + g^{\omega^{4d}}(4dn)$. Hence, the same bound applies to $(\mathcal{Q}_i)_{i \leq \ell}$.

▶ **Lemma 4.7.** *Any (g,n)-controlled chain of congruences in $\mathbb{N}^d$ has length $\leq 1 + g^{\omega^{4d}}(4dn)$.*

Putting together Theorem 4.4 and Lemma 4.7 we can conclude that Algorithm 1 terminates after $H^{\omega^{4d+3}}(e(n))$ iterations for some elementary function $e(n)$.

▶ **Proposition 4.8.** *Reachability in bidirected pushdown VAS is in* ACKERMANN*, and in* F_{4d+3} *if the dimension d is fixed.*

The detailed complexity analysis can be found in the full version [20]. The result above also holds for bidirected pushdown *VASS* (Theorem 1.1) by simulating the state in two additional counters. Hence the complexity for dimension d increases from F_{4d+3} to F_{4d+11}.

5 One-dimensional pushdown VASS

In this section we prove that reachability is in polynomial space for bidirected 1-PVASS (Theorem 1.2). For the rest of this section consider a 1-PVASS $\mathcal{P} = (Q, \Gamma, T)$ of $|Q| = n$ states. To simplify our bounds, we assume $|\Gamma| = 2$. This can be achieved with a simple encoding: To simulate stack letters $a_1, \ldots, a_k$, we can encode each a_i by the string $ab^iab^{k-i}a$.

Preliminaries. We extend the usual component-wise ordering $\leq$ to tuples $(\mathbb{Z} \cup \{-\infty, \omega\})^k$. Given two functions $f, g \colon X \to (\mathbb{Z} \cup \{-\infty, \omega\})^k$, we write $f \leq g$ to denote that $f(x) \leq g(x)$ for each $x \in X$. We define the *one-step $\mathbb{Z}$ relation* $\hookrightarrow$ for $\mathcal{P}$ similarly to the one-step relation $\to$ but with a $\mathbb{Z}$-counter, i.e., we do not require the counter to remain non-negative. A *path from p to q* consists of the initial state p and a sequence of transitions of $\mathcal{P}$, such that it induces a run $(p_1, x_1, w_1) \hookrightarrow (p_2, x_2, w_2) \hookrightarrow \ldots \hookrightarrow (p_j, x_j, w_j)$, with the requirement that $p_1 = p$, $x_1 = 0$ and $w_1 = w_j = \varepsilon$. Given such a path P, we let

- $\mathit{MaxSH}(P) = \max_i |w_i|$ be the maximum stack height along P,
- $w(P) = x_j$ be the value of the counter at the end of P, and
- $m(P) = \min_i x_i$ be the minimum value of the counter along P. Note that $m(P) \leq 0$, as the counter is 0 at the beginning of P.

We also write $\overline{P}$ for the reverse of P. We denote by $\{p \rightsquigarrow q\}$ the set of paths from p to q, and let $\{p \rightsquigarrow q\}_k = \{P \in \{p \rightsquigarrow q\} \colon \mathit{MaxSH}(P) \leq k\}$ be the set of such paths with stack height at most k. Given two paths P_1 and P_2, we write $P_1 \leq P_2$ to denote that $(m(P_1), w(P_1)) \leq (m(P_2), w(P_2))$.

We say that a state q is reachable from a state p if $(p, \mathbf{0}, \varepsilon) \xrightarrow{*} (q, \mathbf{0}, \varepsilon)$. We say that q is *$\mathbb{Z}$-reachable* from p if there is a path $P \in \{p \rightsquigarrow q\}$ with $w(P) = 0$ (hence state reachability implies $\mathbb{Z}$-reachability). Given additionally a natural number $i \in \mathbb{N}$, we say that p *covers* (q, i) if $(p, 0, \varepsilon) \xrightarrow{*} (q, i + j, \varepsilon)$ for some $j \geq 0$. Thus reachability implies coverability for $i = 0$. The following is a simpler proof of a reduction observed in [31]: For bidirected 1-PVASS, reachability reduces to coverability and $\mathbb{Z}$-reachability.

▶ **Lemma 5.1.** *For any two states p, q of $\mathcal{P}$, we have that p reaches q iff (i) p covers $(q, 0)$, (ii) q covers $(p, 0)$, and (iii) p $\mathbb{Z}$-reaches q.*

Proof. Clearly if p reaches q then conditions (i)-(iii) hold, so we only need to argue about the reverse direction. If p does not cover $(q, 1)$, since p covers $(q, 0)$, we have that p reaches q, and we are done. Similarly, if q does not cover $(p, 1)$, we have that q reaches p and thus we are done. Finally, assume that p covers $(q, 1)$ and q covers $(p, 1)$, and let P_p and P_q be the corresponding paths witnessing coverability. Let P be a path witnessing that p $\mathbb{Z}$-reaches q. We construct the path H_ℓ witnessing the reachability of q from p as $H_\ell = (P_p \circ P_q)^\ell \circ P \circ (\overline{P_p} \circ \overline{P_q})^\ell$, where ℓ is chosen such that $w((P_p \circ P_q)^\ell) \geq -m(P)$. ◀

In light of Lemma 5.1, for Theorem 1.2, it suffices to show that for bidirected 1-PVASS, both $\mathbb{Z}$-reachability and coverability can be decided in PSPACE. The former is known already: Reachability in $\mathbb{Z}$-PVASS belongs to NP [24]; in the bidirected case, it is even decidable in P [21]. Thus, the rest of this section is devoted to deciding coverability in PSPACE.

▶ **Lemma 5.2.** *Coverability in 1-dimensional bidirected pushdown VASS is in* PSPACE.

Summary functions. We define a summary function $\gamma_k \colon Q \times Q \to (\mathbb{Z}_{\leq 0} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{-\infty, \omega\})$, parametric on $k \in \mathbb{N}$, as $\gamma_k(p,q) = (a,b)$, where a and b are defined as follows.

$$a = \max(\{m(P) \colon P \in \{p \rightsquigarrow q\}_k\}) \qquad b = \sup(\{w(P) \colon P \in \{p \rightsquigarrow q\}_k \text{ and } m(P) = a\})$$

with the convention that $\max(\emptyset) = \sup(\emptyset) = -\infty$. We occasionally write $\gamma_k(p,q) = (a, \_)$ to denote that $\gamma_k(p,q) = (a,b)$ for some b. We further define a summary function $\delta_k \colon V \to (\mathbb{Z}_{\leq 0} \cup \{-\infty\})$, parametric on $k \in \mathbb{N}$, as follows.

$$\delta_k(p) = \max(\{m(P) \colon P \in \{p \rightsquigarrow p\}_k \text{ and } w(P) > 0\})$$

Recall that we use $n = |Q|$ for the number of states in $\mathcal{P}$. It is well-known that in any pushdown system of n states (and only two stack letters), a shortest path between two states has length $2^{O(n^2)}$ (e.g. this follows by inspecting a proof of the pumping lemma for context-free languages [28, Lemma 6.1]; a precise bound was obtained in [45]). Since both the weight and the minima of any path are lower-bounded by minus the length of the path, if $\{p \rightsquigarrow q\}_k \neq \emptyset$, then a shortest path in $\{p \rightsquigarrow q\}_k$ witnesses $\gamma_k(p,q) = (a,b)$ where a and b are at most exponentially negative. This is established in the following lemma.

▶ **Lemma 5.3.** *Consider any two states p, q and natural number k, and let $\gamma_k(p,q) = (a,b)$. If $a > -\infty$ then $a, b \geq -\beta$, for $\beta = 2^{O(n^2)}$.*

The bidirectedness of $\mathcal{P}$ also leads to the following lemma.

▶ **Lemma 5.4.** *Consider any two states p, q and natural number k, and let $\gamma_k(p,q) = (a,b)$. There exists a constant α independent of $\mathcal{P}$ and k, such that, if $b > 2^{\alpha n^2}$, then $b = \omega$.*

The intuition behind the summary functions γ_k and δ_k is as follows. Lemma 5.3 and Lemma 5.4 hint on an algorithm to decide coverability by saturating γ_k iteratively for increasing k. The lemmas state that the finite values of γ_k are exponentially bounded, and thus the process is guaranteed to reach a fixpoint within exponentially many iterations. An obstacle to this approach is that γ_k may fail to capture paths that are useful in subsequent iterations. In particular, $\gamma_k(p,q)$ misses paths that can reach q with a larger counter at the cost of a lower minima on the way. The following lemma shows that δ_k captures the effects of all paths missed by γ_k, and allows the two summaries to be *mutually* saturated.

▶ **Lemma 5.5.** *For any states p, q, let $\gamma_k(p,q) = (a,b)$, and assume there exists a path $P \in \{p \rightsquigarrow q\}_k$ with $w(P) > b$. Then $\delta_k(p) \geq m(P)$.*

Moreover, the values of δ_k are also exponentially bounded, and thus the mutual saturation can be carried out in polynomial space.

▶ **Lemma 5.6.** *For any state p, if $\delta_k(p) > -\infty$ then $\delta_k(p) \geq -\beta$, for $\beta = 2^{O(n^2)}$.*

In the remainder of this section we describe the saturation procedure for γ_k and δ_k.

Finite graphs. We consider weighted finite graphs $G = (V_G, E_G, w_G)$ where $w_G \colon E_G \to \mathbb{Z}$. Moreover, we assume that every connected component of G is strongly connected. By a small abuse we extend some of the above notation on $\mathcal{P}$ to such graphs. Given two nodes u, v in G, we write $\{u \rightsquigarrow v\}^G$ for the set of paths from u to v in G. We similarly extend the summary functions to γ^G and δ^G, defined by the corresponding paths $P \in \{u \rightsquigarrow v\}^G$.

▶ **Lemma 5.7.** *Given a graph G as above, the summary functions γ^G and δ^G can be computed in polynomial time.*

Computing γ_k and δ_k. We now describe a dynamic-programming algorithm for computing γ_k and δ_k, for increasing values of k. We let $\Gamma_\perp = \Gamma \cup \{\perp\}$, where $\perp$ is a special symbol, and assume without loss of generality that every transition in $\mathcal{P}$ that manipulates the counter does not affect the stack. Afterwards we will argue that the algorithm terminates within exponentially many iterations.

1. We start with $k = 0$. We construct a graph G_0 that consists of nodes $\langle p, \perp\rangle$ where p is a state of $\mathcal{P}$. Moreover, G_0 contains the corresponding transitions of $\mathcal{P}$ that do not manipulate the stack. In particular, for every transition $(p, i, \varepsilon, q) \in T$ we have an edge $\langle p, \perp\rangle \xrightarrow{i} \langle q, \perp\rangle$ in G_0. We use Lemma 5.7 to compute γ^{G_0} and δ^{G_0}, and report that, for all states p and q, we have $\gamma_0(p, q) = \gamma^{G_0}(\langle p, \perp\rangle, \langle q, \perp\rangle)$ and $\delta_0(p) = \delta^{G_0}(\langle p, \perp\rangle)$.
2. We repeat the following until γ^{G_k} and δ^{G_k} have converged. We construct a graph G_k as follows. For every state p and every $\sigma \in \Gamma_\perp$, we have a node $\langle p, \sigma\rangle$ in G_k. We then do the following.
   a. Let $\delta^{G_{k-1}}(\langle p, \perp\rangle) = c$. We insert a node $\langle p', \sigma\rangle$, and if $-\infty < c$, we insert two edges manipulating the counter $\langle p, \sigma\rangle \xrightarrow{c} \langle p', \sigma\rangle$ and $\langle p', \sigma\rangle \xrightarrow{-c+1} \langle p, \sigma\rangle$.
   b. For every state q, let $\gamma^{G_{k-1}}(\langle p, \perp\rangle, \langle q, \perp\rangle) = (a, b)$. If $-\infty < a$, we insert a node $\langle p, q, \sigma\rangle$ and two edges manipulating the counter $\langle p, \sigma\rangle \xrightarrow{a} \langle p, q, \sigma\rangle$ and $\langle p, q, \sigma\rangle \xrightarrow{-a+b'} \langle q, \sigma\rangle$, where $b' = b$ if $b < \omega$ and $b' = 0$ otherwise.
3. Finally, for each stack letter $\sigma \in \Gamma$, we connect nodes of the form $\langle p, \perp\rangle$ and $\langle q, \sigma\rangle$ using the transitions of $\mathcal{P}$ that manipulate the stack. That is, for every transition $(p, 0, \sigma, q) \in T$, we insert an edge $\langle p, \perp\rangle \to \langle q, \sigma\rangle$, and for every transition $(p, 0, \overline{\sigma}, q) \in T$, we insert an edge $\langle p, \sigma\rangle \to \langle q, \perp\rangle$.
4. We use Lemma 5.7 to compute γ^{G_k} and δ^{G_k}, and report that, for all states p and q, we have $\gamma_k(p, q) = \gamma^{G_k}(\langle p, \perp\rangle, \langle q, \perp\rangle)$ and $\delta_k(p) = \delta^{G_k}(\langle p, \perp\rangle)$.

We first argue that the graphs G_k consist of strongly connected components, and thus Lemma 5.7 is applicable.

▶ **Lemma 5.8.** *For all $k \in \mathbb{N}$, every connected component of G_k is strongly connected.*

Given some $\sigma \in \Gamma_\perp$ and $k \geq 1$, we denote by $G_k \restriction \sigma$ the subgraph of G_k induced by the nodes whose last element is σ. It follows directly from the construction of G_k that, for every pair of states p, q of $\mathcal{P}$ and $\sigma \in \Gamma_\perp$, for every path P that goes from $\langle p, \sigma\rangle$ to $\langle q, \sigma\rangle$ and is contained in $G_k \restriction \sigma$, there is a path $H \in \{p \rightsquigarrow q\}_{k-1}$ with $P \leq H$. In turn, this implies that the summary functions γ^{G_k} and δ^{G_k} are always dominated by paths in $\mathcal{P}$ of stack height at most k, i.e., for all states p and q, we have $\gamma^{G_k}(\langle p, \perp\rangle, \langle q, \perp\rangle) \leq \gamma_k(p, q)$ and $\delta^{G_k}(\langle p, \perp\rangle) \leq \delta_k(p)$ for all $k \in \mathbb{N}$. The following lemma states that γ^{G_k} and δ^{G_k} compute γ_k and δ_k exactly, by arguing that γ^{G_k} and δ^{G_k} also dominate all paths of $\mathcal{P}$ with stack height at most k. In turn, this establishes the correctness of the algorithm.

▶ **Lemma 5.9.** *For all $k \in \mathbb{N}$ and states $p, q \in Q$, we have $\gamma_k(p, q) = \gamma^{G_k}(\langle p, \perp\rangle, \langle q, \perp\rangle)$ and $\delta_k(p) = \delta^{G_k}(\langle q, \perp\rangle)$.*

Thus, to decide whether p covers $(q, 0)$, we saturate γ_k and δ_k and check whether $\gamma_k(p, q) = (0, \_)$. The termination and complexity of the algorithm follows from the boundedness of the finite values of γ_k and δ_k (Lemma 5.3, Lemma 5.4 and Lemma 5.6), which concludes Lemma 5.2.

▶ **Lemma 5.10.** *The above algorithm terminates and uses polynomial space.*

Finally, note that if we have a polynomial bound on the stack height, then the saturation procedure runs in polynomial time, which also leads to reachability in polynomial time (a closer analysis yields an $O(n^3)$ bound per iteration). In particular, the PSPACE-hardness proof for 1-dimensional *directed* PVASS from [15] cannot be directly transferred to bidirected PVASS: The 1-PVASS constructed in [15] has a polynomially bounded stack height. See the full version [20] for details on how exactly the reduction fails. Without a bound on the stack height, the saturation might indeed take exponential time: There are 1-dimensional bidirected PVASS on which shortest coverability witnesses require an exponential stack height (see the full version [20] for an example).

6 Tower lower bound

In this section, we show that reachability in bidirected PVASS is TOWER-hard with respect to elementary reductions, and k-EXPSPACE-hard in dimension $2k + 6$. Recall that TOWER is the class of all problems computable by deterministic Turing machines in time (or space) bounded by a tower of exponentials of elementary height.

Lower bound for directed PVASS. We first recall the TOWER-hardness proof by Lazić and Totzke [33] for reachability in directed PVASS. They reduce the $\exp^n(1)$-bounded halting problem for counter programs of size n, which is TOWER-complete with respect to elementary reductions (which allow us to replace the parameter n with an arbitrary elementary function $e(n)$) [19]. A *counter program* is a finite sequence of commands which manipulate non-negative counters, initially set to zero. The commands include increments $x := x + 1$, decrements $x := x - 1$, conditionals **if** $x = 0$ **then goto** L_1 **else goto** L_2 (where L_1 and L_2 are line numbers), and **halt**. If a counter of value 0 is decremented, the program aborts. The $\exp^n(1)$-bounded halting problem asks whether given a counter program of size n, starting from the first command and all counters set to zero, a command **halt** can be reached using a run during which all counters are bounded by $\exp^n(1)$ and are all zero at the end.

As in most lower bounds for vector addition systems and their extensions, for each counter x we store a complement counter $\bar{x}$, maintaining the invariant $x + \bar{x} = B$ for some (large) bound B. This can be achieved by complementing every in/decrement of x by a de/increment of $\bar{x}$, and vice versa. Then, a zero test **if** $x = 0$ **then goto** L_1 **else goto** L_2 can be replaced by guessing whether $x = 0$ and $x \neq 0$: In the former case we add and subtract B to x and continue with L_1. In the latter case we add and subtract B to $\bar{x}$ and continue with L_2.

The challenge is to implement the addition (and subtraction) with a large number B, here $B = \exp^n(1)$, using a polynomially large system. Suppose we have counters $c_1, \ldots, c_n$ with complement counters $\bar{c}_1, \ldots, \bar{c}_n$ satisfying $c_k + \bar{c}_k = \exp^k(1)$ for all k. Lazić and Totzke [33] show how to construct for all $k = 1, \ldots, n$ a $\mathrm{poly}(k)$-sized PVASS that adds $\exp^k(1)$ to c_k. It operates by pushing exactly $\exp^{k-1}(1)$ many zeros to the stack, repeatedly incrementing the $\exp^{k-1}(1)$-bit binary counter on the stack, while simultaneously decrementing c_k, and finally popping exactly $\exp^{k-1}(1)$ many ones from the stack. Observe that the operations on the

stack can be implemented with the help of c_{k-1} that can be in/decremented by $\exp^{k-1}(1)$ by induction hypothesis. Before simulating the counter program, each complement counter $\bar{c}_k$ has to be set to $\exp^k(1)$, which can be done in a similar fashion.

Simulation by bidirected systems. In the following we will make the above construction outlined by Lazić and Totzke [33] explicit and show that the simulation is correct even after making the PVASS bidirected. To this end we need the following argument already found in Post's undecidability proof of the word problem for Thue systems [46, Lemma II]. Consider a deterministic transition system where the final state does not have any outgoing transitions. To such a system we now add reverse edges to make it bidirected. Clearly, any original run is present in the bidirected system. Conversely, consider a run to the final state in the bidirected system, which may use the new reverse edges. It cannot end on a reverse edge, since there is no outgoing forward edge from the final state. So as long as the run contains reverse edges, at least one of these edges must be followed by one in the forward direction. Let us call them $p \xrightarrow{\bar{a}} q$ and $q \xrightarrow{b} r$. As the original system was deterministic q has exactly one outgoing edge, and hence $(q \xrightarrow{b} r) = (q \xrightarrow{a} p)$. Since the effects of $\bar{a}$ and b cancel out, we can omit both of them from the run. Iterating this argument eventually yields a run with no reverse edges. It follows that adding reverse edges to a fully deterministic system does not change its reachability set (this was originally shown by Mayr and Meyer [43] for their proof of EXPSPACE-hardness of reachability for bidirected VAS).

The construction of Lazić and Totzke is not fully deterministic. However, it only uses very restricted nondeterminism that will not impact our simulation of the counter machine.

Gadget construction. Since we also want to show a k-EXPSPACE lower bound for dimension $2k+6$, we use a slightly more refined analysis: We will assume that two numbers k and n are given as input and construct a system that simulates counters bounded by $\exp^k(n)$ instead of $\exp^k(1)$ as in Lazić and Totzke.

In the following a *gadget* G consists of a PVASS and two distinguished terminal states s and t. We consider vectors $\boldsymbol{x} \in \mathbb{N}^{2k}$ where the first k components are viewed the values of k counters $c_1, \ldots, c_k$ and the last k components are the values of k complementary counters $\bar{c}_1, \ldots, \bar{c}_k$. Without further mention, any update on a counter c is always understood with complementary update on $\bar{c}$ so that the sums $c_i + \bar{c}_i$ remain constant.

Given two numbers k, n (in unary), we will inductively construct a gadget G_k with stack alphabet Γ_k. This gadget will allow us to add $\exp^k(n)$ to a counter. The gadget's size will grow exponentially in k (and polynomially in n), and later, we improve the construction to grow only polynomially in k. The gadget G_1 simply increments c_1 by 2^n. Assuming G_{k-1} is already constructed, we construct the gadget G_k. The gadget $\bar{G}_{k-1}$ is obtained from G_{k-1} by reversing all transitions, and interchanging its terminal states. Its behavior is inverse to that of G_{k-1}, as it subtracts $\exp^{k-1}(n)$ from c_{k-1}. Let $Z_{k-1} = G_{k-1} \circ \bar{G}_{k-1}$ be the gadget obtained by composing G_{k-1} with $\bar{G}_{k-1}$, which is a zero test of c_{k-1}. We can naturally view G_{k-1}, $\bar{G}_{k-1}$ and Z_{k-1} as gadgets with $2k$ counters, where c_k and $\bar{c}_k$ are untouched. The gadget G_k is displayed in Figure 1 where $\mathtt{0}$ and $\mathtt{1}$ are fresh stack symbols and Inc_k is a subprocedure which increments the binary counter on the stack.

To prove correctness of the gadget G_k we need a bit of notation. For brevity we write $[x_1, \ldots, x_k]$ for $(x_1, \ldots, x_k, \exp^1(n) - x_1, \ldots, \exp^k(n) - x_k)$. Our gadgets will always assume that the "lower" counters c_j are set to zero and that the invariant is satisfied. A counter vector of the form $[0, \ldots, 0, x_i, \ldots, x_k]$ is called *i-initialized*. Moreover, we call a run $(s, \boldsymbol{u}, w) \xrightarrow{*} (t, \boldsymbol{v}, w')$ in a gadget *i-initialized* if either $\boldsymbol{u}$ or $\boldsymbol{v}$ is i-initialized.

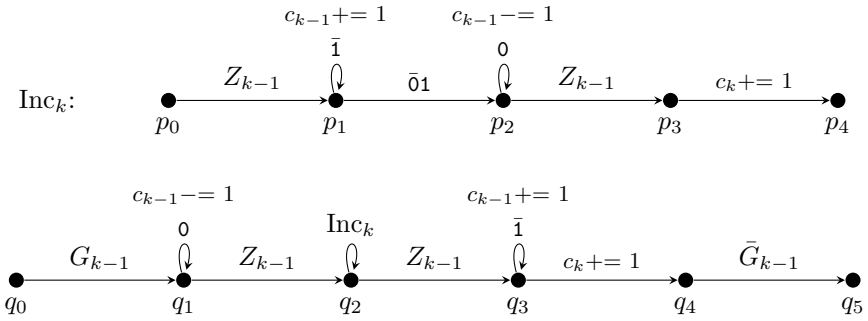


Figure 1 Gadgets Inc$_k$ and G_k.

▶ **Proposition 6.1.** *The k-initialized runs in G_k from q_0 to q_5 are precisely the runs*

$$(q_0, [0, \dots, 0], w) \xrightarrow{*}_{G_k} (q_5, [0, \dots, 0, \exp^k(n)], w) \quad \text{for } w \in \Gamma_{k-1}^*.$$

Next we will analyse the bidirected version of G_k. In order to distinguish the original transitions from the reverse transitions we define for a PVASS G the relations $\leftrightarrow_G = \rightarrow_G \cup \leftarrow_G$ and $\overset{*}{\leftrightarrow}_G$, denoting the reflexive transitive closure of $\leftrightarrow_G$. Similarly to the argument by Post [46, Lemma II], we can prove the following:

▶ **Proposition 6.2.** *Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^{2k}$ where $\boldsymbol{u}$ or $\boldsymbol{v}$ is k-initialized.*

- *If $(q_0, \boldsymbol{u}, w) \overset{*}{\leftrightarrow}_{G_k} (q_5, \boldsymbol{v}, w')$ then $(q_0, \boldsymbol{u}, w) \xrightarrow{*}_{G_k} (q_5, \boldsymbol{v}, w')$.*
- *If $q \in \{q_0, q_5\}$ and $(q, \boldsymbol{u}, w) \overset{*}{\leftrightarrow}_{G_k} (q, \boldsymbol{v}, w')$ then $\boldsymbol{u} = \boldsymbol{v}$ and $w = w'$.*

We need to reduce the size of G_k so that it can be constructed in time polynomial in k. Since G_k uses ten copies of the subgadget G_{k-1} (each zero test Z_{k-1} uses two copies of G_{k-1}), we cannot simply insert G_{k-1} by copying it, as this would induce exponential growth of the number of states of our system. Instead, we instantiate each gadget G_{k-1} once. Then, whenever a gadget would be used between two states p, q, we push a fresh stack symbol $t_{p,q}$ and move to G_{k-1}. When exiting G_{k-1} we pop $t_{p,q}$ and return to q. Since this symbol is unique for every pair of states, it uniquely determines where we can leave the gadget to, even if there are multiple incoming and outgoing transitions at the gadget G_{k-1}. Finally, one can verify that Proposition 6.2 still holds for this adapted version of G_k.

Simulating the counter program. We are ready to finish the lower bound proof. We are given a counter program of size n with three counters x_1, x_2, x_3 and want to reduce the $\exp^{k+1}(n)$-bounded halting problem to the reachability problem for bidirected PVASS using $2k + 6$ counters. To this end, we construct the gadget G_{k+1} three times: Each of these three instances has, instead of c_{k+1} (and its complement), a counter x_i (and its complement) for some $i \in \{1, 2, 3\}$. However, the three instances of G_{k+1} share the counters $c_1, \dots, c_k$ (and their complements). Thus, in total, we have $2 \cdot k + 2 \cdot 3 = 2k + 6$ counters. If k is fixed, this yields our k-EXPSPACE lower bound ([19]). If k is part of the input, the problem becomes TOWER-complete. We start by initializing the complement counters $\bar{c}_1, \dots, \bar{c}_k$ in sequence, using variants of the gadgets G_i that (i) operate on the balance counter $\bar{c}_i$ instead of c_i, (ii) do not decrement c_i when incrementing $\bar{c}_i$, and (iii) operate on the lower $i - 1$ counters as normal. Similarly we initialize $\bar{x}_1, \bar{x}_2, \bar{x}_3$ to $\exp^{k+1}(n)$. Finally, in order to have an all-zero configuration in the final state, we de-initialize these counters before entering the final state.

Increments and decrements in the counter program are directly translated into counter updates in the PVASS. A conditional **if** $x_i = 0$ **then goto** L_1 **else goto** L_2 is replaced by a nondeterministic guess of whether $x_i = 0$ or $x_i \neq 0$, verifying this (in)equality, and jumping

to L_1 or L_2. Here we use variants of the zero tests $Z_{k+1} = G_{k+1} \circ \bar{G}_{k+1}$ which on their highest level operate on x and $\bar{x}$ (instead of c_{k+1} and $\bar{c}_{k+1}$). The question of reachability of **halt** is then a reachability instance on the bidirected version of the PVASS.

If the counter program halts then we can find a corresponding computation in the PVASS. Conversely, consider a successful run of the bidirected PVASS which uses a minimal number of reverse transitions. By Proposition 6.2 we can assume that no gadget G_{k+1} and $\bar{G}_{k+1}$ (and their variants) is entered and exited through the same terminal state. Furthermore, any subrun passing through such a gadget can be assumed to use only forward transitions. Hence the only reverse transitions remaining are from increments or decrements. Observe that the last occurrence of such a reverse transition $\bar{\tau}$ must be followed by its corresponding forward transition τ. Hence we can cancel τ with $\bar{\tau}$, contradiction.

7 Conclusion

We have shown that the reachability problem in bidirected pushdown VASS is decidable, with an Ackermann upper bound and a TOWER lower bound. Moreover, in the one-dimensional case, the problem is in PSPACE, whereas P-hardness was shown in [21]. Thus, the exact complexity, both in the general and the one-dimensional case, remains open.

Another direction for future research is to study bidirected versions of other infinite-state models. For example, pushdown VASS are the simplest level in a hierarchy of infinite-state models for which decidability of the reachability problem is open [53]. Perhaps the techniques from this paper can be applied to show decidability of all levels in the bidirected setting.

References

1 Mohamed Faouzi Atig and Pierre Ganty. Approximating Petri net reachability along context-free traces. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 152–163. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.152.

2 Thomas Becker, Volker Weispfenning, and Heinz Kredel. *Gröbner bases - a computational approach to commutative algebra*, volume 141 of *Graduate texts in mathematics*. Springer, 1993.

3 Rémi Bonnet. *Theory of well-structured transition systems and extended vector-addition systems*. PhD thesis, ENS Cachan, France, 2013. Thèse de doctorat.

4 Rémi Bonnet, Alain Finkel, Jérôme Leroux, and Marc Zeitoun. Model checking vector addition systems with one zero-test. *Log. Methods Comput. Sci.*, 8(2), 2012. doi:10.2168/LMCS-8(2:11)2012.

5 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997. doi:10.1007/3-540-63141-0_10.

6 Zakaria Bouziane and Alain Finkel. Cyclic Petri net reachability sets are semi-linear effectively constructible. In *Second International Workshop on Verification of Infinite State Systems, Infinity 1997, Bologna, Italy, July 11-12, 1997*, volume 9 of *Electronic Notes in Theoretical Computer Science*, pages 15–24. Elsevier, 1997. doi:10.1016/S1571-0661(05)80423-2.

7 Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal*. PhD thesis, Universität Innsbruck, 1965.

8 Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. Optimal Dyck Reachability for Data-Dependence and Alias Analysis. *Proc. ACM Program. Lang.*, 2(POPL), December 2018. doi:10.1145/3158118.

9 Swarat Chaudhuri. Subcubic algorithms for recursive state machines. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 159–169. ACM, 2008. doi:10.1145/1328438.1328460.

10 David C Cooper. Theorem proving in arithmetic without multiplication. *Machine intelligence*, 7(91-99):300, 1972.

11 Wojciech Czerwiński, Slawomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

12 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.

13 Thomas W Dubé. The structure of polynomial ideals and Gröbner bases. *SIAM Journal on Computing*, 19(4):750–773, 1990. doi:10.1137/0219053.

14 Samuel Eilenberg and M. P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13(2):173–191, 1969. doi:10.1016/0021-8693(69)90070-2.

15 Matthias Englert, Piotr Hofman, Slawomir Lasota, Ranko Lazić, Jérôme Leroux, and Juliusz Straszynski. A lower bound for the coverability problem in acyclic pushdown VAS. *Inf. Process. Lett.*, 167:106079, 2021. doi:10.1016/j.ipl.2020.106079.

16 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 269–278. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.39.

17 Alain Finkel, Jérôme Leroux, and Grégoire Sutre. Reachability for two-counter machines with one test and one reset. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.31.

18 Alain Finkel and Grégoire Sutre. Decidability of reachability problems for classes of two counters automata. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 346–357. Springer, 2000. doi:10.1007/3-540-46541-3_29.

19 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Mathematical systems theory*, 1968. doi:10.1007/BF01694011.

20 Moses Ganardi, Rupak Majumdar, Andreas Pavlogiannis, Lia Schütze, and Georg Zetzsche. Reachability in bidirected pushdown VASS. Full version of this paper. arXiv:2204.11799.

21 Moses Ganardi, Rupak Majumdar, and Georg Zetzsche. The complexity of bidirected reachability in valence systems. To appear in Proc. of the Thirty-Seventh Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2022). arXiv:2110.03654.

22 Jan Grabowski. An algorithm to identify slices, with applications to vector replacement systems. In *International Conference on Fundamentals of Computation Theory (FCT 1981)*, pages 425–432. Springer, 1981. doi:10.1007/3-540-10854-8_46.

23 Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. doi:10.1145/3242953.3242964.

24 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 743–759. Springer, 2011. doi:10.1007/978-3-642-22110-1_60.

25 Tero Harju, Oscar H. Ibarra, Juhani Karhumäki, and Arto Salomaa. Some decision problems concerning semilinearity and commutation. *J. Comput. Syst. Sci.*, 65(2):278–294, 2002. doi:10.1006/jcss.2002.1836.

26 Nevin Heintze and David A. McAllester. On the cubic bottleneck in subtyping and flow analysis. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science (LICS), Warsaw, Poland, June 29 - July 2, 1997*, pages 342–351. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614960.

27 Yoram Hirshfeld. *Congruences in commutative semigroups*. LFCS, Department of Computer Science, University of Edinburgh Edinburgh, 1994.

28 John E Hopcroft and Jeffrey D Ullman. *Introduction to Automata Theory, Languages and Computation*. Adison-Wesley, Reading, Mass, 1979.

29 Dung T. Huynh. A superexponential lower bound for Gröbner bases and Church-Rosser commutative Thue systems. *Inf. Control.*, 68(1-3):196–206, 1986. doi:10.1016/S0019-9958(86)80035-3.

30 Jarkko Kari. Reversible cellular automata: From fundamental classical results to recent developments. *New Gener. Comput.*, 36(3):145–172, 2018. doi:10.1007/s00354-018-0034-6.

31 Adam Husted Kjelstrøm and Andreas Pavlogiannis. The decidability and complexity of interleaved bidirected Dyck reachability. *Proc. ACM Program. Lang.*, 6(POPL):1–26, 2022. doi:10.1145/3498673.

32 Ulla Koppenhagen and Ernst W. Mayr. The complexity of the coverability, the containment, and the equivalence problems for commutative semigroups. In *Fundamentals of Computation Theory, 11th International Symposium, FCT '97, Kraków, Poland, September 1-3, 1997, Proceedings*, volume 1279 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 1997. doi:10.1007/BFb0036189.

33 Ranko Lazić and Patrick Totzke. *What Makes Petri Nets Harder to Verify: Stack or Data?*, pages 144–161. Springer International Publishing, 2017. doi:10.1007/978-3-319-51046-0_8.

34 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 307–316. ACM, 2011. doi:10.1145/1926385.1926421.

35 Jérôme Leroux. Vector addition system reversible reachability problem. *Log. Methods Comput. Sci.*, 9(1), 2013. doi:10.2168/LMCS-9(1:5)2013.

36 Jérôme Leroux. Distance between mutually reachable Petri net configurations. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.47.

37 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.

38 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.

39 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

40 Yuanbo Li, Qirun Zhang, and Thomas W. Reps. Fast graph simplification for interleaved Dyck-reachability. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 780–793. ACM, 2020. doi:10.1145/3385412.3386021.

41 Yuanbo Li, Qirun Zhang, and Thomas W. Reps. On the complexity of bidirected interleaved Dyck-reachability. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021. doi:10.1145/3434340.

42 Markus Lohrey and Benjamin Steinberg. An automata theoretic approach to the generalized word problem in graphs of groups. *Proceedings of the American Mathematical Society*, 138(2):445–453, 2010. doi:10.1090/S0002-9939-09-10126-0.

43 Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in mathematics*, 46(3):305–329, 1982. doi:10.1016/0001-8708(82)90048-2.

44 Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.*, 16(3):323–332, 1978. doi:10.1016/0022-0000(78)90021-1.

45 Laurent Pierre. Rational indexes of generators of the cone of context-free languages. *Theoretical Computer Science*, 95(2):279–305, 1992. doi:10.1016/0304-3975(92)90269-L.

46 Emil L. Post. Recursive unsolvability of a problem of Thue. *J. Symb. Log.*, 12(1):1–11, 1947. doi:10.2307/2267170.

47 Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electron. Notes Theor. Comput. Sci.*, 223:239–264, 2008. doi:10.1016/j.entcs.2008.12.042.

48 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

49 Sylvain Schmitz. *Algorithmic Complexity of Well-Quasi-Orders. (Complexité algorithmique des beaux pré-ordres)*. Habilitation thesis, École normale supérieure Paris-Saclay, 2017. URL: https://tel.archives-ouvertes.fr/tel-01663266.

50 Sylvain Schmitz and Georg Zetzsche. Coverability is undecidable in one-dimensional pushdown vector addition systems with resets. In *Reachability Problems - 13th International Conference, RP 2019, Brussels, Belgium, September 11-13, 2019, Proceedings*, volume 11674 of *Lecture Notes in Computer Science*, pages 193–201. Springer, 2019. doi:10.1007/978-3-030-30806-3_15.

51 Guoqing Xu, Atanas Rountev, and Manu Sridharan. Scaling CFL-reachability-based points-to analysis using context-sensitive must-not-alias analysis. In *ECOOP 2009 - Object-Oriented Programming, 23rd European Conference, Genoa, Italy, July 6-10, 2009. Proceedings*, volume 5653 of *Lecture Notes in Computer Science*, pages 98–122. Springer, 2009. doi:10.1007/978-3-642-03013-0_6.

52 Dacong Yan, Guoqing Xu, and Atanas Rountev. Demand-driven context-sensitive alias analysis for Java. In *Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA 2011, Toronto, ON, Canada, July 17-21, 2011*, pages 155–165. ACM, 2011. doi:10.1145/2001420.2001440.

53 Georg Zetzsche. The emptiness problem for valence automata over graph monoids. *Information and Computation*, 277, 2021. doi:10.1016/j.ic.2020.104583.

54 Qirun Zhang, Michael R. Lyu, Hao Yuan, and Zhendong Su. Fast algorithms for Dyck-CFL-reachability with applications to alias analysis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pages 435–446. ACM, 2013. doi:10.1145/2491956.2462159.

55 Qirun Zhang and Zhendong Su. Context-sensitive data-dependence analysis via linear conjunctive language reachability. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 344–358. ACM, 2017. doi:10.1145/3009837.3009848.

Distributed Controller Synthesis for Deadlock Avoidance

Hugo Gimbert
Université de Bordeaux, CNRS, France

Corto Mascle
Université de Bordeaux, France

Anca Muscholl
Université de Bordeaux, France

Igor Walukiewicz
Université de Bordeaux, CNRS, France

Abstract

We consider the distributed control synthesis problem for systems with locks. The goal is to find local controllers so that the global system does not deadlock. With no restriction this problem is undecidable even for three processes each using a fixed number of locks. We propose two restrictions that make distributed control decidable. The first one is to allow each process to use at most two locks. The problem then becomes complete for the second level of the polynomial time hierarchy, and even in PTIME under some additional assumptions. The dining philosophers problem satisfies these assumptions. The second restriction is a nested usage of locks. In this case the synthesis problem is NEXPTIME-complete. The drinking philosophers problem falls in this case.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Distributed Synthesis, Concurrency, Lock Synchronisation, Deadlock Avoidance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.125

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2204.12409`

Funding Work partially funded by ANR project FREDDA (ANR-17-CE40-0013).

1 Introduction

Synthesis of distributed systems has a big potential since such systems are difficult to write, test, or verify. The state space and the number of different behaviors grow exponentially with the number of processes. This is where distributed synthesis can be more useful than centralized synthesis, because an equivalent, sequential system may be very big. The other important point is that distributed synthesis produces by definition a distributed system, while a synthesized sequential system may not be implementable on a given distributed architecture. Unfortunately, very few settings are known for which distributed synthesis is decidable, and those that we know require at least exponential time.

The problem was first formulated by Pnueli and Rosner [27]. Subsequent research showed that, essentially, the only decidable architectures are pipelines, where each process can send messages only to the next process in the pipeline [20, 23, 11]. In addition, the complexity is non-elementary in the size of the pipeline. These negative results motivated the study of distributed synthesis for asynchronous automata, and in particular synthesis with so called causal information. In this setting the problem becomes decidable for co-graph action alphabets [12], and for tree architectures of processes [14, 25]. Yet the complexity

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 125; pp. 125:1–125:20


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is again non-elementary, this time w.r.t. the depth of the tree. Worse, it has been recently established that distributed synthesis with causal information is undecidable for unconstrained architectures [17]. Distributed synthesis for (safe) Petri nets [10] has encountered a similar line of limited advances, and due to [17], is also undecidable in the general case, since it is inter-reducible to distributed synthesis for asynchronous automata [3]. This situation raised the question if there is any setting for distributed synthesis that covers some standard examples of distributed systems, and is manageable algorithmically.

In this work we consider distributed systems with locks; each process can take or release a lock from a pool of locks. Locks are one of the most classical concepts in distributed systems. They are also probably the most frequently used synchronization mechanism in concurrent programs. We formulate our results in a control setting rather than synthesis – this avoids the need for a specification formalism. The objective is to find a local strategy for each process so that the global system does not get stuck. For unrestricted systems with locks we hit again an undecidability barrier, as for the models discussed above. Yet, we find quite interesting restrictions making distributed control synthesis for systems with locks decidable, and even algorithmically manageable.

The first restriction we consider is to limit the number of locks available to each process. The classical example are dining philosophers, where each philosopher has two locks corresponding to the left and the right fork. Observe that we do not limit the total number of processes, or the total number of locks in the system. We show that the complexity of this synthesis problem is at the second level of the polynomial hierarchy. The problem gets even simpler when we restrict it to strategies that cannot block a process when all locks are available. We call them *locally live strategies*. We obtain an NP-algorithm for locally live strategies, and even a Ptime algorithm when the access to locks is *exclusive*. This means that once a process tries to acquire a lock it cannot switch to some other action before getting the lock.

The second restriction is nested lock usage. This is a very common restriction in the literature [19], simply saying that acquiring and releasing locks should follow a stack discipline. Drinking philosophers [4] are an example of a system of this kind. We show that in this case distributed synthesis is Nexptime-complete, where the exponent in the algorithm depends only on the number of locks.

We formalize the distributed synthesis problem as a control problem [28]. A process is given as a transition graph where transitions can be local actions, or acquire/release of a lock. Some transitions are controllable, and some are not. A controller for a process decides which controllable transitions to allow, depending on the local history. In particular, the controller of a process does not see the states of other processes. Our techniques are based on analyzing patterns of taking and releasing locks. In decidable cases there are finite sets of patterns characterizing potential deadlocks.

The notion of patterns resembles locking disciplines [7], a tool frequently used to prevent deadlocks. An example of a locking discipline is "take the left fork before the right one" in the dining philosophers problem. Our results allow to check if a given locking discipline may result in a deadlock, and in some cases even list all deadlock-avoiding locking disciplines.

In summary, the main results of this work are:

- Σ_2^P-completeness of the deadlock avoidance control problem for systems where each process has access to at most 2 locks.
- An NP algorithm when additionally strategies need to be locally live.
- A Ptime algorithm when moreover lock access is exclusive.
- A Nexptime algorithm and the matching lower bound for the nested lock usage case.
- Undecidability of the deadlock avoidance control problem for systems with unrestricted access to locks.

Related work

Distributed synthesis is an old idea motivated by the Church synthesis problem [5]. Actually, the logic CTL has been proposed with distributive synthesis in mind [6]. Given this long history, there are relatively few results on distributed synthesis. Three main frameworks have been considered: synchronous networks of input/output automata, asynchronous automata, Petri games.

The synchronous network model has been proposed by Pnueli and Rosner [27]. They established that controller synthesis is decidable for pipeline architectures and undecidable in general. The undecidability result holds for very simple architectures with only two processes. Subsequent work has shown that in terms of network shape pipelines are essentially the only decidable case [20, 23, 11]. Several ways to circumvent undecidability have been considered. One was to restrict to local specifications, specifying the desired behavior of each automaton in the network separately. Unfortunately, this does not extend the class of decidable architectures substantially [23]. A further-going proposal was to consider only input-output specifications. A characterization, still very restrictive, of decidable architectures for this case is given in [13].

The asynchronous (Zielonka automata) model was proposed as a reaction to these negative results [12]. The main hope was that causal memory helps to prevent undecidability arising from partial information, since the synchronization of processes in this model makes them share information. Causal memory indeed allowed to get new decidable cases: co-graph action alphabets [12], connectedly communicating systems [24], and tree architectures [14, 25]. There is also a weaker condition covering these three cases [16]. This line of research suffered however from a very recent result showing undecidability in the general case [17].

Distributed synthesis in the Petri net model, Petri games, has been proposed recently in [10]. The idea is that some tokens are controlled by the system and some by the environment. Once again causal memory is used. Without restrictions this model is inter-reducible with the asynchronous automata model [3], hence the undecidability result [17] applies. The problem is EXPTIME-complete for one environment token and arbitrary many system tokens [10]. This case stays decidable even for global safety specifications, such as deadlock, but undecidable in general [9]. As a way to circumvent the undecidability, bounded synthesis has been considered in [8, 18], where the bound on the size of the resulting controller is fixed in advance. The approach is implemented in the tool ADAMSYNT [15].

The control formulation of the synthesis problem comes from the control theory community [28]. It does not require to talk about a specification formalism, while retaining most useful aspects of the problem. A frequently considered control objective is avoidance of undesirable states. In the distributed context, deadlock avoidance looks like an obvious candidate, since it is one of the most basic desirable properties. The survey [32] discusses the relation between the distributed control problem and Church synthesis. Some distributed versions of the control problem have been considered, also hitting the undecidability barrier very quickly [29, 31, 30, 1].

We would like to mention two further results that do not fit into the main threads outlined above. In [33] the authors consider a different synthesis problem for distributed systems: they construct a centralized controller for a scheduler that would guarantee absence of deadlocks. This is a very different approach to deadlock avoidance. Another recent work [2] adds a new dimension to distributed synthesis by considering communication errors in a model with synchronous processes that can exchange their causal memory. The authors show decidability of the synthesis problem for 2 processes.

Outline of the paper

In the next section we define systems with locks, strategies, and the control problem. We introduce locally live strategies as well as the 2-lock, exclusive, and nested locking restrictions. This permits to state the main results of the paper. The following three sections consider systems with the 2-lock restriction. First, we briefly give intuitions behind the Σ_2^p-completeness in the general case. Section 4 presents an NP algorithm for the distributed synthesis problem for locally live strategies. Section 5 gives a PTIME algorithm under the exclusive restriction. Next, we consider the nested locking case, and show that the problem is NEXPTIME-complete. Finally, we prove that without any restrictions the synthesis problem for systems with locks is undecidable. The full version can be found at `https://arxiv.org/4272787`.

2 Main definitions and results

A *lock-sharing system* is a distributed system with components (processes) synchronizing over locks. Processes do not communicate, but they synchronize using locks from a global pool. Some transitions of processes are uncontrollable, intuitively the environment decides if such a transition is taken. The goal is to find a local strategy for each process so that the entire system never deadlocks. The strategy can observe only local transitions – it does not see transitions performed by other processes, nor states other processes are in. While the system is finite state, the challenge comes from the locality of strategies. Indeed, the unrestricted problem is undecidable. The main contribution of this work are restrictions that make the problem decidable, and even solvable in PTIME.

In this section we define lock-sharing systems, strategies, and the deadlock avoidance control problem, that is the topic of this paper. We then introduce restrictions on the general problem and state the main decidability and complexity results.

A finite-state *process* p is an automaton $\mathcal{A}_p = (S_p, \Sigma_p, T_p, \delta_p, init_p)$ with a set of locks T_p that it can acquire or release. The transition function $\delta_p : S_p \times \Sigma_p \dot{\to} Op(T_p) \times S_p$ associates with a state from S_p and an action from Σ_p an operation on some lock and a new state; it is a partial function. The lock operations are acquire ($\mathtt{acq}_t$) or release ($\mathtt{rel}_t$) some lock t from T_p, or do nothing: $Op(T_p) = \{\mathtt{acq}_t, \mathtt{rel}_t \mid t \in T_p\} \cup \{nop\}$. Figure 1 gives an example.

A *local configuration* of process p is a state from S_p together with the locks p currently owns: $(s, B) \in S_p \times 2^{T_p}$. The initial configuration of p is $(init_p, \emptyset)$, namely the initial state with no locks. A transition between configurations $(s, B) \xrightarrow{a,op} (s', B')$ exists when $\delta_p(s, a) = (op, s')$ and one of the following holds:

- $op = nop$ and $B = B'$;
- $op = \mathtt{acq}_t$, $t \notin B$ and $B' = B \cup \{t\}$;
- $op = \mathtt{rel}_t$, $t \in B$, and $B' = B \setminus \{t\}$.

A *local run* $(a_1, op_1)(a_2, op_2) \cdots$ of $\mathcal{A}_p$ is a finite or infinite sequence over $\Sigma_p \times Op(T_p)$ such that there exists a sequence of configurations $(init_p, \emptyset) = (s_0, B_0) \xrightarrow{(a_1,op_1)}_p (s_1, B_1) \xrightarrow{(a_2,op_2)}_p \cdots$ While the run is determined by the sequence of actions, we prefer to make lock operations explicit. We write $Runs_p$ for the set of runs of $\mathcal{A}_p$.

A *lock-sharing system* $\mathcal{S} = ((\mathcal{A}_p)_{p \in Proc}, \Sigma^s, \Sigma^e, T)$ is a set of processes together with a partition of actions between *controllable* and *uncontrollable* actions, and a set T of locks. We have $T = \bigcup_{p \in Proc} T_p$, for the set of all locks. Controllable and uncontrollable actions belong to the system and to the environment, respectively. We write $\Sigma = \bigcup_{p \in Proc} \Sigma_p$ for the set of actions of all processes and require that (Σ^s, Σ^e) partitions Σ. The sets of states and action alphabets of processes should be disjoint: $S_p \cap S_q = \emptyset$ and $\Sigma_p \cap \Sigma_q = \emptyset$ for $p \neq q$. The sets of locks are not disjoint, in general, since processes may share locks.

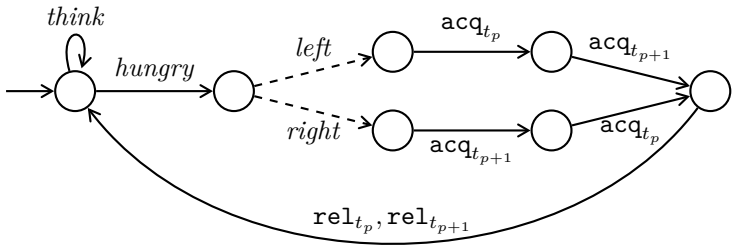


Figure 1 A dining philosopher p. Dashed transitions are controllable.

▶ **Example 1.** The dining philosophers problem can be formulated as control problem for a lock-sharing system $\mathcal{S} = ((\mathcal{A}_p)_{p\in Proc}, \Sigma^s, \Sigma^e, T)$. We set $Proc = \{1, \dots, n\}$ and $T = \{t_1, \dots, t_n\}$ as the set of locks. For every process $p \in Proc$, process $\mathcal{A}_p$ is as in Figure 1, with the convention that $t_{n+1} = t_1$. Actions in Σ^s are marked by dashed arrows. These are controllable actions. The remaining actions are in Σ^e. Once the environment makes a philosopher p hungry, she has to get both the left (t_p) and the right (t_{p+1}) fork to eat. She may however choose the order in which she takes them; actions *left* and *right* are controllable.

A *global configuration* of $\mathcal{S}$ is a tuple of local configurations $C = (s_p, B_p)_{p\in Proc}$ provided the sets B_p are pairwise disjoint: $B_p \cap B_q = \emptyset$ for $p \neq q$. This is because a lock can be taken by at most one process at a time. The initial configuration is the tuple of initial configurations of all processes.

Such systems are *asynchronous*, with transitions between two configurations done by a single process: $C \xrightarrow{(p,a,op)} C'$ if $(s_p, B_p) \xrightarrow{(a,op)}_p (s'_p, B'_p)$ and $(s_q, B_q) = (s'_q, B'_q)$ for every $q \neq p$. A global run is a sequence of transitions between global configurations. Since our systems are deterministic we usually identify a global run by the sequence of transition labels. A global run w *determines a local run* of each process: $w|_p$ is the subsequence of p's actions in w.

A *control strategy* for a lock-sharing system is a tuple of local strategies, one for each process: $\sigma = (\sigma_p)_{p\in Proc}$. A *local strategy* σ_p says which actions p can take depending on a local run so far: $\sigma_p : Runs_p \to 2^{\Sigma_p}$, provided $\Sigma^e \cap \Sigma_p \subseteq \sigma_p(u)$, for every $u \in Runs_p$. This requirement says that a strategy cannot block environment actions.

A local run u of a system *respects* σ_p if for every non-empty prefix $v(a, op)$ of u, we have $a \in \sigma_p(v)$. Observe that local runs are affected only by the local strategy. A global run w respects σ if for every process p, the local run $w|_p$ respects σ_p. We often say just σ-run, instead of "run respecting σ".

As an example consider the system for two philosophers from Example 1. Suppose that both local strategies always say to take the *left* transition. So $hungry^1, left^1, \mathtt{acq}^1_{t_1}, \mathtt{acq}^1_{t_2}$ is a local run of process 1 respecting the strategy; similarly $hungry^2, left^2, \mathtt{acq}^2_{t_2}, \mathtt{acq}^2_{t_1}$ for process 2. (We use superscripts to indicate the process doing an action.) The global run $hungry^1, hungry^2, left^1, left^2, \mathtt{acq}^1_{t_1}, \mathtt{acq}^2_{t_2}$ respects the strategy and blocks, since each philosopher needs a lock the other one owns.

▶ **Definition 2** (Deadlock avoidance control problem). *A σ-run w* leads to a deadlock in *σ if w cannot be prolonged to a σ-run. A control strategy σ is* winning *if no σ-run leads to a deadlock in σ. The* deadlock avoidance control problem *is to decide if for a given system there is some winning control strategy.*

In this work we consider several variants of the deadlock avoidance control problem. Maybe surprisingly, in order to get more efficient algorithms we need to exclude strategies that can block a process by itself:

▶ **Definition 3** (Locally live strategy). *A local strategy σ_p for process p is* locally live *if every local σ_p-run u can be prolonged to a σ_p-run: there is some $b \in \Sigma_p$ such that ub is a local run respecting σ_p. A strategy σ is* locally live *if every local strategy is so.*

In other words, a locally live strategy guarantees that a process does not block if it runs alone. Coming back to Example 1: a strategy always offering one of the *left* or *right* actions is locally live. A strategy that offers none of the two is not. Observe that blocking one process after the hungry action is a very efficient strategy to avoid a deadlock, but it is not the intended one. This is why we consider locally live to be a desirable property rather than a restriction.

Note that being locally live is not exactly equivalent to a strategy always proposing at least one outgoing transition. In our semantics, a process blocks if it tries to acquire a lock that it already owns, or to release a lock it does not own. But it becomes equivalent thanks to the following remark:

▶ Remark 4. We can assume that each process keeps track in its state which locks it owns. Note that this assumption does not compromise the complexity results when the number of locks a process can access is fixed. We will not use this assumption in Section 6, where a process can access arbitrarily many locks (in nested fashion).

Without any restrictions our synthesis problem is undecidable.

▶ **Theorem 5.** *The deadlock avoidance control problem for lock-sharing systems is undecidable. It remains so when restricted to locally live strategies.*

We propose two cases when the control problem becomes decidable. The two are defined by restricting the usage of locks.

▶ **Definition 6** (2LSS). *A process $\mathcal{A}_p = (S_p, \Sigma_p, T_p, \delta_p, init_p)$* uses two locks *if $|T_p| = 2$. A system $\mathcal{S} = ((\mathcal{A}_p)_{p \in Proc}, \Sigma^s, \Sigma^e, T)$ is 2LSS if every process uses two locks.*

Note that in the above definition we do not bound the total number of locks in the system, just the number of locks per process. The process from Figure 1 is 2LSS. Our first main result says that the control problem is decidable for 2LSS.

▶ **Theorem 7.** *The deadlock avoidance control problem for 2LSS is Σ_2^p-complete.*

For the lower bound we use strategies that take a lock and then block. This does not look like a very desired behavior, and this is the reason for introducing the concept of locally live strategies. The second main result says that restricting to locally live strategies helps.

▶ **Theorem 8.** *The deadlock avoidance control problem for 2LSS is in* NP *when strategies are required to be locally live.*

We do not know if the above problem is in Ptime. We can get a Ptime algorithm under one more assumption.

▶ **Definition 9** (Exclusive systems). *A process p is* exclusive *if for every state $s \in S_p$: if s has an outgoing transition with some $\mathtt{acq}_t$ operation then all outgoing transitions of s have the same $\mathtt{acq}_t$ operation. A system is exclusive if all its processes are.*

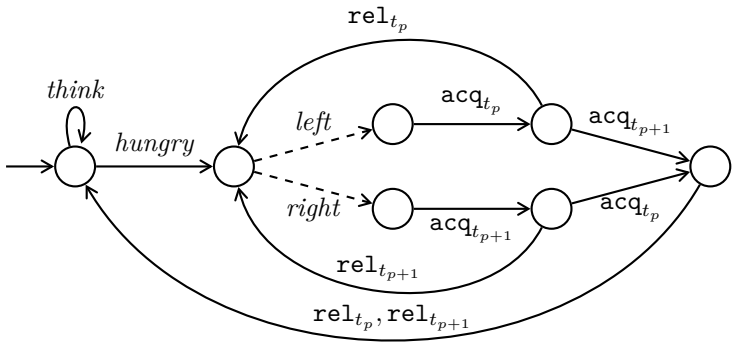


Figure 2 A flexible philosopher p. She can release a fork if the other fork is not available.

▶ **Example 10.** The process from Figure 1 is exclusive, while the one from Figure 2 is not. The latter has a state with one $\texttt{acq}_{t_{p+1}}$ and one $\texttt{rel}_{t_p}$ outgoing transition. Observe that in this state the process cannot block, and has the possibility to take a lock at the same time. Exclusive systems do not have such a possibility, so their analysis is much easier.

▶ **Theorem 11.** *The deadlock avoidance control problem for exclusive 2LSS is in* PTIME*, when strategies are required to be locally live.*

Without local liveness, the problem stays Σ_2^p-hard for exclusive 2LSS. Our last result uses a classical restriction on the usage of locks:

▶ **Definition 12** (Nested-locking)**.** *A local run is* nested-locking *if the order of acquiring and releasing locks in the run respects a stack discipline, i.e., the only lock a process can release is the last one it acquired. A local strategy is nested-locking if all local runs respecting the strategy are nested-locking. A strategy is nested-locking if all local strategies are nested-locking.*

The process from Figure 1 is nested-locking, while the one from Figure 2 is not.

▶ **Theorem 13.** *The deadlock avoidance control problem is* NEXPTIME*-complete when strategies are required to be nested-locking.*

3 Two locks per process

We give some intuitions as to why the deadlock avoidance problem for 2LSS is Σ_2^p-complete (Theorem 7).

When every process uses only two locks there are only few patterns of local lock usage that are relevant for deadlocks. A finite local run u of process p using locks t_1, t_2 can be of one of the following four types:

- p owns both locks at the end of u;
- p owns no lock at the end of u;
- p owns only one lock, say t_1, at the end of u, and the last lock operation of u is $\texttt{acq}_{t_1}$;
- p owns only one lock, say t_1, at the end of u, and the last lock operation of u is $\texttt{rel}_{t_2}$.

A *pattern* of a run is its type, and the set of available actions at the end. If a run reaches a deadlock then the only available actions are to acquire locks owned by other processes.

We fix a 2LSS $(\{\mathcal{A}_p\}_{p\in Proc}, \Sigma^s, \Sigma^e, T)$ over the set of processes *Proc*. We assume that it satisfies Remark 4.

Given a strategy $\sigma = (\sigma_p)_{p \in Proc}$, we call a local σ-run *risky* if it ends in a state from which every outgoing action allowed by σ acquires some lock (this includes states with no outgoing transition). A local σ-run is *neutral* if it ends in a configuration (s, B) with $B = \emptyset$.

▶ **Definition 14.** *We define the* pattern *of a* ***risky*** σ_p*-run* u_p *as follows. Let* T_{owns} *be the set of locks that* p *owns after executing* u_p *and* T_{blocks} *the set of locks that outgoing transitions allowed by* σ_p *after* u_p *need to acquire.*

The pattern of u_p *is the tuple* $(T_{owns}, T_{blocks}, ord)$ *with:*

- *If* u_p *is of the form* $u_1(a, \mathtt{acq}_{t_1})u_2(b, \mathtt{rel}_{t_2})u_3$ *with no action on* t_1 *in* u_2 *and no action on either* t_1 *or* t_2 *in* u_3 *then* $ord = (t_1, t_2)$*.*
- *Otherwise* $ord = \bot$*.*

Note that in light of Remark 4, T_{owns} *and* T_{blocks} *are necessarily disjoint. Furthermore if* ord *is of the form* (t_1, t_2) *then* $T_{owns} = \{t_1\}$*, and either* $T_{blocks} = \emptyset$ *or* $T_{blocks} = \{t_2\}$*.*

A strategy $\sigma = (\sigma_p)_{p \in Proc}$ respects *a family of sets of patterns* $(Patt_p)_{p \in Proc}$ *if for all* $p \in Proc$*, the patterns of all risky* σ_p*-runs belong to* $Patt_p$*.*

In this definition, T_{owns} and T_{blocks} serve as witnesses of deadlock configurations, in which all required locks are owned by another process, and no lock is owned by two different processes. Further, the *ord* component indicates the fourth case described before the definition.

Our key result in this part is Lemma 15. It gives simple, necessary and sufficient, conditions on the family of patterns of local σ-runs $(u_p)_{p \in Proc}$ that lead to a deadlock under a suitable scheduling. The difficulty is to verify if there exists a global run which is a combination of those local runs. For that, all processes must own disjoint sets of locks at the end. The rest can be inferred from the types of runs listed above.

We describe how to schedule local runs into a global one depending on the four types listed before Definition 14.

- In the first case we can assume that p's run is scheduled at the end of the global run, as it ends up keeping both locks anyway, so no other process will use them after p.
- In the second case, we can assume that p's run is scheduled at the beginning of the global run, as it is neutral.
- In the third case, we can split p's run in two parts: a first, neutral part which can be scheduled at the beginning, and a second part in which p acquires t_1 and there is no lock operation afterwards. The second part can be scheduled at the end, because no other process will use t_1 after p.
- In the final case, p acquires t_1, never releases it but later uses t_2. This can be a problem if for instance another process does the same with t_1 and t_2 reversed. The first process that takes its first lock would prevent the other from finishing its local run. We express these constraints by requiring the existence of a global order in which process take locks without releasing them.

▶ **Lemma 15.** *Let* $\sigma = (\sigma_p)_{p \in Proc}$ *be a control strategy. For all* p *let* $Patt_p$ *be the set of patterns of local risky* σ_p*-runs of* p*. The control strategy* σ *is* ***not*** *winning if and only if there exists for each* p *a pattern* $(T^p_{owns}, T^p_{blocks}, ord_p) \in Patt_p$ *such that:*

- $\bigcup_{p \in Proc} T^p_{blocks} \subseteq \bigcup_{p \in Proc} T^p_{owns}$*,*
- *the sets* T^p_{owns} *are pairwise disjoint,*
- *there exists a total order* $\leq$ *on* T *such that for all* p*, if* $ord_p = (t, t')$ *then* $t \leq t'$*.*

Proof. Suppose σ is not winning, let u be a run ending in a deadlock. For each process p let u_p be the corresponding local run. The local run u_p is risky, as otherwise u_p could be extended in a longer run consistent with σ. Thus u_p has a pattern $(T^p_{owns}, T^p_{blocks}, ord_p) \in Patt_p$.

We check that those patterns $(u_p)_{p \in Proc}$ meet the requirements of the lemma. Clearly as we are in a deadlock, all locks that some process wants are taken, hence the first condition is satisfied. Furthermore, no two processes can own the same lock, implying the second condition. Finally, let $\leq$ be a total order on locks given by the order of the last operations on each lock in u: we set $t \leq t'$ iff the last operation on t in u is before the last one on t'. Let p be a process, and suppose ord_p is (t, t'). Then u_p has the form $u_1(a, \mathtt{acq}_t)u_2(b, \mathtt{rel}_{t'})u_3$ with no action on t in u_2 or u_3. Hence, $t \leq t'$.

The other direction is a bit more complicated. Suppose that for each p there is a pattern $(T^p_{owns}, T^p_{blocks}, ord_p) \in Patt_p$ such that those patterns satisfy all three conditions of the lemma. Let $\leq$ be a suitable total order on locks for the third condition, and let $<$ be its strict part. For every p there exists a risky local run u_p yielding the chosen pattern for p.

We start by executing all neutral runs u_p one by one in some order. All locks are free after these executions.

For all p such that $T^p_{owns} = \{t\}$ and $ord_p = \bot$, we can decompose u_p as $u_1(a, \mathtt{acq}_t)u_2$ with no action on locks in u_2. We execute all runs u_1, which are neutral and thus leave all locks free after execution.

Finally, we execute all u_p such that $ord_p \neq \bot$ in increasing order on the first component of ord_p according to $\leq$. For all such p, let $(t, t') = ord_p$, so we have $T^p_{owns} = \{t\}$ and $t < t'$. As all T^p_{owns} are disjoint, before executing u_p all locks greater or equal to t according to $\leq$ are free. In particular, t and t' are free, thus we can execute u_p. In the end all locks are free except the ones belonging to T^p_{owns} for those processes p.

Now we execute the remaining part of the u_p with $T^p_{owns} = \{t\}$ and $ord_p = \bot$ (referred to as $(a, \mathtt{acq}_t)u_2$ before). Those runs do not contain any action on locks besides the first acquire. As all T^p_{owns} are disjoint, the locks they acquire are free, hence all those runs can be executed.

The remaining runs are the ones such that $T^p_{owns} = \{t, t'\}$. As all T^p_{owns} are disjoint, both these locks are free, hence u_p can be executed as p can only use these two locks.

We have combined all local runs into one global run reaching a configuration where all processes have to acquire a lock from $\bigcup_{p \in Proc} T^p_{blocks}$ to keep running, and all locks in $\bigcup_{p \in Proc} T^p_{owns}$ are taken. As $\bigcup_{p \in Proc} T^p_{blocks} \subseteq \bigcup_{p \in Proc} T^p_{owns}$, we have reached a deadlock. ◀

The algorithm for Theorem 7 proceeds in four phases:

- guess a set of patterns $Patt_p$, one for each process p,
- check that there are local strategies σ_p such that the patterns of all runs belong to $Patt_p$,
- let the adversary guess a pattern in each $Patt_p$,
- check whether those patterns satisfy the conditions of Lemma 15.

The alternation between guessing and adversarial guessing yields a Σ^p_2 algorithm.

The lower bound is obtained by a reduction from $\exists\forall$-SAT. The system controls existential variables, the environment controls universal ones. There are two locks for each variable, acquiring one of them is interpreted as choosing the value of the variable. Note that this construction relies on processes that take a lock and then block on their own in states with no outgoing transitions. In the following section we will forbid such unnatural behavior by considering only locally live strategies.

We use some extra processes to enforce that the system wins if and only if the valuation given by the choices of the two players satisfies the SAT formula. The interesting part is that even though it looks like the guessing values of variables is done concurrently by the system and the environment, the whole setting enforces a $\exists\forall$ dependency.

4 Two locks per process with locally live strategies

We describe how to solve the control problem for 2LSS and locally live strategies in NP, as stated in Theorem 8.

We fix a 2LSS satisfying the assumption discussed in Remark 4. We will show that the relevant information about a strategy σ can be formalized as a finite lock graph G_σ and a lockset family $Locks_\sigma$; the latter is a family of sets of sets of locks (see definitions below). This information is very similar to the one described by patterns in the previous section. As we work with locally live strategies, the set of possible patterns of local runs is more restricted and we can view this more conveniently as a graph.

Our algorithm first guesses an abstract lock graph G and lockset family $Locks$. Then it performs two checks:

Step 1 check if there is some strategy σ with $G = G_\sigma$ and $Locks = Locks_\sigma$, and

Step 2 check if there is no deadlock scheme for G and $Locks$ (see Definition 21 below).

A deadlock scheme is some kind of forbidden situation. It is easy to get a co-NP algorithm for the second step: just guess the scheme and check that it has the right shape. The challenge is to do this in PTIME. This is necessary if we want to get an NP algorithm.

We introduce now some notions in order to define G_σ and $Locks_\sigma$ conveniently. Consider a local run u of a process p:

$$(init_p, \emptyset) = (s_0, B_0) \xrightarrow{(a_1, op_1)}_p (s_1, B_1) \cdots \xrightarrow{(a_i, op_i)}_p (s_i, B_i) \ .$$

We say that u *has set of locks* B if $B = B_i$. A σ_p-run u is B*-locked* by the local strategy σ_p if every transition in $\sigma_p(u)$ has as operation $\mathtt{acq}_t$ for some $t \in B$. Process p is B*-lockable* by σ_p if it has a neutral, B-locked σ_p-run.

The intuition is that in order to get a deadlock, a B-lockable process can be scheduled first. It can do a run leading to a state where it requires some of the locks in B without holding any locks. So, the process will be blocked if we ensure that all locks in B are already taken. For example, consider the process in Figure 1. The run *hungry*, *left* is $\{t_p\}$-locked, as the unique next action is $\mathtt{acq}_{t_p}$. The process is $\{t_p\}$-lockable by σ_p if e.g. σ_p always chooses the *left* action. Indeed, in this case the run *hungry*, *left* is a neutral σ_p-run, which is $\{t_p\}$-locked. Process p is not $\{t_{p+1}\}$-lockable by a strategy σ_p choosing always the *left* action, as there is no neutral σ_p-run leading to $\mathtt{acq}_{t_{p+1}}$.

▶ **Definition 16** (Lockset family $Locks_\sigma$)**.** *A* lockset *for a local strategy σ_p is a set $L_p \subseteq 2^{T_p}$ of sets B such that p is B-lockable by σ_p. A* lockset family *for σ is $Locks_\sigma = (L_p)_{p \in Proc}$.*

▶ **Definition 17** (Lock graph G_σ)**.** *For a strategy σ, a* lock graph $G_\sigma = \langle T, E_\sigma \rangle$ *has an edge $t_1 \xrightarrow{p} t_2$ whenever there is some σ_p-run u of p that has $\{t_1\}$ and is $\{t_2\}$-locked. If there is such a run u where the last lock operation in u is $\mathtt{acq}_{t_1}$ then the edge is called* green*, and otherwise it is called* blue*.*

We will say that σ allows *a blue edge $t_1 \overset{p}{\hookrightarrow} t_2$ or a green edge $t_1 \overset{p}{\mapsto} t_2$. We write $t_1 \xrightarrow{p} t_2$ when the color of the edge is irrelevant.*

For example, a strategy choosing the *left* action in Figure 1 yields the green edge $t_p \overset{p}{\mapsto} t_{p+1}$. Lockset families say on which sets of locks each process can block while not holding any lock. An edge $t_1 \xrightarrow{p} t_2$ in the lock graph corresponds to a run of p where P owns lock t_1 (the source of the edge) and waits for the other lock t_2 (the target of the edge).

A lockset represents a run of the second type in the previous section, a green edge a run of the third type, and a blue edge a run of the fourth type with no similar run of the third type. The first type cannot appear in a deadlock when strategies are locally live, as processes always have an available action.

Since we have assumed nothing about how strategies are given, it is not clear how to compute G_σ. Instead of restricting to, say, finite memory strategies, we will work with arbitrary lock graphs and lockset families. This is possible thanks to Lemma 19 below, that allows to check if a graph is the lock graph of some strategy. For this we need to define lockset families and lock graphs abstractly. Notice that the size of both these objects is bounded, as the set of locks per process is fixed for 2LSS.

▶ **Definition 18.** *A* lockset family *is a tuple of sets of locks indexed by processes* $(L_p)_{p\in Proc}$*, with* $L_p \subseteq 2^{T_p}$*. A* lock graph *is an edge-labeled graph* $G = \langle T, E \subseteq T\times Proc\times\{blue, green\}\times T\rangle$ *where nodes are locks from the set* T *and every edge is labeled by a process and a color. A cycle in* G *is called* proper *if all its edges are labeled by different processes. It is denoted as* green *if it contains at least* ***one*** *green edge; otherwise, so if* ***all*** *edges are blue, it is denoted* blue.

At this point we have enough notions to carry out the first step on page 10.

▶ **Lemma 19.** *Given a lock graph* G *and a lockset family Locks, it is decidable in* Ptime *if there is a locally live strategy* σ *such that* $G = G_\sigma$ *and* $Locks_\sigma = Locks$*.*

The proof is by reduction to model-checking a fixed-size MSOL formula over a given regular tree. For every process p we need to check if there is a local strategy σ_p satisfying the conditions imposed by G and $Locks = (L_p)_{p\in Proc}$. Consider the regular tree of all local runs of process p. The formula says that there is a strategy tree inside this regular tree such that L_p contains exactly those sets B such that the subtree has some neutral, B-locked path; and for every edge in G labelled by p there is a path of the required shape in the subtree. This can be expressed by an MSOL formula of constant size, as the process uses only 2 locks. From the MSOL formula we get a tree automaton of constant size. The emptiness check of its product with the tree automaton accepting the unfolding of the automaton $\mathcal{A}_p$ can be done in Ptime.

In the rest of the section we discuss the second step. We first define a Z-deadlock scheme for some set Z of locks. Intuitively, this is a situation showing that there is a run blocking all locks in Z. Then a deadlock scheme is a Z-deadlock scheme for some Z big enough to block all processes.

▶ **Definition 20** (Z-deadlock scheme)**.** *Let* $G = \langle T, E\rangle$ *be a lock graph,* $Locks = (L_p)_{p\in Proc}$ *a lockset family, and* Z *a set of locks. We define* $Proc_Z$ *as the set of processes whose both accessible locks are in* Z*,* $Proc_Z = \{p \in Proc : T_p \subseteq Z\}$*. A* Z-deadlock scheme *is a function* $ds_Z : Proc_Z \to E \cup \{\bot\}$ *such that:*

- *For all* $p \in Proc_Z$*, if* $ds_Z(p) \neq \bot$ *then* $ds_Z(p)$ *is an edge of* G *labeled by* p*.*
- *If* $p \in Proc_Z$ *and* $L_p = \emptyset$ *then* $ds_Z(p) \neq \bot$*.*
- *For all* $t \in Z$ *there exists a unique* $p \in Proc_Z$ *such that* $ds_Z(p)$ *is an outgoing edge from* t*.*
- *The subgraph of* G*, restricted to* $ds_Z(Proc_Z)$ *does not contain any blue cycle.*

The main point of this definition is that for every lock in Z there is an outgoing edge in ds_Z. Intuitively, it means that we have a run where every lock from Z is taken, and every process in $Proc_Z$ requires a lock from Z.

▶ **Definition 21** (Deadlock scheme)**.** *A* deadlock scheme *for* G *and* $Locks = (L_p)_{p\in Proc}$ *is a* Z*-deadlock scheme such that for every process* $p \in Proc \setminus Proc_Z$ *there is* $B \in L_p$ *with* $B \subseteq Z$*.*

Thus a deadlock scheme represents a situation where all processes are blocked, since every process not in $Proc_Z$ can be brought into a state where it needs a lock from Z, but all these locks are taken.

The next lemma says that the absence of deadlock schemes characterizes winning strategies. We could reuse the patterns defined above to obtain a shorter proof but we prefer to give a slightly longer but elementary one.

▶ **Lemma 22.** *A locally live control strategy σ is winning if and only if there is no deadlock scheme for its lock graph G_σ and its lockset family $Locks_\sigma$.*

Proof. Suppose σ is not winning. Then there exists a global σ-run u leading to a deadlock. As a consequence, in the deadlock configuration all processes must be trying to acquire some lock that is already taken.

We then construct a deadlock scheme (BT, ds) as follows. Let BT be the set of locks taken in the deadlock configuration, and for all $p \in Proc$, define $ds(p)$ as:

- $\bot$ if p does not own any lock in the deadlock configuration,
- $t_1 \xrightarrow{p} t_2$ if p owns t_1 and is trying to acquire t_2 in the deadlock configuration (the color of the edge is determined by the run, it is irrelevant for the argument).

Clearly for all $p \in Proc$ the value $ds(p)$ is either $\bot$ or a p-labeled edge of the lock graph G_σ.

Suppose $ds(p) = \bot$, and let t_1, t_2 be the two locks accessible by p. As the final configuration is a deadlock, all actions allowed by σ_p are necessarily $\texttt{acq}_{t_1}$ or $\texttt{acq}_{t_2}$. So p is $\{t_1, t_2\}$-lockable. Furthermore, as we are in a deadlock, the lock(s) blocking p are in BT (if they were free, p would be able to advance), therefore p is BT-lockable.

For every $t \in BT$, there is a process p holding t in the final configuration. As we are in a deadlock, p is trying to acquire its other accessible lock t' (recall that the definition of control strategy demands that at least one action be available to each process at all times). Thus $ds(p)$ is an edge from t to t'. Furthermore t' cannot be free as we are in a deadlock, thus $t' \in BT$. There are no other outgoing edges from t as no other process can hold t while p does.

Finally let $t_1 \xrightarrow{p_1} t_2 \cdots \xrightarrow{p_k} t_{k+1}$ be a cycle with $t_1 = t_{k+1}$ in the subgraph of G_σ restricted to BT and $ds(Proc)$. One of the locks t_i was the last lock taken in the run u (say by process p_i). We show now by contradiction that the edge $t_i \xrightarrow{p_i} t_{i+1}$ is green. If p_i would have released t_{i+1} after the last $\texttt{acq}_{t_i}$ in u, then p_{i+1} would have done its last $\texttt{acq}_{t_{i+1}}$ later, a contradiction. The subgraph of G_σ restricted to BT and $ds(Proc)$ has therefore no blue cycles, therefore (BT, ds) is a deadlock scheme.

For the other direction, suppose we have a deadlock scheme (BT, ds) for the lock graph G_σ. As $(BT, ds(Proc))$ does not contain a blue cycle, we can pick a total order $\leq$ on locks such that for all blue edges $t_1 \overset{p}{\hookrightarrow} t_2 \in ds(Proc)$, we have $t_1 \leq t_2$.

By definition of the lock graph, for each process $p \in Proc$ we can take a local run u_p of $\mathcal{A}_p$ respecting σ with the following properties.

- If $ds(p) = \bot$ then p is BT-lockable. So there exists a neutral run u_p leading to a state where all outgoing transitions require locks from BT.
- If $ds(p) = t_p^1 \xrightarrow{p} t_p^2$ then there is u_p of the form $u_p^1(a, \texttt{acq}_{t_p^1})u_p^2(a', \texttt{acq}_{t_p^2})$ without $\texttt{rel}_{t_p^1}$ transition in u_p^2. Moreover if $ds(p)$ is green then we know that there is no $\texttt{rel}_{t_p^2}$ transition in u_p^2.

We now combine these runs to get a run respecting σ ending in a deadlock configuration. For each process p such that $ds(p) = \bot$, execute the local run u_p. Since u_p is neutral, all locks are available after executing it. The only possible actions of p after this run are to acquire some locks from BT.

Next, for every process p such that $ds(p)$ is a green edge, execute the local run u_p^1. This is also a neutral run. After this run p is in a state where σ_p allows to take lock t_p^1, but p does not own any lock.

Next, in increasing order according to $\leq$, for every lock t with an outgoing blue edge $ds(p) = t \overset{p}{\hookrightarrow} t'$ execute the run u_p, except for the last $\mathtt{acq}_{t'}$ action. After this run lock t is taken by p, and all actions allowed by σ_p are $\mathtt{acq}_{t'}$ actions. Since there is only one outgoing edge from every lock, and since we are respecting the order $\leq$, both t and t' are free before executing that run. Hence it is possible to execute this run.

Finally, we come back to processes p such that $ds(p)$ is a green edge. For every such process we execute $\mathtt{acq}_{t_p^1}$ followed by u_p^2. This is possible because t_p^1 is free as there is a unique outgoing edge from t_p^1. After executing these runs every process p with $ds(p) \neq \bot$ is in a state when the only possible action is $\mathtt{acq}_{t_p^2}$.

At this stage all locks that are sources of edges from $ds(Proc)$ are taken. Since every lock in BT is a source of an edge, all locks from BT are taken. Thus no process p with $ds(p) = \bot$ can move as it needs some lock from BT. Similarly, no process p with $ds(p) \neq \bot$ can move, as they need locks pointed by targets of the edges $ds(p)$, and these are in BT too. So we have constructed a run respecting σ and reaching a deadlock. ◀

From now on we concentrate on deciding if there is some deadlock scheme for a given graph G along with a lockset family *Locks*. Our approach will be to repeatedly eliminate edges from G or add locks to Z, and construct a deadlock scheme on Z at the same time.

As a preparatory step we observe that we can almost ignore the lockset family. Examining the definition of Z-deadlock scheme we see that the only information about *Locks* it uses is whether $L_p = \emptyset$ or not. Hence we call a process *solid* if $L_p = \emptyset$, and *fragile* otherwise. The second condition in the definition of Z-deadlock scheme becomes: if $p \in Proc_Z$ is solid then $ds_Z(p) \neq \bot$.

The next lemma gives an important composition principle for deadlock schemes. Suppose we already have a set of "kernel" locks Z on which we know how to construct a Z-deadlock scheme. Then the lemma says that in order to get a deadlock scheme for G it is enough to consider the remaining part $G \setminus Z$.

▶ **Lemma 23.** *Let $Z \subseteq T$ be such that there is no edge labeled by a solid process from a lock of Z to a lock of $T \setminus Z$ in G. Suppose $ds_Z : Proc_Z \to E \cup \{\bot\}$ is a Z-deadlock scheme. Then there is a deadlock scheme for G if and only if there is one equal to ds_Z over $Proc_Z$.*

The rest of the proof is a sequence of stages. We start with $H = G$ and $Z = \emptyset$. At each stage we remove some edges in H or extend Z. This process continues till some obstacle to the existence of a deadlock scheme is found, or till Z is big enough to be a deadlock scheme. We use three invariants:

▶ **Invariant 1.** *G admits a deadlock scheme if and only if H does.*

▶ **Invariant 2.** *There are no edges labeled by a solid process from Z to $T \setminus Z$ in H.*

▶ **Invariant 3.** *There exists a Z-deadlock scheme.*

▶ **Proposition 24.** *There is a polynomial time algorithm to decide if a lock graph G and a lockset family Locks have a deadlock scheme.*

The final argument behind Theorem 8 is as follows. We start by non-deterministically guessing G and *Locks*. These are of polynomial size with respect to the size of the 2LSS. We can check in polynomial time that there exists a strategy σ giving G and *Locks* (Lemma 19). If that is not the case, we reject the input. Otherwise we check if G and *Locks* admit a deadlock scheme (Proposition 24). By Lemma 22, the strategy σ is winning if and only if the check says that there is no deadlock scheme in G and *Locks*.

5 Solving the exclusive case in PTIME

In this section we study exclusive 2LSS. We have shown an NP algorithm for the deadlock avoidance control problem when restricting to locally live strategies. Here we show that the problem is in PTIME if the 2LSS is exclusive (Definition 9). This is possible because the exclusive assumption simplifies the structure of lock graphs, and makes the lockset family unnecessary.

Throughout this section we fix an exclusive 2LSS, call it $\mathcal{S}$. The exclusive property prohibits situations as in Figure 2 where a state has one outgoing $\texttt{acq}_{t_{p+1}}$ transition, and one $\texttt{rel}_{t_p}$ transition. Compared to the previous section we do not need to make a difference between solid and fragile processes. We can even ignore colors on the arrows. This is a consequence of the following two lemmas.

▶ **Lemma 25.** *Let σ be a locally live control strategy and G_σ its lock graph. For all $t_1, t_2 \in T$, if G_σ has a blue edge $t_1 \overset{p}{\hookrightarrow} t_2$ then it has a green edge $t_2 \overset{p}{\mapsto} t_1$.*

▶ **Lemma 26.** *Let σ be a locally live control strategy and G_σ its lock graph. For every edge $t_1 \overset{p}{\to} t_2$ in G, process p is $\{t_1, t_2\}$-lockable.*

Thanks to these simplifications there is a much more direct way of checking if a strategy is winning. Take a locally live strategy σ. Consider a decomposition of G_σ into strongly connected components (SCC). We say that an SCC is a *direct deadlock* if it contains at least two nodes, and:

- either it has an edge that is not a double edge: $t_1 \xrightarrow{p} t_2$ but not $t_1 \xleftarrow{p} t_2$, for some p;
- or all edges in the component are double edges and there is a proper cycle, i.e., all edges are labeled by different processes.

A *deadlock* SCC is a direct deadlock SCC or an SCC that can reach some direct deadlock SCC. Let BT_σ be the set of all the locks appearing in some deadlock SCC. We obtain a simple characterization of winning strategies.

▶ **Proposition 27.** *A strategy σ is winning if and only if there exists a process that is not BT_σ-lockable.*

Building on this result we can give a method to decide if there is a winning strategy in the system $\mathcal{S}$. For every process p and every set of edges between two locks of p we check if there is a local strategy inducing exactly these edges. This can be done in a similar way as Lemma 19. We say that an edge la belled by p is *unavoidable* if all the local strategies σ_p induce this edge. Let $G_\mathcal{S}$ be the graph whose nodes are locks and edges are unavoidable edges.

We calculate a set $BT_\mathcal{S}$ in a similar way as BT_σ in the previous proposition except that we use slightly more general basic SCCs of $G_\mathcal{S}$. A *direct semi-deadlock SCC* is either a direct deadlock SCC or an SCC containing at least two nodes, only double edges, and two locks t_1 and t_2 such that for some process p not inducing a double edge between t_1, t_2 in $G_\mathcal{S}$: every strategy for p induces at least one edge between t_1 and t_2. Then a *semi-deadlock SCC* is an SCC that can reach some direct semi-deadlock SCC, or is itself a direct semi-deadlock SCC.

Let $BT_\mathcal{S}$ be the set of locks appearing in semi-deadlock SCCs of $G_\mathcal{S}$. Theorem 11 follows from the next proposition.

▶ **Proposition 28.** *Let $\mathcal{S}$ be an exclusive 2LSS. There is a winning locally live strategy for the system if and only if there exists a locally live strategy σ_p for some process p preventing it from acquiring any lock from $BT_\mathcal{S}$.*

The algorithm computes $BT_{\mathcal{S}}$, and then checks if for some process p the condition from the proposition holds. This check amounts to solving a safety game on a finite graph – the transition graph of process p.

6 Nested-locking strategies

We switch to another decidable case, where we require that locks are acquired and released in stack-like manner. Our goal is Theorem 13 saying that the deadlock avoidance control problem is NEXPTIME-complete when restricted to nested-locking strategies (cf. Definition 12).

In the context of this section we cannot assume that a process knows which locks it has (cf. Remark 4). In consequence, it is not realistic to require that a strategy is locally live. Yet, the lower bound works also for locally live strategies.

We will use some notions about local runs as defined on page 10.

▶ **Definition 29.** *A* stair decomposition *of a local run u is*

$$u = u_1 \mathtt{acq}_{t_1} u_2 \mathtt{acq}_{t_2} \dots u_k \mathtt{acq}_{t_k} u_{k+1}$$

where in the configuration reached by $u_1 \mathtt{acq}_{t_1} u_2 \mathtt{acq}_{t_2} \dots u_i$ the set of locks held by the process is $\{t_1, \dots, t_{i-1}\}$ for every $i > 0$, and there is no operation on t_i in $u_{i+1} \dots u_{k+1}$. (We omit the actions associated with each operation as they are irrelevant here).

Every nested-locking run has a unique stair decomposition.

Without the locally live assumption we may have runs simply ending because there are no outgoing actions. Recall that given a strategy σ, a *risky σ-run* is a local σ-run ending in a state from which every outgoing action allowed by σ acquires some lock. We define patterns of risky local runs that will serve as witnesses of reachable deadlocks.

▶ **Definition 30.** *Consider a stair decomposition $u_1 \mathtt{acq}_{t_1} u_2 \mathtt{acq}_{t_2} \cdots u_k \mathtt{acq}_{t_k} u_{k+1}$ of a risky σ-run u of a process p. Suppose the run is T_{blocks}-blocked, and let $T_{owns} = \{t_1, \dots, t_k\}$. We associate with u a* stair pattern *$(T_{owns}, T_{blocks}, \preceq)$, where $\preceq$ is the smallest partial order on the set T_p of locks of p satisfying: for all i, for all $t \in T_p$, if the last operation on t in the run is after the last $\mathtt{acq}_{t_i}$ then $t_i \preceq t$. A* behavior *of σ is a family of sets of stair patterns $(\mathcal{P}_p)_{p \in Proc}$, where $\mathcal{P}_p$ is the set of stair patterns of local risky σ-runs of p.*

Similarly to Lemma 22 we can show that the family of patterns for a strategy determines if it is winning.

▶ **Lemma 31.** *A nested-locking control strategy σ with behavior $(\mathcal{P}_p)_{p \in Proc}$ is **not** winning if and only if for every $p \in Proc$ there is a stair pattern $(T^p_{owns}, T^p_{blocks}, \preceq^p) \in \mathcal{P}_p$ such that:*

- $\bigcup_{p \in Proc} T^p_{blocks} \subseteq \bigcup_{p \in Proc} T^p_{owns}$,
- *the sets T^p_{owns} are pairwise disjoint,*
- *there exists a total order $\preceq$, on the set of all locks T, compatible with all $\preceq^p$.*

Similarly to Lemma 19 we can check if there is a strategy whose set of patterns has only patterns from a given family. Observe that the depth of nesting is bounded by the number of locks.

▶ **Lemma 32.** *Given a lock-sharing system $((\mathcal{A}_p)_{p \in Proc}, \Sigma^s, \Sigma^e, T)$, a process $p \in Proc$ and a set of patterns $\mathcal{P}_p$, we can check in polynomial time in $|\mathcal{A}_p|$ and $2^{|T|}$ whether there exists a nested-locking local strategy σ_p with set of patterns included in $\mathcal{P}_p$.*

▶ **Proposition 33.** *The deadlock avoidance control problem is decidable for lock-sharing systems with nested-locking strategies in non-deterministic exponential time.*

Proof. The decision procedure guesses a set of patterns $\mathcal{P}_p$ for each process p, of size at most $2^{2|T|}|T|! \leq 2^{O(|T|\log(|T|)}$. Then it checks if there exist local strategies yielding subsets of those sets of patterns. This takes exponential time by Lemma 32. If the result is negative then the procedure rejects. Otherwise, it checks if some condition from Lemma 31 does not hold. It it finds one then it accepts, otherwise it rejects.

Clearly, if there is a winning nested-locking strategy then the procedure can accept by guessing the family of patterns corresponding to this strategy. For this family the check from Lemma 32 does not fail, and one of the conditions of Lemma 31 must be violated.

Conversely, if the decision procedure concludes that there exists a winning strategy, then let $(\mathcal{P}_p)_{p\in Proc}$ be the guessed family of sets of patterns. We know that there exists a strategy σ with behaviors $(\mathcal{P}'_p)_{p\in Proc}$ such that $\mathcal{P}'_p \subseteq \mathcal{P}_p$ for all $p \in Proc$. Furthermore, as there are no patterns in $(\mathcal{P}_p)_{p\in Proc}$ satisfying the requirements of Lemma 31, there cannot be any in the $\mathcal{P}'_p$ either. Hence σ is a winning strategy. ◀

7 Undecidability for unrestricted lock-sharing systems

In this section we show that the deadlock avoidance control problem for lock-sharing systems is undecidable for three processes with a fixed number of locks. Three locks used in non-nested fashion allow to synchronize two processes in lock-step manner. This is an essential ingredient for the undecidability proof.

We have defined lock-sharing systems so that initially all locks are free. First we show the undecidability result supposing that we are allowed to start with a designated distribution of locks. Later we describe how to implement initial lock distributions using extra locks.

▶ **Lemma 34.** *The control problem for lock-sharing systems with 3 processes, fixed initial configuration and fixed number of locks per process is undecidable.*

The proof uses the usual recipe for the undecidability of distributed synthesis [26, 27]. Two processes P and $\overline{P}$ synchronize with a third process C over a stream of bits chosen by their strategy. The process C is partially controlled by the environment, which selects non-deterministically an interleaving of the two streams and parses the interleaving with a finite automaton. This is enough to get undecidability by a reduction from an infinite Post Correspondence Problem (PCP).

Consider an instance $(\alpha_i, \beta_i)_{i\in I}$ of PCP on the alphabet $\{0, 1\}$. A solution is an infinite sequence $i_1 i_2 \ldots \in I^\omega$ such that $\alpha_{i_1}\alpha_{i_2}\ldots = \beta_{i_1}\beta_{i_2}\ldots$. The two streams sent by P and $\overline{P}$ to C, are $\alpha = \alpha_{i_1} i_1 \alpha_{i_2} i_2 \ldots$ and $\beta = \beta_{j_1} j_1 \beta_{j_2} j_2 \ldots$, resp. With finite memory C can check equality of the two words $(\alpha_{i_1}\alpha_{i_2}\cdots = \beta_{j_1}\beta_{j_2}\ldots)$ or equality of the two index sequences $(i_1 i_2 \ldots = j_1 j_2 \ldots)$. Since P and $\overline{P}$ are not aware of what C does, the streams are fixed by the strategies and do not depend on what C is checking.

The locks used in the proof are $\{c, s_0, s_1, p, \overline{c}, \overline{s}_0, \overline{s}_1, \overline{p}\}$. Process C and P use locks from $\{c, s_0, s_1, p\}$ to synchronize and similarly for C, $\overline{P}$ and $\{\overline{c}, \overline{s}_0, \overline{s}_1, \overline{p}\}$.

It remains to explain the synchronization mechanism. The two processes P and C synchronize over a bit of information, say bit 0, by executing specific finite runs using the locks $\{s_0, c, p\}$ in non-nested fashion. Initially, C owns $\{s_0, c\}$ and P owns $\{p\}$. First, C releases lock s_0 and P acquires it, which we denote as $C \xrightarrow{s_0} P$. Here, P is waiting for C to release s_0, and the two actions $\texttt{rel}_{s_0}$ of C and $\texttt{acq}_{s_0}$ of P are ordered. The rest of the run follows a similar pattern: at each step, one of the processes is waiting to take a lock released by the other process. With the same notation, the run proceeds with $P \xrightarrow{p} C$, and continues until each process owns the same locks it owned at the start: each lock is sent

twice, from its initial owner to the other process, and back. To sum up, the exchange of bit 0 between C and P corresponds to $C \xrightarrow{s_0} P \xrightarrow{p} C \xrightarrow{c} P \xrightarrow{s_0} C \xrightarrow{p} P \xrightarrow{c} C$. In other words, processes C and P respectively perform two local runs:

$$C: \texttt{rel}_{s_0}\,\texttt{acq}_p\,\texttt{rel}_c\,\texttt{acq}_{s_0}\texttt{rel}_p\,\texttt{acq}_c \qquad P: \texttt{acq}_{s_0}\texttt{rel}_p\,\texttt{acq}_c\,\texttt{rel}_{s_0}\texttt{acq}_p\texttt{rel}_c$$

Observe that P and C need to execute these sequences in lock-step manner, as one of the two processes waits for a lock from the other.

In order to synchronize over bit 1, the two processes perform a similar synchronization, using s_1 instead of s_0. The communication between C and $\overline{P}$ is identical, except that it uses locks from $\{\overline{c}, \overline{s}_0, \overline{s}_1, \overline{p}\}$.

In each round, P and C must agree beforehand on a bit they are going to synchronize on, either s_0 or s_1. Otherwise the two processes get blocked, and $\overline{P}$ will get blocked too, as it needs locks held by C. A bit stream between C and P is encoded as a concatenation of such runs, and similarly for $C, \overline{P}$. The content of the two bit streams is chosen by the strategies of $P, \overline{P}, C$. Since the strategy has infinite memory, there is no upper bound on the complexity of the streams. Interestingly, two locks are not enough for two processes to synchronize over a bit stream.

▶ **Lemma 35.** *There is a polynomial-time reduction from the control problem for lock-sharing systems with initial configuration to the control problem where all locks are initially free. The reduction adds $|Proc|$ new locks.*

We sketch the proof idea. Assume that we have pairwise disjoint sets $(I_p)_{p \in Proc}$ of locks, and a lock-sharing system $\mathcal{S}$ in which each process p initially owns exactly the locks in I_p. We build another lock-sharing systems $\mathcal{S}_\emptyset$ that starts with all locks initially free, makes every process acquire all locks in I_p, and then simulates $\mathcal{S}$.

It is important that the initialization phase of $\mathcal{S}_\emptyset$ does not interfere with the simulation of $\mathcal{S}$. We ensure this by using one additional lock k_p per process, called the "key" of p.

For process p, the initialization sequence consists of three steps.

1. First, p takes one by one (in a fixed arbitrary order), all its initial locks in I_p.
2. Second, p takes and releases, one by one (in a fixed arbitrary order) all the keys of the other processes $(k_q)_{q \neq p}$.
3. Finally, p acquires its key k_p and keeps it forever.

After acquiring k_p process p reaches the initial state in $\mathcal{S}$.

In order to prevent the initialization phase to create extra deadlocks, there is a local *nop* loop on every state of the initialization sequence. This way, a deadlock may only occur if all processes have finally completed their initialization sequences. Note that the initialization phase does not interfere with the simulation of $\mathcal{S}$. This is because the exchange of keys guarantees that up to the moment where a process p has completed the initialization in $\mathcal{S}_\emptyset$, no other process has used any lock from I_p.

8 Conclusions

Motivated by a recent undecidability result for distributed control synthesis [17] we have considered a model for which the problem has not been investigated yet. With hindsight it is strange that the well-studied model of lock synchronization has not been considered in the context of distributed synthesis. One reason may be the "non-monotone" nature of the synthesis problem. It is not the case that for a less expressive class of systems the problem is necessarily easier because the controllers get less powerful, too.

The two decidable classes of lock-sharing systems presented here are rather promising. Especially because the low complexity results cover already non-trivial problems. All our algorithms are based on analyzing lock patterns. While in this paper we consider only finite state processes, the same method applies to more complex systems, as long as solving the centralized control problem in the style of Lemma 19 is decidable. This is for example the case for pushdown systems.

There are numerous directions that need to be investigated further. We have focused on deadlock avoidance because this is a central property, and deadlocks are difficult to discover by means of testing or verification. Another option is partial deadlock, where some, but not all, processes are blocked. The concept of Z-deadlock scheme from Definition 20 should help here, but the complexity results may be different. Reachability, and repeated reachability properties need to be investigated, too.

We do not know if the upper bound from Theorem 8 is tight. The algorithm for verifying if there is a deadlock in a given strategy graph, Proposition 24, is already quite complicated, and it is not clear how to proceed when a strategy is not given.

Another research direction is to consider probabilistic controllers. It is well known that there are no symmetric solutions to the dining philosophers problem but there is a randomized one [21, 22]. Symmetric solutions are quite important for resilience issues as it is preferable that every process runs the same code. The Lehmann-Rabin algorithm is essentially the system presented in Figure 2 where the choice between *left* and *right* is made randomly. This is one of the examples where randomized strategies are essential. Distributed synthesis has a potential here because it is even more difficult to construct distributed randomized systems and prove them correct.

References

1 A. Arnold and I. Walukiewicz. Nondeterministic controllers of nondeterministic processes. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 29–52. Amsterdam University Press, 2007.

2 B. Bérard, B. Bollig, P. Bouyer, M. Függer, and N. Sznajder. Synthesis in presence of dynamic links. In J-F. Raskin and D. Bresolin, editors, *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020*, volume 326 of *EPTCS*, pages 33–49, 2020. To appear in Information and Computation. doi:10.4204/EPTCS.326.3.

3 R. Beutner, B. Finkbeiner, and J. Hecking-Harbusch. Translating asynchronous games for distributed synthesis. In W. J. Fokkink and R. van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019*, volume 140 of *LIPIcs*, pages 26:1–26:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.26.

4 K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Trans. Program. Lang. Syst.*, 6(4):632–646, October 1984. doi:10.1145/1780.1804.

5 A. Church. Applications of recursive arithmetic to the problem of cricuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume I, pages 3–50. Cornell Univ., Ithaca, N.Y., 1957.

6 E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer Verlag, 1981.

7 M. D. Ernst, A. Lovato, D. Macedonio, F. Spoto, and J. Thaine. Locking discipline inference and checking. In L. K. Dillon, W. Visser, and L. A. Williams, editors, *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016*, pages 1133–1144. ACM, 2016. doi:10.1145/2884781.2884882.

8 B. Finkbeiner. Bounded synthesis for Petri games. In R. Meyer, A. Platzer, and H. Wehrheim, editors, *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday. Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2015. doi:10.1007/978-3-319-23506-6_15.

9 B. Finkbeiner, M. Gieseking, J. Hecking-Harbusch, and E.-R. Olderog. Global winning conditions in synthesis of distributed systems with causal memory. In F. Manea and A. Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, Virtual Conference*, volume 216 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.20.

10 B. Finkbeiner and E.-R. Olderog. Petri games: Synthesis of distributed systems with causal memory. *Inf. Comput.*, 253:181–203, 2017.

11 B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS) 2005, Proceedings*, pages 321–330. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.53.

12 P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In K. Lodaya and M. Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.

13 P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. *Formal Methods in System Design*, 34(3):215–237, June 2009.

14 B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Asynchronous games over tree architectures. In F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.

15 M. Gieseking, J. Hecking-Harbusch, and A. Yanich. A web interface for Petri nets with transits and Petri games. In J. F. Groote and K. G. Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 381–388. Springer, 2021. doi:10.1007/978-3-030-72013-1_22.

16 H. Gimbert. On the control of asynchronous automata. In S. V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, volume 93 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.30.

17 H. Gimbert. Distributed asynchronous games with causal memory are undecidable. *CoRR*, abs/2110.14768, 2021. Submitted. arXiv:2110.14768.

18 J. Hecking-Harbusch and N. O. Metzger. Efficient trace encodings of bounded synthesis for asynchronous distributed systems. In Y. F. Chen, C. H. Cheng, and J. Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 369–386. Springer, 2019. doi:10.1007/978-3-030-31784-3_22.

19 V. Kahlon and A. Gupta. An automata-theoretic approach for model checking threads for LTL properties. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), Proceedings*, pages 101–110. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.11.

20 O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *16th Annual IEEE Symposium on Logic in Computer Science, Proceedings*, pages 389–398. IEEE Computer Society, 2001. doi:10.1109/LICS.2001.932514.

21 D. Lehmann and M. O. Rabin. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In J. White, R. J. Lipton, and P. C. Goldberg, editors, *Conference Record of the Eighth Annual ACM Symposium on Principles of Programming Languages*, pages 133–138. ACM Press, 1981. doi:10.1145/567532.567547.

22 N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

23 P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2001. doi:10.1007/3-540-48224-5_33.

24 P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In R. Ramanujam and S. Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference*, volume 3821 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2005. doi:10.1007/11590156_16.

25 A. Muscholl and I. Walukiewicz. Distributed synthesis for acyclic architectures. In V. Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014*, volume 29 of *LIPIcs*, pages 639–651. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.639.

26 G. L. Peterson and J. H. Reif. Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science*, pages 348–363. IEEE Computer Society, 1979. doi:10.1109/SFCS.1979.25.

27 A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science*, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

28 P. J.G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(2):81–98, 1989.

29 K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. on Automat. Control*, 37(11):1692–1708, 1992.

30 J. G. Thistle. Undecidability in decentralized supervision. *Systems & Control Letters*, 54(5):503–509, 2005. doi:10.1016/j.sysconle.2004.10.002.

31 S. Tripakis. Undecidable problems in decentralized observation and control for regular languages. *Information Processing Letters*, 90(1):21–28, 2004.

32 I. Walukiewicz. Synthesis with finite automata. In J.-É. Pin, editor, *Handbook of Automata Theory*, pages 1217–1260. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/Automata-2/11.

33 Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, and S. A. Mahlke. The theory of deadlock avoidance via discrete control. In Z. Shao and B. C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 252–263. ACM, 2009. doi:10.1145/1480881.1480913.

Lower Bounds for Unambiguous Automata via Communication Complexity

Mika Göös
EPFL, Lausanne, Switzerland

Stefan Kiefer
University of Oxford, UK

Weiqiang Yuan
EPFL, Lausanne, Switzerland

Abstract

We use results from communication complexity, both new and old ones, to prove lower bounds for unambiguous finite automata (UFAs). We show three results.

1. *Complement:* There is a language L recognised by an n-state UFA such that the complement language $\overline{L}$ requires NFAs with $n^{\tilde{\Omega}(\log n)}$ states. This improves on a lower bound by Raskin.
2. *Union:* There are languages L_1, L_2 recognised by n-state UFAs such that the union $L_1 \cup L_2$ requires UFAs with $n^{\tilde{\Omega}(\log n)}$ states.
3. *Separation:* There is a language L such that both L and $\overline{L}$ are recognised by n-state NFAs but such that L requires UFAs with $n^{\Omega(\log n)}$ states. This refutes a conjecture by Colcombet.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases Unambiguous automata, communication complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.126

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Acknowledgements We thank anonymous ICALP reviewers for many comments.

1 Introduction

Given two finite automata recognising languages $L_1, L_2 \subseteq \Sigma^*$ a basic question is to determine the *state complexity* of various language operations. How many states are needed in an automaton that recognises the union $L_1 \cup L_2$? How about the intersection $L_1 \cap L_2$? The complement $\overline{L}_1 := \Sigma^* \setminus L_1$? The answer depends on the type of automaton considered, such as deterministic (DFA), nondeterministic (NFA), or unambiguous (UFA). Recall that a UFA is an NFA that has at most one accepting computation on any input.

State complexities have been extensively studied for various types of automata and language operations; see, e.g., [9, 15] and their references, or the excellent compendium on Wikipedia [22]. For example, complementing an NFA with n states may require 2^n states [3], even for automata with binary alphabet [14]. Surprisingly, several extremely basic questions about UFAs remain open. For example, it was shown only in 2018 by Raskin [20] that the state complexity for UFA complementation is not polynomial: for any $n \in \mathbb{N}$ there exists a language L recognised by an n-state UFA such that any UFA (or even NFA) that recognises $\overline{L}$ has at least $n^{(\log\log\log n)^{\Omega(1)}}$ states. This superpolynomial blowup refuted a conjecture that it may be possible to complement UFAs with a polynomial blowup [5].

In this paper, as our main results, we prove three new blowup theorems.

▶ **Theorem 1** (Complement). *For every $n \in \mathbb{N}$ there is a language $L \subseteq \{0,1\}^*$ recognised by an n-state UFA such that any NFA that recognises $\overline{L}$ requires $n^{\tilde{\Omega}(\log n)}$ states.*

▶ **Theorem 2** (Union). *For every $n \in \mathbb{N}$ there are languages $L_1, L_2 \subseteq \{0,1\}^*$ recognised by n-state UFAs such that any UFA that recognises $L_1 \cup L_2$ requires $n^{\tilde{\Omega}(\log n)}$ states.*

▶ **Theorem 3** (Separation). *For every $n \in \mathbb{N}$ there is a language $L \subseteq \{0,1\}^*$ such that both L and $\overline{L}$ are recognised by n-state NFAs but any UFA that recognises L requires $n^{\Omega(\log n)}$ states.*

Discussion of main results. **Theorem 1** upgrades Raskin's slightly-superpolynomial bound into a quasipolynomial bound $n^{\tilde{\Omega}(\log n)}$. (Here we use the notation $\tilde{\Omega}(m)$ to suppress $\mathrm{poly}(\log m)$ factors.) However, we note that Raskin's language is unary, $|\Sigma| = 1$, while ours is binary, $|\Sigma| = 2$, and hence the two results are incomparable in this sense. As for positive results, it is known that the trivial 2^n upper bound for UFA complementation can be improved: the complement of any n-state UFA can be recognised by a UFA with at most $\mathrm{poly}(n) \cdot 2^{n/2}$ states [15, 13]. Closing the exponential gap here between the lower and upper bounds remains a tantalising open problem. It was highlighted as one of the foremost challenges in the recent Dagstuhl workshop *Unambiguity in Automata Theory* [6].

Theorem 2 establishes the first superpolynomial lower bound for the union operation. Letting $\sqcup$ denote disjoint union, observe that

$$L_1 \cup L_2 \;=\; L_1 \sqcup (L_2 \cap \overline{L}_1). \tag{1}$$

Since disjoint union and intersection are polynomial for UFAs, it follows from (1) and Theorem 2 that the same $n^{\tilde{\Omega}(\log n)}$ lower bound holds for complementing UFAs. However, we stress that Theorem 1 has a stronger conclusion than this, since it proves a lower bound against NFAs, not just UFAs. The observation (1) also yields the upper bound $\mathrm{poly}(n) \cdot 2^{n/2}$ by using the complement construction from [15, 13].

Theorem 3 refutes a conjecture by Colcombet [5, Conjecture 2]. Indeed, he conjectured that for any pair of NFAs recognising languages L_1, L_2 such that $L_1 \cap L_2 = \emptyset$, there is a polynomial-sized UFA that recognises some L that *separates* L_1 and L_2 in the sense that $L_1 \subseteq L$ and $L \cap L_2 = \emptyset$. Theorem 3 refutes this even in the special case $L_1 = \overline{L}_2$. Related separability questions are classical in formal language theory and have attracted renewed attention; see, e.g, [8] and the references therein. Separating automata have also been used recently to elegantly describe quasipolynomial time algorithms for solving parity games in an automata theoretic framework; see [4, Chapter 3] and [7].

1.1 Technique: Communication complexity

Our three main theorems rely on results – both new and old – in communication complexity; see [18, 19] for the standard textbooks. In communication complexity, one studies functions of the form $F\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ that determine the following two-party communication problem: Alice holds $x \in \{0,1\}^n$, Bob holds $y \in \{0,1\}^n$, and their goal is to output $F(x,y)$ while communicating as few bits as possible between them. Communication complexity is a classical tool to prove lower bounds for automata. Indeed, it is well known that if the language $\{xy : F(x,y) = 1\}$ is recognised by a small DFA (resp. NFA, UFA) then F admits an efficient deterministic (resp. nondeterministic, unambiguous) protocol. We revisit this connection in light of recent developments in communication complexity.

Theorem 1 is a ***relatively straightforward consequence*** of a recent result of Balodis et al. [2]. They exhibited a two-party function whose co-nondeterministic communication complexity is nearly quadratic in its unambiguous complexity (which matches an upper bound due to Yannakakis [23]). We translate this separation into the language of automata theory, virtually in a black-box fashion.

Theorem 2, by contrast, is our ***main technical contribution***. We will show that it follows from the following analogous communication result, which we prove in this paper.

▶ **Theorem 4.** *For every $m \in \mathbb{N}$ there exists a function $F(x, y)$ with unambiguous communication complexity at most m such that the logical-or of two copies of F, namely, $F^{\vee}(xx', yy') \coloneqq F(x, y) \vee F(x', y')$, has unambiguous communication complexity $\tilde{\Omega}(m^2)$.*

This is a new result in communication complexity; the unambiguous complexity of $F^{\vee}$ has not been studied previously. We prove Theorem 4 using the popular *query-to-communication lifting* technique that has been wildly successful in the past decade to prove communication lower bounds (including in [2]). In this technique, one starts by proving a lower bound on the query (aka decision tree) complexity of a boolean function $f\colon \{0,1\}^n \to \{0,1\}$. A lifting theorem (e.g., [12]) then transforms f into an analogous communication problem F in such a way that the communication complexity of F is characterised by the query complexity of f. This reduces the task of proving communication lower bounds into the much easier task of proving query lower bounds.

Interestingly, our proof of Theorem 4 formalises a kind of *converse* to the observation (1) above (saying that union can be computed via a complement). Namely, we show that unambiguously computing the union necessarily requires computing a complement, and therefore we can rely on an existing query lower bound for complementation [11].

Theorem 3, finally, is a straightforward consequence of a classical quadratic separation between two-sided nondeterministic communication complexity and unambiguous communication complexity due to Razborov [21].

1.2 Bonus result: Approximate nonnegative rank

Along the way to Theorem 4 we inadvertently stumbled upon another separation result that addresses a question raised by Kol et al. [16]. They studied the *ϵ-approximate nonnegative rank* $\mathrm{rk}^{+}_{\epsilon}(M)$ of a nonnegative matrix $M \in \mathbb{R}^{n\times n}$. Here, $\mathrm{rk}^{+}_{\epsilon}(M)$ is defined as the least nonnegative rank $\mathrm{rk}^{+}(N)$ of a matrix $N \in \mathbb{R}^{n\times n}$ such that $|M_{ij} - N_{ij}| \le \epsilon$ for all i, j; see Section 3 for precise definitions. In particular, Kol et al. [16] asked whether for all error parameters $0 < \epsilon < \delta < 1/2$ and boolean matrices $M \in \{0,1\}^{n\times n}$ we have the polynomial relationship $\mathrm{rk}^{+}_{\epsilon}(M) \le O(\mathrm{rk}^{+}_{\delta}(M)^C)$ where $C = C(\epsilon, \delta)$ is a constant. In short, does approximate nonnegative rank admit efficient error reduction? (It is known that the more usual notion, *approximate rank*, does [1].) We provide the following negative answer.

▶ **Theorem 5** (No efficient error reduction)**.** *For every $m \in \mathbb{N}$ there exists a boolean matrix M with $\mathrm{rk}^{+}_{1/4}(M) \le m$ but such that $\mathrm{rk}^{+}_{10^{-5}}(M) \ge m^{\tilde{\Omega}(\log m)}$.*

Previously, a negative answer was known only for *partial* boolean matrices $M \in \{0, 1, *\}^{n\times n}$ that allow "don't care" entries $M_{ij} = *$ [12]. Our Theorem 5 still leaves open the possibility (also raised by [16]) that, for a total boolean matrix M, we can bound $\mathrm{rk}^{+}_{\epsilon}(M)$ as a polynomial function of $\mathrm{rk}^{+}_{\delta}(M) + \mathrm{rk}^{+}_{\delta}(\overline{M})$ where $\overline{M}$ is the boolean complement.

1.3 Open problems

Our quasipolynomial lower bounds for automata are not known to be tight; in all cases the best known upper bounds are exponential. Curiously enough, the analogous communication results are tight for communication protocols. This suggests two opportunities.

- Can other techniques from communication complexity improve the lower bounds further? Perhaps via multi-party communication complexity?
- Can techniques for proving upper bounds on communication complexity be adapted to prove upper bounds on the size of automata?

1.4 Definitions of automata

An *NFA* is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where Q is the finite set of states, Σ is the finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $I \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of accepting states. We write $q \xrightarrow{a} r$ to denote that $(q, a, r) \in \delta$. A finite sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ is called a *run*; it can be summarized as $q_0 \xrightarrow{a_1 \cdots a_n} q_n$. The NFA $\mathcal{A}$ *recognizes* the language $L(\mathcal{A}) := \{w \in \Sigma^* \mid \exists\, q_0 \in I \,.\, \exists\, f \in F \,.\, q_0 \xrightarrow{w} f\}$. The NFA $\mathcal{A}$ is a *DFA* if $|I| = 1$ and for every $q \in Q$ and $a \in \Sigma$ there is exactly one q' with $q \xrightarrow{a} q'$. The NFA $\mathcal{A}$ is a *UFA* if for every word $w = a_1 \cdots a_n \in \Sigma^*$ there is at most one *accepting* run for w, i.e., a run $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$ with $q_0 \in I$ and $q_n \in F$. Any DFA is a UFA.

2 UFA Complementation

In this section we prove Theorem 1.

▶ **Theorem 1** (Complement). *For every $n \in \mathbb{N}$ there is a language $L \subseteq \{0,1\}^*$ recognised by an n-state UFA such that any NFA that recognises $\overline{L}$ requires $n^{\tilde{\Omega}(\log n)}$ states.*

The proof uses concepts from communication complexity, in particular a recent result from [2] and a nondeterministic lifting theorem from [12]. We start by recalling these tools.

2.1 DNFs and nondeterministic protocols

Unambiguous DNFs. Let $D = C_1 \vee \cdots \vee C_m$ be an n-variate boolean formula in disjunctive normal form (DNF). DNF D has *width* k if every C_i is a conjunction of at most k literals. We call such D a *k-DNF*. For conjunctive normal form (CNF) formulas the width and k-CNFs are defined analogously. DNF D is said to be *unambiguous* if for every input $x \in \{0,1\}^n$ at most one of the conjunctions C_i evaluates to true, $C_i(x) = 1$. For any boolean function $f\colon \{0,1\}^n \to \{0,1\}$ define

- $\mathrm{C}_1(f)$ as the least k such that f can be written as a k-DNF;
- $\mathrm{C}_0(f)$ as the least k such that f can be written as a k-CNF;
- $\mathrm{UC}_1(f)$ as the least k such that f can be written as an unambiguous k-DNF.

Note that $\mathrm{C}_0(f) = \mathrm{C}_1(\neg f)$. The following recent result separates two of these measures.

▶ **Theorem 6** ([2, Theorem 1]). *For every $k \in \mathbb{N}$ there exists a function $f\colon \{0,1\}^n \to \{0,1\}$ where $n \leq \mathrm{poly}(k)$ and such that $\mathrm{UC}_1(f) \leq k$ and $\mathrm{C}_0(f) \geq \tilde{\Omega}(k^2)$.* ◀

In words, for every k there is an unambiguous k-DNF such that any equivalent CNF requires width $\tilde{\Omega}(k^2)$. The bound is almost tight, as every unambiguous k-DNF has an equivalent k^2-CNF; see [10, Section 3].

Nondeterministic protocols and rectangle covers. Next we recall standard notions from two-party communication complexity; see [18, 19] for textbooks. Consider a two-party function $F\colon X \times Y \to \{0,1\}$. A set $A \times B \subseteq X \times Y$ (with $A \subseteq X$ and $B \subseteq Y$) is called a *rectangle*. Rectangles $R_1, \ldots, R_k$ *cover* a set $S \subseteq X \times Y$ if $\bigcup_i R_i = S$. For $b \in \{0,1\}$, the *cover number*

$\mathrm{Cov}_b(F)$ is the least number of rectangles that cover $F^{-1}(b)$. The *nondeterministic (resp., co-nondeterministic) communication complexity* of F is defined as $\mathrm{N}_1(F) \coloneqq \log_2 \mathrm{Cov}_1(F)$ (resp., $\mathrm{N}_0(F) \coloneqq \log_2 \mathrm{Cov}_0(F)$). Note that $\mathrm{N}_0(F) = \mathrm{N}_1(\neg F)$. The nondeterministic communication complexity can be interpreted as the number of bits that two parties (Alice and Bob), holding inputs $x \in X$ and $y \in Y$, respectively, need to communicate in a nondeterministic (i.e., based on guessing and checking) protocol in order to establish that $F(x,y) = 1$; see [18, Chapter 2] for details.

Nondeterministic lifting. Next we formulate a *lifting theorem*, which allows us to transfer lower bounds on the DNF width of an n-bit boolean function f to the nondeterministic communication complexity of a related two-party function F. We first choose a small two-party function $g\colon \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$, often called a *gadget*. Then we compose f with g to construct the function $F \coloneqq f \circ g^n$ that maps $\{0,1\}^{bn} \times \{0,1\}^{bn} \to \{0,1\}$ where Alice gets as input $x \in \{0,1\}^{bn}$, Bob gets as input $y \in \{0,1\}^{bn}$, and their goal is to compute

$$F(x,y) \;\coloneqq\; f(g(x_1,y_1),\dots,g(x_n,y_n)) \qquad \text{where } x_i, y_j \in \{0,1\}^b.$$

The following is a nondeterministic lifting theorem [12, 10].

▶ **Theorem 7** ([10, Theorem 4]). *For any $n \in \mathbb{N}$ there is a gadget $g\colon \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$ with $b = \Theta(\log n)$ such that for any function $f\colon \{0,1\}^n \to \{0,1\}$ we have, for $F \coloneqq f \circ g^n$,*

$$\mathrm{N}_0(F) \;=\; \Omega(\mathrm{C}_0(f) \cdot b)$$

(and thus also $\mathrm{N}_1(F) = \Omega(\mathrm{C}_1(f) \cdot b)$). ◀

Protocols can simulate automata. Finally, we need a simple folklore connection between automata and protocols. To formalise this, we tacitly identify a function $F\colon \{0,1\}^{m_1} \times \{0,1\}^{m_2} \to \{0,1\}$ with the language $F^{-1}(1) \;=\; \{xy \in \{0,1\}^{m_1+m_2} \mid F(x,y) = 1\}$.

▶ **Lemma 8.** *If a two-party function $F\colon \{0,1\}^{m_1} \times \{0,1\}^{m_2} \to \{0,1\}$ admits an NFA with s states, then $\mathrm{Cov}_1(F) \le s$ (that is, $\mathrm{N}_1(F) \le \log s$).*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be an NFA with $L(\mathcal{A}) = \{xy \in \{0,1\}^{m_1+m_2} \mid F(x,y) = 1\}$. We show that $F^{-1}(1)$ is covered by at most $|Q|$ rectangles. Indeed, $F^{-1}(1)$ equals

$$\bigcup_{q \in Q} \left(\{x \in \{0,1\}^{m_1} \mid \exists\, q_0 \in I \,.\, q_0 \xrightarrow{x} q\}\right) \times \left(\{y \in \{0,1\}^{m_2} \mid \exists\, f \in F \,.\, q \xrightarrow{y} f\}\right).$$

(Alternatively, in terms of a nondeterministic protocol, the first party, holding $x \in \{0,1\}^{m_1}$, produces a run for x from an initial state to a state q and then sends the name of q, which takes $\log_2 |Q|$ bits, to the other party. The other party then produces a run for y from q to an accepting state.) ◀

2.2 Proof of Theorem 1

For $k \in \mathbb{N}$, let $f\colon \{0,1\}^n \to \{0,1\}$ be the function from Theorem 6. That is, f has an unambiguous k-DNF with $k = n^{\Omega(1)}$ (hence, $\log n = O(\log k)$) and $\mathrm{C}_0(f) = \tilde{\Omega}(k^2)$. Let $g\colon \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$ with $b = \Theta(\log n)$ and $F \coloneqq f \circ g^n\colon \{0,1\}^{bn} \times \{0,1\}^{bn} \to \{0,1\}$ be the two-party functions from the lifting theorem Theorem 7. We will show that Theorem 1 holds for the language $F^{-1}(1)$.

First we argue that F has an unambiguous DNF of small width. Indeed, g and $\neg g$ have unambiguous $2b$-DNFs, which can be extracted from the deterministic decision tree of g. By plugging these unambiguous $2b$-DNFs for g and $\neg g$ into the unambiguous k-DNF for f (and "multiplying out"), one obtains an unambiguous $2bk$-DNF, say D, for F.

Over the $2bn$ variables of F, there exist at most $(2(2bn)+1)^{2bk}$ different conjunctions of at most $2bk$ literals. So D consists of at most $n^{O(bk)}$ conjunctions. From D we obtain a UFA $\mathcal{A}$ that recognizes $F^{-1}(1) \subseteq \{0,1\}^{2bn}$, as follows. Each initial state of $\mathcal{A}$ corresponds to a conjunction in D. When reading the input $x \in \{0,1\}^{2bn}$, the UFA checks that the corresponding assignment to the variables satisfies the conjunction represented by the initial state. This check requires at most $O(bn)$ states for each initial state. Thus, $\mathcal{A}$ has at most $n^{O(bk)} = 2^{\tilde{O}(k)} =: N$ states in total. (We use N in place of n in the statement of Theorem 1.)

On the other hand, by Theorem 7, we have $\mathrm{N}_0(F) = \Omega(\mathrm{C}_0(f) \cdot b) = \tilde{\Omega}(k^2)$. So by Lemma 8 any NFA that recognizes $F^{-1}(0)$ has at least $2^{\tilde{\Omega}(k^2)}$ states. Any NFA that recognizes $\{0,1\}^* \setminus L(\mathcal{A})$ can be transformed into an NFA that recognizes $F^{-1}(0) = \{0,1\}^{2bn} \setminus L(\mathcal{A})$ by taking a product with a DFA that has $2bn+2$ states. It follows that any NFA that recognizes $\{0,1\}^* \setminus L(\mathcal{A})$ has at least $2^{\tilde{\Omega}(k^2)}/(2bn+2) = 2^{\tilde{\Omega}(k^2)} = N^{\tilde{\Omega}(\log N)}$ states. ◀

3 UFA Union

In this section, we prove Theorem 2.

▶ **Theorem 2** (Union). *For every $n \in \mathbb{N}$ there are languages $L_1, L_2 \subseteq \{0,1\}^*$ recognised by n-state UFAs such that any UFA that recognises $L_1 \cup L_2$ requires $n^{\tilde{\Omega}(\log n)}$ states.*

We follow the same high-level approach that we already saw in Section 2. Namely, we will first show that computing the $\vee$-operation is hard for unambiguous DNFs and then lift that hardness to unambiguous protocols, which then implies the same hardness for UFAs. There are, however, two challenges in carrying out this plan.

1. It is an open problem to prove an unambiguous lifting theorem. That is, it is not known whether the unambiguous communication complexity of $f \circ g^n$ is at least $\Omega(\mathrm{UC}_1(f))$. To circumvent this issue, we study instead a *linear relaxation* of unambiguous DNFs. These objects are called *conical juntas* and they do admit a lifting theorem [12, 17].
2. There is no existing result showing that the $\vee$-operation is hard for unambiguous DNFs and/or conical juntas. We show a result of this type. The proof is by a reduction to the hardness of negating conical juntas, which *is* a known result [11].

3.1 Conical juntas

A nonnegative function $h\colon \{0,1\}^n \to \mathbb{R}_{\geq 0}$ is a *d-junta* if h depends on at most d variables. For example, a conjunction of d literals is a d-junta. Moreover, we say $f\colon \{0,1\}^n \to \mathbb{R}_{\geq 0}$ is a *conical d-junta* if it can be written as a nonnegative linear combination of d-juntas. Equivalently, f is a conical d-junta if it can be written as $f = \sum_i w_i C_i$ where each C_i is a width-d conjunction and $w_i \in \mathbb{R}_{\geq 0}$ are nonnegative coefficients. For example, if f can be written as an unambiguous d-DNF, $f = C_1 \vee \cdots \vee C_m$, then $f = \sum_i C_i$ is a conical d-junta with 0/1 coefficients. The *nonnegative degree* of f, denoted $\deg^+(f)$, is the least d such that f is a conical d-junta. In particular, if f is boolean-valued, then $\deg^+(f) \leq \mathrm{UC}_1(f)$.

We also need to work with *approximate* conical juntas that compute a given function only to within some point-wise error $\epsilon > 0$. This is important because the available lifting theorems for $\deg^+$ incur some error, and hence we need to prove lower bounds that are robust to this error. Indeed, we define the *ϵ-approximate nonnegative degree* of f, denoted $\deg^+_\epsilon(f)$, as the least nonnegative degree of a conical junta g such that

$$|f(x) - g(x)| \;\leq\; \epsilon \qquad \text{for all } x \in \{0,1\}^n.$$

▶ Remark. An awkward aspect of working with approximate conical juntas is that the error parameter ϵ is not well behaved. For $0 < \epsilon < \delta < 1/2$ we of course have $\deg^+_\delta(f) \leq \deg^+_\epsilon(f)$ but it is not a priori clear whether the converse inequality holds with a modest loss in the degree. In fact, in Section 5, we will end up showing that there can be a polynomial gap between the nonnegative degrees corresponding to two different error parameters – and this is related to our bonus result discussed in the introduction. As a consequence, our theorems in this section have to track the error parameters with some care.

Linear programming formulation. Approximate nonnegative degree can be captured using an LP. Write $\mathcal{C}^n_d$ for the set of all conjunctions of width at most d over n variables. In the (Primal) programme below, we have a variable $w_C \in \mathbb{R}$ for every $C \in \mathcal{C}^n_d$. In the associated (Dual) programme, we have a variable $\Phi(x) \in \mathbb{R}$ for each $x \in \{0,1\}^n$.

$$\begin{aligned} \min \quad & \epsilon \\ \textit{subject to} \quad & \textstyle|\sum_C w_C C(x) - f(x)| \;\leq\; \epsilon, && \forall x \in \{0,1\}^n \\ & w_C \;\geq\; 0, && \forall C \in \mathcal{C}^n_d \end{aligned} \tag{Primal}$$

$$\begin{aligned} \max \quad & \langle \Phi, f\rangle \;\coloneqq\; \textstyle\sum_x \Phi(x) f(x) \\ \textit{subject to} \quad & \|\Phi\| \;\coloneqq\; \textstyle\sum_x |\Phi(x)| \;\leq\; 1 \\ & \langle \Phi, C\rangle \;\leq\; 0, && \forall C \in \mathcal{C}^n_d \end{aligned} \tag{Dual}$$

We have that $\deg^+_\delta(f) \leq d$ iff the optimal value of (Primal) is at most δ. Alternatively, by strong LP duality, we have $\deg^+_\delta(f) > d$ iff there exists a feasible solution Φ to (Dual) such that $\langle \Phi, f\rangle > \delta$. It is typical to think of such feasible $\Phi\colon \{0,1\}^n \to \mathbb{R}$ as a *dual certificate* that witnesses a lower bound on approximate nonnegative degree.

3.2 Hardness of $\vee$

The goal of this subsection is to prove Theorem 9 below, which states that the $\vee$-operation is hard for unambiguous DNFs and even approximate conical juntas. Given an n-bit boolean function f we define a $2n$-bit function by $f^\vee(xy) \coloneqq f(x) \vee f(y)$ where $x, y \in \{0,1\}^n$.

▶ **Theorem 9** (Hardness of $\vee$). *For every $m \in \mathbb{N}$, there exists a boolean function $f\colon \{0,1\}^n \to \{0,1\}$ with $n \leq \mathrm{poly}(m)$ such that $\mathrm{UC}_1(f) \leq m$ and $\deg^+_{1.5\times 10^{-5}}(f^\vee) \geq \tilde{\Omega}(m^2)$.*

We show Theorem 9 by combining two lemmas, Lemmas 10 and 11, below. The first lemma, proved in [11], states that unambiguous DNFs are hard to negate, even by approximate conical juntas. The second lemma, which remains for us to prove, states that, for conical juntas, computing $f^\vee$ is at least as hard as computing the negation $\neg f$. Hence Theorem 9 follows immediately by combining these lemmas.

▶ **Lemma 10** (Hardness of ¬ [11, Lemma 8]). *For every $m \in \mathbb{N}$, there exists a boolean function $f\colon \{0,1\}^n \to \{0,1\}$ with $n \le \mathrm{poly}(m)$ such that $\mathrm{UC}_1(f) \le m$ and $\deg^+_{0.05}(\neg f) \ge \tilde{\Omega}(m^2)$.* ◀

▶ **Lemma 11** ($\vee$ harder than $\neg$). *For every $\delta > 0$ there exists an $\epsilon = \epsilon(\delta) > 0$ such that for every boolean function f, we have $\deg^+_\epsilon(f^\vee) \ge \Omega(\deg^+_\delta(\neg f))$. Moreover, $\epsilon \coloneqq \left(\frac{\ln(1+\delta)}{\lceil \log_{3/4} \delta \rceil}\right)^2$.*

It remains to prove Lemma 11. We do it in two steps. In Claim 12 we show that the approximate nonnegative degree of $f^\vee$ is at least that of $2 - f$ by exhibiting a dual certificate. Then in Claim 13 we show that the approximate nonnegative degree of $2 - f = 1 + \neg f$ is at least that of $\neg f$ via a powering trick. The error parameter ϵ will degrade in both of these steps. (We will later see that this degradation is, in fact, unavoidable; see Section 5.)

▷ Claim 12. We have $\deg^+_{\epsilon^2}(f^\vee) \ge \deg^+_\epsilon(2 - f)$ for any boolean-valued f and error ϵ.

Proof. Let $d \coloneqq \deg^+_\epsilon(2 - f)$ and let $\Phi\colon \{0,1\}^n \to \{0,1\}$ be a dual certificate witnessing this. That is, $\langle \Phi, 2 - f\rangle > \epsilon$, $\|\Phi\| \le 1$, and $\langle \Phi, C\rangle \le 0$ for all $C \in \mathcal{C}^n_{d-1}$. To construct a dual certificate $\Phi^\vee\colon \{0,1\}^{2n} \to \{0,1\}$ witnessing $\deg^+_{\epsilon^2}(f^\vee) \ge d$, we consider the negated tensor product (which was found by an educated guess)

$$\Phi^\vee(x,y) \;\coloneqq\; -\Phi(x)\Phi(y).$$

It remains to check that this is feasible for the dual programme and also that $\langle \Phi^\vee, f^\vee\rangle > \epsilon^2$.

1. $\|\Phi^\vee\| = \sum_{x,y} |\Phi^\vee(x,y)| = \sum_{x,y} |\Phi(x)\Phi(y)| = \sum_{x,y} |\Phi(x)| \cdot |\Phi(y)| = \|\Phi\|^2 \le 1$.
2. For any conjunction $C \in \mathcal{C}^{2n}_{d-1}$, we write $C(x,y) = C_1(x)C_2(y)$ where $C_1, C_2 \in \mathcal{C}^n_{d-1}$. Now

$$\begin{aligned}
\langle \Phi^\vee, C\rangle &= \textstyle\sum_{x,y} \Phi^\vee(x,y)C(x,y) \\
&= \textstyle\sum_{x,y} -\Phi(x)\Phi(y) \cdot C_1(x)C_2(y) \\
&= -\big[\textstyle\sum_x \Phi(x)C_1(x)\big]\big[\textstyle\sum_y \Phi(y)C_2(y)\big] \\
&= -\langle \Phi, C_1\rangle \langle \Phi, C_2\rangle \\
&\le 0.
\end{aligned}$$

3. Observe that $\langle \Phi, -f\rangle \ge \langle \Phi, 2 - f\rangle$ since $1 \in \mathcal{C}^n_{d-1}$ for the constant-1 function. Thus

$$\begin{aligned}
\langle \Phi^\vee, f^\vee\rangle &= \textstyle\sum_{x,y} \Phi^\vee(x,y) f^\vee(x,y) \\
&= \textstyle\sum_{x,y} -\Phi(x)\Phi(y)(f(x) + f(y) - f(x)f(y)) \\
&= \textstyle\sum_{x,y} -\Phi(x)\Phi(y)(2f(x) - f(x)f(y)) \\
&= \textstyle\sum_x -\Phi(x)f(x) \cdot \big[\sum_y \Phi(y)(2 - f(y))\big] \\
&= \langle \Phi, -f\rangle \cdot \langle \Phi, 2 - f\rangle \\
&\ge \langle \Phi, 2 - f\rangle \cdot \langle \Phi, 2 - f\rangle && \text{(above observation)} \\
&> \epsilon^2.
\end{aligned}$$

   where the third equality holds since we have $\sum_{x,y} \Phi(x)\Phi(y)f(x) = \sum_{x,y} \Phi(x)\Phi(y)f(y)$ by exchanging x and y. ◁

▷ Claim 13. For any $\delta > 0$ define $\epsilon \coloneqq \frac{\ln(1+\delta)}{\lceil \log_{3/4} \delta \rceil} > 0$. Then for any boolean-valued function f we have $\deg^+_\epsilon(1 + f) \ge \Omega(\deg^+_\delta(f))$.

Proof. We may assume $\delta < 1/2$ (and hence $\epsilon < 1/4$) as otherwise the claim is trivial. Suppose $\deg^+_\epsilon(1 + f) = d$ is witnessed by a conical d-junta g that ϵ-approximates $1 + f$. Define $g' \coloneqq ((g + \epsilon)/2)^k$ where the exponent is $k \coloneqq \lceil \log_{3/4} \delta \rceil$. By multiplying out the terms in

this definition, we see that g' has nonnegative degree $kd = O(d)$. We claim that g' is a δ-approximation of f. Indeed, if $f(x) = 0$, then $g'(x) \le (1/2+\epsilon)^k \le (3/4)^k \le \delta$. If $f(x) = 1$, then $1 \le (g(x)+\epsilon)/2 \le 1+\epsilon$, and thus $1 \le g'(x) \le (1+\epsilon)^k \le \exp(\epsilon k) \le 1+\delta$. ◁

Proof of Lemma 11. Using Claim 12 and Claim 13 (but with $\neg f$ in place of f), we have, for any $\delta > 0$ and $\epsilon := \frac{\ln(1+\delta)}{\lceil \log_{3/4} \delta \rceil}$:

$$\deg^+_{\epsilon^2}(f^\vee) \;\ge\; \deg^+_\epsilon(2-f) \;=\; \deg^+_\epsilon(1+\neg f) \;\ge\; \Omega(\deg^+_\delta(\neg f)).$$ ◀

3.3 Unambiguous protocols and nonnegative rank

Our goal will be to *lift* the hardness of the $\vee$-operation (Theorem 9) to communication complexity. In this subsection, we recall the concepts that are needed for this goal, namely, unambiguous protocols, (approximate) nonnegative rank, and a lifting theorem from nonnegative degree to nonnegative rank [12, 17].

Unambiguous protocols. Recall from Section 2.1 the notions of nondeterministic protocols and rectangle covers. For a two-party function $F\colon X \times Y \to \{0,1\}$, the *partition number* $\mathrm{Par}_1(F)$ is the least number of *pairwise disjoint* rectangles that cover $F^{-1}(1)$. Note that $\mathrm{Cov}_1(F) \le \mathrm{Par}_1(F)$. The *unambiguous communication complexity* of F is defined as $\mathrm{U}_1(F) := \log_2 \mathrm{Par}_1(F)$. Note that $\mathrm{N}_1(F) \le \mathrm{U}_1(F)$. Unambiguous communication complexity can be interpreted as the least communication cost of a nondeterministic protocol that has at most one accepting computation on every input. We also have the following folklore lemma, proved the same way as Lemma 8, which states that UFAs are simulated by unambiguous protocols.

▶ **Lemma 14.** *If a two-party function $F\colon \{0,1\}^{m_1} \times \{0,1\}^{m_2} \to \{0,1\}$ admits an UFA with s states, then $\mathrm{Par}_1(F) \le s$ (that is, $\mathrm{U}_1(F) \le \log s$).* ◀

Nonnegative rank. We often think of a two-party function $F\colon X \times Y \to \{0,1\}$ as a boolean matrix $F \in \{0,1\}^{X\times Y}$, sometimes called the *communication matrix* of F. For a nonnegative matrix $M \in \mathbb{R}^{X\times Y}_{\ge 0}$ we define its *nonnegative rank*, denoted $\mathrm{rk}^+(M)$, as the least r such that M can be written as a sum of r nonnegative rank-1 matrices, i.e., $M = \sum_{i=1}^r u_i v_i^{\mathrm{T}}$, where $u_i \in \mathbb{R}^X_{\ge 0}$ and $v_i \in \mathbb{R}^Y_{\ge 0}$ are nonnegative vectors. Note that for a boolean matrix F,

$$\mathrm{Par}_1(F) \;\ge\; \mathrm{rk}^+(F) \qquad \text{and thus} \qquad \mathrm{U}_1(F) \;\ge\; \log \mathrm{rk}^+(F). \tag{2}$$

Indeed, if $F^{-1}(1)$ can be partitioned into r rectangles, $F^{-1}(1) = R_1 \sqcup \cdots \sqcup R_r$, then F can be written as a sum of r nonnegative rank-1 matrices, $F = M_1 + \cdots + M_r$, where M_i is 1 on the rectangle R_i and 0 elsewhere. As with nonnegative degree, we define an approximate version of nonnegative rank. The *ϵ-approximate nonnegative rank* of M, denoted $\mathrm{rk}^+_\epsilon(M)$, is defined as the least $\mathrm{rk}^+(N)$ over all nonnegative matrices N that ϵ-approximate M, i.e.,

$$|M_{ij} - N_{ij}| \;\le\; \epsilon \qquad \text{for all } i,j.$$

Nonnegative lifting. Finally, we formulate a theorem that lifts lower bounds on the nonnegative degree of an n-bit boolean function f to the nonnegative rank of the composed function $F = f \circ g^n$ (which was defined in Section 2.1).

▶ **Theorem 15** ([12, 17])**.** *Fix constants $\delta > \epsilon > 0$. For any $n \in \mathbb{N}$ there is a gadget $g\colon \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$ with $b = \Theta(\log n)$ such that for any $f\colon \{0,1\}^n \to \{0,1\}$ we have*

$$\log \mathrm{rk}^+_\epsilon(f \circ g^n) \;\ge\; \Omega(\deg^+_\delta(f) \cdot b).$$ ◀

3.4 Proof of Theorem 2 (and also Theorem 4)

We start with the function $f\colon \{0,1\}^n \to \{0,1\}$ given by Theorem 9 such that for $m = \mathrm{poly}(n)$,

$$\mathrm{UC}_1(f) \;\leq\; m, \tag{3}$$
$$\deg^+_{1.5\times 10^{-5}}(f^\vee) \;\geq\; \tilde{\Omega}(m^2). \tag{4}$$

We then use the gadget g on $b = \Theta(\log n)$ bits from the lifting theorem Theorem 15 to construct $F := f \circ g^n$. By the same argument as in Section 2.2 we see that the resulting $F\colon \{0,1\}^{nb} \times \{0,1\}^{nb} \to \{0,1\}$ enjoys the following upper bounds, derived from (3).

- F admits an unambiguous DNF of width $2bm = \tilde{O}(m)$.
- F admits an UFA of size $2^{\tilde{O}(m)}$.
- F admits an unambiguous protocol of cost $\mathrm{U}_1(F) \leq \tilde{O}(m)$.

On the other hand, we note that $F^\vee = (f \circ g^n)^\vee = f^\vee \circ g^n$. Hence, we may combine (2), Theorem 15, and (4) to conclude that

$$\mathrm{U}_1(F^\vee) \;\geq\; \log \mathrm{rk}^+_{10^{-5}}(F^\vee) \;\geq\; \Omega(\deg^+_{1.5\times 10^{-5}}(f^\vee)) \;\geq\; \tilde{\Omega}(m^2). \tag{5}$$

This finishes the proof of Theorem 4. We proceed with the proof of Theorem 2. To this end, we define two languages

$$L_1 \;:=\; \{xx'yy' : x,x',y,y' \in \{0,1\}^{bn} \text{ and } F(x,y) = 1\},$$
$$L_2 \;:=\; \{xx'yy' : x,x',y,y' \in \{0,1\}^{bn} \text{ and } F(x',y') = 1\}.$$

Both L_1 and L_2 admit UFAs of size $\mathrm{poly}(n) \cdot 2^{\tilde{O}(m)} = 2^{\tilde{O}(m)} =: N$. By contrast, we have $L_1 \cup L_2 = (F^\vee)^{-1}(1)$, and this union language requires UFAs of size $2^{\tilde{\Omega}(m^2)} = N^{\tilde{\Omega}(\log N)}$ by (5) and Lemma 14. This concludes the proof of Theorem 2. ◀

4 UFA Separation

In this section, we prove Theorem 3.

▶ **Theorem 3** (Separation). *For every $n \in \mathbb{N}$ there is a language $L \subseteq \{0,1\}^*$ such that both L and $\overline{L}$ are recognised by n-state NFAs but any UFA that recognises L requires $n^{\Omega(\log n)}$ states.*

Loosely speaking, in our construction, we define NFAs $\mathcal{A}_1, \mathcal{A}_2$ that recognize *(sparse) set disjointness* and its complement. For $n \in \mathbb{N}$ and $k \leq n$ we define

$$\textsc{Disj}^n_k \;:=\; \{(S,T) \mid S \subseteq [n],\ T \subseteq [n],\ |S| = |T| = k,\ S \cap T = \emptyset\}.$$

Define also $\langle \textsc{Disj}^n_k \rangle := \{\langle S\rangle\langle T\rangle \mid (S,T) \in \textsc{Disj}^n_k\}$ where $\langle S\rangle \in \{0,1\}^n$ is such that the ith letter of $\langle S\rangle$ is 1 if and only if $i \in S$, and similarly for $\langle T\rangle$. Note that $\langle S\rangle, \langle T\rangle$ each contain k times the letter 1. To prove Theorem 3 it suffices to prove the following lemma.

▶ **Lemma 16.** *For any $n \in \mathbb{N}$ let $k := \lceil \log_2 n \rceil$. There are NFAs $\mathcal{A}_1, \mathcal{A}_2$ with $n^{O(1)}$ states such that $L(\mathcal{A}_1) = \langle \textsc{Disj}^n_k \rangle$ and $L(\mathcal{A}_2) = \{0,1\}^* \setminus \langle \textsc{Disj}^n_k \rangle$. Furthermore, any UFA that recognizes $\langle \textsc{Disj}^n_k \rangle$ has at least $n^{\Omega(\log n)}$ states.*

In the rest of the section we prove Lemma 16 by following Razborov's analysis of sparse set disjointness [21]. In particular, we will give a self-contained proof of the existence of polynomial-sized NFAs for $\langle \textsc{Disj}^n_k \rangle$ and its complement, but the main argument also comes from communication complexity.

4.1 Proof of Lemma 16

First we prove the statement on UFAs. Write $\binom{[n]}{k} := \{S \subseteq [n] \mid |S| = k\}$. Let $F\colon \binom{[n]}{k} \times \binom{[n]}{k} \to \{0,1\}$ be the two-party function with $F(S,T) = 1$ if and only if $(S,T) \in \textsc{Disj}_k^n$. It is shown, e.g., in [18, Example 2.12] that the communication matrix of F has full rank, $\mathrm{rk}(F) = \binom{n}{k}$. Let $F'\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be such that $F'(x,y) = 1$ if and only if $xy \in \langle \textsc{Disj}_k^n \rangle$. Then F is a principal submatrix of F', so $\binom{n}{k} \le \mathrm{rk}(F')$. Using (2) and Lemma 14 it follows that any UFA, say $\mathcal{A}$, that recognizes $\langle \textsc{Disj}_k^n \rangle$ has at least $\binom{n}{k} \ge (\frac{n}{k})^k$ states. With $k := \lceil \log_2 n \rceil$, it follows that $\mathcal{A}$ has $n^{\Omega(\log n)}$ states.

It is easy to see that there is an NFA, $\mathcal{A}_2$, with $n^{O(1)}$ states and $L(\mathcal{A}_2) = \{0,1\}^* \setminus \langle \textsc{Disj}_k^n \rangle$. Indeed, we can assume that the input is of the form $\langle S \rangle \langle T \rangle$; otherwise $\mathcal{A}_2$ accepts. NFA $\mathcal{A}_2$ guesses $i \in [n]$ such that $i \in S \cap T$ and then checks it.

Finally, we show that there is an NFA, $\mathcal{A}_1$, with $n^{O(1)}$ states and $L(\mathcal{A}_1) = \langle \textsc{Disj}_k^n \rangle$. We can assume that the input is of the form $\langle S \rangle \langle T \rangle$; otherwise $\mathcal{A}_1$ rejects. NFA $\mathcal{A}_1$ "hard-codes" polynomially many sets $Z_1, \dots, Z_\ell \subseteq [n]$. It guesses $i \in [\ell]$ such that $S \subseteq Z_i$ and $Z_i \cap T = \emptyset$ and then checks it. It remains to show that there exist $\ell = n^{O(1)}$ sets $Z_1, \dots, Z_\ell \subseteq [n]$ such that for any $(S,T) \in \textsc{Disj}_k^n$ there is $i \in [\ell]$ with $S \subseteq Z_i$ and $Z_i \cap T = \emptyset$. The argument uses the probabilistic method and is due to [21]; see also [18, Example 2.12]. We reproduce it here due to its elegance and brevity.

Fix $(S,T) \in \textsc{Disj}_k^n$. Say that a set $Z \subseteq [n]$ *separates* (S,T) if $S \subseteq Z$ and $Z \cap T = \emptyset$. A random set $Z \subseteq [n]$ (each i is in Z with probability $1/2$) separates (S,T) with probability 2^{-2k}. Thus, choosing $\ell := \lceil 2^{2k} \ln \binom{n}{k}^2 \rceil = n^{O(1)}$ random sets $Z \subseteq [n]$ independently, the probability that none of them separates (S,T) is

$$(1 - 2^{-2k})^\ell \;<\; e^{-2^{-2k}\ell} \;\le\; \tbinom{n}{k}^{-2}.$$

By the union bound, since $|\textsc{Disj}_k^n| < \binom{n}{k}^2$, the probability that there exists $(S,T) \in \textsc{Disj}_k^n$ such that none of ℓ random sets separates (S,T) is less than 1. Equivalently, the probability that for all $(S,T) \in \textsc{Disj}_k^n$ at least one of ℓ random sets separates (S,T) is positive. It follows that there are $Z_1, \dots, Z_\ell \subseteq [n]$ such that each $(S,T) \in \textsc{Disj}_k^n$ is separated by some Z_i. ◀

5 Bonus result: Approximate nonnegative rank

In this section, we prove Theorem 5.

▶ **Theorem 5** (No efficient error reduction). *For every $m \in \mathbb{N}$ there exists a boolean matrix M with $\mathrm{rk}^+_{1/4}(M) \le m$ but such that $\mathrm{rk}^+_{10^{-5}}(M) \ge m^{\tilde{\Omega}(\log m)}$.*

We first illustrate the idea in the context of nonnegative degree. In contrast to Theorem 9 (which states that $\vee$ is hard to approximate to within tiny error), we show that the $\vee$-operation is, in fact, easy to approximate when we allow large enough error.

▷ Claim 17. For any boolean-valued f, we have $\deg^+_{1/4}(f^\vee) \le \deg^+(f)$.

Proof. Let $g\colon \{0,1\}^{2n} \to \mathbb{R}_{\ge 0}$ be given by $g(x,y) := (f(x) + f(y))/2 + 1/4$. Then

$$g(x,y) \;=\; \begin{cases} 1/4 & \text{if } f(x) = f(y) = 0, \\ 5/4 & \text{if } f(x) = f(y) = 1, \\ 3/4 & \text{otherwise.} \end{cases}$$

Thus g is a $1/4$-approximation to $f^\vee$. Note also that $\deg^+(g) \le \deg^+(f)$, as desired. ◁

We can now repeat the same idea for nonnegative rank. In Section 3.4 we constructed a boolean matrix (two-party function) F such that $\log \mathrm{rk}^+(F) \leq \mathrm{U}_1(F) \leq m$ and $\log \mathrm{rk}^+_{10^{-5}}(F^\vee) \geq \tilde{\Omega}(m^2)$. We claim that $\log \mathrm{rk}^+_{1/4}(F^\vee) \leq O(m)$, which would finish the proof of Theorem 5. Indeed, analogously to Claim 17, we can define a nonnegative matrix by $G(xx, yy') \coloneqq (F(x,y) + F(x',y'))/2 + 1/4$. This is a $1/4$-approximation to $F^\vee$ and we have $\mathrm{rk}^+(G) \leq 2 \cdot \mathrm{rk}^+(F) + 1 \leq 2^{m+1} + 1$, as claimed.

References

1 Noga Alon. Problems and results in extremal combinatorics–I. *Discrete Mathematics*, 273(1–3):31–53, 2003. doi:10.1016/S0012-365X(03)00227-9.

2 Kaspars Balodis, Shalev Ben-David, Mika Göös, Siddhartha Jain, and Robin Kothari. Unambiguous DNFs and Alon-Saks-Seymour. In *62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2021. To appear. Available at arXiv:2102.08348.

3 Jean-Camille Birget. Partial orders on words, minimal elements of regular languages, and state complexity. *Theoretical Computer Science*, 119(2):267–291, 1993. doi:10.1016/0304-3975(93)90160-U.

4 Mikołaj Bojańczyk and Wojciech Czerwiński. An automata toolbox, 2018. Available at https://www.mimuw.edu.pl/~bojan/paper/automata-toolbox-book.

5 Thomas Colcombet. Unambiguity in automata theory. In *17th International Workshop on Descriptional Complexity of Formal Systems (DCFS)*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3_1.

6 Thomas Colcombet, Karin Quaas, and Michał Skrzypczak. Unambiguity in Automata Theory (Dagstuhl Seminar 21452). *Dagstuhl Reports*, 11(10):57–71, 2022. doi:10.4230/DagRep.11.10.57.

7 Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Pawel Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2333–2349, 2019. doi:10.1137/1.9781611975482.142.

8 Wojciech Czerwiński and Sławomir Lasota. Regular separability of one counter automata. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:20)2019.

9 Yuan Gao, Nelma Moreira, Rogério Reis, and Sheng Yu. A survey on operational state complexity. *Journal of Automata, Languages and Combinatorics*, 21(4):251–310, 2016. doi:10.25596/jalc-2016-251.

10 Mika Göös. Lower bounds for clique vs. independent set. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1066–1076. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.69.

11 Mika Göös, T.S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Transactions on Computation Theory (TOCT)*, 10(1):1–20, 2018. doi:10.1145/3170711.

12 Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. *SIAM Journal on Computing*, 45(5):1835–1869, 2016. doi:10.1137/15M103145X.

13 Emil Indzhev and Stefan Kiefer. On complementing unambiguous automata and graphs with many cliques and cocliques. Technical report, arxiv.org, 2021. Available at arXiv:2105.07470.

14 Galina Jirásková. State complexity of some operations on binary regular languages. *Theoretical Computer Science*, 330(2):287–298, 2005. doi:10.1016/j.tcs.2004.04.011.

15 Jozef Jirásek Jr., Galina Jirásková, and Juraj Šebej. Operations on unambiguous finite automata. *International Journal of Foundations of Computer Science*, 29(5):861–876, 2018. doi:10.1142/S012905411842008X.

16 Gillat Kol, Shay Moran, Amir Shpilka, and Amir Yehudayoff. Approximate nonnegative rank is equivalent to the smooth rectangle bound. *Computational Complexity*, 28(1):1–25, 2019. Preliminary version in *ICALP '14*. doi:10.1007/s00037-018-0176-4.

17 Pravesh Kothari, Raghu Meka, and Prasad Raghavendra. Approximating rectangles by juntas and weakly exponential lower bounds for LP relaxations of CSPs. *SIAM Journal on Computing*, pages STOC17–305–STOC17–332, May 2021. Preliminary version in *STOC '17*. doi:10.1137/17m1152966.

18 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574948.

19 Anup Rao and Amir Yehudayoff. *Communication Complexity: And Applications*. Cambridge University Press, 2020. doi:10.1017/9781108671644.

20 Mikhail Raskin. A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 138:1–138:11, 2018. doi:10.4230/LIPIcs.ICALP.2018.138.

21 Alexander Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1):81–93, 1990. doi:10.1007/BF02122698.

22 Wikipedia. State complexity. URL: https://en.wikipedia.org/wiki/State_complexity.

23 Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991. doi:10.1016/0022-0000(91)90024-Y.

Satisfiability Problems for Finite Groups

Paweł M. Idziak
Jagiellonian University, Kraków, Poland

Piotr Kawałek
Jagiellonian University, Kraków, Poland

Jacek Krzaczkowski
Maria Curie-Sklodowska University, Lublin, Poland

Armin Weiß
Universität Stuttgart, FMI, Germany

Abstract

Over twenty years ago, Goldmann and Russell initiated the study of the complexity of the equation satisfiability problem (PolSat) and the NUDFA program satisfiability problem (ProgramSat) in finite groups. They showed that these problems are in P for nilpotent groups while they are NP-complete for non-solvable groups.

In this work we completely characterize finite groups for which the problem ProgramSat can be solved in randomized polynomial time under the assumptions of the Randomized Exponential Time Hypothesis and the Constant Degree Hypothesis. We also determine the complexity of PolSat for a wide class of finite groups. As a by-product, we obtain a classification for ListPolSat, a version of PolSat where each variable can be restricted to an arbitrary subset. Finally, we also prove unconditional algorithms for these problems in certain cases.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Complexity classes

Keywords and phrases Satisifiability, Solvable groups, ProgramSat, PolSat, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.127

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding This research is partially supported by Polish NCN Grant # 2014/14/A/ST6/00138 and German DFG Grant WE 6835/1-2.

1 Introduction

Non-uniform deterministic finite automata (NUDFA) are a well-known concept introduced by Barrington [1], which proves its usefulness in describing important classes of languages defined by Boolean circuits such as NC^1 [2], ACC^0 and AC^0 [6]. Formally, a NUDFA (also called $\mathbf{G}$-program) over a group $\mathbf{G} = (G, \cdot)$ computing an n-ary function $f\colon \{0,1\}^n \longmapsto \{0,1\}$ consists of an l-ary polynomial $\mathbf{p}$ over $\mathbf{G}$ (i.e. the term over $\mathbf{G}$ with some variables replaced with constants from G), a set $S \subseteq G$ (the accepting set) and a sequence of l triples (instructions) of the form $\langle i_j, g_0^j, g_1^j \rangle$ where $1 \leqslant i_j \leqslant n$ and $g_0^j, g_1^j \in G$ for $0 \leqslant j \leqslant l$ (the number of the triple) such that $f(\overline{x}) = 1$ iff $\mathbf{p}(g^1_{x_{i_1}}, g^2_{x_{i_2}}, \ldots, g^l_{x_{i_l}}) \in S$. Note that this definition of NUDFA is not exactly the same as the one introduced in [1] but it is equivalent to the original one. In particular, there are obvious linear time transformations between the two program definitions. Moreover, this new definition does not change the class of languages recognized by NUDFA's over a fixed group.


The natural problem to decide whether the language recognized by a given NUDFA over a fixed group is non-empty (ProgramSat) was introduced in [14] and considered together with the problem of the existence of a solution to a given equation of polynomials over some fixed finite group (PolSat – here the input consists of polynomials $\mathbf{p}$ and $\mathbf{q}$ with variables $x_1, \ldots, x_n$, the question is whether there is some $\overline{a} \in G^n$ with $\mathbf{p}(\overline{a}) = \mathbf{q}(\overline{a})$). In the case of (finite) groups PolSat can be reduced to ProgramSat. In [14] Goldmann and Russell established a polynomial-time algorithm for ProgramSat in nilpotent groups and concluded that also PolSat for nilpotent groups is in P. On the other hand, their proof that PolSat for non-solvable groups is NP-complete implies NP-completeness of ProgramSat for such groups. This last fact was proved directly in [4]. These results reflect the ability of expressing all the functions AND_n by short polynomials of a particular group or by short NUDFA programs. It was known that over any fixed non-solvable group AND_n can be computed by NUDFAs (of size polynomial in n) [2], while over nilpotent groups NUDFAs are able to compute AND_n only of bounded arity n [5]. Note also that over solvable but non-nilpotent groups NUDFA programs are in fact able to compute all of the AND_n's [2], but sometimes these programs are of exponential length, namely $2^{\Omega(n)}$ [5].

It turned out that for a fixed group $\mathbf{G}$ the length of the shortest NUDFA program (over $\mathbf{G}$) computing AND_n plays a crucial role in classifying the computational complexity of ProgramSat($\mathbf{G}$). On the one hand, it was proved in [4] that, if the size of the shortest NUDFA program over a group $\mathbf{G}$ computing AND_n is $2^{\Omega(n)}$ (following [4] we say that $\mathbf{G}$ is AND-weak), then there exists a quasi-polynomial time algorithm solving ProgramSat($\mathbf{G}$). On the other hand, the same paper shows that, if $\mathbf{G}$ is AND-strong (i.e. for every n there exists an efficiently computable NUDFA over $\mathbf{G}$ of polynomial size computing AND_n), then ProgramSat for a wreath product $\mathbf{G} \wr \mathbb{Z}_k$ is NP-complete if $k \geqslant 4$. The problem is that determining the length of a shortest NUDFA program over a fixed group $\mathbf{G}$ computing AND_n is often highly non-trivial. Because of this difficulty with estimating the size of AND_n (occurring, in fact, in many models of computations), [5] introduced the so-called Constant Degree Hypothesis (CDH). In a circuit language it can be stated as follows: Consider a circuit of depth three where the input gates are connected to bounded fan-in AND gates followed by a layer of MOD_p gates and a MOD_q gate as output gate. Under CDH such a circuit needs size $2^{\Omega(n)}$ for computing AND_n (see [15, 16]). In this paper we will only use that CDH implies that groups of the form $\mathbf{G} = \mathbf{G}_p \rtimes \mathbf{N}$, where $\mathbf{G}_p$ is a p-group and $\mathbf{N}$ is a nilpotent group, are AND-weak (see Theorem 10 in [5]).

The difficulty of determining which groups are AND-weak is probably the reason why for almost 20 years after publishing [14] and [4] not too much progress has been made in characterizing the complexity of ProgramSat and PolSat for solvable but non-nilpotent groups. A number of results [19, 18, 11] were proved but all of them were restricted to showing polynomial time algorithms solving PolSat for some subclasses of groups with so-called Fitting length at most 2 (groups $\mathbf{G}$ having a nilpotent normal subgroup $\mathbf{N}$ such that $\mathbf{G}/\mathbf{N}$ is nilpotent), see [12] for a most comprehensive result. An important step towards the full classification of the computational complexity of ProgramSat for finite groups was made in 2020 when we proved in [20, 30, 23] that for every finite group of Fitting length at least 3 the problem PolSat (and in a consequence also ProgramSat) is not in P, unless the Exponential Time Hypothesis (ETH) fails. Shortly thereafter in [21] the first three authors gave examples of groups with Fitting length 2 and non-tractable PolSat (and ProgramSat), again, under the assumption of ETH.

This paper (among other things) gives a full characterization of finite groups for which ProgramSat is tractable in randomized polynomial time. Our classification also works for a related problem we call ListPolSat($\mathbf{G}$): given polynomials $\mathbf{p}$ and $\mathbf{q}$ with variables $x_1, \ldots, x_n$

and a list of subsets $A_1, \ldots, A_n \subseteq G$, decide whether there is $\overline{a} = (a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$ with $\mathbf{p}(\overline{a}) = \mathbf{q}(\overline{a})$. While we are not aware of any previous results on ListPolSat for finite algebras, it has been studied in the form of equations with rational constraints in [29] for word equations and in [10] for groups. Our classification of the complexity of these problems relies on both above-mentioned complexity assumptions (hypotheses): rETH and CDH.

▶ **Theorem 1.** *Under the assumption of rETH and CDH, the problems* ProgramSat($\mathbf{G}$) *and* ListPolSat($\mathbf{G}$) *for a finite group* $\mathbf{G}$ *are in* RP *if and only if there is a prime* p *and a normal* p*-subgroup* $\mathbf{G}_p$ *of* $\mathbf{G}$ *with* $\mathbf{G}/\mathbf{G}_p$ *being nilpotent.*

Note that our results partially confirm the intuition behind CDH, which was stated over 30 years ago: we show that in all cases in which there is a chance for AND-weakness, it is implied by CDH.

The proof of Theorem 1 is based on two main ideas. The first one is to show that, if some group $\mathbf{G}$ is AND-weak, then ProgramSat($\mathbf{G}$) satisfies what we call the *none-or-many property*: if a language recognized by a given program over $\mathbf{G}$ is not empty, then this language contains at least a polynomial fraction of all words (see Lemma 10). This gives us a randomized polynomial time algorithm solving ProgramSat($\mathbf{G}$) for groups which are AND-weak. Now, assuming CDH we immediately obtain the upper bound from Theorem 1. The second idea is to use polynomials witnessing the non-nilpotency of $\mathbf{G}$ to produce relatively short (of subexponential size) programs (and polynomials) expressing CNF-formulas. This, together with rETH ensures us that there is no polynomial time randomized algorithm solving ProgramSat for a group $\mathbf{G}$ which does not have a normal p-subgroup $\mathbf{G}_p$ with a nilpotent quotient $\mathbf{G}/\mathbf{G}_p$. In fact, we prove the main lemmas of this part of the proof (Lemma 14 and Theorem 16) in the much more general setting of solvable algebras (in a sense universal algebraic sense) from a so-called congruence permutable variety. We use two powerful universal algebraic tools: Tame Congruence Theory [17] and Commutator Theory [13] to prove Lemma 14, which tells us how to use non-nilpotency to produce polynomials over non-nilpotent algebras which imitate polynomials over finite fields. We conclude with a subexponential reduction from 3-CNF-SAT to ProgramSat which is based on the polynomials of small degree from [3] that describe symmetric periodic functions and were used to construct relatively small modular circuits for AND.

The situation for PolSat is more involved: indeed, we are far from getting full classification. The main reason is that in this case we cannot restrict the arguments of a polynomial to certain values. Hence, we need much more control to be able to use the polynomials from [3] in the proof of Theorem 16. In order to state Theorem 16 in the group case, we write $C_{\mathbf{G}}(\mathbf{A}) = \{\, g \in G \mid ga = ag \text{ for all } a \in A \,\}$ for the centralizer of a normal subgroup $\mathbf{A}$ of $\mathbf{G}$:

▶ **Theorem 2.** *Let* $\mathbf{G}$ *be a finite solvable group with two minimal non-trivial normal subgroups* $\mathbf{A}$ *and* $\mathbf{B}$ *such that* $|A|$ *and* $|B|$ *are coprime and* $C_{\mathbf{G}}(\mathbf{A}) \cdot C_{\mathbf{G}}(\mathbf{B}) \neq G$*. Then the problem* PolSat($\mathbf{G}$) *is not in* RP *under rETH.*

▶ **Corollary 3.** *If a finite group* $\widetilde{\mathbf{G}}$ *has a quotient as in Theorem 2, then* PolSat($\widetilde{\mathbf{G}}$) *is not in* RP *under rETH.*

Notice that the conditions for hardness in Theorem 1 and Theorem 2 are quite similar. Indeed, if we are not in the RP-case of Theorem 1, then there are two different primes in $|[\mathbf{G}, \mathbf{G}]|$ witnessed by two normal subgroups $\mathbf{A}$ and $\mathbf{B}$ of $\mathbf{G}$ (but contained in $[\mathbf{G}, \mathbf{G}]$) of coprime order with $C_{\mathbf{G}}(\mathbf{A}) \neq G \neq C_{\mathbf{G}}(\mathbf{B})$ as opposed to $C_{\mathbf{G}}(\mathbf{A}) \cdot C_{\mathbf{G}}(\mathbf{B}) \neq G$ in Theorem 2. This subtle difference prevents us from giving a classification as Theorem 1 for the case of PolSat (for details, we refer to the proof of Theorem 16). Moreover, Theorem 4 shows that this is not due to our ignorance but there are indeed groups for which PolSat is in RP and ProgramSat is not (under rETH).

▶ **Theorem 4.** *Let* $\mathbf{G}$ *be the semidirect product* $\mathbf{N} \rtimes \mathbf{H}$ *of nilpotent groups* $\mathbf{N}$ *and* $\mathbf{H}$ *and let* $|G|$ *have at most two prime factors. Then* PolSat($\mathbf{G}$) *is in* RP *under CDH. Moreover, if* $\mathbf{H}$ *is abelian,* PolSat($\mathbf{G}$) *is in* RP *unconditionally.*

Note here that the last sentence of Theorem 4 gives an unconditional upper bound for the complexity of PolSat. This together with Theorem 1 enables us to classify the computational complexity of PolSat for dihedral groups (i.e the groups of symmetries of regular polygons, where $\mathbf{D}_m = \mathbb{Z}_m \rtimes \mathbb{Z}_2$ denotes the symmetry group of the m-gon).

▶ **Corollary 5** (Classification of dihedral groups)**.**

(i) *If* $m = 2^\alpha p^\beta$ *for* $\alpha, \beta \in \mathbb{N}$ *and an odd prime* p*, then* PolSat($\mathbf{D}_m$) *is in* RP*.*

(ii) *Otherwise* PolSat($\mathbf{D}_m$) *cannot be solved in* RP *under rETH (resp.* P *under ETH).*

Furthermore, based on Theorem 2 we obtain (under the assumption of CDH and rETH) a classification of PolSat for wreath products of nilpotent groups.

▶ **Corollary 6.** *Let* $\mathbf{G}$ *and* $\mathbf{H}$ *be nilpotent groups. Under CDH and rETH,* PolSat($\mathbf{G} \wr \mathbf{H}$) *is in* RP *if an only if* $\mathbf{G}$ *is a p-group or* $|G| \cdot |H|$ *has at most two prime divisors.*

In [12] a question was asked about the computational complexity of PolSat for several examples of groups. Among them were four groups of order 24 (complexity of PolSat for groups of smaller order was known previously). In this paper we determine the computational complexity of PolSat for three of these groups: $\mathbf{D}_{12}$, $(\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_3) \rtimes \mathbb{Z}_2$ and $\mathbb{Z}_3 \rtimes \mathbf{Q}$, where $\mathbf{Q}$ is the quaternion group. It turns out that PolSat for these groups is in RP (see Corollary 5 and Example 21). On the other hand, $\mathbf{S}_4$, the fourth of the groups mentioned above, was shown in [20, 23] to have non-tractable PolSat problem (assuming ETH).

2 Preliminaries

We use $[m \,..\, n]$ to denote the interval of integers $\{m, \ldots, n\}$, the difference of sets is denoted by $A - B$. We use standard notation from complexity theory, which can be found in any textbook on complexity, e.g. [27]. In particular, we write RP for randomized polynomial time (with one-sided error).

ETH and rETH. The Exponential Time Hypothesis (ETH) and its randomized version rETH (see e.g. [9]) is the conjecture that there is some $\delta > 0$ such that every (randomized) algorithm for 3-CNF-SAT needs time $\Omega(2^{\delta n})$ in the worst case where n is the number of variables of the given 3-CNF-SAT instance. By the Sparsification Lemma [25, Thm. 1] this is equivalent to the existence of some $\epsilon > 0$ such that every algorithm for 3-CNF-SAT needs (randomized) time $\Omega(2^{\epsilon(m+n)})$ in the worst case where m is the number of clauses of the given 3-CNF-SAT instance (see also [8, Thm. 14.4]). In particular, under ETH/rETH there is no (randomized) algorithm for 3-CNF-SAT running in time $2^{o(n+m)}$. Here we only consider one-sided errors; clearly our results also remain true when understanding rETH with two-sided error.

Group Theory. Throughout, we only consider finite groups $\mathbf{G}$ with underlying set (universe) G. We follow the notation of [28]. For groups $\mathbf{G}$ and $\mathbf{H}$ we write $\mathbf{H} \leqslant \mathbf{G}$ if $\mathbf{H}$ is a subgroup of $\mathbf{G}$ and $\mathbf{H} < \mathbf{G}$ if $\mathbf{H}$ is a proper subgroup of $\mathbf{G}$. For a subset $X \subseteq G$ we write $\langle X \rangle$ for the subgroup generated by X. We write $[x, y] = x^{-1}y^{-1}xy$ for the *commutator*. The commutator of subgroups $\mathbf{X}, \mathbf{Y} \leqslant \mathbf{G}$ is defined by $[\mathbf{X}, \mathbf{Y}] = \langle\, [x, y] \mid x \in X, y \in Y \,\rangle$.

Let $\mathbf{N} \leqslant \mathbf{H}$ be a normal subgroup of $\mathbf{G}$. We define the *centralizer* of $\mathbf{H}$ modulo $\mathbf{N}$ as $C_{\mathbf{G}}(\mathbf{H}/\mathbf{N}) = \{ g \in G \mid [g,h] \in N \text{ for all } h \in H \}$. The center of a group is $Z(\mathbf{G}) = C_{\mathbf{G}}(\mathbf{G})$. Nilpotent groups can be defined inductively: $\mathbf{G}$ is nilpotent of class 1 if it is abelian; $\mathbf{G}$ is *nilpotent* of class c if $\mathbf{G}/Z(\mathbf{G})$ is nilpotent of class $c-1$. A group is called a *p-group* (for p prime) if its order is p^k for some $k \in \mathbb{N}$. It is well-known that a finite group is nilpotent iff it is a direct product of p-groups (see e.g. 5.1.3 and 5.2.4 in [28]). We will use this fact without further reference. If $\mathbf{G} = \mathbf{NH}$ where $\mathbf{N}$ is a normal subgroup of $\mathbf{G}$ and $\mathbf{H}$ a subgroup with $N \cap H = \{1\}$, $\mathbf{G}$ is called a *semidirect product* and we write $\mathbf{G} = \mathbf{N} \rtimes \mathbf{H}$. Notice that $\mathbf{G}/\mathbf{N} = \mathbf{H}$ in this case. Elements of $\mathbf{N} \rtimes \mathbf{H}$ can be viewed as pairs (n,h) with $n \in N$ and $h \in H$ with the multiplication rule $(n_1,h_1)(n_2,h_2) = (n_1\, {}^{h_1}n_2, h_1h_2)$ where ${}^{h_1}n_2 = h_1n_2h_1^{-1}$ is an action (via automorphisms) of $\mathbf{H}$ on $\mathbf{N}$. We will use the following classical result (see e.g. [28, 9.1.2]):

▶ **Fact 7** (Schur-Zassenhaus Theorem). *If* $\mathbf{G}$ *is a group with a normal subgroup* $\mathbf{N}$ *and* $|N|$ *and* $|\mathbf{G}/\mathbf{N}|$ *are coprime, then* $\mathbf{G} = \mathbf{N} \rtimes \mathbf{G}/\mathbf{N}$.

To define the wreath product $\mathbf{G} \wr \mathbf{H}$ of groups $\mathbf{G}$ and $\mathbf{H}$, we start with the direct power $\mathbf{G}^H = \{ f : H \to G \}$. Now $\mathbf{H}$ has a natural left action on $\mathbf{G}^H$ given by $({}^hf)(y) = f(h^{-1}y)$ for $f \in \mathbf{G}^H$, $h \in H$ and $y \in H$. Now the *wreath product* $\mathbf{G} \wr \mathbf{H}$ is defined to be the corresponding semidirect product $\mathbf{G}^H \rtimes \mathbf{H}$.

Algebra. A finite algebra $\mathbf{A}$ is a finite universe A together with a finite number of k_i-ary fundamental operations $f_i : A^{k_i} \to A$ for $i \in I$. A *term* is a composition of fundamental operations and a *polynomial* is a term with some variables replaced by constants from A; we write $\mathrm{Pol}(\mathbf{A})$ for the set of polynomials over $\mathbf{A}$. An equivalence relation which preserves all operations of $\mathbf{A}$ is called a *congruence* (more precisely, an equivalence relation $\alpha \subseteq A \times A$ is a congruence iff for every fundamental operation f, say r-ary, of $\mathbf{A}$ and $(a_1,b_1), \ldots, (a_r,b_r) \in \alpha$ we have $(f(\overline{a}), f(\overline{b})) \in \alpha$). We usually write $a \overset{\alpha}{\equiv} b$ to express that $(a,b) \in \alpha$. For a congruence relation α on $\mathbf{A}$ the congruence class containing an element a is denoted by a/α and A/α is the set of all congruence classes of α. The congruences of an algebra $\mathbf{A}$, when ordered by inclusion (denoted by $\alpha \leqslant \beta$), form the complete lattice with the top element $1_{\mathbf{A}} = A \times A$ and the bottom element $0_{\mathbf{A}}$ consisting of all pairs (a,a) for $a \in A$. We frequently omit the subscripts in $0_{\mathbf{A}}$ and $1_{\mathbf{A}}$. Given two congruences β and γ, we write $\beta \vee \gamma$ for the *join*, which is the smallest congruence containing both β and γ. A congruence α is called *join-irreducible* iff, whenever $\alpha = \beta \vee \gamma$ for congruences β and γ, already $\alpha \in \{\beta, \gamma\}$. Note that a join-irreducible α has a unique congruence α_- such that $\alpha_- < \alpha$ and there is no β with $\alpha_- < \beta < \alpha$.

A normal subgroup $\mathbf{N}$ of a group $\mathbf{G}$ defines a congruence $\alpha_{\mathbf{N}} = \{ (a,b) \in G^2 \mid a^{-1}b \in N \}$. As $\mathbf{G}/\mathbf{N} = \mathbf{G}/\alpha_{\mathbf{N}}$, we do not distinguish between congruences and normal subgroups.

The commutator $[\mathbf{H}, \mathbf{K}]$ of normal subgroups of a group has been generalized as an operation on congruences for arbitrary algebras (see [13]) and then used to extend notions of solvability and nilpotency onto such algebras. For a precise definition of the commutator $[\alpha, \beta]$ of congruences α and β we refer to [13] and simply note that for groups it is the usual commutator. We say that a congruence α *centralizes* β modulo γ iff $[\alpha, \beta] \leqslant \gamma$. The biggest α which centralizes β modulo γ is denoted by $(\gamma : \beta)$ and called the *centralizer* of β modulo γ. In the group case this agrees exactly with the usual definition of the centralizer.

Another important concept of universal algebra derived from group theory is a *Malcev term*: it is a term $\mathbf{d}$ satisfying $\mathbf{d}(y,x,x) = \mathbf{d}(x,x,y) = y$. The existence of such a term implies many nice properties of an algebra (e.g. connected with the behaviour of commutators and the congruence lattice). In the group case, the term $\mathbf{d}(x,y,z) = xy^{-1}z$ is an example of a Malcev term. An algebra with a Malcev term is called a *Malcev algebra*.

Some of our proofs use advanced tools of universal algebra: the Commutator Theory and the Tame Congruence Theory as presented in the books [13] and [17]. The reader may find a not too long introduction to the needed notions and facts from these theories in [24].

Satisfiability problems. For the definition of a $\mathbf{G}$-program (aka. NUDFA over $\mathbf{G}$) and ProgramSat we refer to the introduction. Usually, we denote both polynomials and $\mathbf{G}$-programs by $\mathbf{p}$ and, for a simpler notation, also use $\mathbf{p}$ to denote the function $f : \{0,1\}^n \to \{0,1\}$ computed by the $\mathbf{G}$-program. For a unified treatment of ListPolSat and ProgramSat, we further define 2-ListPolSat which is like ListPolSat but the A_i all may contain only two elements. Notice that in groups, as we can multiply by inverses, we can assume that the input for these problems consists of only one polynomial (plus the lists restricting variables). Thus, in the group setting we call $\overline{a} \in G^n$ a solution to an instance $\mathbf{p}(\overline{x})$ for PolSat($\mathbf{G}$) if $\mathbf{p}(\overline{a}) = 1$. Likewise, we call some $\overline{a} \in \{0,1\}^n$ a solution for a $\mathbf{G}$-program $\mathbf{p}$ if $\mathbf{p}(\overline{a}) = 1$ (recall that $\mathbf{p}$ computes a function $\{0,1\}^n \to \{0,1\}$). It is obvious that we can treat 2-ListPolSat as a "subproblem" of ProgramSat in which NUDFA's programs have the following property: for every two instructions (i_j, g_0^j, g_1^j), (i_k, g_0^k, g_1^k) of the program, if $i_j = i_k$, then $g_0^j = g_0^k$ and $g_1^j = g_1^k$. Also note that even for general algebras PolSat($\mathbf{A}$) is the special case of ListPolSat($\mathbf{A}$) where all the A_i are A. These observations and some other well-known results are summarized as follows (see also [14, 4]):

▶ **Lemma 8.** *Let* $\mathbf{G}$ *be a group. Then*

- *If* $\mathbf{H} = \mathbf{G}/\mathbf{N}$, *then* PolSat($\mathbf{H}$) $\leqslant_{dtt}$ PolSat($\mathbf{G}$).
- *If* $\mathbf{H} \leqslant \mathbf{G}$ *or* $\mathbf{H} = \mathbf{G}/\mathbf{N}$, *then* ProgramSat($\mathbf{H}$) $\leqslant_m$ ProgramSat($\mathbf{G}$) *and* ListPolSat($\mathbf{H}$) $\leqslant_m$ ListPolSat($\mathbf{G}$).
- PolSat($\mathbf{G}$) $\leqslant_m$ 2-ListPolSat($\mathbf{G}$)
- 2-ListPolSat($\mathbf{G}$) $\leqslant_m$ ProgramSat($\mathbf{G}$) *and* 2-ListPolSat($\mathbf{G}$) $\leqslant_m$ ListPolSat($\mathbf{G}$).

Here $A \leqslant_m B$ denotes a polynomial-time many-one reduction and $A \leqslant_{dtt} B$ denotes a polynomial-time disjunctive truth-table reduction: one instance x for A is reduced to several instances $y_1, \ldots, y_k$ of B such that $x \in A$ if and only if there is some i with $y_i \in B$.

3 CDH and the Many-Solutions Property

We wish to treat NUDFA programs and polynomials (with variables restricted to lists) in a unified setting. For this we have to face the problem that programs are defined on Boolean domains whereas the domain of a variable of a polynomial is (a subset of) a group. So with an instance of ListPolSat / ProgramSat we associate an indicator function $f : \prod_{i=1}^n A_i \to \{0,1\}$ with $|A_i| \geqslant 2$ ($A_i \subseteq G$ resp. $A_i = \{0,1\}$) such that $f(\overline{x}) = 1$ iff $\overline{x}$ is a solution to the corresponding NUDFA program/polynomial equation with lists. In these cases we measure the size of f (denoted by size(f)) by the size of the smallest program/polynomial representing it (in the second case together with the sizes of lists). More precisely, in the definition of the size we take the smallest possible representation for f.

Note that independently of the model the function f was created in, if we replace some variables by constants, or restrict some of the A_i's from the domain to smaller sets $A_i' \subseteq A_i$ with $|A_i'| \geqslant 2$, we still obtain an indicator function (of possibly smaller arity) for another ListPolSat / ProgramSat instance. Moreover, the size of a function after such operations does not increase. This is the crucial property we use in Proposition 9 below. We will use the following notation:

- $f[x_J/\overline{a}]$ for a function obtained from f by substituting x_j by a_j for all $j \in J$,
- $f_{|\mathcal{B}}$ for a function obtained from f by restricting the domain of f, that is $\prod_{i=1}^n A_i$, to $\mathcal{B} = \prod_{i=1}^n B_j$ (for $B_j \subseteq A_j$),
- $f[x_i/a]$ for the substitution of one variable.

Also we combine these notations, e.g. by writing $f[x_I/\overline{b}, x_i/a]_{|\mathcal{B}}$.

Our aim is to bound the size of a general function using previous knowledge about the size for describing the AND function. In order to do so, we call $f : \prod_{i=1}^n A_i \to \{0,1\}$ with $|A_i| \geqslant 2$ for all i an n-ary *spike* if there is some $\overline{a} \in \prod_{i=1}^n A_i$ with $f(\overline{a}) = 1$ and for all $\overline{x} \neq \overline{a}$ we have $f(\overline{x}) = 0$. Let $\gamma : \mathbb{N} \to \mathbb{N}$ be the function $\gamma(n) = \min\{\,\mathrm{size}(f) \mid f \text{ is an } n\text{-ary spike}\,\}$. Notice that γ is monotone. Thus, its inverse $\gamma^{-1}(m) = \max\{\,n \in \mathbb{N} \mid \gamma(n) \leqslant m\,\}$ as a function $\mathbb{N} \to \mathbb{N}$ is well-defined and also monotone.

Sometimes we will also write $\gamma_{\mathrm{Prog},\mathbf{G}}$ or $\gamma_{\mathrm{Pol},\mathbf{G}}$ when we want to specify the model.

▶ **Proposition 9.** *Let f be an n-ary indicator function with domain $\mathcal{A} = \prod_{i=1}^n A_i$. Then either f is constant zero or $\left|f^{-1}(1)\right| / |\mathcal{A}| \geqslant 1/|\mathcal{A}|^{\gamma^{-1}(\mathrm{size}(f))}$. In particular, if $\gamma(n) \in 2^{\Omega(n)}$, then $\left|f^{-1}(1)\right| / |\mathcal{A}| \geqslant 1/\mathrm{size}(f)^{\mathcal{O}(1)}$.*

Proof. Our proof relies on the same idea as [21, Theorem 6.1]; however, we have to overcome the difficulty to deal with non-boolean domains. If f is constantly 1 the statement holds; otherwise f is non-nonstant. The idea is to successively substitute constants for variables while not increasing the density of $f^{-1}(1)$ in the full domain: If there is some $i \in [1\,..\,n]$ such that for all $a \in A_i$ the function $f[x_i/a]$ is not constant 0, we choose $b \in A_i$ such that $\left|f[x_i/b]^{-1}(1)\right|$ is minimal among all $\left|f[x_i/a]^{-1}(1)\right|$ for $a \in A_i$. Observe that $f[x_i/b]$ is not a constant function and $\left|f[x_i/b]^{-1}(1)\right| \leqslant \left|f^{-1}(1)\right| / |A_i|$. Now, we proceed by induction eventually obtaining some $I \subseteq [1\,..\,n]$ and $\overline{b} \in \prod_{i\in I} A_i$ such that $f[x_I/\overline{b}]$ is not constant, $1 \leqslant \left|f[x_I/\overline{b}]^{-1}(1)\right| \cdot \prod_{i\in I} |A_i| \leqslant \left|f^{-1}(1)\right|$, and for every $i \in [1\,..\,n] - I$ there is some $a \in A_i$ such that $f[x_I/\overline{b}, x_i/a]$ is constant zero. Now, we can restrict all remaining variables to two-element sets and we obtain a $(n - |I|)$-ary spike. As during this process the size of f does not increase, we obtain:

$$\frac{\left|f^{-1}(1)\right|}{|\mathcal{A}|} \geqslant \frac{\prod_{i\in I} |A_i|}{\prod_{i=1}^n |A_i|} = \frac{1}{\prod_{[1..n]-I} |A_i|} \geqslant \frac{1}{|\mathcal{A}|^{n-|I|}} \geqslant \frac{1}{|\mathcal{A}|^{\gamma^{-1}(\mathrm{size}(f))}}.$$ ◀

We say that ProgramSat($\mathbf{G}$), ListPolSat($\mathbf{G}$) or PolSat($\mathbf{G}$) has the *none-or-many property* if for any instance $\mathbf{p}$ of length m either $\mathbf{p}$ has no solution or a random assignment will be a solution with $1/m^{\mathcal{O}(1)}$ probability (recall that we call $\overline{a} \in G^n$ a solution to an instance $\mathbf{p}(\overline{x})$ for PolSat($\mathbf{G}$) if $\mathbf{p}(\overline{a}) = 1$ and similarly for ListPolSat($\mathbf{G}$) and ProgramSat).

▶ **Lemma 10.** *Let G be a group with $\gamma_{\mathrm{Prog},G}(n) \in 2^{\Omega(n)}$. Then, the problems* ProgramSat($\mathbf{G}$)*,* ListPolSat($\mathbf{G}$) *and* PolSat($\mathbf{G}$) *have the none-or-many property.*

In particular, ProgramSat($\mathbf{G}$)*,* ListPolSat($\mathbf{G}$) *and* PolSat($\mathbf{G}$) *are in* RP*.*

For the proof of Lemma 10 notice that a G-program for a spike is never longer than a polynomial for a spike. Therefore, the requirement $\gamma_{\mathrm{Prog},G}(n) \in 2^{\Omega(n)}$ also can be used for ListPolSat. The result for PolSat then follows by Lemma 8. For some polynomial or $\mathbf{G}$-program $\mathbf{p}$ we use Proposition 9 to get a bound $\left|\mathbf{p}^{-1}(1)\right| / |G^n| \geqslant 1/|G|^{\gamma^{-1}(|\mathbf{p}|)} \in 1/|G|^{\mathcal{O}(\log|\mathbf{p}|)} \subseteq 1/|\mathbf{p}|^{\mathcal{O}(1)}$.

▶ Remark 11. Notice that, if $\mathbf{G}$ is a group with $\gamma_{\mathrm{Prog},\mathbf{G}}(n) \in 2^{\Omega(n)}$, then by [4, Theorem 2] ProgramSat($\mathbf{G}$), ListPolSat($\mathbf{G}$) and PolSat($\mathbf{G}$) also can be solved in deterministic quasi-polynomial time (notice that [4, Theorem 2] technically is not for ListPolSat($\mathbf{G}$) but the proof can easily be adapted).

CDH-based conditional algorithms. Let us recall the consequence of the constant degree hypothesis (CDH) which has been proved in Theorem 10 of [5]:

(A) Let $\mathbf{G}_p$ be a p-group, $\mathbf{N}$ a nilpotent group and $\mathbf{G} = \mathbf{G}_p \rtimes \mathbf{N}$. If CDH is true, then every $\mathbf{G}$-program computing the n-ary AND function has length $2^{\Omega(n)}$.

Note that [5] gives a proof of CDH in the case that $\mathbf{N}$ is abelian, see (C) in Section 6 below. As an immediate consequence of Lemma 10 and (A), we obtain our next result.

▶ **Corollary 12.** *Let $\mathbf{G}_p$ be a p-group, $\mathbf{N}$ a nilpotent group and $\mathbf{G} = \mathbf{G}_p \rtimes \mathbf{N}$. If CDH is true, then* ProgramSat($\mathbf{G}$), ListPolSat($\mathbf{G}$) *or* PolSat($\mathbf{G}$) *have the none-or-many property.*

In particular, if CDH is true, then ProgramSat($\mathbf{G}$), ListPolSat($\mathbf{G}$) *and* PolSat($\mathbf{G}$) *are in* RP.

4 Lower Bounds

In this section we will prove our hardness conditions. Instead of ProgramSat we will prove intractability of 2-ListPolSat. For that we need go deeply into the local structure of finite algebras (called sometimes as Tame Congruence Theory) as described in [17]. In our setting, i.e. solvable Malcev algebras, this local structure, (relative to a pair of congruences $\alpha \prec \beta$, i.e. $\alpha < \beta$ with no congruence γ in between: $\alpha < \gamma < \beta$) reduces to a vector space over a finite field. The prime that is the characteristic of this field, is also called the *characteristic* of the pair (α, β). For a join-irreducible congruence γ we define its *characteristic* to be the characteristic of the pair (γ_-, γ).

Different characteristics of two join-irreducible congruences is one of two important ingredients we use in the proof of Theorem 16. The second one is what we call congruence collaboration: Two congruences α and β of an algebra $\mathbf{A}$ are called *collaborating* if there are 3 different elements $a, c, b \in A$ satisfying $a \overset{\alpha-\beta}{\frown} c \overset{\beta}{\frown} b$ or $a \overset{\alpha}{\frown} c \overset{\beta-\alpha}{\frown} b$, where $x \overset{R}{\frown} y$ stands for $(x, y) \in R$. Notice that in a group case any two distinct non-trivial normal subgroups are collaborating. We will also use following easy observation.

▶ **Lemma 13.** *For two congruences $\varphi \neq 1_{\mathbf{A}} \neq \psi$ of an algebra $\mathbf{A}$ we have $\varphi \cup \psi \neq 1_{\mathbf{A}}$.*

Proof. Suppose $\varphi \cup \psi = 1_{\mathbf{A}} \neq \varphi$. To see that then $(a, b) \in \psi$ for all $a, b \in A$, note first that $(a, b) \in \psi$ whenever $(a, b) \notin \varphi$. Now let $(a, b) \in \varphi$. Obviously $\varphi \neq 1_{\mathbf{A}}$ gives an element $c \in A - a/\varphi = A - b/\varphi$. But the previous case puts both (c, a) and (c, b) into ψ, so that $(a, b) \in \psi$ as claimed. ◀

Note that Lemma 13 cannot be extended to more than 2 congruences: in a finitely dimensional vector space the union of all 1-dimensional subspaces covers the entire space.

In the next Lemma we use the polynomial which witnesses the lack of a centralization to produce a polynomial of the algebra which imitates a given polynomial over the field $GF(p)$. A similar argument was used in [22].

▶ **Lemma 14.** *Let $\mathbf{A}$ be a finite solvable Malcev algebra. Moreover, let γ be a join-irreducible congruence of characteristic p such that $(\gamma_- : \gamma) \neq 1$. Then for:*

- *every pair $(e, a) \in \gamma$ of different elements,*
- *every subset $\top \subseteq A$ that is a union of $(\gamma_- : \gamma)$-cosets,*
- *and every n-ary polynomial $w(\overline{x})$ of degree s over the field $GF(p)$ that sends the set $\{0, 1\}^n$ to $\{0, 1\}$*

there is an n-ary polynomial $[w]_{\gamma, \top, a, e}$ of the algebra $\mathbf{A}$ such that

- *both the size of* $[w]_{\gamma,\top,a,e}$ *and the time needed to compute it are bounded by* $2^{O(s\cdot\log n)}$,
- *for any tuple* $\overline{x}\in A^n$ *we have*

$$[w]_{\gamma,\top,a,e}(\overline{x}) = \begin{cases} a, & \text{if } w(b_\top(x_1),\dots,b_\top(x_n)) = 1,\\ e, & \text{if } w(b_\top(x_1),\dots,b_\top(x_n)) = 0,\end{cases}$$

where $b_\top : A \longrightarrow \{0,1\}$ *is defined by* $b_\top(x) = 1$ *iff* $x\in\top$.

Proof. First put $\gamma^\star = (\gamma_- : \gamma)$ and let $\{d_0, d_1, \dots, d_r\}$ be a transversal of the quotient $\mathbf{A}/\gamma^\star$. Moreover, pick a (γ_-,γ)-minimal set V and two elements $(e',a')\in\gamma|_V-\gamma_-$. By N denote the trace of V containing both e' and a'. Note that $(\mathbf{A}|_N)/\gamma_-$ is polynomially equivalent to a one-dimensional vector space over a field of characteristic p. In fact we will use only its additive structure $((\mathbf{A}|_N)/\gamma_-,+)$ which is isomorphic to a certain power of $(\mathbb{Z}_p,+)$. We can assume that e'/γ_- is the neutral element of this group and that addition in this group is realized (modulo γ_-) by the polynomial $x+y=\mathbf{d}(x,e',y)$, where $\mathbf{d}$ is a Malcev term for $\mathbf{A}$. Note that, despite the fact that the addition defined above is guaranteed to behave nicely only modulo γ_-, the properties of the Malcev term give us that $e'+e' = \mathbf{d}(e',e',e') = e'$. Since we will be summing the large amount of summands, to keep the sum relatively short (i.e. of the length which is polynomial in the sum of the summands' lengths) we will compose the addition in a balanced binary way, i.e. the tree of a polynomial realizing the sum of m summands is supposed to be the complete binary tree with m leaves – we will point out when this is necessary. Now we use the argument for the second half of Lemma 3.1 in [22] to show that for any pair $(c,d)\notin\gamma^\star$ there is a binary polynomial $\mathbf{s}_{cd}(x,y)$ of $\mathbf{A}$, satisfying

$$\begin{aligned} \mathbf{s}_{cd}(e',y) &= e', && \text{for all } y\in A,\\ \mathbf{s}_{cd}(a',c) &\overset{\gamma_-}{\equiv} e',\\ \mathbf{s}_{cd}(a',d) &= a'. \end{aligned} \tag{1}$$

Because $(e',a')\notin\gamma_-$ and γ is join-irreducible, we know that $\gamma=\Theta(e',a')$. Now if $(c,d)\notin\gamma^\star$ then $[\Theta(c,d),\Theta(e',a')]\not\leqslant\gamma_-$ so that Exercise 6.6 in [13] supplies us with a binary polynomial $\mathbf{s}(x,y)$ of $\mathbf{A}$ such that $\mathbf{s}(e',c)\overset{\gamma_-}{\equiv}\mathbf{s}(a',c)$ but $\mathbf{s}(e',d)\overset{\gamma_-}{\not\equiv}\mathbf{s}(a',d)$. The second property gives $\Theta(\mathbf{s}(e',d),\mathbf{s}(a',d))=\gamma\ni(e',a')$ and therefore there is a unary polynomial $\mathbf{p}$ of $\mathbf{A}$ that takes the pair $(\mathbf{s}(e',d),\mathbf{s}(a',d))$ to (e',a') (see e.g. Lemma 3.2 in [24]). Now it should be easy to check that for the polynomial $\mathbf{s}_{cd}(x,y)=\mathbf{d}(\mathbf{ps}(x,y),\mathbf{ps}(e',y),e')$ we have

$$\begin{aligned} \mathbf{s}_{cd}(e',y) &= \mathbf{d}(\mathbf{ps}(e',y),\mathbf{ps}(e',y),e') &&= e',\\ \mathbf{s}_{cd}(a',c) &= \mathbf{d}(\mathbf{ps}(a',c),\mathbf{ps}(e',c),e') &&\overset{\gamma_-}{\equiv} \mathbf{d}(\mathbf{ps}(e',c),\mathbf{ps}(e',c),e') &&= e',\\ \mathbf{s}_{cd}(a',d) &= \mathbf{d}(\mathbf{ps}(a',d),\mathbf{ps}(e',d),e') &&= \mathbf{d}(a',e',e') &&= a', \end{aligned}$$

as claimed in (1).

Using the fact that $[\gamma,\gamma^\star]\leqslant\gamma_-$ we can keep conditions (1) modulo γ_- by varying the second variable modulo $\gamma^\star$:

$$\begin{aligned} \mathbf{s}_{cd}(e',y) &= e', && \text{for each } y\in A,\\ \mathbf{s}_{cd}(a',y) &\overset{\gamma_-}{\equiv} e', && \text{for each } y\in c/\gamma^\star,\\ \mathbf{s}_{cd}(a',y) &\overset{\gamma_-}{\equiv} a', && \text{for each } y\in d/\gamma^\star. \end{aligned} \tag{2}$$

Now we want c and d to range over our transversal of $\mathbf{A}/\gamma^\star$ so that for $i\neq j$ we put $\mathbf{s}_{ij}(x,y)=\mathbf{e}_V\mathbf{s}_{d_id_j}(\mathbf{e}_V(x),y)$, where $\mathbf{e}_V$ is the unary idempotent polynomial of $\mathbf{A}$ with range V. Obviously $\mathbf{s}_{ij}$ satisfies all the properties of $\mathbf{s}_{d_id_j}$ listed in (2), but the polynomial $\mathbf{s}_{ij}$ has its range contained in V and for any fixed $y\in A$ the mapping $V\ni v\longmapsto\mathbf{s}_{ij}(v,y)\in V$ is either a permutation of the (γ_-,γ)-minimal set V or collapses $\gamma|_V$ to γ_-, i.e. it is constant modulo γ_- on $\gamma|_V$-classes.

Hence, as for $v \in N$ and $y \in d_i/\gamma_-$ the map $v \longmapsto \mathbf{s}_{ij}(v,y)$ is not a permutation, we have $\mathbf{s}_{i,j}(v,y) \overset{\gamma_-}{\equiv} \mathbf{s}_{i,j}(a',y) \overset{\gamma_-}{\equiv} e' = \mathbf{s}_{i,j}(e',y)$, which allows us to replace the second line in (2) by:

$$\mathbf{s}_{ij}(v,y) \overset{\gamma_-}{\equiv} e', \quad \text{for each } v \in N \text{ and } y \in d_i/\gamma^\star. \tag{3}$$

Now for each $j = 0,\dots,r$ put $\mathbf{s}_j(v,y) = \mathbf{s}_{i_1 j}(\dots \mathbf{s}_{i_{r-1} j}(\mathbf{s}_{i_r j}(v,y),y)\dots,y)$, with $\{i_1,\dots,i_r\} = \{0,1,\dots,r\} - \{j\}$. Obviously $\mathbf{s}_j$ has its range contained in V. We will show that

$$\begin{array}{lcll} \mathbf{s}_j(e',y) & = & e', & \text{for each } \ y \in A, \\ \mathbf{s}_j(v,y) & \overset{\gamma_-}{\equiv} & e', & \text{for each } \ v \in N \text{ and } \ y \in A - d_j/\gamma^\star, \\ \mathbf{s}_j(a',y) & \overset{\gamma_-}{\equiv} & a', & \text{for each } \ y \in d_j/\gamma^\star. \end{array} \tag{4}$$

Indeed, the first and the last item follow directly from the definition of $\mathbf{s}_j$. For the middle one, note that for $v \in N$, $y \in d_{i_\ell}/\gamma^\star$, and $v' = \mathbf{s}_{i_{\ell+1} j}(\dots \mathbf{s}_{i_{r-1} j}(\mathbf{s}_{i_r j}(v,y),y)\dots,y)$ we have $v' \overset{\gamma}{\equiv} \mathbf{s}_{i_{\ell+1}}(\dots \mathbf{s}_{i_{r-1} j}(\mathbf{s}_{i_r j}(e',y),y)\dots,y) = e'$, i.e. $v' \in N$ so that (2) yields $\mathbf{s}_{i_\ell j}(v',y) \overset{\gamma_-}{\equiv} e'$, and consequently $\mathbf{s}_j(v,y) = \mathbf{s}_{i_1 j}(\dots \mathbf{s}_{i_{\ell-1} j}(\mathbf{s}_{i_\ell j}(v',y),y)\dots,y) \overset{\gamma_-}{\equiv} \mathbf{s}_{i_1 j}(\dots \mathbf{s}_{i_{\ell-1} j}(e',y)\dots,y) = e'$. This establishes (4).

Now for a positive integer m and a $\gamma^\star$-block, i.e. the product $Q_{i_1,\dots,i_n} = d_{i_1}/\gamma^\star \times \dots \times d_{i_m}/\gamma^\star \subseteq A^m$ we define $(1+m)$-ary polynomial $\mathbf{q}_{i_1,\dots,i_m}(v,y_1,\dots,y_m)$ by putting

$$\mathbf{q}_{i_1,\dots,i_m}(v,y_1,\dots,y_m) = \mathbf{s}_{i_m}(\mathbf{s}_{i_{m-1}}(\dots \mathbf{s}_{i_2}(\mathbf{s}_{i_1}(v,y_1),y_2)\dots,y_{m-1}),y_m).$$

Then we observe that due to (4) we have

$$\begin{array}{lcll} \mathbf{q}_{i_1,\dots,i_m}(e',\overline{y}) & = & e', & \text{for all } \overline{y} \in A^m, \\ \mathbf{q}_{i_1,\dots,i_m}(v,\overline{y}) & \overset{\gamma_-}{\equiv} & e', & \text{for } v \in N \text{ and } \overline{y} \notin Q_{i_1,\dots,i_m}, \\ \mathbf{q}_{i_1,\dots,i_m}(a',\overline{y}) & \overset{\gamma_-}{\equiv} & a', & \text{for } \overline{y} \in Q_{i_1,\dots,i_m}. \end{array} \tag{5}$$

Note that the length of the polynomial $\mathbf{q}_{i_1,\dots,i_m}$ is bounded by $2^{O(m)}$, as it is a composition of m polynomials of the form $\mathbf{s}_j$ each of which having the size bounded by the same constant that depends only on $\mathbf{A}$.

Now, if $Q = Q_1 \cup \dots \cup Q_l$, with each Q_i being a single n-dimensional $\gamma^\star$-block, we sum up (in a balanced binary way) the polynomials $\mathbf{q}_i(\overline{x})$ produced, as above, separately for each block Q_i to get $\mathbf{q}_Q(\overline{x})$ satisfying

$$\begin{array}{lcll} \mathbf{q}_Q(e',\overline{x}) & = & e', & \text{for each } \overline{x} \in A^m, \\ \mathbf{q}_Q(v,\overline{x}) & \overset{\gamma_-}{\equiv} & e', & \text{if } v \in N \text{ and } \overline{x} \notin Q, \\ \mathbf{q}_Q(a',\overline{x}) & \overset{\gamma_-}{\equiv} & a', & \text{if } \overline{x} \in Q. \end{array} \tag{6}$$

The balanced way in which the $\mathbf{q}_i$'s are summed up guaranties that the resulting polynomial $\mathbf{q}_Q$ has its length bounded by $2^{O(m)}$, even if there are exponentially many summands determined by the blocks contained in A^m.

Now, let w be an n-ary polynomial over the field $GF(p)$ of degree s. To produce the associated polynomial $[w]_{\gamma,\top,a,e}$ of $\mathbf{A}$ we first produce $[w]_{\gamma,\top,a',e'}$ and then compose it with a unary polynomial $\mathbf{p}(x)$ that maps a' to a and e' to e (the algebra $\mathbf{A}$ has such a polynomial $\mathbf{p}$ as $(e,a) \in \gamma = \Theta(e',a')$). To produce the required n-ary polynomial $[w]_{\gamma,\top,a',e'}$ we first construct a $(1+n)$-ary polynomial $(w)_{\gamma,\top,a',e'}$ satisfying

$$\begin{array}{lcll} (w)_{\gamma,\top,a',e'}(e',\overline{x}) & = & e', & \text{for each } \overline{x} \in A^n, \\ (w)_{\gamma,\top,a',e'}(v,\overline{x}) & = & e', & \text{if } v \in N \text{ and } w(\mathsf{b}_\top(x_1),\dots,\mathsf{b}_\top(x_n)) = 0, \\ (w)_{\gamma,\top,a',e'}(v,\overline{x}) & = & v, & \text{if } v \in V \text{ and } w(\mathsf{b}_\top(x_1),\dots,\mathsf{b}_\top(x_n)) = 1, \end{array} \tag{7}$$

to put $[w]_{\gamma,\top,a',e'}(\overline{x}) = (w)_{\gamma,\top,a',e'}(a',\overline{x})$, which, by the last two lines in (7), will do the job.

To transform the polynomial $w(\overline{x})$ over the field $GF(p)$ into $(w)_{\gamma,\top,a',e'}(a',\overline{x})$, we first assume that $w(\overline{x})$ is given as sum of monomials and that the monomials of $w(\overline{x})$ are of the form $x_{i_1}\cdot\ldots\cdot x_{i_m}$ (with $m\leqslant s$) or are constant 1 (i.e., the monomials carry no leading constant – we can achieve this by replacing $2x$ by $x+x$ etc.). When passing from w to $(w)_{\gamma,\top,a',e'}(a',\overline{x})$ the constant 1 will be represented by the unary polynomial $\mathbf{e}_V(v)$, while the monomial $x_{i_1}\cdot\ldots\cdot x_{i_m}$ turns into $\mathbf{q}_{T^m}(v,x_{i_1},\ldots x_{i_m})$. Since $\top$ has been assumed to be a join of $\gamma^\star$-cosets, $\top^m$ is obviously a sum of $\gamma^\star$-blocks. Note also that for $\overline{y}\in A^m$ we have $\mathbf{q}_{\top^m(\overline{y})}(a',\overline{y})\subseteq e'/\gamma_-\cup a'/\gamma_-$ and, due to (6), we have

$$\mathbf{q}_{\top^m(\overline{y})}(a',\overline{y})\overset{\gamma_-}{\equiv}a'\quad\text{iff}\quad\overline{y}\in\top^m\quad\text{iff}\quad\mathsf{b}_\top(y_1)\cdot\mathsf{b}_\top(y_2)\cdot\ldots\cdot\mathsf{b}_\top(y_m)=1. \tag{8}$$

Now we sum up (appropriate amount of) the polynomials $\mathbf{e}_V(v)$ and appropriate polynomials $\mathbf{q}_Q(v,x_{i_1},\ldots x_{i_m})$ to get $(w)_{\gamma,\top,a',e'}(a',\overline{x})$ (again we have to be careful to use a balanced summation tree). Next we are using the fact that there is an isomorphism of the group $\mathbb{Z}_p$ with the subgroup of $((\mathbf{A}|_N)/\gamma_-,+)$ generated by a'/γ_- that sends 1 to a'/γ_-. Applying this isomorphism to (8) we get

$$\begin{array}{lcll}(w)_{\gamma,\top,a',e'}(e',\overline{x}) & = & e', & \text{for each } \overline{x}\in A^m,\\ (w)_{\gamma,\top,a',e'}(a',\overline{x}) & \overset{\gamma_-}{\equiv} & e', & \text{if } w(\mathsf{b}_\top(x_1),\ldots,\mathsf{b}_\top(x_n))=0,\\ (w)_{\gamma,\top,a',e'}(a',\overline{x}) & \overset{\gamma_-}{\equiv} & a', & \text{if } w(\mathsf{b}_\top(x_1),\ldots,\mathsf{b}_\top(x_n))=1.\end{array} \tag{9}$$

Since there are at most $O(n^s)=O(2^{s\cdot\log n})$ monomials of degree at most s, while the polynomials $\mathbf{q}_{\top^m}$ representing them has sizes bounded by $2^{O(s)}$ the length of $(w)_{\gamma,\top,a',e'}(v,\overline{x})$ is at most $2^{O(s\cdot\log n)}$.

Finally to pass from (9) to (7) we start with reminding that for a fixed $\overline{x}\in A^n$ the mapping $V\ni v\longmapsto(w)_{\gamma,\top,a',e'}(v,\overline{x})\in V$ either permutes the set V or collapses $\gamma|_V$ to γ_-, i.e. it is constant modulo γ_- on $\gamma|_V$-classes. Thus, iterating $(w)_{\gamma,\top,a',e'}(v,\overline{x})$ in the first variable a sufficient number of times, we may additionally assume that $(w)_{\gamma,\top,a',e'}(v,\overline{x})$ is either the identity map on V or it is constant, modulo γ_-, on $\gamma|_V$-classes, depending on whether $w(\mathsf{b}(\overline{x}))$ is 1 or 0. Thus, $w(\mathsf{b}(\overline{x}))=1$ gives us the third equality in (7).

In the other case, $(w)_{\gamma,\top,a',e'}(v,\overline{x})$ collapses the entire trace N to e'/γ_- so that we have $(w)_{\gamma,\top,a',e'}(v,\overline{x})\overset{\gamma_-}{\equiv}e'$ for $v\in N$. Now we go down along the chain $0=\theta_0\prec\theta_1\prec\ldots\prec\theta_l=\gamma_-$ to show that $(w)_{\gamma,\top,a',e'}(v,\overline{x})\overset{\theta_i}{\equiv}e'$ yields $(w)_{\gamma,\top,a',e'}(v,\overline{x})\overset{\theta_{i-1}}{\equiv}e'$. Since the unary polynomial $\mathbf{f}:V\ni v\longmapsto(w)_{\gamma,\top,a',e'}(v,\overline{x})\in V$ does not permute V, $\mathbf{f}(A)=\mathbf{f}(V)\subsetneq V$. This gives that for each $i=l,\ldots,1$ the polynomial $\mathbf{f}$ collapses θ_i to θ_{i-1}, as otherwise V would properly contain a (θ_{i-1},θ_i)-minimal set. But this is impossible in view of Lemma 4.30 in [17]. Therefore, composing l times the polynomial $\mathbf{f}$ we get that this composition satisfies $(w)_{\gamma,\top,a',e'}(v,\overline{x})=e'$ so that (7) is shown. This iteration inflates the size of $(w)_{\gamma,\top,a',e'}(v,\overline{x})$ by raising it to the l-th power so that it is still bounded by $2^{O(s\cdot\log n)}$. ◀

We are going to use the following fact that is borrowed from [3] (see [21, Fact 3.4] for a recent proof).

▶ **Fact 15.** *Let p be a prime and $\nu\geqslant 1$ be an integer. Then there is a polynomial $w(\overline{x})\in GF(p)[\overline{x}]$ of degree at most $p^\nu-1$, such that for $\overline{x}\in\{0,1\}^n\subseteq\mathbb{Z}_p^n$ we have*

$$w(\overline{x})=\begin{cases}0, & \textit{if } \left|\overline{x}^{-1}(0)\right|\equiv 0 \textit{ modulo } p^\nu,\\ 1, & \textit{else.}\end{cases}$$

Now we are in a position to prove the following.

▶ **Theorem 16.** *Let* $\mathbf{A}$ *be a finite solvable Malcev algebra and* α, β *be two collaborating join-irreducible congruences of different characteristics. Then, assuming the (randomized) Exponential Time Hypothesis, the following hold:*

- *if* $(\alpha_- : \alpha) \vee (\beta_- : \beta) < 1$*, then* PolSat($\mathbf{A}$) *is not in* P *(resp.* RP*),*
- *if* $(\alpha_- : \alpha) < 1$ *and* $(\beta_- : \beta) < 1$*, then* 2-ListPolSat($\mathbf{A}$) *is not in* P *(resp.* RP*).*

More precisely, we show that under rETH there is no randomized algorithm of running time $2^{o((\log n/\log\log n)^2)}$ for these problems. Before we prove Theorem 16, let us point out how it implies Theorem 2 from the introduction:

Proof of Theorem 2. As $\mathbf{A}$ and $\mathbf{B}$ are minimal normal subgroups, they are join-irreducible. As $\mathbf{A} \neq \mathbf{B}$, they are collaborating and by assumption they are of different characteristic. The condition $C_{\mathbf{G}}(\mathbf{A}) \cdot C_{\mathbf{G}}(\mathbf{B}) \neq \mathbf{G}$ is just the same as $(\alpha_- : \alpha) \vee (\beta_- : \beta) < 1$ written in a group language (here $\mathbf{A} = \alpha$, $\mathbf{B} = \beta$ and $\alpha_- = \beta_- = \{1\}$). Now we apply Theorem 16. ◀

Proof of Theorem 16. Since α, β are collaborating, without loss of generality we may assume that there are 3 different elements $a, e, b \in A$ such that $(a, e) \in \alpha$ and $(e, b) \in \beta - \alpha$. Let $\mathbf{d}$ denote a Malcev term for $\mathbf{A}$. Observe here that

(*) for $u \in \{a, e\}$ and $v \in \{b, e\}$ we have $\mathbf{d}(u, e, v) = e$ iff $u = e = v$.

Indeed, $\mathbf{d}(a, e, e) = a$ and $\mathbf{d}(e, e, b) = b$, while $\mathbf{d}(a, e, b) = e$ would give $b = \mathbf{d}(e, e, b) \overset{\alpha}{\equiv} \mathbf{d}(a, e, b) = e$, contrary to our choice of $(b, e) \in \beta - \alpha$.

To unify our arguments for both PolSat($\mathbf{A}$) and 2-ListPolSat($\mathbf{A}$), observe that $\alpha^* = (\alpha_- : \alpha) < 1$ and $\beta^* = (\beta_- : \beta) < 1$ gives $\alpha^* \cup \beta^* \neq 1$, by Lemma 13. This allows us to pick a pair $(c, d) \in A^2$ satisfying

[P] $(c, d) \notin \alpha^* \vee \beta^*$,

[LP] $(c, d) \notin \alpha^* \cup \beta^*$,

depending on which of the two problems PolSat($\mathbf{A}$), 2-ListPolSat($\mathbf{A}$) we are considering.

With the help of Lemma 14 and Fact 15, to each 3-CNF-SAT formula $\Phi(\overline{x})$ having n variables $\overline{x} = (x_1, \ldots, x_n)$ and ℓ clauses we are going to associate two n-ary polynomials $\mathbf{t}^\Phi(\overline{x})$ and $\mathbf{s}^\Phi(\overline{x})$ of length $2^{O(\sqrt{\ell} \cdot \log \ell)}$ such that

[P] $\Phi(\overline{x})$ is satisfiable iff the equation $\mathbf{t}^\Phi(\overline{x}) = e$ has a solution $\overline{x}$ in A^n,

[LP] $\Phi(\overline{x})$ is satisfiable iff the equation $\mathbf{s}^\Phi(\overline{x}) = e$ has a solution $\overline{x}$ in the set $\{c, d\}^n$.

To do that, let p and q be the characteristics of α and β, respectively, and pick $\mu, \nu \in \mathbb{N}$ with $p^{\mu-1} \leqslant \sqrt{\ell} < p^\mu$ and $q^{\nu-1} \leqslant \sqrt{\ell} < q^\nu$. Now Fact 15 supplies us with ℓ-ary polynomials (with their variables c_i's later to be substituted by the clauses C_i's of the formula $\Phi(\overline{x})$):

- $w_p(c_1, \ldots, c_\ell) \in GF(p)[\overline{c}]$, with degree bounded by $p^\mu - 1$,
- $w_q(c_1, \ldots, c_\ell) \in GF(q)[\overline{c}]$, with degree bounded by $q^\nu - 1$,

which on $\overline{c} \in \{0, 1\}^\ell$ take values from $\{0, 1\}$, such that $w_p(\overline{c}) = 0$ iff $\left|\overline{c}^{-1}(0)\right| \equiv 0 \mod p^\mu$ (resp. $w_q(\overline{c}) = 0$ iff $\left|\overline{c}^{-1}(0)\right| \equiv 0 \mod q^\mu$).

With a 3-ary clause $C(z^1, z^2, z^3)$ we associate a polynomial $C'(z^1, z^2, z^3)$ of degree 3 over $GF(p)$ (resp. $GF(q)$) in an obvious way, so that for example the clause $z^1 \vee z^2 \vee \neg z^3$ goes to $1 - ((1 - z^1) \cdot (1 - z^2) \cdot z^3)$. Now for $\Phi(\overline{x}) = \bigwedge_{i=1}^{\ell} C_i$ we feed up the polynomials w_p and w_q by substituting C_i' for the variable c_i to produce (at most 3ℓ-ary) polynomials $w_p^\Phi(\overline{z})$ and $w_q^\Phi(\overline{z})$ of degrees bounded by $3(p^\mu - 1)$ and $3(q^\nu - 1)$, respectively. Note that again the new polynomials w_p^Φ (or w_q^Φ) on arguments from $\{0, 1\}$ return values from the same set $\{0, 1\}$, but this time 0 is taken exactly on valuations of variables in Φ under which the number of unsatisfied clauses is divisible by p^μ (or by q^ν, respectively). The important feature is that

(**) the polynomials w_p^Φ and w_q^Φ simultaneously return 0 on a valuation from the set $\{0, 1\}$ iff this valuation satisfies Φ.

Indeed, $w_p^\Phi(\overline{z}) = 0 = w_q^\Phi(\overline{z})$ tells us that the number of unsatisfied clauses in Φ is divisible both by p^μ and q^ν. Since $p \neq q$, this number is divisible by $p^\mu \cdot q^\nu > \ell$. However, there are only ℓ clauses, so that none of them is unsatisfied by $\overline{z}$.

Now with the help of Lemma 14 we are able to define

$$\mathbf{t}^\Phi(\overline{x}) = \mathbf{d}\left(\left[w_p^\Phi\right]_{\alpha,\top_\alpha,a,e}(\overline{x}), e, \left[w_q^\Phi\right]_{\beta,\top_\beta,b,e}(\overline{x})\right),$$

where a and b are as above and the sets $\top_\alpha, \top_\beta \neq \emptyset, A$ are chosen to be sums of α^*- or β^*-cosets, respectively. Since our assumption for the problem PolSat says that $\alpha^* \vee \beta^* < 1$, one way to ensure this is by putting $\top_\alpha = \top_\beta$ to be a single $\alpha^* \vee \beta^*$-coset, e.g. $d/(\alpha^* \vee \beta^*)$. Now, combining Lemma 14 with the properties (*) and (**), we know that for $\overline{x} = (x_1, \ldots, x_n) \in A^n$ we have $\mathbf{t}^\Phi(\overline{x}) = e$ iff $\Phi(\mathsf{b}(x_1), \ldots, \mathsf{b}(x_n)) = 1$, where $\mathsf{b} = \mathsf{b}_{\top_\alpha} = \mathsf{b}_{\top_\beta}$.

In case of 2-ListPolSat the polynomial $\mathbf{s}^\Phi$ is defined in a similar way, i.e. we put

$$\mathbf{s}^\Phi(\overline{x}) = \mathbf{d}\left(\left[w_p^\Phi\right]_{\alpha,\top_\alpha,a,e}(\overline{x}), e, \left[w_q^\Phi\right]_{\beta,\top_\beta,b,e}(\overline{x})\right),$$

but this time we cannot ensure $\alpha^* \vee \beta^* < 1$. Instead we can bound the range for the x_i's in A to be smaller. In fact, we restrict these ranges to $\{c, d\}$ so that, by the very same argument as before, the sets $\top_\alpha = d/\alpha^*$ and $\top_\beta = d/\beta^*$ will do the job.

Finally, Fact 15 together with Lemma 14 ensure us that the sizes of $\left[w_p^\Phi\right]_{\alpha,\top_\alpha,a,e}$, and $\left[w_q^\Phi\right]_{\beta,\top_\beta,b,e}$, and therefore of both $\mathbf{t}^\Phi(\overline{x})$ and $\mathbf{s}^\Phi(\overline{x})$, are bounded by $2^{O(\sqrt{\ell}\cdot\log\ell)}$. Thus, ETH (resp. rETH) puts both the problems PolSat($\mathbf{A}$) and 2-ListPolSat($\mathbf{A}$) outside P (resp. RP). Indeed, assume that PolSat($\mathbf{A}$) or 2-ListPolSat($\mathbf{A}$) could be decided in (randomized) time $2^{o((\log m/\log\log m)^2)}$ where m is the length of the input polynomial. Then we could decide a 3-CNF-SAT instance of length ℓ in (randomized) time

$$2^{o((\log 2^{O(\sqrt{\ell}\cdot\log\ell)}/\log\log 2^{O(\sqrt{\ell}\cdot\log\ell)})^2)} = 2^{o(\sqrt{\ell}^2\cdot\log^2\ell/\log^2(\sqrt{\ell}\cdot\log\ell))} = 2^{o(\ell)}.$$ ◀

5 A Dichotomy for ProgramSat

Proof of Theorem 1. First, consider the case that $\mathbf{G}$ has a normal p-subgroup $\mathbf{G}_p$ such that $\mathbf{G}/\mathbf{G}_p$ is nilpotent (possibly $\mathbf{G}_p$ is trivial). Then $\mathbf{G}/\mathbf{G}_p = \mathbf{H}_p \times \mathbf{H}$ for some maximal p-group $\mathbf{H}_p$ (also $\mathbf{H}_p$ might be trivial). We denote the preimage of $\mathbf{H}_p$ in $\mathbf{G}$ by $\widetilde{\mathbf{G}}_p = \mathbf{H}_p\mathbf{G}_p$. Since $|H|$ and $\left|\widetilde{G}_p\right|$ are coprime, by the Schur-Zassenhaus theorem (Fact 7), we conclude that $\mathbf{G} = \widetilde{\mathbf{G}}_p \rtimes \mathbf{H}$ and so, by Corollary 12, ListPolSat($\mathbf{G}$) and ProgramSat($\mathbf{G}$) are in RP under CDH.

On the other hand, assume that $\mathbf{G}$ is not of the above form. If $\mathbf{G}$ is non-solvable, ListPolSat($\mathbf{G}$) and ProgramSat($\mathbf{G}$) are NP-complete by [14], hence, not in RP under rETH. If $\mathbf{G}$ is solvable but does not have a nilpotent normal subgroup $\mathbf{N}$ with nilpotent quotient $\mathbf{G}/\mathbf{N}$, then [23] (for certain cases also [20, 30]) shows that PolSat($\mathbf{G}$) (hence, also ListPolSat($\mathbf{G}$) and ProgramSat($\mathbf{G}$)) is not in P under ETH. The same proof shows that these problems are not in RP under rETH (as a randomized algorithm for ListPolSat($\mathbf{G}$) or ProgramSat($\mathbf{G}$) would lead to a randomized algorithm for 3-CNF-SAT).

Finally, let $\mathbf{N}$ denote the smallest normal subgroup such that $\mathbf{G}/\mathbf{N}$ is nilpotent. Such an $\mathbf{N}$ exists and it is nilpotent as we excluded already all the other cases. Moreover, we know that $|N|$ has at least two distinct prime divisors p and q since otherwise, we would be in the RP case. Notice that $\mathbf{N}$ is the direct product of its Sylow subgroups. Thus, after taking

a quotient (recall that ListPolSat($\mathbf{G}/\mathbf{H}$) $\leqslant$ ListPolSat($\mathbf{G}$) and ProgramSat($\mathbf{G}/\mathbf{H}$) $\leqslant$ ProgramSat($\mathbf{G}$), see Lemma 8), we may assume that there are precisely two non-trivial normal subgroups $\mathbf{A}$ and $\mathbf{B}$ of $\mathbf{G}$ below $\mathbf{N}$ and $|A| = p^\alpha$ and $|B| = q^\beta$ for some $\alpha, \beta \geqslant 1$ and $\mathbf{N} = \mathbf{A} \times \mathbf{B}$. Clearly $\mathbf{A}$ and $\mathbf{B}$ are join-irreducible.

Now assume for a contradiction that $C_{\mathbf{G}}(\mathbf{A}) = G$. Observe that $C_{\mathbf{G}}(\mathbf{A}) = C_G(\mathbf{N}/\mathbf{B})$ because $[g, a] = 1$ if and only if $[g, ab] \in B$ for $b \in B$. This means that $\mathbf{N}/\mathbf{B}$ is in the center of $\mathbf{G}/\mathbf{B}$. Thus, as $\mathbf{G}/\mathbf{N} = (\mathbf{G}/\mathbf{B})/(\mathbf{N}/\mathbf{B})$ is nilpotent, this implies that $\mathbf{G}/\mathbf{B}$ is nilpotent contradicting that $\mathbf{N}$ is the smallest normal subgroup such that $\mathbf{G}/\mathbf{N}$ is nilpotent.

By symmetry we also have $C_{\mathbf{G}}(\mathbf{B}) \neq G$. Since $\mathbf{A}$ and $\mathbf{B}$ are collaborating, we have verified the requirements of Theorem 16 showing that 2-ListPolSat($\mathbf{G}$) is not in RP under rETH. By Lemma 8 also ListPolSat($\mathbf{G}$) and ProgramSat($\mathbf{G}$) are not in RP under rETH. ◀

6 Unconditional Algorithms

Proving lower bounds for a non-trivial computational model is usually a challenging task. Rare examples of results of such kind are either proven in some very restricted settings, or rely on additional assumptions. The same issue affects programs over groups and their expressiveness of AND, where essentially our knowledge can be summarized as follows:

(B) For a nilpotent group $\mathbf{N}$ there is a constant $d_{\mathbf{N}}$ such that there is no $\mathbf{N}$-program for the n-ary AND function for $n \geqslant d_{\mathbf{N}}$ ([5, Corollary to Theorem 6]).

(C) If $\mathbf{Q} = \mathbf{G}_q \rtimes \mathbf{A}$ for some q-group $\mathbf{G}_q$ and abelian group $\mathbf{A}$, then $\gamma_{\mathrm{Prog},\mathbf{Q}}(n) \in 2^{\Omega(n)}$ ([5, Corollary to Theorem 9], recall the definition of $\gamma_{\mathrm{Prog},\mathbf{Q}}(n)$ in Section 3).

Lemma 10 implies that these two cases lead to RP algorithms for ProgramSat, ListPolSat, and PolSat. In fact, for ProgramSat for nilpotent groups and for PolSat, in both cases (B) and (C), even polynomial time algorithms have been obtained [14, 12]; however, as for now, ProgramSat in case (C) is only known to have quasi-polynomial time algorithms [4].

The main aim of the next theorem is to present a new lower bound result for groups that are direct products of these presented in (B), (C), which then by Lemma 10 also leads to RP algorithms. To the best of our knowledge this is the first result going beyond the cases (B) and (C) in this direction.

▶ **Theorem 17.** *Let $\mathbf{N}$ be a nilpotent group, $\mathbf{A}$ an abelian group, $\mathbf{Q} = \mathbf{G}_q \rtimes \mathbf{A}$ for some q-group $\mathbf{G}_q$ and let $\mathbf{G} \leqslant \mathbf{N} \times \mathbf{Q}$. Then $\gamma_{\mathrm{Prog},\mathbf{G}}(n) \in 2^{\Omega(n)}$.*

The fact that this lower bound applies also to subgroups of the product will allow us to construct a series of natural examples of groups for which we achieve efficient algorithms. Of particular interest is the case that $\mathbf{A}$ is also a subgroup of $\mathbf{N}$, so there is some non-trivial interaction between $\mathbf{N}$ and $\mathbf{Q}$.

The proof of Theorem 17 relies on a similar construction as used in [7] for showing that for every low degree polynomial there is a large affine subspace on which the polynomial is constant. For the proof we need some preparation: For $\overline{a}, \overline{b} \in \prod_{i=1}^n |A_i|$ with $\overline{a} = (a_1, \ldots, a_n)$ and $\overline{b} = (b_1, \ldots, b_n)$ we define $\mathrm{HammingDist}(\overline{a}, \overline{b}) = |\{\, i \in [1\,..\,n] \mid a_i \neq b_i \,\}|$. The following easy combinatorial observation has been used in slightly different forms in [4, Theorem 2] or [21, Theorem 6.1].

▶ **Fact 18.** *Let f be an n-ary indicator function with domain $\mathcal{A} = \prod_{i=1}^n A_i$ with $|A_i| \geqslant 2$ for all i and $\mathrm{size}(f) < \gamma(n)$. Let $\overline{b} \in f^{-1}(1)$. Then there is some $\overline{a} \in f^{-1}(1)$ with $\overline{b} \neq \overline{a}$ and $\mathrm{HammingDist}(\overline{a}, \overline{b}) \leqslant \gamma^{-1}(\mathrm{size}(f)) + 1$.*

Proof. As $\mathrm{size}(f) < \gamma(n)$, there is some $\overline{c} \in f^{-1}(1)$ with $\overline{c} \neq \overline{b}$. Writing $\overline{b} = (b_1, \ldots, b_n)$ and $\overline{c} = (c_1, \ldots, c_n)$ that means $b_i \neq c_i$ for some i. Now, set $\overline{b}' = (b_1, \ldots, b_{i-1}, c_i, b_{i+1}, \ldots, b_n)$, i.e., $\overline{b}'$ agrees with $\overline{b}$ on all but the i-th coordinate. Consider some $\overline{a}$ with $f[x_i/c_i](\overline{a}) = 1$ of minimal Hamming distance k to $\overline{b}'$ (note that possibly $\overline{a} = \overline{b}'$). Then setting all variables on which $\overline{a}$ and $\overline{b}'$ agree to this constant and restricting all other variables x_j (with $a_j \neq b_j$) to $\{a_j, b_j\}$, we obtain a k-ary spike. Thus, $\mathrm{size}(f) \geqslant \gamma(k)$ and Fact 18 follows. ◀

Proof of Theorem 17. It is enough to consider $\mathbf{G} = \mathbf{N} \times \mathbf{Q}$, as lower bounds for subgroups are inferred from their containing group. We can think of a $\mathbf{G}$-program in the direct product as a system of two separate programs sharing the same variables. Thus, let $\mathbf{p}$ be an $\mathbf{N}$-program and $\mathbf{q}$ a $\mathbf{Q}$-program with variables $x_1, \ldots, x_n$ for a spike (i.e., the n-ary AND function). Without loss of generality, we can assume that the all-zero vector $\overline{0}$ is the only satisfying assignment to the conjunction of $\mathbf{p}$ and $\mathbf{q}$ (we can achieve this by simply replacing variables by their negations if necessary – this does not increase the size of the programs). Thus, we can also associate $\mathbf{p}, \mathbf{q}$ with one-element accepting sets $\{s_{\mathbf{p}}\}, \{s_{\mathbf{q}}\}$ respectively. Now we show that $|\mathbf{q}| \in 2^{\Omega(n)}$.

We will identify a vector $\overline{b} \in \{0,1\}^n$ with the subset $\overline{b}^{-1}(1)$ of $[1\,..\,n]$. By (B) there is a constant d such that no $\mathbf{N}$-program can be a d-ary spike. We are going to exploit this fact by proving that, if $\mathbf{q}$ is relatively short, then there must be a relatively large boolean cube on which $\mathbf{p}$ behaves like an AND. Note that for $\overline{b}_1, \ldots, \overline{b}_d \in \{0,1\}^n$ with $\overline{b}_i \cap \overline{b}_j = \varnothing$ for all $i \neq j$ we can simulate the behaviour of $\mathbf{p}$ on $B = \left\{ \sum_{i=1}^d \alpha_i \overline{b}_i \;\middle|\; \alpha_i \in \{0,1\} \right\}$ by creating a new d-ary program $\hat{\mathbf{p}}(y_1, \ldots, y_d) = \mathbf{p}(\sum_{i=1}^d y_i \overline{b}_i)$. Indeed, consider an instruction $\langle j, g, h\rangle$ of $\mathbf{p}$: if j is not in any of the $\overline{b}_i$ just replace it with constant g and if $j \in \overline{b}_i$ for some i replace it with the instruction $\langle i, g, h\rangle$. To find a cube of our interest we start with the following claim.

▷ Claim 19. Let $\tilde{\mathbf{Q}} = \mathbf{Q}^{2^d}$. If $|\mathbf{q}| < \gamma_{\mathrm{Prog},\tilde{\mathbf{Q}}}(n/d - 1)$, then there are $\overline{b}_1, \ldots, \overline{b}_d \in \{0,1\}^n$ with $\overline{b}_i \cap \overline{b}_j = \varnothing$ for all $i \neq j$ such that $\mathbf{q}(\overline{x}) = 1$ for all $\overline{x} \in \left\{ \sum_{i=1}^d \alpha_i \overline{b}_i \;\middle|\; \alpha_i \in \{0,1\} \right\} \subseteq \{0,1\}^n$.

Proof. For $k \in [1\,..\,d]$ consider the group $\mathbf{Q}_k = \mathbf{Q}^{2^k}$. As having more coordinates clearly gives more expressive power to a program, we have $\gamma_{\mathrm{Prog},\tilde{\mathbf{Q}}}(n) \leqslant \gamma_{\mathrm{Prog},\mathbf{Q}_k}(n)$.

Assume we already constructed $\overline{b}_1, \ldots, \overline{b}_k$ for some $k \in [0\,..\,d-1]$ (for $k = 0$ that means we have no $\overline{b}_i$'s). Let $X_k = \bigcup_{i=1}^k \overline{b}_i$. We additionally require by induction that $|X_k| \leqslant kn/d$. First observe that a new vector $\overline{b}_{k+1}$ which could extend the sequence $\overline{b}_1, \ldots, \overline{b}_k$ needs to satisfy a system of 2^k equations $\mathbf{q}(\alpha_1\overline{b}_1 + \alpha_2\overline{b}_2 + \ldots + \alpha_k\overline{b}_k + \overline{x}) = 1$, one equation for each $\overline{\alpha} \in \{0,1\}^k$. As we expect $\overline{b}_{k+1}$ to have disjoint support with all preceding $\overline{b}_i$'s, we just put $x_i = 0$ for $i \in X_k$, so that each $\mathbf{q}_\alpha(\overline{x}) = \mathbf{q}(\alpha_1\overline{b}_1 + \ldots + \alpha_k\overline{b}_k + \overline{x})$ is of arity $n - |X_k|$. Notice that we can encode those 2^k conditions as one program condition in the group $\mathbf{Q}_k = \mathbf{Q}^{2^k}$. To produce a program $\hat{\mathbf{q}}(\overline{x}) = (\mathbf{q}_\alpha(\overline{x}))_{\alpha \in \{0,1\}^k}$ with associated accepting set $\{s_{\mathbf{q}}\}^{2^k}$, we replace each instruction of $\mathbf{q}$ with one in the new domain: whenever $j \notin X_k$ we just replace $\langle j, g, h\rangle$ with $\langle j, (g, \ldots, g), (h, \ldots, h)\rangle$ and whenever $j \in \overline{b}_i$ (for some $i \in [1\,..\,k]$) we replace it by the constant $(c_\alpha)_{\alpha \in \{0,1\}^k}$, where $c_\alpha = g$ when $\alpha_i = 0$ and $c_\alpha = h$ when $\alpha_i = 1$.

Since $|\hat{\mathbf{q}}| \leqslant |\mathbf{q}| \leqslant \gamma_{\mathrm{Prog},\tilde{\mathbf{Q}}}(n/d - 1)$ and $n - |X_k| \geqslant \frac{n}{d}$, we know by Fact 18 that not only $\overline{0}$ is a solution to $\hat{\mathbf{q}}(x) = 1$ but we have another solution with Hamming weight at most $\gamma^{-1}_{\mathrm{Prog},\tilde{\mathbf{Q}}}(|\hat{\mathbf{q}}|) + 1 \leqslant \gamma^{-1}_{\mathrm{Prog},\tilde{\mathbf{Q}}}(\gamma_{\mathrm{Prog},\tilde{\mathbf{Q}}}(n/d - 1)) + 1 = n/d$. We can clearly choose this solution to become $\overline{b}_{k+1}$ and finish the induction by noticing that $|X_{k+1}| = |X_k| + \left|\overline{b}_{k+1}\right| \leqslant kn/d + n/d = (k+1)n/d$. ◁

Finally, by (C), we know that $\gamma_{\mathrm{Prog},\tilde{\mathbf{Q}}}(n) \geqslant 2^{\delta n - C}$ for some suitable constants δ and C. Assume for a contradiction that $|\mathbf{q}| \leqslant 2^{\delta n/d - C - d - 1} \leqslant \gamma_{\mathrm{Prog},\tilde{\mathbf{Q}}}(n/d - 1)$. By Claim 19 we obtain a boolean cube $B = \left\{ \sum_{i=1}^d \alpha_i \overline{b}_i \;\middle|\; (\alpha_1, \ldots, \alpha_d) \in \{0,1\}^d \right\} \subseteq \{0,1\}^n$ with $\mathbf{q}(B) = \{1\}$.

It means that the equation $\mathbf{p}(x) = 1$ has only one solution $\overline{0}$ in the set B, otherwise the system $(\mathbf{p}(\overline{x}), \mathbf{q}(\overline{x}))$ would not define the spike (with accepting set $\{(s_{\mathbf{p}}, s_{\mathbf{q}})\}$). But now to get a contradiction define a program $\hat{\mathbf{p}}(y_1, \ldots, y_d) = \mathbf{p}(y_1\overline{b}_1 + \ldots + y_d\overline{b}_d)$, which must be a d-ary spike – contrary to the choice of d. ◀

As an immediate consequence of Theorem 17 and Lemma 10 we get the following.

▶ **Corollary 20.** *Let* $\mathbf{N}$ *be a nilpotent group,* $\mathbf{A}$ *abelian,* $\mathbf{Q} = \mathbf{G}_q \rtimes \mathbf{A}$ *for some* q*-group* $\mathbf{G}_q$ *and let* $\mathbf{G} \leqslant \mathbf{N} \times \mathbf{Q}$*. Then* ProgramSat($\mathbf{G}$)*,* ListPolSat($\mathbf{G}$) *or* PolSat($\mathbf{G}$) *have the none-or-many property.*

In particular, ProgramSat($\mathbf{G}$)*,* ListPolSat($\mathbf{G}$) *and* PolSat($\mathbf{G}$) *are in* RP*.*

Proof of Corollary 5. We can apply Corollary 20 to dihedral groups of order $2^\alpha p^\beta$. Indeed, each such group is a subgroup of $\mathbf{D}_{2^\alpha} \times \mathbf{D}_{p^\beta}$, where $\mathbf{D}_{2^\alpha}$ is nilpotent and $\mathbf{D}_{p^\beta}$ is isomorphic to the semidirect product $\mathbb{Z}_{p^\beta} \rtimes \mathbb{Z}_2$. So for each such group PolSat is in RP (if $\alpha \in \{0, 1\}$ it is even in P by [18, Corollary 2]). On the other hand, all other dihedral groups $\mathbf{D}_m$ have a quotient $\mathbf{D}_k$ where k is odd and has exactly two different prime divisors. By [21, Theorem 7.1], PolSat($\mathbf{D}_k$) is not in P under ETH. The same argument also shows that it is not in RP under rETH. By Lemma 8 this transfers to $\mathbf{D}_m$. We can see this also as a consequence of Theorem 2: the minimal normal subgroups $\mathbf{A}$ and $\mathbf{B}$ are just cyclic subgroups of $\mathbb{Z}_m$ of coprime odd order. The centralizer of both of them is $\mathbb{Z}_m < \mathbf{D}_m$. Hence, Theorem 2 tells us that PolSat($\mathbf{D}_m$) is not in RP under rETH. ◀

The positive case of the proof of Corollary 5 obviously generalizes to groups $\mathbf{G}$ with a nilpotent normal subgroup $\mathbf{N}$ of order $p^\alpha q^\beta$ such that $\mathbf{G}/\mathbf{N}$ is abelian of order p^γ (just apply the Schur-Zassenhaus Theorem, Fact 7). In particular, this applies as follows:

▶ **Example 21.** Note that the complexity of PolSat for three natural examples of order 24, namely the dihedral group $\mathbf{D}_{12}$, the quaternion group $\mathbf{Q}$ acting over $\mathbb{Z}_3$ (i.e. $\mathbb{Z}_3 \rtimes \mathbf{Q}$), and the group $(\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_3) \rtimes \mathbb{Z}_2$ was left unsolved in [12, Problem 3]. Now our Corollary 20 covers all of these examples.

7 Extending Randomized Algorithms for PolSat

The smallest example of a group of Fitting length two for which we know superpolynomial lower bounds under ETH is $\mathbf{D}_{15}$ [21]. We can embed $\mathbf{D}_{15}$ into the group $\mathbf{D}_3 \times \mathbf{D}_5$ which has polynomial-time decidable PolSat. On the contrary, both ProgramSat and ListPolSat share superpolynomial complexities under ETH in this case. Moreover, for a given group $\mathbf{G}$, under assumption of CDH and ETH the problem ProgramSat($\mathbf{G}$) is in P whenever ListPolSat($\mathbf{G}$) is. So, in a sense, complexities of ProgramSat and ListPolSat seem to coincide, while the complexity of PolSat may differ in certain cases. It is due to the fact that upper bounds for PolSat are not inherited by the subgroups in a very strong way:

▶ **Observation 22.** *Every group of Fitting length two can be embedded into a group with* PolSat *in* RP *under CDH. Moreover, if* $\mathbf{G}$ *is nilpotent-by-abelian, it can be embedded into a group with* PolSat *in* P *(unconditionally).*

Proof. Let $\mathbf{G}$ be a group of Fitting length two. Thus, there is some nilpotent normal subgroup $\mathbf{N}$ such that $\mathbf{G}/\mathbf{N}$ is nilpotent. Since $\mathbf{N}$ is nilpotent, we can write it as a direct product $\mathbf{N} = \mathbf{G}_{p_1} \times \cdots \times \mathbf{G}_{p_k}$ for p_i-groups $\mathbf{G}_{p_i}$. Let $\mathbf{N}_i = \prod_{j \neq i} \mathbf{G}_{p_j}$. Then $\mathbf{G}_{p_i}$ is a

normal nilpotent subgroup of $\mathbf{G}/\mathbf{N}_i$ and $(\mathbf{G}/\mathbf{N}_i)/\mathbf{G}_{p_i}$ is nilpotent. Thus, by Theorem 1 PolSat$(\mathbf{G}/\mathbf{N}_i)$ is in RP under CDH. Finally, notice that $\mathbf{G}$ embeds into the direct product $\prod_{i=1}^{k} \mathbf{G}/\mathbf{N}_i$, which by Lemma 8 has also PolSat in RP under CDH.

For the second part just observe that, if $\mathbf{G}$ is nilpotent-by-abelian, $\mathbf{G}/\mathbf{N}_i$ has a normal p_i-subgroup with abelian quotient and so by [12, Theorem 1] PolSat is in P. ◀

The unusual properties of PolSat allow us to create larger classes of groups with polynomial time algorithms than for the other two problems.

Proof of Theorem 4. We start with some preparation and decompose $\mathbf{G}$ into smaller groups. Since $|G|$ has only two prime factors, say p and q, we can write $\mathbf{N} = \mathbf{N}_q \times \mathbf{N}_p$ and $\mathbf{H} = \mathbf{H}_p \times \mathbf{H}_q$ where $\mathbf{N}_p, \mathbf{H}_p$ are p-groups and $\mathbf{N}_q, \mathbf{H}_q$ are q-groups. Notice that $\mathbf{N}_p$ and $\mathbf{N}_q$ are also normal in $\mathbf{G}$ (this is because they are characteristic subgroups of $\mathbf{N}$). We define subsets L, R, P, Q of G by putting $L = N_q H_p$, $R = N_p H_q$, $P = N_p H_p$ and $Q = N_q H_q$. Notice that we defined indeed subgroups $\mathbf{L}, \mathbf{R}, \mathbf{P}$, and $\mathbf{Q}$ since each of them is a product of a normal subgroup and a subgroup (though they might not be normal subgroups) and $LR = G = PQ$. In particular, we can write each $g \in G$ uniquely as $g = \ell r$ where $\ell \in L$ and $r \in R$. Moreover, they are all semidirect products: $\mathbf{L} = \mathbf{N}_q \rtimes \mathbf{H}_p$, $\mathbf{R} = \mathbf{N}_p \rtimes \mathbf{H}_q$, $\mathbf{P} = \mathbf{N}_p \rtimes \mathbf{H}_p$ and $\mathbf{Q} = \mathbf{N}_q \rtimes \mathbf{H}_q$, where the actions of $\mathbf{H}_p$ and $\mathbf{H}_q$ on $\mathbf{N}_p$ and $\mathbf{N}_q$ are the restrictions of the actions of $\mathbf{H}$ on $\mathbf{N}$.

We will now prove the many-solutions property for $\mathbf{G}$ by restricting the variables to the subgroups $\mathbf{L}$ and $\mathbf{R}$. However, we cannot apply Corollary 12 directly; instead, we will construct another group $\mathbf{G}_1$ with the desired properties for which we can apply Corollary 12 for ListPolSat.

We define $\mathbf{G}_1 = \mathbf{P} \rtimes \mathbf{Q}$ where the action is given by the action of $\mathbf{H}_q \leqslant \mathbf{Q}$ on $\mathbf{N}_p \leqslant \mathbf{P}$ (and $\mathbf{P}$ and $\mathbf{N}_q \leqslant \mathbf{Q}$ commute). Notice that there is a canonical bijection (in general not an isomorphism) between G and G_1 and we have $\mathbf{R} \leqslant \mathbf{G}_1$. Moreover, notice that $\mathbf{G}_1$ meets the requirements of Corollary 12 and $\mathbf{G}/\mathbf{N}_q = (\mathbf{N}_p \rtimes (\mathbf{H}_p \times \mathbf{H}_q) = (\mathbf{N}_p \rtimes \mathbf{H}_p) \rtimes \mathbf{H}_q = \mathbf{G}_1/\mathbf{N}_q$.

Our next step is to transform a polynomial $\mathbf{q} \in \mathrm{Pol}(\mathbf{G})$ to a polynomial $\theta(\mathbf{q}) \in \mathrm{Pol}(\mathbf{G}_1)$ which, when restricting variables to R, has the same solution set (this makes sense as $\mathbf{R}$ is both a subgroup of $\mathbf{G}$ and $\mathbf{G}_1$).

In order to do so, write $\mathbf{q}(\overline{x}) = g_0 h_0 \xi_1 g_1 h_1 \cdots \xi_m g_m h_m$ where $g_i \in N_q$ and $h_i \in H_p$ (i.e., $g_i h_i \in L$) and the ξ_i are constants from $\mathbf{R} = \mathbf{N}_p \rtimes \mathbf{H}_q$ or variables. We define $\theta : \mathrm{Pol}(\mathbf{G}) \to \mathrm{Pol}(\mathbf{G}_1)$ by $\theta(\mathbf{q}) = g_0 h_0 \xi_1 ({}^{h_0}g_1) h_1 \cdots \xi_m ({}^{h_0 \cdots h_{m-1}}g_m) h_m$, where ${}^{h}g = hgh^{-1}$ denotes the action of $h \in H_p$ on $g \in N_q$ (considered in $\mathbf{G}$); we view ${}^{h}g$ as a fixed element of $N_q \leqslant G_1$ and forget that it comes from the action in $\mathbf{G}$. Notice that up to a constant factor $|\theta(\mathbf{q})|$ and $|\mathbf{q}|$ are equal. We say $\overline{x}$ is a solution of $\mathbf{q}$ if $\mathbf{q}(\overline{x}) = 1$.

▷ Claim 23. Let $\overline{x} \in R^n$. Then $\overline{x}$ is a solution of $\mathbf{q}$ if and only if $\overline{x}$ is a solution of $\theta(\mathbf{q})$.

Proof. As $\mathbf{G}/\mathbf{N}_q = \mathbf{G}_1/\mathbf{N}_q$, we may assume that $\mathbf{q}(\overline{x})$ and $\theta(\mathbf{q})(\overline{x})$ are both in N_q. Then, in $\mathbf{G}$ we have

$$\begin{aligned}\mathbf{q}(\overline{x}) &= g_0 h_0 \xi_1 g_1 h_1 \cdots \xi_m g_m h_m \\ &= g_0 \, {}^{h_0\xi_1}g_1 \cdots {}^{h_0\xi_1 \cdots h_{m-1}\xi_m}g_m \cdot h_0 \xi_1 \cdots h_{m-1}\xi_m h_m \\ &= g_0 \, {}^{h_0\xi_1}g_1 \cdots {}^{h_0\xi_1 \cdots h_{m-1}\xi_m}g_m.\end{aligned}$$

On the other hand, in $\mathbf{G}_1$ we have

$$\begin{aligned}\theta(\mathbf{q})(\overline{x}) &= g_0 h_0 \xi_1 ({}^{h_0}g_1) h_1 \cdots \xi_m ({}^{h_0 \cdots h_{m-1}}g_m) h_m \\ &= g_0 \, {}^{\xi_1}({}^{h_0}g_1) \cdots {}^{\xi_1 \cdots \xi_m}({}^{h_0 \cdots h_{m-1}}g_m) \cdot h_0 \xi_1 \cdots h_{m-1}\xi_m h_m \\ &= g_0 \, {}^{\xi_1}({}^{h_0}g_1) \cdots {}^{\xi_1 \cdots \xi_m}({}^{h_0 \cdots h_{m-1}}g_m).\end{aligned}$$

Now, we can read the last line as an element of $\mathbf{G}$ interpreting ${}^{h}g = hgh^{-1}$ again as the action of $h \in H_p$ on $g \in N_q$. As in $\mathbf{G}$ the ξ_i commute with h_i modulo $\mathbf{N}_p$, which is contained in the centralizer of $\mathbf{N}_q$ in $\mathbf{G}$, we conclude that as an equality in $\mathbf{G}$ we have

$$\theta(\mathbf{q})(\overline{x}) = g_0 {}^{h_0\xi_1}g_1 \cdots {}^{h_0\xi_1\cdots h_{m-1}\xi_m}g_m.$$

This proves the claim. ◁

If a polynomial $\mathbf{q} \in \mathrm{Pol}(\mathbf{G})$ has a solution with variables restricted to R, by Claim 23, $\theta(\mathbf{q})$ also has a solution with variables restricted to R. Now, we can apply Corollary 12 (in the CDH case) or Corollary 20 (if $\mathbf{H}$ is abelian), which gives us that a polynomial fraction $1/|\theta(\mathbf{q})|^{\mathcal{O}(1)}$ of all assignments $\overline{y} \in R^n$ are satisfying for $\theta(\mathbf{q})$ (i.e., there are at least $|R|^n/|\theta(\mathbf{q})|^{\mathcal{O}(1)}$ satisfying assignments among $|R|^n$ possible assignments). By Claim 23 also at least a polynomial fraction $1/|\mathbf{q}|^{\mathcal{O}(1)}$ of all assignments $\overline{y} \in R^n$ are satisfying for $\mathbf{q}$.

By symmetry the same argument applies to a polynomial with variables restricted to L: if a polynomial $\mathbf{q} \in \mathrm{Pol}(\mathbf{G})$ with variables restricted to L has a solution, at least $1/|\mathbf{q}|^{\mathcal{O}(1)}$ of all assignments $\overline{y} \in L^n$ are satisfying for $\mathbf{q}$.

Since $\mathbf{G} = \mathbf{LR}$, we can show the none-or-many property for $\mathbf{G}$ as follows: assume $\mathbf{p}$ is a polynomial with a solution $\overline{a} = (a_1, \ldots, a_n)$. We can write each $a_i = \ell_i r_i$ with $\ell_i \in L$ and $r_i \in R$. Let $\mathbf{q}$ be the polynomial obtained from $\mathbf{p}$ by substituting every variable x_i by $\ell_i y_i$ where y_i is a new variable. We know that $\mathbf{q}$ has a solution when restricting all variables to R – hence, it has at least $|R|^n/|\mathbf{p}|^{\mathcal{O}(1)}$ solutions in R^n. For each of these solutions $\overline{r}' = (r'_1, \ldots, r'_n) \in R^n$ again we obtain a polynomial $\mathbf{r}$ from $\mathbf{p}$ by replacing each variable x_i by $z_i r'_i$ where z_i is a new variable restricted to L. Now, $\mathbf{r}$ has at least $|L|^n/|\mathbf{p}|^{\mathcal{O}(1)}$ many solutions. Since any of these solutions gives us a solution to $\mathbf{p}$, we obtain at least $|R|^n/|\mathbf{p}|^{\mathcal{O}(1)} \cdot |L|^n/|\mathbf{p}|^{\mathcal{O}(1)}$ solutions for $\mathbf{p}$.

Therefore, picking random assignments leads to an RP algorithm (like in Lemma 10). ◀

A straightforward (though not the smallest) example for Theorem 4 not covered by previous results is the wreath product $\mathbb{Z}_6 \wr \mathbb{Z}_6 = (\mathbb{Z}_6)^6 \rtimes \mathbb{Z}_6$. By Theorem 4 we know that PolSat is in RP for this group, whereas ProgramSat is not in RP under rETH by Theorem 1.

As we show in Corollary 6, we can even classify the complexity of PolSat for arbitrary wreath products. Before we outline the proof, let us remark that, if $\mathbf{G}$ is nilpotent and $\mathbf{H}$ abelian, then PolSat$(\mathbf{G} \wr \mathbf{H})$ is in RP as soon as $\mathbf{G}$ is a p-group or $|G|$ and $|H|$ only have the same two prime divisors – without requiring CDH. This is an immediate consequence of Corollary 20 and Theorem 4.

Proof sketch of Corollary 6. The CDH-based RP algorithms are due to Corollary 12 and Theorem 4. Being not in the RP case, $|G|$ has at least two prime divisors $q \neq r$. Moreover, $|H|$ has a third prime divisor $p \neq q, r$. Thus, we will find a wreath product $(\mathbb{Z}_q \times \mathbb{Z}_r) \wr \mathbb{Z}_p$ as a subgroup of a quotient of $\mathbf{G} \wr \mathbf{H}$. By [26, Theorm 4.1.10] neither $\mathbb{Z}_q \wr \mathbb{Z}_p$ nor $\mathbb{Z}_r \wr \mathbb{Z}_p$ is nilpotent. Thus, we can find covering pairs of normal subgroups $\mathbf{B}_q, \mathbf{A}_q$ and $\mathbf{B}_r, \mathbf{A}_r$ such that $C_{\mathbb{Z}_q \wr \mathbb{Z}_p}(\mathbf{B}_q/\mathbf{A}_q) \neq \mathbb{Z}_q \wr \mathbb{Z}_p$. It remains to lift them to $\mathbf{G} \wr \mathbf{H}$ and apply Corollary 3. ◀

8 Conclusion

In this paper, under the assumptions of rETH and CDH, we fully classified in which cases the computational complexity of ProgramSat and ListPolSat for finite groups is in RP. It seems that eliminating the assumptions (especially rETH) can be really hard, but there is still a chance to improve Theorem 1 by showing polynomial time deterministic algorithms instead of the randomized ones:

▶ **Problem 1.** *Is there a polynomial time* deterministic *algorithm solving* ProgramSat($\mathbf{G}$) *and* ListPolSat($\mathbf{G}$) *for* $\mathbf{G}$ *such that there is a prime* p *and a normal* p*-subgroup* $\mathbf{G}_p$ *of* $\mathbf{G}$ *with* $\mathbf{G}/\mathbf{G}_p$ *being nilpotent?*

We took a step towards full classification of the complexity of PolSat for finite groups. Our study reveals that the interactions of normal subgroups of different characteristics play a crucial role. To conclude we present an example of a group of Fitting length 2 for which the complexity of PolSat can not be resolved by our results.

▶ **Problem 2.** *What is the computational complexity of* PolSat($\mathbf{G}$) *for*

$$\mathbf{G} = (\mathbb{Z}_3 \times \mathbb{Z}_5 \times \mathbb{Z}_7) \rtimes (\mathbb{Z}_2 \times \mathbb{Z}_2),$$

where the first $\mathbb{Z}_2$ *acts on* $\mathbb{Z}_3 \times \mathbb{Z}_5$ *by inversion and the second* $\mathbb{Z}_2$ *acts on* $\mathbb{Z}_5 \times \mathbb{Z}_7$ *by inversion?*

Note that the group $\mathbf{G}$ from Problem 2 has $\mathbb{Z}_3$, $\mathbb{Z}_5$, and $\mathbb{Z}_7$ as normal subgroups of different characteristics with $C_{\mathbf{G}}(\mathbb{Z}_p) \neq G$ for $p = 3, 5, 7$ and $C_{\mathbf{G}}(\mathbb{Z}_p) \cdot C_{\mathbf{G}}(\mathbb{Z}_q) = G$ for $p \neq q$. In particular, the last property prevents us from using Theorem 2 or Corollary 3. On the other hand, four different primes dividing the size of $\mathbf{G}$ blocks Theorem 4 from being applied here. Moreover, also Corollary 12 cannot be applied here since the largest nilpotent quotient of $\mathbf{G}$ is $\mathbb{Z}_2 \times \mathbb{Z}_2$ and the kernel of the projection is clearly not a p-group.

References

1. David A. Mix Barrington. Width-3 permutation branching programs. Technical Report TM-293, MIT Laboratory for Computer Science, 1985.
2. David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proceedings of STOC'86*, pages 1–5, 1986. doi:10.1145/12130.12131.
3. David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing Boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4:367–382, 1994. doi:10.1007/BF01263424.
4. David A. Mix Barrington, Pierre McKenzie, Cristopher Moore, Pascal Tesson, and Denis Thérien. Equation satisfiability and program satisfiability for finite monoids. In *Proceedings of MFCS'00*, pages 172–181, 2000. doi:10.1007/3-540-44612-5_13.
5. David A. Mix Barrington, Howard Straubing, and Denis Thérien. Non-uniform automata over groups. *Inf. Comput.*, 89(2):109–132, 1990. doi:10.1016/0890-5401(90)90007-5.
6. David A. Mix Barrington and Denis Thérien. Finite monoids and the fine structure of NC^1. *J. ACM*, 35(4):941–952, 1988. doi:10.1145/48014.63138.
7. Gil Cohen and Avishay Tal. Two structural results for low degree polynomials and applications. In *Proceedings of APPROX/RANDOM'15*, pages 680–709, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.680.
8. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
9. Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.
10. Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Inf. Comput.*, 202(2):105–140, 2005. doi:10.1016/j.ic.2005.04.002.

11 Attila Földvári. The complexity of the equation solvability problem over semipattern groups. *IJAC*, 27(2):259, 2017. doi:10.1142/S0218196717500126.

12 Attila Földvári and Gábor Horváth. The complexity of the equation solvability and equivalence problems over finite groups. *IJAC*, 30(03):607–623, 2020. doi:10.1142/S0218196720500137.

13 Ralph Freese and Ralph McKenzie. *Commutator Theory for Congruence Modular Varieties*. London Mathematical Society Lecture Notes, No. 125. Cambridge University Press, 1987.

14 Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Inf. Comput.*, 178(1):253–262, 2002. doi:10.1006/inco.2002.3173.

15 Vince Grolmusz. A degree-decreasing lemma for (MOD_p-MOD_m) circuits. *Discret. Math. Theor. Comput. Sci.*, 4(2):247–254, 2001. doi:10.46298/dmtcs.289.

16 Vince Grolmusz and Gábor Tardos. Lower bounds for (MOD_p-MOD_m) circuits. *SIAM J. Comput.*, 29(4):1209–1222, 2000. doi:10.1137/S0097539798340850.

17 David Hobby and Ralph McKenzie. *Structure of Finite Algebras*. Contemporary Mathematics vol. 76. American Mathematical Society, 1988. doi:10.1090/conm/076.

18 Gábor Horváth. The complexity of the equivalence and equation solvability problems over meta-Abelian groups. *J. Algebra*, 433:208–230, 2015. doi:10.1016/j.jalgebra.2015.03.015.

19 Gábor Horváth and Csaba A. Szabó. The complexity of checking identities over finite groups. *IJAC*, 16(5):931–940, 2006. doi:10.1142/S0218196706003256.

20 Paweł M. Idziak, Piotr Kawałek, and Jacek Krzaczkowski. Intermediate problems in modular circuits satisfiability. In *Proceedings of LICS'20*, pages 578–590, 2020. doi:10.1145/3373718.3394780.

21 Pawel M. Idziak, Piotr Kawalek, and Jacek Krzaczkowski. Complexity of modular circuits. In *37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (LICS '22), August 2–5, 2022, Haifa, Israel*. ACM, New York, NY, USA, 2022. doi:10.1145/3531130.3533350.

22 Pawel M. Idziak, Piotr Kawalek, and Jacek Krzaczkowski. Satisfiability of circuits and equations over finite Malcev algebras. In *Proceedings of STACS'22*, pages 37:1–37:14, 2022. doi:10.4230/LIPIcs.STACS.2022.37.

23 Pawel M. Idziak, Piotr Kawalek, Jacek Krzaczkowski, and Armin Weiß. Equation satisfiability in solvable groups. *Theory Comput. Syst.*, to appear, 2022. arXiv:2010.11788.

24 Paweł M. Idziak and Jacek Krzaczkowski. Satisfiability in multi-valued circuits. *SIAM J. Comp.*, 51(3):337–378, 2022. doi:10.1137/18M1220194.

25 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

26 J. D. P. Meldrum. *Wreath products of groups and semigroups*, volume 74 of *Pitman Monographs and Surveys in Pure and Applied Mathematics*. Longman, Harlow, 1995.

27 Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

28 Derek J. S. Robinson. *A course in the theory of groups*, volume 80 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1996. doi:10.1007/978-1-4419-8594-1.

29 Klaus U. Schulz. Makanin's algorithm for word equations – two improvements and a generalization. In *Proceedings of Word Equations and Related Topics, First International Workshop, IWWERT*, pages 85–150, 1990. doi:10.1007/3-540-55124-7_4.

30 Armin Weiß. Hardness of Equations over Finite Solvable Groups Under the Exponential Time Hypothesis. In *Proceedings of ICALP'20*, pages 102:1–102:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.102.

Linearly Ordered Colourings of Hypergraphs

Tamio-Vesa Nakajima
Department of Computer Science, University of Oxford, UK

Stanislav Živný
Department of Computer Science, University of Oxford, UK

Abstract

A linearly ordered (LO) k-colouring of an r-uniform hypergraph assigns an integer from $\{1, \dots, k\}$ to every vertex so that, in every edge, the (multi)set of colours has a unique maximum. Equivalently, for $r = 3$, if two vertices in an edge are assigned the same colour, then the third vertex is assigned a larger colour (as opposed to a different colour, as in classic non-monochromatic colouring). Barto, Battistelli, and Berg [STACS'21] studied LO colourings on 3-uniform hypergraphs in the context of promise constraint satisfaction problems (PCSPs). We show two results.

First, given a 3-uniform hypergraph that admits an LO 2-colouring, one can find in polynomial time an LO k-colouring with $k = O(\sqrt{n \log \log n}/\log n)$, where n is the number of vertices of the input hypergraph. This is established by building on ideas from algorithms designed for approximate graph colourings.

Second, given an r-uniform hypergraph that admits an LO 2-colouring, we establish NP-hardness of finding an LO 3-colouring for every constant uniformity $r \geq 5$. In fact, we determine the precise relationship of polymorphism minions for all uniformities $r \geq 3$, which reveals a key difference between $r = 3, 4$ and $r \geq 5$ and which may be of independent interest. Using the algebraic approach to PCSPs, we actually show a more general result establishing NP-hardness of finding an LO $(k+1)$-colouring for LO k-colourable r-uniform hypergraphs for $k \geq 2$ and $r \geq 5$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Problems, reductions and completeness; Theory of computation → Constraint and logic programming

Keywords and phrases hypegraph colourings, promise constraint satisfaction, PCSP, polymorphisms, minions, algebraic approach

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.128

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Extended version (with stronger results)*: https://arxiv.org/abs/2204.05628

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

1 Introduction

The computational complexity of the *approximate graph colouring* problem [16] is an outstanding open problem in theoretical computer science. Given a 3-colourable graph G on n vertices, is it possible to find a k-colouring of G? On the tractability side, the current best is a polynomial-time algorithm of Kawarabayashi and Thorup [23] that finds a k-colouring with $k = k(n) = n^{0.199}$ colours. On the intractability side, the state-of-the-art for constant k has only recently been improved from $k = 4$, due to Khanna, Linial, and Safra [24] and Guruswami and Khanna [17] to $k = 5$, due to Barto, Bulín, Krokhin, and Opršal [5]. The authors of [5] introduced a general algebraic methodology for studying the computational complexity of so-called promise constraint satisfaction problems (PCSPs). Going beyond the


work in [5], for graphs with a promised higher chromatic number than three, the current best intractability results for constantly many extra colours is due to Wrochna and Živný [27], building on the work of Huang [22].

The situation is much better understood for the approximate *hypergraph* colouring problem with the classic notion of a colouring leaving no edge monochromatic. A celebrated result of Dinur, Regev, and Smyth established that finding an ℓ-colouring of a 3-uniform hypergraph that is k-colourable is NP-hard for every constant $2 \leq k \leq \ell$ [14] (and this also implies the same result on r-uniform hypergraphs for every constant uniformity $r \geq 3$).

Different variants of approximate hypergraph colourings, such as rainbow colourings, were studied, e.g. in [2, 10, 18, 19, 12], but most complexity classifications related to these problems are open. Some intractability results are also known for colourings with a super-constant number of colours. For graphs, conditional hardness was established by Dinur and Shinkar [15]. For hypergraphs, intractability results were obtained by Bhangale [9] and by Austrin, Bhanghale, and Potukuchi [1].

Barto, Battistelli, and Berg have recently studied systematically a certain type of PCSPs on non-Boolean domains and identified a very natural variant of k-colourings of 3-uniform hypergraphs, called *linearly ordered* (LO) k-colourings [4]. A k-colouring of a 3-uniform hypergraph with colours $[k] = \{1, \dots, k\}$ is an LO colouring if, for every edge, it holds that, if two vertices are coloured with the same colour, then the third vertex is coloured with a larger colour. (In the classic non-monochromatic colouring, the requirement is that the third vertex should be coloured with a *different* colour, but not necessarily a larger one.) An LO 2-colouring is thus a "1-in-3" colouring. Barto et al. asked whether finding an LO k-colouring of a 3-uniform hypergraph is NP-hard for a fixed $k \geq 3$ if the input hypergraph is promised to admit an LO 2-colouring.

Contributions

While we do not resolve the question raised in [4], we obtain non-trivial results, both positive (algorithmic) and negative (hardness).

First, we present an efficient algorithm for finding an LO colouring of a 3-uniform hypergraph with super-constantly many colours if an LO 2-colouring is promised to exist. In more detail, for a given 3-uniform hypergraph H on n vertices that admits an LO 2-colouring, we present a polynomial-time algorithm that finds an LO k-colouring of H with $k = k(n) = O(\sqrt{n \log \log n} / \log n)$ colours. As mentioned above, there are only a few results on hypergraph colourings with super-constantly many colours.

Second, we establish intractability of finding an LO 3-colouring of an r-uniform hypergraph if an LO 2-colouring is promised for every constant uniformity $r \geq 5$. In fact, we prove a more general result that finding an LO $(k+1)$-colouring of an r-uniform hypergraph admitting an LO k-colouring is intractable for every constant $k \geq 2$ and $r \geq 5$. This result is based the algebraic approach to PCSPs and in particular on minions [5]. As a matter of fact, we establish the precise relationships of the polymorphism minions of the LO 2- vs 3-colourings on r-uniform hypergraphs for $r \geq 3$, which may be of independent interest. This gives the advertised intractability result but also an impossibility result on certain types of polynomial-time reductions (namely pp-constructions [5]) between LO 2- vs 3-colourings on r-uniform hypergraphs for $r \geq 5$ and $r = 3, 4$.

2 Preliminaries

An *r-uniform hypergraph* H is a pair (V, E) where V is the set of *vertices* of the hypergraph, and $E \subseteq V^r$ is the set of *edges* of the hypergraph. In our context the order of the vertices in each edge is irrelevant. We will allow vertices to appear multiple times in edges; however, we exclude edges of form $(v, \dots, v)$ – such edges would be impossible in the problems we will consider anyway. We say that two distinct vertices u, v are *neighbours* if they both belong to some edge $e \in E$. Let $N(u)$ be the set of neighbours of u. Call a set S an *independent set* of a hypergraph H if and only if no two members of S are neighbours.

A *linearly ordered* (LO) k-colouring of an r-uniform hypergraph $H = (V, E)$ is an assignment $c : V \to [k]$ of colours from $[k] = \{1, \dots, k\}$ to the vertices of H such that, for each edge $(v_1, \dots, v_r) \in E$, the sequence $c(v_1), \dots, c(v_r)$ has a unique maximum. We omit the "k-" if the number of colours is unimportant.

▶ **Example 1.** Consider the hypergraph $H = (V, E)$, where $V = [4] = \{1, 2, 3, 4\}$ and $E = \{(1, 2, 3), (1, 2, 4)\}$. The assignment $c = \{1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 2\}$ is an LO 2-colouring, and $c'(x) = x$ is an LO 4-colouring. On the other hand, $c''(x) = 3 - c(x)$ is not an LO colouring at all, since both of the edges have two equal maximal elements when mapped through c''.

Finding an LO k-colouring, for constant $k \geq 3$, of a 3-uniform hypergraph that admits an LO 2-colouring was studied by Barto et al. [4] in the context of promise constraint satisfaction problems (PCSPs), which we define next.

2.1 Promise CSPs

Promise CSPs have been introduced in the works of Austrin, Guruswami, and Håstad [3] and Brakensiek and Guruswami [11]. We follow the notation and terminology of Barto, Bulín, Krokhin, and Opršal [5], adapted to structures consisting of a single relation.

An *r-ary structure* is a pair $\mathbf{D} = (D, R^{\mathbf{D}})$, where $R^{\mathbf{D}} \subseteq D^r$ and D is finite. We call D the *domain* of the structure, and $R^{\mathbf{D}}$ the *relation* of the structure. For two r-ary structures $\mathbf{A}, \mathbf{B}$, a *homomorphism from* $\mathbf{A}$ *to* $\mathbf{B}$ is a function $h : A \to B$ such that, for each $(a_1, \dots, a_r) \in R^{\mathbf{A}}$, we have $(h(a_1), \dots, h(a_r)) \in R^{\mathbf{B}}$. This is written $h : \mathbf{A} \to \mathbf{B}$. If we wish to assert only the existence of such a homomorphism, we write $\mathbf{A} \to \mathbf{B}$.

We now define the search version of the fixed-template PCSP problem. Given two r-ary structures $\mathbf{A} \to \mathbf{B}$, the problem PCSP$(\mathbf{A}, \mathbf{B})$ is the following: given an r-ary structure $\mathbf{I} \to \mathbf{A}$, find a homomorphism $h : \mathbf{I} \to \mathbf{B}$. The decision version of this problem is: given an r-ary structure $\mathbf{I}$, output YES if $\mathbf{I} \to \mathbf{B}$, and output NO if $\mathbf{I} \not\to \mathbf{A}$. Observe that the decision version can be reduced to the search version: to solve the decision version, run an algorithm for the search version, then check if it gives a correct answer. We will use PCSP$(\mathbf{A}, \mathbf{B})$ to mean the decision version when proving hardness, and the search version when showing algorithmic results.

LO colourings can be readily seen as PCSPs. First, observe that an r-uniform hypergraph can be seen as an r-ary structure. Second, define an r-ary structure $\mathbf{LO}_k^r$ with domain $[k]$, and whose relation contains a tuple $(c_1, \dots, c_r)$ if and only if the sequence $c_1, \dots, c_r$ has a unique maximum. Then, an LO k-colouring of an r-uniform hypergraph $\mathbf{H}$ is the same as a homomorphism from $\mathbf{H}$ (viewed as an r-ary structure) to $\mathbf{LO}_k^r$. Thus, the problem of finding an LO k-colouring of an r-uniform hypergraph that has an LO 2-colouring is the same as PCSP$(\mathbf{LO}_2^r, \mathbf{LO}_k^r)$.

In Section 3, we study the computational complexity of $\mathrm{PCSP}(\mathbf{LO}_2^3, \mathbf{LO}_{k(n)}^3)$, were $k(n)$ depends on the input size; here n denotes the number of vertices of the input (3-uniform) hypergraph. As in Example 1, this is obviously possible for $k(n) = n$. As our first contribution, we will present an efficient algorithm with $k(n) = O(\sqrt{n \log\log n}/\log n)$.

In Section 5, we study the computational complexity of $\mathrm{PCSP}(\mathbf{LO}_k^r, \mathbf{LO}_{k+1}^r)$ for constant uniformity $r \geq 3$ and constant arity $k \geq 2$. We establish intractability of $\mathrm{PCSP}(\mathbf{LO}_k^r, \mathbf{LO}_{k+1}^r)$ for $r \geq 5$ and $k \geq 2$, cf. Corollary 23 (for $k = 2$) and Corollary 25 (for $k \geq 2$) in Section 5. These results are based on the algebraic theory of minions [5], briefly introduced in Section 4. In fact, we establish the precise relationships of the polymorphism minions of $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ for all $r \geq 3$ (cf. Theorem 22 in Section 5).

In the full version of this paper [25] we show stronger results, namely NP-hardness of $\mathrm{PCSP}(\mathbf{LO}_k^r, \mathbf{LO}_\ell^r)$ for every constant k and ℓ with $2 \leq k \leq \ell$ and every constant uniformity $r \geq \ell - k + 4$.

3 Algorithmic results

Within this section, we will prove algorithmic results for finding LO colourings of 3-uniform hypergraphs that have LO 2-colourings, using a non-constant number of colours. We will describe two algorithms: a simpler one that uses $O(\sqrt{n})$ colours; and a more complicated one that builds on the first, that uses $O(\sqrt{n \log\log n}/\log n)$ colours. The initial algorithm is inspired by Wigderson's algorithm for the approximate graph colouring problem [26]: like that algorithm it considers a "sparse case" and a "dense case"; however, our algorithm is more complex due to the fact that an LO colouring is different to a normal graph colouring. A particularly salient difference is that our algorithm picks a high-degree *edge*, whereas Wigderson's algorithm picks a high-degree *vertex*. Our second, more complex algorithm refines the first one by selecting more than one edge at a time (similarly to the work of Berger and Rompel [8], although, like Wigderson's algorithm [26], they choose several vertices at a time, not several edges as we do), and by using an improved algorithm for finding independent sets (studied by Halldórsson [21]).

In both algorithms, the general strategy will be to colour some vertices with large colours, then to recursively colour the rest of the hypergraph. The exact choice of "large colour" is made only after the recursive colouring (since only at that point do we know how large the colours need to be). We will not keep track of the bookkeeping needed to do this, in order to make the algorithms easier to explain.

3.1 First algorithm

Fix some 3-uniform hypergraph $H = (V, E)$, with n vertices and m edges. Suppose that this hypergraph admits an LO 2-colouring c^*. We first show a way to extend a partial colouring consistent with c^* until it intersects each edge of H in zero, one or three vertices.

▶ **Lemma 2.** *There exists a polynomial-time algorithm that, given a partial LO 2-colouring c that coincides with c^* on its domain, extends c into a partial LO 2-colouring* extend(c) *that does not intersect any edge of H in exactly 2 vertices and remains consistent with c^* on its domain. The algorithm will also run in polynomial time even if c isn't consistent with c^* or even an LO 2-colouring, in which case it will return an arbitrary extension of c.*

Proof. The following algorithm suffices: while c intersects an edge (x, y, z) in precisely two vertices (say x, y), extend c with $z \mapsto 4 - c(x) - c(y)$. This algorithm clearly extends c until it intersects no edges in precisely 2 vertices. Furthermore, since c^* is an LO 2-colouring, for

each edge (x, y, z) we have $c^*(x) + c^*(y) + c^*(z) = 4$. Thus, if we assume that c is initially consistent with c^*, we can show that each extension keeps c consistent with c^*. We conclude by noting that each extension can be done in polynomial time, and that there are at most n extensions – and that this is the case even if c doesn't coincide with c^* on its domain, or isn't even an LO colouring. ◀

We now show how to find a large independent set if the hypergraph is sparse enough.

▶ **Lemma 3.** *If every edge in H contains a vertex with at most $\Delta \geq 0$ neighbours, then we can find an independent set with $n/(\Delta + 1) = \Omega(n/\Delta)$ vertices in polynomial time. This holds even if Δ happens not to be an integer.*

Proof. This condition implies that there exists at least one vertex with most Δ neighbours: if there exists at least one edge then one of the vertices in it must be this vertex; otherwise, all vertices have $0 \leq \Delta$ neighbours. By adding that vertex to the independent set, removing it and its neighbours from the hypergraph, and repeating this process until no vertices remain, we can find the required independent set. ◀

Our algorithm will colour a certain part of the hypergraph with $O(1)$ colours, colouring the rest recursively, with strictly smaller colours. The following lemma gives us a sufficient goal for the size of what must be coloured in one step.

▶ **Lemma 4.** *A recursive procedure that colours $\Omega(\sqrt{n})$ vertices (where n refers to the* current *size of the hypergraph, not the initial size) with $O(1)$ colours at each step will use $O(\sqrt{n})$ colours overall.*

Proof. Let $T(n)$ be the number of colours needed to colour a hypergraph with at most n vertices. It is sufficient to prove that $T(n) = O(\sqrt{n})$ for n a power of two, since for arbitrary n we can consider the next largest power of two, which is at most $2n$. For some such n, consider how many colours are needed to colour half the hypergraph. At each step until the hypergraph is halved, we colour at least $\Omega(\sqrt{n/2}) = \Omega(\sqrt{n})$ vertices with $O(1)$ colours. Equivalently, we colour half the hypergraph with $O(\sqrt{n})$ colours. Thus we deduce that $T(n) \leq T(n/2) + O(\sqrt{n})$. Applying the Master method of Cormen et al. [13] to the recurrence $U(n) = U(n/2) + \alpha\sqrt{n}$, where α comes from the constant hidden in the recurrence for T, we can deduce that $T(n) = O(\sqrt{n})$. (Applying this method requires satisfying the "regularity condition": $\sqrt{n/2} \leq c\sqrt{n}$ for some $c < 1$ and large enough n; this holds for $c = 1/\sqrt{2}$.) ◀

These results together help us create the algorithm we want.

▶ **Theorem 5.** *There exists a polynomial-time algorithm that finds an LO $O(\sqrt{n})$-colouring for a hypergraph with an LO 2-colouring.*

Proof. We provide a recursive algorithm.

1. If the hypergraph has no edges (x, y, z) where all of x, y and z have at least $\sqrt{n}$ neighbours, then find an independent set with $\Omega(n/\sqrt{n}) = \Omega(\sqrt{n})$ vertices, colour that independent set with a large colour, and recursively colour the rest of the hypergraph with smaller colours. This is possible by Lemma 3.
2. Otherwise, let (x, y, z) be an edge where all of x, y and z have at least $\sqrt{n}$ neighbours.
3. Iterate over $u \in \{x, y, z\}$.
4. Construct a partial colouring c_u where $c_u(u) = 2$ and $c_u(v) = 1$ for $v \in N(u)$.
5. Construct extend(c_u), and check if it is a partial LO 2-colouring. This is possible by Lemma 2.

6. If it is, colour the vertices in the domain of extend(c_u) with two large colours, depending on the colours they got in extend(c_u) (i.e. for some large enough C, colour v with $C + \text{extend}(c_u)(v)$), and then recursively colour the rest of the hypergraph with smaller colours.

We establish the correctness of this algorithm by showing its soundness and completeness.

We begin by showing completeness. The only way we could possibly fail to return something is if, for all $u \in \{x, y, z\}$, the function extend(c_u) is not a proper partial LO 2-colouring. However, for at least one $u \in \{x, y, z\}$, it must be the case that $c^*(u) = 2$, and thus that $c^*(v) = 1$ for $v \in N(u)$. Thus, for this value of u, c^* is consistent with c_u on its domain, and therefore extend(c_u) must be a proper partial LO 2-colouring.

Now we show soundness: whenever we return a colouring, we return an LO $O(\sqrt{n})$-colouring. We first show that we return an LO colouring. Observe that in the case covered in step 1, all the edges that do not intersect the independent set that we find are properly coloured recursively. Furthermore all edges that do intersect the independent set intersect it in exactly one vertex. This vertex is assigned a colour larger than the colours of the other vertices in the edge, so all such edges are also properly coloured. In the case covered in steps 2–6, note that if we return anything, then extend(c_u) is a proper partial LO 2-colouring. Note that all edges intersecting the domain of extend(c_u) in zero or three vertices are correctly coloured (either recursively, or since extend(c_u) is an LO colouring). Furthermore, since the vertices in the domain of extend(c_u) are assigned large colours, those edges that intersect this domain in one vertex are also properly coloured. Thus in this case, since no edge intersects extend(c_u) in two vertices, we also return a proper LO colouring. Thus in all cases we return an LO colouring.

To see that we return an LO $O(\sqrt{n})$-colouring, note that in all cases we colour $\Omega(\sqrt{n})$ vertices with $O(1)$ colours at each iteration (in the first case because we colour an independent set with this size; in the second because c_u must be defined on $N(u)$, which has at least $\sqrt{n}$ vertices). Thus, by Lemma 4, we use $O(\sqrt{n})$ colours overall.

We conclude by noting that this algorithm has recursive depth at most n, and does polynomial work at each step – thus it works in polynomial time. ◀

3.2 Second algorithm

As before, fix some 3-uniform hypergraph $H = (V, E)$, with n vertices and m edges, and suppose that this hypergraph admits an LO 2-colouring c^*. We first reduce our general problem to the special case of finding an LO 3-colouring for a *linear* 3-uniform hypergraph that admits an LO 2-colouring. We call a hypergraph *linear* if no pair of vertices belongs to more than one edge.

▶ **Example 6.** The hypergraph from Example 1 is not linear since vertices 1 and 2 belong to two edges.

The key property of linear hypergraphs we will use is the following: if a vertex v in an r-uniform hypergraph has d neighbours $v_1, \ldots, v_d$, then v is contained in at most d edges. This is the case since each pair (v, v_i) can belong to at most one edge, and $(v, \ldots, v)$ cannot appear as an edge. If H weren't linear, then v could have belonged to at least $\binom{d}{r-1}$ edges, one for each set of $r - 1$ of the d neighbours. (In fact, there are even more possible edges, since vertices are allowed to appear multiple times.)

▶ **Theorem 7.** *If H is a 3-uniform hypergraph with an LO 2-colouring, then we can find, in polynomial time, a linear 3-uniform hypergraph H' with an LO 2-colouring, such that an LO 3-colouring of H can be computed from an LO 3-colouring of H' in polynomial time.*

Proof. The following procedure is sufficient: while H contains two edges of form (x, y, z) and (x, y, z'), identify z with z'. This works because z and z' must be coloured the same in any LO 2-colouring of H (as, in any LO 2-colouring c, $c(x) + c(y) + c(z) = c(x) + c(y) + c(z') = 4$, and thus $c(z) = c(z')$). Thus H' also remains LO 2-colourable. Furthermore, by reversing the identifications, any LO 3-colouring of H' can be made into an LO 3-colouring of H. Finally, this procedure clearly takes polynomial time and results in a linear hypergraph H'. ◀

Thus we will assume that H is linear. We now show how to find a large independent set when the hypergraph is sparse enough, using a series of lemmata and an algorithm of Halldórsson [21].

▶ **Lemma 8.** *Suppose each edge of H has at least one vertex with at most $\Delta \geq 1$ neighbours. Then H has $\Omega(n)$ vertices with at most Δ neighbours.*

Proof. Let x be the number of vertices with at most Δ neighbours. We will show $x = \Omega(n)$ by double counting $|E|$. Note that each edge in $|E|$ contains at least one vertex with at most Δ neighbours; furthermore, each such vertex belongs to at most Δ edges (since H is linear). Thus $|E| \leq x\Delta$. Now note that each vertex with more than Δ neighbours belongs to more than $\Delta/2$ edges, and each edge contains 3 vertices. Thus $3|E| > (n - x)\Delta/2$. Therefore $(n - x)\Delta/6 < x\Delta$, which implies $x = \Omega(n)$. ◀

▶ **Lemma 9.** *Suppose all vertices in H have at most $\Delta \geq 1$ neighbours. Then H has an independent set with at least $\Omega(|E|/\Delta)$ vertices.*

Proof. Let S be the set of values mapped to 2 by c^*; note that S is an independent set. Consider the mapping $f : V \to \mathbb{N}$, where $f(u)$ is zero if $u \notin S$, and otherwise is equal to the number of edges that u belongs to. Since each edge contains exactly one vertex mapped to 2 by c^*, we deduce that $\sum_{u \in V} f(u) = \sum_{u \in S} f(u) = |E|$. Now, each value of $f(u)$ is at most Δ (since H is linear); thus we deduce that $|S|\Delta \geq \sum_{u \in S} f(u) = |E|$, and thus $|S| \geq |E|/\Delta$. ◀

▶ **Theorem 10** ([21, Theorem 4.4])**.** *There exists a polynomial-time algorithm that, if given a graph G with average degree $\overline{d}$, which has an independent set with size s, can find an independent set with size $\Omega(s \log \overline{d}/\overline{d} \log \log \overline{d})$.*

The *primal graph* $P(H)$ of H is a graph with the same vertices as H, and where two vertices are linked by an edge if and only if they are neighbours in H. Since the primal graph preserves neighbours, an independent set in H is independent in $P(H)$ and vice versa.

▶ **Example 11.** The primal graph $P(H)$ of the hypergraph H from Example 1 has $\{1, 2, 3, 4\}$ as vertices and $\{(1, 2), (2, 3), (3, 1), (2, 4), (4, 1)\}$ as edges.

▶ **Theorem 12.** *There exists a polynomial-time algorithm that, if given a linear hypergraph H with an LO 2-colouring where each vertex has most $\Delta = O(\sqrt{n/\log \log n})$ neighbours, can find an independent set with $\Omega(\sqrt{n} \log n/\sqrt{\log \log n})$ vertices.*

Proof. Consider the average degree $\overline{d}$ of the primal graph $P(H)$ of H. Observe that the independent sets of H and $P(H)$ coincide. Suppose $\overline{d} \leq \sqrt[6]{n}$. Thus $P(H)$ has $O(n\sqrt[6]{n})$ edges, and thus $O(n/\sqrt[6]{n}) = o(n)$ vertices of $P(H)$ have degree larger than $\sqrt[3]{n}$. By a simple greedy algorithm applied to the $\Omega(n)$ vertices of $P(H)$ with degree at most $\sqrt[3]{n}$ (repeatedly select the vertex with minimum degree), we can find an independent set of $P(H)$ (and thus of H) with $\Omega(n/\sqrt[3]{n})$ vertices, which is sufficient.

Now suppose $\overline{d} \geq \sqrt[6]{n}$. Note that there exists an independent set with $s = \Omega(|E|/\Delta)$ vertices. Since each edge of H gives rise to at most three edges of $P(H)$, we deduce that $\overline{d} = O(|E|/n)$. Thus $s/\overline{d} = \Omega(|E|/\Delta)/O(|E|/n) = \Omega(n/\Delta) = \Omega(\sqrt{n \log\log n})$. Also, $\log \overline{d}/\log\log \overline{d} \geq \log \sqrt[6]{n}/\log\log \sqrt[6]{n} = \Omega(\log n/\log\log n)$ for large enough n. Thus, the independent set algorithm from [21] applied to $P(H)$ will give us an independent set of $P(H)$ (and thus of H) with $\Omega(\sqrt{n}\log n/\sqrt{\log\log n})$ vertices. ◀

▶ **Corollary 13.** *If we are given instead a hypergraph where each edge has a vertex with at most $\Delta = O(\sqrt{n/\log\log n})$ neighbours, then, by applying the algorithm of Theorem 12 to the subhypergraph formed by taking only the $\Omega(n)$ vertices with at most Δ neighbours, we can still get an independent set with size $\Omega(\sqrt{n}\log n/\sqrt{\log\log n})$.*

As before, our algorithm will be recursive. We now investigate how many vertices must be coloured in one recursive step.

▶ **Lemma 14.** *A recursive procedure that colours $\Omega\left(\sqrt{n}\log n/\sqrt{\log\log n}\right)$ vertices (where n refers to the* current *size of the hypergraph, not the initial size) with $O(1)$ colours at each step will use $O(\sqrt{n\log\log n}/\log n)$ colours overall.*

Proof. Use the same strategy and notation as in Lemma 4. For n a large power of two, until halving n, in one step, we colour $\Omega(\sqrt{n/2}\log(n/2)/\sqrt{\log\log(n/2)}) = \Omega(\sqrt{n}\log n/\sqrt{\log\log n})$ vertices with $O(1)$ colours. The "large enough" part is important to get $n/2$ to the interval where $x \mapsto \sqrt{x}\log x/\sqrt{\log\log x}$ is increasing. Thus, with the same logic as before, $T(n) \leq T(n/2) + O(\sqrt{n\log\log n}/\log n)$ for large enough n. Applying the Master method of Cormen et al. [13] in the same way as in Lemma 4, $T(n) = O(\sqrt{n\log\log n}/\log n)$. (Again, we need the regularity condition: $\sqrt{n/2\log\log(n/2)}/\log(n/2) \leq c\sqrt{n\log\log n}/\log n$ for some $c < 1$ and large enough n; this can be shown for e.g. $c = 0.9$, $n \geq 6$.) ◀

With this result in hand, we now give a stronger algorithm for our problem, inspired by the strategy of Berger and Rompel [8]: we select multiple edges at once, not only one.

▶ **Theorem 15.** *There is a polynomial-time algorithm that finds an LO $O(\sqrt{n\log\log n}/\log n)$-colouring for a hypergraph that has an LO 2-colouring.*

Proof. Consider the following nondeterministic, recursive algorithm.

1. Make H linear by identifying vertices.
2. Let $k = \lceil \log_3 n \rceil$.
3. Let c_0 be an empty partial colouring.
4. For i from 1 to k:
   a. Let (x_i, y_i, z_i) be an edge for which x_i, y_i and z_i do not belong to the domain of c_{i-1}, and the minimum of the numbers of neighbours of x_i, y_i and z_i *not in the domain of* c_{i-1} is as large as possible. If such an edge does not exist, then exit this loop, setting $c_k = c_{i-1}$.
   b. Nondeterministically choose $u_i \in \{x_i, y_i, z_i\}$.
   c. Augment c_{i-1} with $u_i \mapsto 2$ and $v \mapsto 1$ for $v \in N(u_i)$; let the result of "extend"-ing this new function be c_i.

5. If c_k is not a proper LO 2-colouring, then this nondeterministic execution fails.
6. Colour the vertices in the domain of c_k with two large colours according to c_k (i.e. colour u with $C + c_k(u)$ for some large enough C). Remove the domain of c_k from H, letting the result be H'
7. If each edge in H' has a vertex with at most $O(\sqrt{n/\log\log n})$ neighbours, then find an independent set in H' with size $\Omega(\sqrt{n}\log n/\sqrt{\log\log n})$ and colour it with a large colour (but smaller then those used in the previous step), removing it from H'.
8. Recursively colour the rest of the hypergraph with smaller colours.

Since only logarithmically many nondeterministic choices from among a constant number of options are made, and polynomial work is done otherwise, this algorithm can be made into a polynomial-time deterministic one. Thus we show that it is correct, by showing that it is sound and complete.

To show completeness we need to show that at least one sequence of nondeterministic choices doesn't fail. It is therefore sufficient to show that, for one choice of $u_1, \ldots, u_k$, all of c_i are consistent with c^*. We see that, if c_{i-1} is consistent with c^*, then one choice of u_i exists that makes c_i consistent with c^* (the member of $\{x_i, y_i, z_i\}$ that is assigned 2 by c^*). Since c_0 is empty and thus consistent with c^* on its domain, we deduce inductively that one set of nondeterministic choices that keeps c_i consistent with c^* exists, and thus that the algorithm is complete.

Now we show soundness. Just as in the first algorithm, we always colour either by colouring the domain of some LO 2-colouring that doesn't intersect any edges in precisely 2 vertices with two large colours, according to that colouring; or by colouring an independent set with one large colour. (The LO colouring c_k intersects each edge in zero, one or three vertices due to the use of "extend".) We showed in the first algorithm that these procedures, followed by colouring what remains with smaller colours, lead to a proper LO colouring. We now need to show that we use $O(\sqrt{n\log\log n}/\log n)$ colours. To this end, we show that we colour $\Omega(\sqrt{n}\log n/\sqrt{\log\log n})$ vertices in each iteration – since colouring this many vertices with $O(1)$ colours at each recursive step will lead to an $O(\sqrt{n\log\log n}/\log n)$ colouring overall, as shown in Lemma 14. If the condition in step 7 is satisfied, then we clearly colour the required number of vertices; thus suppose it is not satisfied. In this case, there exists an edge e in H' whose vertices have $\Omega(\sqrt{n/\log\log n})$ neighbours. This implies that, during step a., the edge (x_i, y_i, z_i) could have been selected to be e. Since the number of neighbours of a vertex in H' is a lower bound for the number of neighbours of a vertex in H that do not belong to the domain of c_{i-1}, we find that selecting e would have made the minimum of the numbers of neighbours of x_i, y_i and z_i to have been at least $\Omega(\sqrt{n/\log\log n})$. Thus the actual values of x_i, y_i and z_i must all have at least this many neighbours not in the domain of c_{i-1}. Also, an edge is always found in step a. – since e can always be selected. This implies that c_0 is augmented by at least $\Omega(\sqrt{n/\log\log n})$ vertices at each of the $k = \Theta(\log n)$ iterations in step 4; thus c_k is defined on $\Omega(\sqrt{n}\log n/\sqrt{\log\log n})$ vertices. We thus deduce that in this case we also colour enough vertices. We conclude that the algorithm is sound.

Regarding the running time, note that if a (non-deterministic) guess in Step 4.b passes the test in Step 5, then no more guesses are made in this recursive call because the recursive call made in Step 8 cannot fail via our completeness proof. Thus the algorithm runs in polynomial time. ◀

4 Algebraic theory of fixed-template promise CSPs

We recount the algebraic theory of fixed-template PCSPs developed in [5] and specialised to structures with a single relation (of arity r).

The *p-the power* of an r-ary structure $\mathbf{A} = (A, R^{\mathbf{A}})$ is a structure $\mathbf{A}^p = (A^p, R^{\mathbf{A}^p})$ where

$$R^{\mathbf{A}^p} = \{((a_1^1, \dots, a_1^p), \dots, (a_r^1, \dots, a_r^p)) \mid (a_1^1, \dots, a_r^1) \in R^{\mathbf{A}}, \dots, (a_1^p, \dots, a_r^p) \in R^{\mathbf{A}}\}.$$

In other words, a tuple of $R^{\mathbf{A}^p}$ contains r vectors of p elements of A, such that if these are written as a matrix with r rows and p columns, each column is a member of $R^{\mathbf{A}}$. For two r-ary structures $\mathbf{A}, \mathbf{B}$, a p-ary polymorphism from $\mathbf{A}$ to $\mathbf{B}$ is a homomorphism $f : \mathbf{A}^p \to \mathbf{B}$.

▶ **Example 16.** Consider the binary structure $\mathbf{A} = ([2], R^{\mathbf{A}})$, where $R^{\mathbf{A}}$ is the binary disequality relation $\neq$ (restricted to $[2]^2$). The power $\mathbf{A}^5$ has domain $[2]^5$ and relation $\{(\mathbf{a}, \mathbf{b}) \mid \mathbf{a}, \mathbf{b} \in [2]^5, a_i \neq b_i, i = 1, \dots, 5\}$. This relation is constructed as follows: $(\mathbf{a}, \mathbf{b})$ belongs to the relation if and only if every column of a matrix with 5 columns and 2 rows constructed out of $\mathbf{a}, \mathbf{b}$ satisfies $\neq$. The matrix is the following one:

$$\begin{pmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 \end{pmatrix}.$$

Thus, for each column to satisfy $\neq$, we must have $a_i \neq b_i$ for $i = 1, \dots, 5$, as indicated above.

Now, consider a quinary polymorphism $f : \mathbf{A}^5 \to \mathbf{A}$. This is a function $f : [2]^5 \to [2]$ that satisfies the following property: if given a matrix with 2 rows and 5 columns, such that each column is a member of $R^{\mathbf{A}}$, then by applying f to the rows of this matrix we also get a member of $R^{\mathbf{A}}$. For instance, for the matrix

$$\begin{pmatrix} 1 & 2 & 2 & 1 & 1 \\ 2 & 1 & 1 & 2 & 2 \end{pmatrix},$$

we deduce that the pair $(f(1,2,2,1,1), f(2,1,1,2,2)) \in R^{\mathbf{A}}$ i.e. $f(1,2,2,1,1) \neq f(2,1,1,2,2)$. One such polymorphism is given by selecting the values of f from $[2]$ such that $f(x_1, \dots, x_5) \equiv \sum_{i=1}^{5} x_i \pmod 2$.

The real power of this theory comes from *minions*.[1] A *minion* $\mathcal{M}$ is a sequence of sets $\mathcal{M}^{(0)}, \mathcal{M}^{(1)}, \dots$, equipped with an operation that, for $f \in \mathcal{M}^{(p)}$ and $\pi : [p] \to [q]$, yields $f_\pi \in \mathcal{M}^{(q)}$. The operation must satisfy the following conditions:

- For $f \in \mathcal{M}^{(p)}$, if $id : [p] \to [p]$ is the identity on $[p]$, then $f_{id} = f$.
- For $f \in \mathcal{M}^{(p)}$, $\pi : [p] \to [q]$ and $\sigma : [q] \to [t]$, we have $f_{\sigma \circ \pi} = (f_\pi)_\sigma$.

An important class of minion is the *polymorphism minion*. The polymorphism minion $\mathcal{M} = \mathrm{Pol}(\mathbf{A}, \mathbf{B})$ for two structures $\mathbf{A}, \mathbf{B}$ with the same arity is a minion where $\mathcal{M}^{(p)}$ is the set of p-ary polymorphisms from $\mathbf{A}$ to $\mathbf{B}$, and where, for $f : A^p \to B, \pi : [p] \to [q]$, f_π is given by $f_\pi(x_1, \dots, x_q) = f(x_{\pi(1)}, \dots, x_{\pi(p)})$. It is not difficult to check that, if $f : \mathbf{A}^p \to \mathbf{B}$ and $\pi : [p] \to [q]$, then $f_\pi : \mathbf{A}^q \to \mathbf{B}$, as required.

In order to be able to relate polymorphism minions with the complexity of PCSPs, we use *minion homomorphisms*.[2] A *minion homomorphism from $\mathcal{M}$ to $\mathcal{N}$* is a mapping ξ that takes each $\mathcal{M}^{(p)}$ to $\mathcal{N}^{(p)}$ and that satisfies the following condition: for any $\pi : [p] \to [q]$ and $f \in \mathcal{M}^{(p)}$, $\xi(f)_\pi = \xi(f_\pi)$. The following theorem links minion homomorphisms to PCSPs. In particular, minion homomorphisms capture precisely a certain type of polynomial-time reductions, know as primitive-positive constructions,[3] studied for CSPs [6] and PCSPs [5].

---

[1] In category-theoretic terms, a minion is a functor from the skeleton of the category of finite sets to the category of sets. The objects of the first category are sets $[p]$ for $p \in \mathbb{N}$, and the arrows are functions between them. The functor equivalent to a minion $\mathcal{M}$ takes $[p]$ to $\mathcal{M}^{(p)}$, and $\pi : [p] \to [q]$ to $f \mapsto f_\pi$.

[2] In category-theoretic terms, a minion homomorphism is just a natural transformation.

[3] Primitive-positive constructions (or pp-constructions, for short) capture so-called "gadget reductions", cf. [6, Section 3].

▶ **Theorem 17** ([5, Theorems 3.1 and 4.12]). *For r-ary structures $\mathbf{A}, \mathbf{B}$ and r'-ary structures $\mathbf{A}', \mathbf{B}'$, a primitive-positive construction-based polynomial-time reduction from* $\mathrm{PCSP}(\mathbf{A}', \mathbf{B}')$ *to* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *exists if and only if* $\mathrm{Pol}(\mathbf{A}, \mathbf{B}) \to \mathrm{Pol}(\mathbf{A}', \mathbf{B}')$.

Unfortunately, it is usually a complex task to explicitly construct minion homomorphisms. An auxiliary construction called the *free structure* allows us to construct them more easily. For an arbitrary minion $\mathcal{M}$ and an r-ary structure $\mathbf{A} = (A, R^{\mathbf{A}})$, the *free structure* $\mathbf{F} = \mathbf{F}_{\mathcal{M}}(\mathbf{A})$ is an r-ary structure whose domain is $F = \mathcal{M}^{(|A|)}$. To construct its relation $R^{\mathbf{F}}$, first identify A with $[n]$ for $n = |A|$, and then enumerate the tuples of $R^{\mathbf{A}}$ as vectors $\mathbf{r}^1, \ldots, \mathbf{r}^k$, where $k = |R^{\mathbf{A}}|$. Construct functions $\pi_1, \ldots, \pi_r : [k] \to [n]$ where $\pi_i(j) = \mathbf{r}^j_i$. (If we were to arrange $\mathbf{r}^1, \ldots, \mathbf{r}^k$ into a matrix with r rows and k columns, then $\pi_i(1), \ldots, \pi_i(k)$ is the i-th row of the matrix.) Now, the tuple $(f_1, \ldots, f_r)$, where $f_1, \ldots, f_r \in \mathcal{M}^{(n)}$, belongs to $R^{\mathbf{F}}$ if and only if, for some $f \in \mathcal{M}^{(k)}$ we have $f_i = f_{\pi_i}$.

▶ **Example 18.** Consider some polymorphism minion $\mathcal{M}$ and the ternary structure $\mathbf{LO}_3^3$. We will construct $\mathbf{F} = \mathbf{F}_{\mathcal{M}}(\mathbf{LO}_3^3)$. The domain is $\mathcal{M}^{(3)}$. To construct the relation of $\mathbf{F}$, we first arrange the 15 tuples of $R^{\mathbf{LO}_3^3}$ into a matrix with 3 rows and 15 columns:

$$\begin{pmatrix} 2 & 1 & 1 & 3 & 1 & 1 & 3 & 2 & 3 & 1 & 2 & 1 & 3 & 2 & 2 \\ 1 & 2 & 1 & 1 & 3 & 1 & 2 & 3 & 1 & 3 & 1 & 2 & 2 & 3 & 2 \\ 1 & 1 & 2 & 1 & 1 & 3 & 1 & 1 & 2 & 2 & 3 & 3 & 2 & 2 & 3 \end{pmatrix}.$$

Row i of this matrix can be seen as a function $\pi_i : [15] \to [3]$. Now the relation $R^{\mathbf{F}}$ contains precisely the tuples $(f_{\pi_1}, f_{\pi_2}, f_{\pi_3})$ for all $f \in \mathcal{M}^{(15)}$. Substituting the definition for f_{π_i}, we find that these polymorphisms $f_{\pi_1}, f_{\pi_2}, f_{\pi_3}$ are:

$$(x, y, z) \mapsto f\,(y, x, x, z, x, x, z, y, z, x, y, x, z, y, y)$$
$$(x, y, z) \mapsto f\,(x, y, x, x, z, x, y, z, x, z, x, y, y, z, y)$$
$$(x, y, z) \mapsto f\,(x, x, y, x, x, z, x, x, y, y, z, z, y, y, z)$$

Observe that the matrix and the arguments of f are actually arranged in the same configuration, with 1 replaced by x, 2 by y and 3 by z.

The following theorem connects minion homomorphisms and the free structure.

▶ **Theorem 19** ([5, Lemma 4.4]). *If $\mathcal{M}$ is a minion and $\mathbf{A}, \mathbf{B}$ are r-ary structures, the homomorphisms $h : \mathbf{F}_{\mathcal{M}}(\mathbf{A}) \to \mathbf{B}$ are in a (natural) 1-to-1 correspondence to the minion homomorphisms $\xi : \mathcal{M} \to \mathrm{Pol}(\mathbf{A}, \mathbf{B})$.*[4] *As a consequence, $\mathbf{F}_{\mathcal{M}}(\mathbf{A}) \to \mathbf{B}$ if and only if $\mathcal{M} \to \mathrm{Pol}(\mathbf{A}, \mathbf{B})$.*

5 Hardness results

In this section we will investigate the hardness of $\mathrm{PCSP}(\mathbf{LO}_k^r, \mathbf{LO}_{k+1}^r)$. First, we will establish that $\mathrm{PCSP}(\mathbf{LO}_k^r, \mathbf{LO}_{k+1}^r)$ is NP-hard for $r \geq 5$ and $k \geq 2$. In particular, this proves intractability of $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ for $r \geq 5$. Second, we will show that $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ with $r \geq 5$ *cannot* be reduced to $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ with $r = 3$ and $r = 4$ using primitive-positive constructions (i.e. gadget reductions [5]).

[4] In category-theoretic terms, $\mathbf{F}_{-}(\mathbf{A})$ and $\mathrm{Pol}(\mathbf{A}, -)$ are functors between (in opposite directions) the category of r-ary structures and the category of minions, and $\mathbf{F}_{-}(\mathbf{A}) \dashv \mathrm{Pol}(\mathbf{A}, -)$.

We will use a hardness result of Guruswami and Trevisan [20] (stated as Theorem 20 below) on the *1-in-r exact hitting set*. An instance of this problem is an r-uniform hypergraph, and an admissible solution is a subset of its vertices. We are then asked to maximise the number of edges e for which exactly one vertex of e belongs to the chosen subset. An instance is called *α-satisfiable* if it is possible to find a subset of vertices such that a proportion of α of all the edges satisfy this condition. An instance is called *satisfiable* if it is 1-satisfiable. We will call the problem of distinguishing a satisfiable 1-in-r exact hitting set instance from one that is not even α-satisfiable the *α-gap 1-in-r exact hitting set problem*. Guruswami and Trevisan state a weaker version of the following theorem, but their proofs establish the following.

▶ **Theorem 20** ([20, Theorem 10, Theorem 12, Lemma 13])**.** *For any $\epsilon > 0$, for large enough r, the $(1/(e-\epsilon))$-gap 1-in-r exact hitting set problem is* NP*-hard.*

▶ **Corollary 21.** $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ *is* NP*-hard for large enough r.*

Proof. Observe that the instances of $(1/2)$-gap 1-in-r exact hitting set and $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ are both r-uniform hypergraphs. We show that the identity function is a reduction from the first problem to the second. For completeness, note that a satisfiable instance of 1-in-r exact hitting set is also immediately a YES-instance of $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$. For soundness, consider a YES-instance of $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ i.e. an r-uniform hypergraph with an LO 2-colouring and thus also an LO 3-colouring. Under this colouring each edge either has a unique vertex assigned 3, or a unique vertex assigned 2; thus, at least half of the edges must contain exactly one 2, or at least half of the edges must contain exactly one 3. We deduce that taking either the set of vertices assigned 2, or the set of vertices assigned 3, gives us a solution that satisfies at least half the edges of the hypergraph, viewed as a hitting set instance. ◀

How can we now leverage this basic hardness result to other values of r? We will use chains of minion homomorphisms to do this. We define $\mathcal{M}_r = \mathrm{Pol}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$. Our main result in this section is the following.

▶ **Theorem 22.** *The relationships between the minions $\mathcal{M}_r = \mathrm{Pol}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ for $r \geq 3$ are as shown in Figure 1, i.e., all homomorphisms not drawn or implied do not exist.*

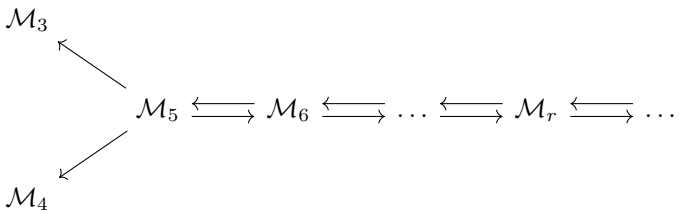


Figure 1 Minion homomorphism order of minions $\mathcal{M}_r$ for $r \geq 3$.

Combining Theorems 17 and 22 gives the following.

▶ **Corollary 23.** $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ *is* NP*-hard for $r \geq 5$. Moreover, there is no polynomial-time reduction using pp-constructions from $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ to $\mathrm{PCSP}(\mathbf{LO}_2^3, \mathbf{LO}_3^3)$ and from $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ to $\mathrm{PCSP}(\mathbf{LO}_2^4, \mathbf{LO}_3^4)$ for $r \geq 5$.*

A simple proof shows the following:

▶ **Theorem 24.** *For every $r \geq 5$ and $\ell > k \geq 2$, $\mathrm{Pol}(\mathbf{LO}_{k+1}^r, \mathbf{LO}_{\ell+1}^r) \to \mathrm{Pol}(\mathbf{LO}_k^r, \mathbf{LO}_\ell^r)$.*

Proof. Consider any p-ary polymorphism $f \in \mathrm{Pol}(\mathbf{LO}^r_{k+1}, \mathbf{LO}^r_{\ell+1})^{(p)}$. Consider the value of f for inputs $a_1, \ldots, a_p \in [k]$; due to the following matrix with $r \geq 3$ rows

$$\begin{pmatrix} a_1+1 & \ldots & a_p+1 \\ a_1 & \ldots & a_p \\ \vdots & \ddots & \vdots \\ a_1 & \ldots & a_p \end{pmatrix},$$

we can deduce that $f(a_1, \ldots, a_p) < f(a_1+1, \ldots, a_p+1) \in [\ell+1]$. This implies that $f(a_1, \ldots, a_p) \in [\ell]$. We claim this implies that f, restricted to $[k]^p$, is a polymorphism of $\mathrm{Pol}(\mathbf{LO}^r_k, \mathbf{LO}^r_\ell)$. Consider matrix of inputs a^j_i where $i \in [p]$, $j \in [r]$, such that each column $a^1_i, \ldots, a^r_i$ is a tuple of $\mathbf{LO}^r_k$. Thus each column is also a tuple of $\mathbf{LO}^r_{k+1}$. Since f is a polymorphism of $\mathrm{PCSP}(\mathbf{LO}^r_{k+1}, \mathbf{LO}^r_{\ell+1})$, we deduce that

$$(f(a^1_1, \ldots, a^1_p), \ldots, f(a^r_1, \ldots, f^r_p))$$

is a tuple of $\mathbf{LO}^r_{\ell+1}$ i.e. has a unique maximum. But we already know these values belong to $[\ell]$. Since they have a unique maximum, they are a tuple of $\mathbf{LO}^r_\ell$. Thus f, restricted to $[k]^p$, is a polymorphism of $\mathrm{Pol}(\mathbf{LO}^r_k, \mathbf{LO}^r_\ell)$.

We now claim that the map $f \mapsto f|_{[k]^p}$ taking a p-ary polymorphism to its restriction on $[k]^p$ is a minion homomorphism $\mathrm{Pol}(\mathbf{LO}^r_{k+1}, \mathbf{LO}^r_{\ell+1}) \to \mathrm{Pol}(\mathbf{LO}^r_k, \mathbf{LO}^r_\ell)$. To see why, consider any polymorphism $f \in \mathrm{Pol}(\mathbf{LO}^r_{k+1}, \mathbf{LO}^r_{\ell+1})^{(p)}$ and a function $\pi : [p] \to [q]$. What we need to prove is that

$$(f_\pi)|_{[k]^p} = (f|_{[k]^p})_\pi.$$

But note that, for $x_1, \ldots, x_p \in [k]$,

$$\begin{aligned}((f_\pi)|_{[k]^p})(x_1, \ldots, x_p) = f_\pi(x_1, \ldots, x_p) &= f(x_{\pi(1)}, \ldots, x_{\pi(p)}) \\ &= (f|_{[k]^p})(x_{\pi(1)}, \ldots, x_{\pi(p)}) = (f|_{[k]^p})_\pi(x_1, \ldots, x_p).\end{aligned}$$

This concludes the proof. ◀

Theorem 24 and Corollary 23 imply the following:

▶ **Corollary 25.** $\mathrm{PCSP}(\mathbf{LO}^r_k, \mathbf{LO}^r_{k+1})$ *is* NP*-hard for* $r \geq 5$ *and* $k \geq 2$.

In order to construct the minion homomorphisms, we first exhibit a simple necessary and sufficient condition for the existence of a minion homomorphism to $\mathrm{Pol}(\mathbf{LO}^r_2, \mathbf{LO}^r_k)$, and a sufficient condition for such a homomorphism to not exist.

▶ **Lemma 26.** *Fix* $r \geq 3$ *and* $k \geq 3$. *Consider any polymorphism minion* $\mathcal{M}$. *For any element* $f \in \mathcal{M}^{(r)}$, *let* $f_1(x, y) = f(y, x, \ldots, x)$, $f_2(x, y) = f(x, y, x, \ldots, x)$, $\ldots$, $f_r(x, y) = f(x, \ldots, x, y)$. *Now,* $\mathcal{M} \to \mathrm{Pol}(\mathbf{LO}^r_2, \mathbf{LO}^r_k)$ *if and only if there exists some* $\omega : \mathcal{M}^{(2)} \to [k]$ *such that, for all* $f \in \mathcal{M}^{(r)}$, *there exists a unique maximum value among* $\omega(f_1), \ldots, \omega(f_r)$.

Proof. We construct $\mathbf{F}_{\mathcal{M}}(\mathbf{LO}^r_2)$. The tuples of the relation of $\mathbf{LO}^r_2$ are all the r-dimensional vectors containing exactly one 2, with the other entries equal to 1. We can arrange these tuples into an r-by-r matrix where the diagonal contains 2 and all the other elements are 1. Replacing 1 with x and 2 with y, and applying f, we get the definitions of $f_1, \ldots, f_r$. Thus the relation of $\mathbf{F}_{\mathcal{M}}(\mathbf{LO}^r_2)$ contains precisely the tuples of form $(f_1, \ldots, f_r)$ for $f \in \mathcal{M}^{(r)}$.

Thus our condition amounts to the existence of a homomorphism $\omega : \mathbf{F}_{\mathcal{M}}(\mathbf{LO}^r_2) \to \mathbf{LO}^r_k$. By Theorem 19, this is equivalent to $\mathcal{M} \to \mathrm{Pol}(\mathbf{LO}^r_2, \mathbf{LO}^r_k)$. ◀

For the next lemma, call a function f *block-symmetric* with respect to a partition of the arguments of f into blocks if f is not changed by arbitrarily permuting the arguments in the blocks. For instance $(x, y, z) \mapsto x + y$ is block-symmetric with respect to the blocks $\{x, y\}$ and $\{z\}$.

▶ **Lemma 27.** *Fix $r \geq 3$ and a polymorphism minion $\mathcal{M}$. If $\mathcal{M}$ has a block-symmetric polymorphism of arity r with respect to a partition in which all blocks consist of at least two elements, then $\mathcal{M} \not\to \mathcal{M}_r$.*

Proof. We use the same notation as in the proof of Lemma 26. Let f be this block-symmetric polymorphism, and note that no polymorphism among $f_1, \ldots, f_r$ appears exactly once, due to block-symmetry. Thus, for any $\omega : \mathcal{M}^{(2)} \to [3]$, the sequence $\omega(f_1), \ldots, \omega(f_r)$ does not have a unique maximum. Applying the previous lemma gives us the required result. ◀

▶ **Theorem 28.** *For any $r \geq 5$, $\mathcal{M}_r \to \mathcal{M}_{r+1}$.*

Proof. We will establish the statement via Lemma 26. We will use the assignment $\omega : \mathcal{M}^{(2)} \to [3]$ given by $\omega(f) = f(1, 2)$. To check that this satisfies our condition, we need to investigate $\mathcal{M}_r^{(r+1)}$. Observe that an $(r + 1)$-ary polymorphism $f \in \mathcal{M}_r^{(r+1)}$ is a function from $[2]^{r+1}$ to $[3]$; if it is applied to the rows of an $r \times (r+1)$ matrix whose columns are tuples of the relation of $\mathbf{LO}_2^r$ (i.e. they contain one 2 and otherwise are 1), then the resulting values contain a unique maximum. Similarly to Barto et. al. [4], we view f as a function from the powerset of $[r + 1]$ to $[3]$. (Thus, the input tuple $(1, 2, 1, 2)$ is seen as equivalent to the input set $\{2, 4\}$.) Under this view, f is a polymorphism if and only if, for any partition $A_1, \ldots, A_r$ of $[r + 1]$, the sequence $f(A_1), \ldots, f(A_r)$ has a unique maximum element. (Observe that each part A_i corresponds to a row in the matrix mentioned earlier). To show that ω satisfies our condition, what we need to prove is that the sequence $f\{1\}, \ldots, f\{r + 1\}$ has a unique maximum. We have three cases depending on the maximum of these values.

Maximum is 3. Suppose that at least one of $f\{1\}, \ldots, f\{r+1\}$ is 3. Without loss of generality, say $f\{1\} = 3$. Suppose for contradiction that another of the values is also 3; without loss of generality, say $f\{2\} = 3$. But consider the partition $\{1\}, \{2\}, \{3, \ldots, r + 1\}, \emptyset, \ldots, \emptyset$ of $[r + 1]$ into r sets. Since f is a polymorphism, the images of these sets must have a unique maximum value. But if $f\{1\} = f\{2\} = 3$ this is impossible! So if $f\{1\} = 3$, then this is the unique maximum of the sequence $f\{1\}, \ldots, f\{r + 1\}$.

Maximum is 2. Now, suppose that all of $f\{1\}, \ldots, f\{r + 1\}$ are either 1 or 2, and that at least one of these (say $f\{1\}$) is 2. Suppose for contradiction that another value (say $f\{2\}$) is also 2. Due to the partitions $\{1\}$, $\{2\}$, $\{3, 4\}$, $\{5\}$, $\ldots$, $\{r + 1\}$; $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5, 6\}$, $\{7\}$, $\ldots$, $\{r + 1\}$ we deduce that $f\{3, 4\} = f\{5, 6\} = 3$. This is impossible because of the partition $\{1, 2, 7, \ldots, r + 1\}, \{3, 4\}, \{5, 6\}, \emptyset, \ldots, \emptyset$. Thus in this case $f\{1\} = 2$ is the unique maximum of the sequence $f\{1\}, \ldots, f\{r + 1\}$.

Maximum is 1. Finally, suppose that $f\{1\} = \ldots = f\{r + 1\} = 1$. Consider the partitions $\{1, 2\}$, $\{3\}$, $\ldots$, $\{r + 1\}$; $\{1\}$, $\{2\}$, $\{3, 4\}$, $\{5\}$, $\ldots$, $\{r + 1\}$; and $\{1\}$, $\ldots$, $\{4\}$, $\{5, 6\}$, $\{7\}$, $\ldots$, $\{r + 1\}$. Since all the singletons have image 1, the two-element sets here must have image 2 or 3. At least two of them must therefore have the same image; if that image is 3, then we have a contradiction like in the last case. Thus suppose, without loss of generality, that $f\{1, 2\} = f\{3, 4\} = 2$. Considering the partition $\{1, 2\}, \{3, 4\}, \{5\}, \ldots, \{r + 1\}, \emptyset$, we find that $f(\emptyset) = 3$. But this cannot happen, due to partition $\emptyset, \ldots, \emptyset, \{1, \ldots, r + 1\}$. Thus this case is impossible.

Our assignment of values to polymorphisms of $\mathcal{M}_r^{(2)}$ is correct. Therefore we deduce that $\mathcal{M}_r \to \mathcal{M}_{r+1}$. ◀

▶ **Theorem 29.** *For any $r \geq 3, k \geq 3$,* $\mathrm{Pol}(\mathbf{LO}_2^{r+2}, \mathbf{LO}_k^{r+2}) \to \mathrm{Pol}(\mathbf{LO}_2^r, \mathbf{LO}_k^r)$. *In particular, for $k = 3$,* $\mathcal{M}_{r+2} \to \mathcal{M}_r$.

Proof. We use the same notation convention as in Theorem 28, and we take $\omega(f) = f(1, 2)$. Thus, we want to prove that, if f is a function that takes subsets of $[r]$ to $[k]$ such that, for any partition $A_1, \ldots, A_{r+2}$, the sequence $f(A_1), \ldots, f(A_{r+2})$ has a unique maximum, then the sequence $f\{1\}, \ldots, f\{r\}$ has a unique maximum. But consider the partition $\{1\}, \ldots, \{r\}, \emptyset, \emptyset$, and note that the largest value cannot be $f(\emptyset)$ (since $f(\emptyset)$ appears twice). Thus we deduce that one of $f\{1\}, \ldots, f\{r\}$ is the maximum, and furthermore that this maximum is strictly larger than all the other values in this sequence. Thus, $\mathcal{M}_{r+2} \to \mathcal{M}$. ◀

Proof of Theorem 22. We have $\mathcal{M}_{r+1} \to \mathcal{M}_{r+2} \to \mathcal{M}_r$, so $\mathcal{M}_{r+1} \to \mathcal{M}_r$ for every $r \geq 5$ by Theorems 28 and 29. Thus, $\mathcal{M}_5 \rightleftarrows \mathcal{M}_6 \rightleftarrows \ldots$; furthermore, by Theorem 29, we have $\mathcal{M}_5 \to \mathcal{M}_3$. Finally, by Theorem 28 and Theorem 29, we have $\mathcal{M}_5 \to \mathcal{M}_6 \to \mathcal{M}_4$.

It remains to show that (i) $\mathcal{M}_3 \not\to \mathcal{M}_r$ for any $r \geq 4$ and that (ii) $\mathcal{M}_4 \not\to \mathcal{M}_r$ for any $r \geq 3$, $r \neq 4$. An auxiliary function will be useful for both: let $f : \mathbb{N} \to \mathbb{N}$ map 0 to 2, 1 to 1, and all other values to 3.

Regarding (i), note that $\mathcal{M}_3$ has a quaternary block-symmetric polymorphism with respect to a partition in which all blocks have size 2, viz. $(a, b, c, d) \mapsto f(a + b)$. Thus, by Lemma 27, $\mathcal{M}_3 \not\to \mathcal{M}_4$. Since $\mathcal{M}_r \to \mathcal{M}_5 \to \mathcal{M}_4$ for $r \geq 4$, we deduce that $\mathcal{M}_3 \not\to \mathcal{M}_r$.

Regarding (ii), note that $\mathcal{M}_4$ has a ternary symmetric polymorphism, viz. $(a, b, c) \mapsto f(a + b + c)$. Thus, by Lemma 27, $\mathcal{M}_4 \not\to \mathcal{M}_3$. Since $\mathcal{M}_r \to \mathcal{M}_5 \to \mathcal{M}_3$ for $r \geq 3, r \neq 4$, we deduce that $\mathcal{M}_4 \not\to \mathcal{M}_r$. ◀

In Theorem 28, using Theorem 17, we showed that hardness of $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ for some large r implies hardness of $\mathrm{PCSP}(\mathbf{LO}_2^5, \mathbf{LO}_3^5)$. We now show that the same is true for $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_k^r)$: hardness of $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_k^r)$ for some large r implies hardness of $\mathrm{PCSP}(\mathbf{LO}_2^{r(k)}, \mathbf{LO}_k^{r(k)})$, for some $r(k)$ that depends only on k.

▶ **Theorem 30.** *For $k \geq 4, r \geq 4k - 3$,* $\mathrm{Pol}(\mathbf{LO}_2^r, \mathbf{LO}_k^r) \to \mathrm{Pol}(\mathbf{LO}_2^{r+1}, \mathbf{LO}_k^{r+1})$.

Note that the lower bound on r in the statement is $4k - 3$, which is worse than the bound for $k = 3$ in Theorem 28. We do not know whether a smaller bound is possible for $k > 3$.

Proof. Fix some k and let $\mathcal{N}_r = \mathrm{Pol}(\mathbf{LO}_2^r, \mathbf{LO}_k^r)$. We show that $\mathcal{N}_r \to \mathcal{N}_{r+1}$ for $r \geq 4k - 3$.

Use the same notation as in Theorem 28, and the same choice of ω; what we want to show is that if $f \in \mathcal{N}_r^{(r+1)}$ then the sequence $f\{1\}, f\{2\}, \ldots, f\{r + 1\}$ has a unique maximum. Recall that f satisfies the property that, if $A_1, \ldots, A_r$ is a partition of $\{1, \ldots, r + 1\}$, then $f(A_1), \ldots, f(A_r)$ has a unique maximum. We now split into three cases, depending on the maximum of $f\{1\}, \ldots, f\{r + 1\}$.

Maximum is k. The maximum must be unique in this case. If we suppose, contrarily and without loss of generality, that $f\{1\} = f\{2\} = k$, then the partition $\{1\}, \{2\}, \{3, \ldots, r + 1\}, \emptyset, \ldots, \emptyset$ yields our contradiction.

Maximum is 1. This case is impossible. Consider the partitions $\{1, 2\}, \{3\}, \ldots, \{r + 1\}$; $\{1\}, \{2\}, \{3, 4\}, \{5\}, \ldots, \{r + 1\}$; $\ldots$; $\{1\}, \ldots, \{4k - 3, 4k - 2\}, \{4k - 1\}, \ldots, \{r + 1\}$. These partitions exist since $4k - 2 \leq r + 1$. All of the two-element sets must be mapped to some values in $\{2, \ldots, k\}$ (they cannot be 1 since then all the parts in the previous partitions map to 1), and there are $2k - 1$ two-element sets. Thus by pidgeonhole principle three sets must be mapped to the same value. Thus suppose $f\{1, 2\} = f\{3, 4\} = f\{5, 6\}$. Considering the partition $\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7\}, \ldots, \{r + 1\}, \emptyset, \emptyset$ we find that none of the images through f can be the unique maximum (since the first three and the last two are equal, and all the rest are 1). Thus we have the required contradiction.

Maximum is neither. Suppose that the maximum is $1 < k' < k$, and suppose that the maximum is not unique. Thus without loss of generality let $f\{1\} = f\{2\} = k'$. Now consider the partitions $\{1\},\{2\},\{3,4\},\{5\},\dots,\{r+1\};\dots;\{1\},\{2\},\dots,\{4k-4\},\{4k-3,4k-2\},\{4k-1\},\dots,\{r+1\}$. The two-element sets must be mapped to a value from $\{k'+1,\dots,k\}$ i.e. to one of at most $k-2$ values, and there are $2k-2$ sets. Thus at least three of these sets are assigned the same value. Suppose without loss of generality that $f\{3,4\} = f\{5,6\} = f\{7,8\}$. Then the partition $\{1\},\{2\},\{3,4\},\{5,6\},\{7,8\},\{9\},\dots,\{r+1\},\emptyset,\emptyset$ gives us a contradiction: the images of the first two sets, the next three sets, and the last two sets form equal blocks; and all other sets have images that are not greater than the images of the first two sets.

Thus, since our choice of ω is valid, by Lemma 26, we find that $\mathcal{N}_r \to \mathcal{N}_{r+1}$. ◀

6 Conclusions

The question about the complexity of $\mathrm{PCSP}(\mathbf{LO}_2^3, \mathbf{LO}_k^3)$ for constant $k \geq 3$ raised in [4] stays open. The complexity of $\mathrm{PCSP}(\mathbf{LO}_k^r, \mathbf{LO}_\ell^r)$ is open except for the hardness obtained in our work. In this paper, we show NP-hardness for $k \geq 2$, $\ell = k+1$, and any $r \geq 5$. In the full version of this paper [25], we show NP-hardness for $2 \leq k \leq \ell$ and any $r \geq \ell - k + 4$.

The minion homomorphisms (and lack thereof) between the polymorphism minions $\mathrm{Pol}(\mathbf{LO}_2^r, \mathbf{LO}_3^r)$ for various values of r have interesting implications for the complexity of PCSPs more broadly. First, if one were to prove that this problem is hard for $r = 3$ or $r = 4$, then our results imply that hardness in linearly ordered colourings does not necessarily follow from minion homomorphisms and thus in particular cannot be obtained via "gadget reductions" [5]. This is in contrast to the case of (non-promise) CSPs, where it is known [7] (cf. also [6])[5] that all NP hardness can be shown using minion homomorphisms.[6] Second, our results show that, if one proves that $\mathrm{PCSP}(\mathbf{LO}_2^r, \mathbf{LO}_\ell^r)$ is hard for some large arity r, then it is hard $\mathrm{PCSP}(\mathbf{LO}_2^{r'}, \mathbf{LO}_\ell^{r'})$ for some arity $r' = r'(\ell)$ that depends only on ℓ.

Going beyond the realm of fixed-template PCSPs [5] (which limits the number of colours by a constant), what is the smallest function $k(n)$ for which $\mathrm{PCSP}(\mathbf{LO}_2^3, \mathbf{LO}_{k(n)}^3)$ is solvable efficiently? There is no clear reason to believe that positive result from the present paper with $k(n) = O(\sqrt{n \log\log n}/\log n)$ is optimal.

References

1 Per Austrin, Amey Bhangale, and Aditya Potukuchi. Simplified inpproximability of hypergraph coloring via t-agreeing families, 2019.

2 Per Austrin, Amey Bhangale, and Aditya Potukuchi. Improved inapproximability of rainbow coloring. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1479–1495, 2020. doi:10.1137/1.9781611975994.90.

3 Per Austrin, Venkatesan Guruswami, and Johan Håstad. $(2+\epsilon)$-Sat is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017. doi:10.1137/15M1006507.

4 Libor Barto, Diego Battistelli, and Kevin M. Berg. Symmetric Promise Constraint Satisfaction Problems: Beyond the Boolean Case. In *Proc. 38th International Symposium on Theoretical Aspects of Computer Science (STACS'21)*, volume 187 of *LIPIcs*, pages 10:1–10:16, 2021. doi:10.4230/LIPIcs.STACS.2021.10.

[5] [6] uses the terminology of height 1 identities.

[6] This would not be the first time though in the context of PCSPs. A recent example of the same phenomenon comes from [27] for the problem of approximate graph colouring.

5 Libor Barto, Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *J. ACM*, 68(4):28:1–28:66, 2021. doi:10.1145/3457606.

6 Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.1.

7 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Isr. J. Math*, 223(1):363–398, February 2018. doi:10.1007/s11856-017-1621-9.

8 Bonnie Berger and John Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5(3):459–466, 1990. doi:10.1007/BF01840398.

9 Amey Bhangale. NP-Hardness of Coloring 2-Colorable Hypergraph with Poly-Logarithmically Many Colors. In *Proc. 45th International Colloquium on Automata, Languages, and Programming (ICALP'18)*, volume 107 of *LIPIcs*, pages 15:1–15:11, 2018. doi:10.4230/LIPIcs.ICALP.2018.15.

10 Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. In *Proc. 31st Conference on Computational Complexity (CCC'16)*, volume 50 of *LIPIcs*, pages 14:1–14:27, 2016. doi:10.4230/LIPIcs.CCC.2016.14.

11 Joshua Brakensiek and Venkatesan Guruswami. Promise Constraint Satisfaction: Algebraic Structure and a Symmetric Boolean Dichotomy. *SIAM J. Comput.*, 50(6):1663–1700, 2021. doi:10.1137/19M128212X.

12 Joshua Brakensiek and Venkatesan Guruswami. The quest for strong inapproximability results with perfect completeness. *ACM Trans. Algorithms*, 17(3):27:1–27:35, 2021. doi:10.1145/3459668.

13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

14 Irit Dinur, Oded Regev, and Clifford Smyth. The hardness of 3-uniform hypergraph coloring. *Comb.*, 25(5):519–535, September 2005. doi:10.1007/s00493-005-0032-4.

15 Irit Dinur and Igor Shinkar. On the Conditional Hardness of Coloring a 4-Colorable Graph with Super-Constant Number of Colors. In *Proc. 13th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'10)*, volume 6302 of *LNCS*, pages 138–151. Springer, 2010. doi:10.1007/978-3-642-15369-3_11.

16 M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *J. ACM*, 23(1):43–49, 1976. doi:10.1145/321921.321926.

17 Venkatesan Guruswami and Sanjeev Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM J. Discret. Math*, 18(1):30–40, 2004. doi:10.1137/S0895480100376794.

18 Venkatesan Guruswami and Euiwoong Lee. Strong inapproximability results on balanced rainbow-colorable hypergraphs. *Comb.*, 38(3):547–599, 2018. doi:10.1007/s00493-016-3383-0.

19 Venkatesan Guruswami and Sai Sandeep. Rainbow coloring hardness via low sensitivity polymorphisms. *SIAM J. Discret. Math*, 34(1):520–537, 2020. doi:10.1137/19M127731X.

20 Venkatesan Guruswami and Luca Trevisan. The complexity of making unique choices: Approximating 1-in- k SAT. In *Proc. 8th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'05)*, volume 3624 of *LNCS*, pages 99–110. Springer, 2005. doi:10.1007/11538462_9.

21 Magnús M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. *J. Graph Algorithms Apppl.*, 4(1):1–16, 2000. doi:10.7155/jgaa.00020.

22 Sangxia Huang. Improved hardness of approximating chromatic number. In *Proc. 16th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'13)*, pages 233–243. Springer, 2013. doi:10.1007/978-3-642-40328-6{_}17.

23 Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *J. ACM*, 64(1):4:1–4:23, 2017. doi:10.1145/3001582.

24 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Comb.*, 20(3):393–415, March 2000. doi:10.1007/s004930070013.

25 Tamio-Vesa Nakajima and Stanislav Živný. Linearly ordered colourings of hypergraphs, 2022. arXiv:2204.05628.

26 Avi Wigderson. Improving the performance guarantee for approximate graph coloring. *J. ACM*, 30(4):729–735, 1983. doi:10.1145/2157.2158.

27 Marcin Wrochna and Stanislav Živný. Improved hardness for H-colourings of G-colourable graphs. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1426–1435, 2020. doi:10.1137/1.9781611975994.86.

The Variance-Penalized Stochastic Shortest Path Problem

Jakob Piribauer
Technische Universität Dresden, Germany

Ocan Sankur
Univ Rennes, Inria, CNRS, IRISA, France

Christel Baier
Technische Universität Dresden, Germany

Abstract

The stochastic shortest path problem (SSPP) asks to resolve the non-deterministic choices in a Markov decision process (MDP) such that the expected accumulated weight before reaching a target state is maximized. This paper addresses the optimization of the variance-penalized expectation (VPE) of the accumulated weight, which is a variant of the SSPP in which a multiple of the variance of accumulated weights is incurred as a penalty. It is shown that the optimal VPE in MDPs with non-negative weights as well as an optimal deterministic finite-memory scheduler can be computed in exponential space. The threshold problem whether the maximal VPE exceeds a given rational is shown to be EXPTIME-hard and to lie in NEXPTIME. Furthermore, a result of interest in its own right obtained on the way is that a variance-minimal scheduler among all expectation-optimal schedulers can be computed in polynomial time.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Markov decision process, variance, stochastic shortest path problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.129

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version An extended version containing all proofs omitted here is available online.
Full Version: `https://arxiv.org/abs/2204.12280`

Funding This work was funded by DFG grant 389792660 as part of TRR 248, the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), and DFG-projects BA-1679/11-1 and BA-1679/12-1.

1 Introduction

Markov decision processes (MDPs) are a standard operational model comprising randomization and non-determinism and are widely used in verification, articifical intelligence, robotics, and operations research. In each state of an MDP, there is a non-deterministic choice from a set of actions. Each action is equipped with a weight and a probability distribution according to which the successor state is chosen randomly. In the analysis of systems modelled as MDPs, one typically is interested in the worst- or best-case behavior, where worst and best case range over all resolutions of the non-determinism. So, the resulting algorithmic problems on MDPs usually ask to resolve non-deterministic choices by specifying a *scheduler* such that the resulting probabilistic behavior is optimized with respect to an objective function. If the weights are used to model one of various quantitative aspects of a system such as costs, resource consumption, rewards, or utility, a frequently encountered such optimization problem is the *stochastic shortest path problem* (SSPP) [5,8]. It asks to optimize the expected


value of the accumulated weight before reaching a target state. Example applications include the analysis of worst-case expected termination times of probabilistic programs or finding the optimal controls in a motion planning scenario with random external influences.

While a solution to the SSPP provides guarantees on the behavior of a system in all environments or indicates the optimal control to maximize expected rewards, it completely disregards all other aspects of the resulting probability distribution of the accumulated weight besides the expected value. In almost all practical applications, however, the uncertainty coming with the probabilistic behavior cannot be neglected. In traffic control systems or energy grids, for example, large variability in the throughput comes at a high cost due to the risk of traffic jams or the difficulty of storing surplus energy. Also a probabilistic program employed in a complex environment might be of more use with a higher expected termination time in exchange for a lower chance of extreme termination times.

To overcome these shortcomings of the SSPP, various additional optimization problems have been studied in the literature: Optimizing conditional expected accumulated weights under the condition that certain system states are reached allows for a more fine-grained system analysis by making it possible to determine the worst- or best-case expectation in different scenarios [4, 20]. Given a probability p, quantiles on the accumulated weight in MDPs, also called *values-at-risk* in the context of risk analysis, are the best bound B such that the accumulated weight exceeds B with probability at most p in the worst or best case [11, 27]. The *conditional value-at-risk* and the *entropic value-at-risk* are more involved measures that have been studied in this context [1, 14]. They quantify how far the probability mass of the tail of the probability distribution lies above the value-at-risk. The arguably most prominent measure for the deviation of a random variable from its expected value is the *variance*. The computation of the variance of accumulated weights has been studied in Markov chains [28] and in MDPs [16, 17]. The investigations of variance in MDPs in the literature is discussed in more detail in the "Related Work" section below.

Variance-penalized expectation (VPE). In this paper, we investigate a variant of the SSPP in which the costs caused by probabilistic uncertainty are priced in to the objective function: We study the optimization of the *variance-penalized expectation* (VPE), a well-known measure that combines the expected value μ and the variance σ^2 into the single objective function $\mu - \lambda \cdot \sigma^2$ where λ is a parameter that can be varied to aim for different tradeoffs between expectation and variance. In the context of optimization problems on MDPs, the VPE has been studied, e.g., in [7, 9].

Furthermore, the VPE finds use in an area of research primarily concerned with the tradeoffs between expected performance and risks, namely, the theory of financial markets and investment decision-making: In 1952, Harry Markowitz introduced *modern portfolio theory* that evaluates portfolios in terms of expected returns and variance of the returns [18], for which he was later awarded the Nobel Prize in economics. A portfolio lies on the *Markowitz efficient frontier* if the expected return cannot be increased without increasing the variance and, vice versa, the variance cannot be decreased without decreasing the expectation. The final choice of a portfolio on the efficient frontier depends on the investors preferences. In this context, the VPE $\mu - \lambda \cdot \sigma^2$ is a simple, frequently used way to express the preference of an investor using the single parameter λ capturing the risk-aversion of the investor (see, e.g., [10]). In more involved accounts, the investor's preference is described in terms of a utility function mapping returns to utilities. For the commonly used exponential utility function $u(x) = -e^{-\alpha x}$ and normally distributed returns, the objective of an investor trying to maximize expected utility turns out to be equivalent to the maximization of the VPE with parameter $\lambda = \alpha/2$ [2, 23].

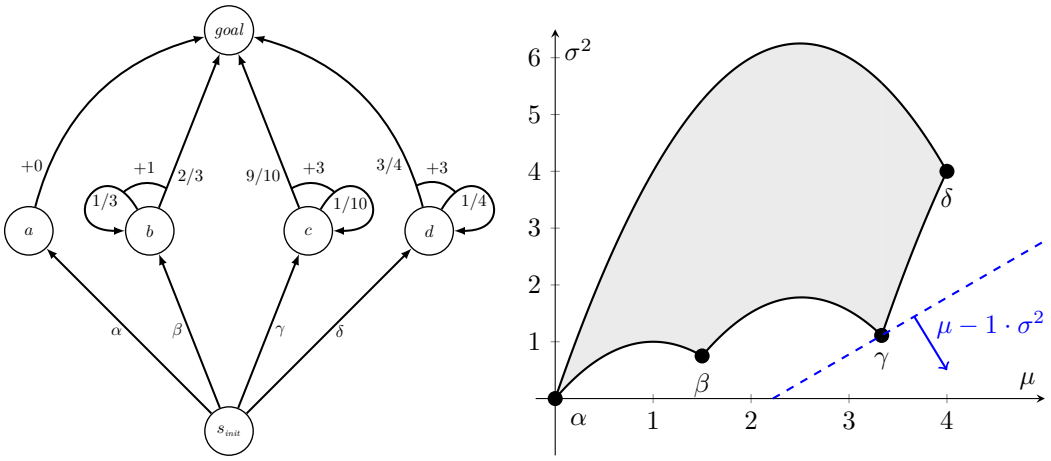


Figure 1 The left hand side shows the MDP $\mathcal{M}$ for Example 1. On the right hand side, all possible combinations of expected accumulated weight and variance for schedulers for $\mathcal{M}$ are depicted. The points corresponding to the four deterministic schedulers are marked by the corresponding action. Furthermore, the blue line indicates all points at which $\mu - 1 \cdot \sigma^2 = 20/9$ and the arrow indicates the direction in which the value of this objective function increases.

For an illustration of the VPE, consider the following example:

▶ **Example 1.** Consider the MDP $\mathcal{M}$ depicted in Figure 1 where non-trivial probability values as well as the weights accumulated are denoted next to the transitions. We want to analyze the possible trade-offs between the variance and the expected value of the accumulated weight that we can achieve in this MDP.

The only non-deterministic choice is in the state s_{init}. Choosing action α leads to *goal* with expected weight and variance 0. For the remaining actions, the accumulated weight follows a geometric distribution where in each step some weight k is accumulated and *goal* is reached with some probability p after the step. For such a distribution, it is well-known that the expected accumulated weight is k/p and the variance is $(k/p)^2 \cdot (1-p)$. Plugging in the respective values for the distributions reached after actions β, γ, and δ, we obtain the pairs of expectations and variances as depicted on the right-hand side of Figure 1. In particular, choosing γ leads to an expectation of $10/3$ and a variance of $10/9$.

Making use of randomization over two different actions τ and σ with probability p and $1-p$, respectively, for some $p \in (0,1)$, we will see in Remark 12 in Section 4 that the expected values and variances under the resulting schedulers lie on a parabolic line segment depicted in black that is uniquely determined by the expected values and variances under τ and σ. By further randomization over multiple actions, combinations of expectation and variance in the gray region in Figure 1 can be realized.

Consider now the VPE with parameter $\lambda = 1$. The dashed blue line in Figure 1 marks all points at which $\mu - 1 \cdot \sigma^2 = 20/9$. The arrow indicates in which direction the value of the VPE increases. So, it turns out that choosing action γ maximizes the VPE in this case; the slightly lower expectation compared to δ is compensated by a significantly lower variance. Geometrically, we can observe that the optimal point for the VPE for any parameter will always lie on the border of the convex hull of the region of feasible points in the μ-σ^2-plane as the VPE is a linear function of expectation and variance. For varying values of λ, also α (for $\lambda \geq 3$) and δ (for $\lambda \leq 1/13$) can constitute the optimal choice in s_{init} for the maximization of the VPE, while β is not optimal for any choice of λ as it lies in the interior of the convex hull of the feasible region. The results of Section 4 will show that in general, the optimal point for the VPE can be achieved by a deterministic finite-memory scheduler. ⌟

Contribution. The main results of this paper are the following:

1. Among all schedulers that optimize the expected accumulated weight before reaching a target, a variance-minimal scheduler can be computed in polynomial time and chosen to be memoryless and deterministic (Section 3).
2. The maximal VPE in MDPs with non-negative weights can be computed in exponential space. The maximum is obtained by a deterministic scheduler that can be computed in exponential space as well (Section 4). As memory, an optimal scheduler only needs to keep track of the accumulated weight up to a bound computable in polynomial time. As soon as the bound is reached, optimal schedulers can switch to the behavior of a variance-minimal scheduler among the expectation-minimal schedulers that can be computed by result 1.
3. The threshold problem whether the maximal VPE is greater or equal to a rational ϑ is in NEXPTIME and EXPTIME-hard (Section 4).

Related work. *Accumulated rewards.* In [16], a characterization of variance-minimal scheulders among the schedulers maximizing the expected accumulated weight in MDPs is given. Here, we provide a simpler proof based on the calculations of [28]; we moreover show how to compute such schedulers in polynomial time. [16] also contains hints for a similar characterization of discounted reward, and developments for mean payoff. Another closely related work is [17] which study the following multi-objective problem for the accumulated weight in finite-horizon MDPs: given η, ν is there a scheduler achieving an expectation of at least η, and a variance of at most ν? This problem is shown to be NP-hard, and exact pseudo-polynomial time algorithm is given for the existence of a scheduler with expectation η and variance $\leq \nu$. Furthermore, pseudo-polynomial approximation algorithms are given for optimizing the expectation under a constraint on the variance, and optimizing the variance under a constraint on the expectation.

Discounted rewards. In [12], the author proves that memoryless *moment-optimal* schedulers exist for the discounted reward, that is, schedulers that maximize the expectation, minimize the variance, maximize the third moment, and so on. Moreover, an algorithm is described to compute such schedulers. In [25], a formula for the variance of the discounted reward is given for memoryless schedulers and for the finite-horizon case, in MDPs and semi-MDPs. Variance-minimal schedulers among those maximizing the expected discounted reward until a target set is reached are studied in [29] for MDPs with varying discount factors. [31] presents a policy iteration algorithm to minimize variance of the discounted weight among schedulers achieving an expectation equal to a given constant.

Mean payoff. For mean payoff objectives, variance was studied in [26] for memoryless strategies, and algorithms were given to compute schedulers that achieve given bounds on the expectation and the variance [6]. The latter paper also considers the minimization of the variability, which is the average of the squared differences between the expected mean-payoff and each observed one-step reward. In [15], the author considers optimizing the expected mean payoff and the average variance. Average variance is defined as the limsup of the variances of the partial sums. They show how to minimize average variance among ϵ-optimal strategies for the expected mean payoff. Policy iteration algorithms were given in [30, 32] to minimize variance or variability of the mean payoff (without constraints on the expectation).

Variance-penalized expectation. The VPE was studied for finite-horizon MDPs with terminal rewards in [7]. In [9], this notion was studied for the expectation and the variability of both mean payoff and discounted rewards. [33] presents a policy iteration algorithm converging against *local* optima for a similar measure.

2 Preliminaries

We give basic definitions and present our notation (for details, see, e.g., [24]). Afterwards, we provide auxiliary results on expected frequencies used in the subsequent sections.

2.1 Notation and definitions

Notations for Markov decision processes. A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, Act, P, s_{init}, goal, wgt)$ where S is a finite set of states, Act a finite set of actions, $P\colon S \times Act \times S \to [0,1] \cap \mathbb{Q}$ the transition probability function, $s_{init} \in S$ the initial state, $goal \in S$ a designated target state, and $wgt\colon S \times Act \to \mathbb{Z}$ the weight function. We require that $\sum_{t \in S} P(s,\alpha,t) \in \{0,1\}$ for all $(s,\alpha) \in S \times Act$. We say that action α is enabled in state s iff $\sum_{t \in S} P(s,\alpha,t) = 1$ and denote the set of all actions that are enabled in state s by $Act(s)$. In this paper, for all MDPs, we assume that $goal$ is the only *trap* state in which no actions are enabled, that $goal$ is reachable from all other states s, and that all states are reachable from s_{init}. The paths of $\mathcal{M}$ are finite or infinite sequences $s_0\,\alpha_0\,s_1\,\alpha_1\ldots$ where states and actions alternate such that $P(s_i,\alpha_i,s_{i+1}) > 0$ for all $i \geq 0$. For $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\alpha_{k-1}\,s_k$, $wgt(\pi) = wgt(s_0,\alpha_0) + \ldots + wgt(s_{k-1},\alpha_{k-1})$ denotes the accumulated weight of π, $P(\pi) = P(s_0,\alpha_0,s_1)\cdot\ldots\cdot P(s_{k-1},\alpha_{k-1},s_k)$ its probability, and $last(\pi) = s_k$ its last state. A path is called *maximal* if it is infinite or ends in the trap state $goal$. The *size* of $\mathcal{M}$ is the sum of the number of states plus the total sum of the logarithmic lengths of the non-zero probability values $P(s,\alpha,s')$ as fractions of co-prime integers and the weight values $wgt(s,\alpha)$.

An *end component* of $\mathcal{M}$ is a strongly connected sub-MDP formalized by a subset $S' \subseteq S$ of states and a non-empty subset $\mathfrak{A}(s) \subseteq Act(s)$ for each state $s \in S'$ such that for each $s \in S'$, $t \in S$ and $\alpha \in \mathfrak{A}(s)$ with $P(s,\alpha,t) > 0$, we have $t \in S'$ and such that in the resulting sub-MDP all states are reachable from each other. An end-component is a 0-end-component if it only contains cycles whose accumulated weight is 0 (so-called 0-cycles) so that the accumulated weight is bounded on all (infinite) paths in the end component. We will further use the *mean payoff* measure as tool to classify end-components. For an infinite path ζ, the mean payoff is defined as $\mathbb{MP}(\zeta) = \liminf_{n\to\infty} \frac{1}{n} wgt(pref(\zeta,n))$ where $pref(\zeta,n)$ is the prefix of length n of ζ.

Scheduler. A *scheduler* for $\mathcal{M}$ is a function $\mathfrak{S}$ that assigns to each non-maximal path π a probability distribution over $Act(last(\pi))$. If the choice of a scheduler $\mathfrak{S}$ depends only on the current state, i.e., if $\mathfrak{S}(\pi) = \mathfrak{S}(\pi')$ for all non-maximal paths π and π' with $last(\pi) = last(\pi')$, we say that $\mathfrak{S}$ is *memoryless*. In this case, we also view schedulers as functions mapping states $s \in S$ to probability distributions over $Act(s)$. A scheduler $\mathfrak{S}$ that satisfies $\mathfrak{S}(\pi) = \mathfrak{S}(\pi')$ for all pairs of finite paths π and π' with $last(\pi) = last(\pi')$ and $wgt(\pi) = wgt(\pi')$ is called *weight-based* and can be viewed as a function from state-weight pairs $S \times \mathbb{Z}$ to probability distributions over actions. If there is a finite set X of memory modes and a memory update function $U : S \times Act \times S \times X \to X$ such that the choice of $\mathfrak{S}$ only depends on the current state after a finite path and the memory mode obtained from updating the memory mode according to U in each step, we say that $\mathfrak{S}$ is a finite-memory scheduler. A scheduler $\mathfrak{S}$ is called deterministic if $\mathfrak{S}(\pi)$ is a Dirac distribution for each path π in which case we also view the scheduler as a mapping to actions in $Act(last(\pi))$. Given a scheduler $\mathfrak{S}$, $\zeta = s_0\,\alpha_0\,s_1\,\alpha_1\ldots$ is a $\mathfrak{S}$-path iff ζ is a path and $\mathfrak{S}(s_0\,\alpha_0\ldots\alpha_{k-1}\,s_k)(\alpha_k) > 0$ for all $k \geq 0$. Given a scheduler $\mathfrak{S}$ and a finite $\mathfrak{S}$-path π, we define the residual scheduler $\mathfrak{S}{\uparrow}\pi$ by $\mathfrak{S}{\uparrow}\pi(\rho) = \mathfrak{S}(\pi \circ \rho)$ for each finite path ρ starting in $last(\pi)$.

Probability measure. We write $\Pr^{\mathfrak{S}}_{\mathcal{M},s}$ to denote the probability measure induced by a scheduler $\mathfrak{S}$ and a state s of an MDP $\mathcal{M}$. It is defined on the σ-algebra generated by the cylinder sets $Cyl(\pi)$ of all maximal extensions of a finite path $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\alpha_{k-1}\,s_k$ starting in state s, i.e., $s_0 = s$, by assigning to $Cyl(\pi)$ the probability that π is realized under $\mathfrak{S}$, which is $\mathfrak{S}(s_0)(\alpha_0)\cdot P(s_0,\alpha_0,s_1)\cdot\mathfrak{S}(s_0\alpha_0 s_1)(\alpha_1)\cdot\ldots\cdot\mathfrak{S}(s_0\alpha_0\ldots s_{k-1})(\alpha_{k-1})\cdot P(s_{k-1},\alpha_{k-1},s_k)$. For details, see [24].

For a random variable X that is defined on (some) maximal paths in $\mathcal{M}$, we denote the expected value of X under the probability measure induced by a scheduler $\mathfrak{S}$ and state s by $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)$. We define $\mathbb{E}^{\min}_{\mathcal{M},s}(X) = \inf_{\mathfrak{S}}\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)$ and $\mathbb{E}^{\max}_{\mathcal{M},s}(X) = \sup_{\mathfrak{S}}\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)$ where $\mathfrak{S}$ ranges over all schedulers for $\mathcal{M}$ under which X is defined almost surely. The variance of X under the probability measure determined by $\mathfrak{S}$ and s in $\mathcal{M}$ is denoted by $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M},s}(X)$ and defined by

$$\mathbb{V}^{\mathfrak{S}}_{\mathcal{M},s}(X) \stackrel{\text{def}}{=} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}((X - \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X))^2) = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X^2) - \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)^2.$$

Furthermore, for a measurable set of paths ψ with positive probability, $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X|\psi)$ denotes the conditional expectation of X under ψ. If $s = s_{init}$, we sometimes drop the subscript s.

These notations are extended to end-components of a given MDP, which are themselves seen as MDPs. We may, for instance, write $\mathbb{E}^{\min}_{\mathcal{E},s}(X)$ where $\mathcal{E}$ is an end-component of $\mathcal{M}$, and s is a state in $\mathcal{E}$, and the minimization ranges over schedulers of $\mathcal{M}$ that do not leave $\mathcal{E}$.

Accumulated weight. For maximal paths ζ of $\mathcal{M}$, we define the following random variable $\Diamond\!\!\!\!+goal$:

$$\Diamond\!\!\!\!+goal(\zeta) = \begin{cases} wgt(\zeta) & \text{if } \zeta \vDash \Diamond goal, \\ undefined & \text{otherwise.} \end{cases}$$

Recall that we only take schedulers under which a random variable is defined almost surely into account when addressing minimal or maximal expected values. For the expected value of $\Diamond\!\!\!\!+goal$ to be defined, it is necessary that $goal$ is reached almost surely. We call a scheduler $\mathfrak{S}$ with $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = 1$ *proper*. So, in the definition of the maximal (or minimal) expected accumulated weight $\mathbb{E}^{\max}_{\mathcal{M}}(\Diamond\!\!\!\!+goal) = \sup_{\mathfrak{S}}\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond\!\!\!\!+goal)$, $\mathfrak{S}$ ranges over all proper schedulers.

2.2 Auxiliary conclusions from results on expected frequencies

In this section, we present conclusions from well-known results on the expected frequencies of state-weight pairs in MDPs in the formulation in which we use them in the paper. Let $\mathcal{M} = (S, Act, P, s_{init}, goal, wgt)$ be an MDP with weights in $\mathbb{Z}$ and let $\mathfrak{S}$ be a scheduler. For each state-weight pair $(s,w) \in S \times \mathbb{Z}$, we define the *expected frequency* $\vartheta^{\mathfrak{S}}_{s,w}$ under $\mathfrak{S}$ by

$$\vartheta^{\mathfrak{S}}_{s,w} \stackrel{\text{def}}{=} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\text{number of visits to } s \text{ with accumulated weight } w)$$

where the random variable "number of visits to s with accumulated weight w" counts the number of prefixes π of a maximal paths ζ with $last(\pi) = s$ and $wgt(\pi) = w$. Note also that in MDPs $\mathcal{M}$ in which all end components have negative maximal expected mean-payoff, the expected frequencies of all state-weight pairs are finite under any scheduler.

▶ **Lemma 2.** *Let $\mathcal{M}$ be an MDP and let $\mathfrak{S}$ be a scheduler such that the expected frequency $\vartheta^{\mathfrak{S}}_{s,w}$ are finite for all state-weight pairs $(s,w) \in S \times \mathbb{Z}$. Then, there is a weight-based (randomized) scheduler $\mathfrak{T}$ with $\vartheta^{\mathfrak{S}}_{s,w} = \vartheta^{\mathfrak{T}}_{s,w}$ for all $(s,w) \in S \times \mathbb{Z}$.*

Proof sketch. Analogous to [24, Theorem 5.5.1]: For each state-weight pair (s, w) and each action $\alpha \in Act(s)$, let $\vartheta^{\mathfrak{S}}_{s,w,\alpha}$ be the expected number of times that α is chosen under $\mathfrak{S}$ after finite path ending in state s with weight w. Define the scheduler $\mathfrak{T}$ as a function from $S \times \mathbb{Z} \to \mathrm{Distr}(Act)$ by letting

$$\mathfrak{T}(s,w)(\alpha) \stackrel{\text{def}}{=} \frac{\vartheta^{\mathfrak{S}}_{s,w,\alpha}}{\vartheta^{\mathfrak{S}}_{s,w}}.$$ ◀

▶ **Corollary 3.** *Let $\mathcal{M}$ be an MDP. Let $\mathfrak{S}$ be a scheduler for which $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal)$ and $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal)$ are defined and for which the expected frequency $\vartheta^{\mathfrak{S}}_{s,w}$ are finite for all state-weight pairs $(s, w) \in S \times \mathbb{Z}$. Then, there is a weight-based scheduler $\mathfrak{T}$ with*

$$\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = \mathbb{E}^{\mathfrak{T}}_{\mathcal{M}}(\Diamond goal) \qquad \text{and} \qquad \mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = \mathbb{V}^{\mathfrak{T}}_{\mathcal{M}}(\Diamond goal).$$

Proof. The expected value and the variance of $\Diamond goal$ under a scheduler $\mathfrak{S}$ depend only on the expected frequencies $\vartheta^{\mathfrak{S}}_{goal,w}$ with $w \in \mathbb{Z}$. ◀

In this paper, we address questions concerning the possible combinations of expected value and variance of the random variable $\Diamond goal$. Due to this corollary, we can restrict our attention to weight-based schedulers for all investigations in the sequel.

Given two scheduler $\mathfrak{S}$ and $\mathfrak{T}$, our definition of schedulers does not directly allow us to define a new scheduler $\mathfrak{R}$ that behaves according to $\mathfrak{S}$ with probability $p \in (0, 1)$ and according to $\mathfrak{T}$ with probability $1-p$. For each state-weight pair (s, w) the expected frequency under the hypothetical scheduler $\mathfrak{R}$ would be $p \cdot \vartheta^{\mathfrak{S}}_{s,w} + (1-p) \cdot \vartheta^{\mathfrak{T}}_{s,w}$. The following lemma states that a scheduler achieving these frequencies exists:

▶ **Lemma 4.** *Let $\mathcal{M}$ be an MDP as above and let $\mathfrak{S}$ and $\mathfrak{T}$ be schedulers such that the expected frequency $\vartheta^{\mathfrak{S}}_{s,w}$ and $\vartheta^{\mathfrak{T}}_{s,w}$ are finite for all state-weight pairs $(s, w) \in S \times \mathbb{Z}$. Further, let $p \in (0, 1)$. Then, there exists a scheduler $\mathfrak{R}$ such that $\vartheta^{\mathfrak{R}}_{s,w} = p \cdot \vartheta^{\mathfrak{S}}_{s,w} + (1-p) \cdot \vartheta^{\mathfrak{T}}_{s,w}$ for all state-weight pairs (s, w).*

Proof sketch. Let $\vartheta^{\mathfrak{S}}_{s,w,\alpha}$ be defined as in the proof above. We define the weight-based scheduler $\mathfrak{R}$ as follows: For all state-weight pairs (s, w) and all $\alpha \in Act(s)$, let

$$\mathfrak{R}(s,w)(\alpha) = \frac{p \cdot \vartheta^{\mathfrak{S}}_{s,w,\alpha} + (1-p) \cdot \vartheta^{\mathfrak{T}}_{s,w,\alpha}}{p \cdot \vartheta^{\mathfrak{S}}_{s,w} + (1-p) \cdot \vartheta^{\mathfrak{T}}_{s,w}}.$$

The proof of the correctness is analogous to [13, Theorem 9.12]. ◀

This lemma allows us to introduce the following notation:

▶ **Definition 5.** *Given $\mathcal{M}$, $\mathfrak{S}$ and $\mathfrak{T}$ as in the previous lemma, we denote the scheduler $\mathfrak{R}$ whose existence is stated in the lemma by $p \cdot \mathfrak{S} \oplus (1-p) \cdot \mathfrak{T}$.*

3 Minimal variance among expectation-optimal schedulers

Let us call a scheduler *expectation-optimal* if it maximizies the expectation of $\Diamond goal$ from a given state s. In this section, we prove a result that is of interest in its own right and that will play a crucial role in our investigation of the optimization of the VPE in the following section. Namely, we show how to compute a scheduler that minimizes the variance among expectation-optimal schedulers in polynomial time. Note that in MDPs with weights in $\mathbb{Z}$, the mimimization of the expectation of $\Diamond goal$ can be reduced to the maximization by multiplying all weights with -1. This change of weights does not affect the variance and hence all results of this section also apply to expectation-minimal schedulers.

We assume that in a given MDP $\mathcal{M} = (S, Act, P, s_{init}, wgt, goal)$, the maximal achievable expectation of $\Diamond goal$ is finite. This can be checked in polynomial time [3] and, when this value is finite, it is achievable by memoryless deterministic strategies. By [3], all end components E of $\mathcal{M}$ are then either 0-end components or satisfy $\mathbb{E}^{\max}_E(\mathbb{MP}) < 0$.

The algorithm proceeds as follows. First, a transformation is applied so as to ensure that the only end-components in $\mathcal{M}$ are such that the maximal achievable expected mean payoff is negative; while preserving the expectation and the variance of $\Diamond goal$ (Lemma 6). We then prune the MDP so that all actions are optimal for maximizing the expected $\Diamond goal$ (Lemma 7). It follows that all schedulers then achieve the same expected $\Diamond goal$. We then derive an equation system in which the variances at each state are unknowns, while the expectations are known constants (Lemma 8). We conclude by showing that this equation system admits a unique solution and is solvable in polynomial time. Omitted proofs can be found in [22].

▶ **Lemma 6** ([3]). *Let $\mathcal{M} = (S, Act, P, s_{init}, wgt, goal)$ be an MDP with $\mathbb{E}^{\max}_{\mathcal{M}}(\Diamond goal) < \infty$. There is a polynomial transformation which outputs an MDP $\mathcal{M}'$ with the following properties:*

1. *$\mathcal{M}'$ has no 0-end-components,*
2. *there is a mapping f from schedulers of $\mathcal{M}$ to those of $\mathcal{M}'$ such that for all* proper *schedulers $\mathfrak{S}$ for $\mathcal{M}$, $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = \mathbb{E}^{f(\mathfrak{S})}_{\mathcal{M}'}(\Diamond goal)$, and $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = \mathbb{V}^{f(\mathfrak{S})}_{\mathcal{M}'}(\Diamond goal)$.*
3. *there is a mapping g from schedulers of $\mathcal{M}'$ to those of $\mathcal{M}$ such that for all* proper *schedulers $\mathfrak{S}$ for $\mathcal{M}'$, $\mathbb{E}^{g(\mathfrak{S})}_{\mathcal{M}}(\Diamond goal) = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond goal)$, and $\mathbb{V}^{g(\mathfrak{S})}_{\mathcal{M}}(\Diamond goal) = \mathbb{V}^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond goal)$.*

From now on, by the previous lemma, we assume that $\mathcal{M}$ only has end-components E with $\mathbb{E}^{\max}_{\mathcal{M}}(\mathbb{MP}) < 0$. We start by computing $\mathbb{E}^{\max}_{\mathcal{M}}(\Diamond goal)$ with the following equation:

$$\mu_s = \begin{cases} 0 & \text{if } s = goal, \\ \max_{a \in Act(s)} \sum_{s' \in S} P(s,a,s')(wgt(s,a) + \mu_{s'}) & \text{otherwise.} \end{cases} \qquad (*)$$

By [5], $(*)$ has the unique solution $\mu_s = \mathbb{E}^{\max}_{\mathcal{M}}(\Diamond goal)$ and this solution is computable in polynomial time via linear programming. Let us define $Act^{\max}(s)$ as the set of actions from s which satisfy $(*)$ with equality, i.e. $Act^{\max}(s) \stackrel{\text{def}}{=} \{a \in Act(s) \mid \mu_s = wgt(s,a) + \sum_{s' \in S} P(s,a,s')\mu_{s'}\}$, and let $\mathcal{M}'$ be obtained by restricting $\mathcal{M}$ to actions from $Act^{\max}$. By standard arguments (see [22]), we can show the following lemma:

▶ **Lemma 7.** *Let $(\mu_s)_{s \in S}$ be the solution of $(*)$ for an MDP $\mathcal{M}$. Let $\mathcal{M}'$ obtained from $\mathcal{M}$ as above. Then, $\mathcal{M}'$ has no end-components. Moreover, for all $s \in S$, all schedulers $\mathfrak{S}$ of $\mathcal{M}'$ achieve $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}'}[\Diamond goal] = \mu_s$.*

So, in order to find the variance-minimal scheduler among expectation optimal schedulers for $\mathcal{M}$, it is sufficient to find a variance-minimal scheduler for $\mathcal{M}'$. We derive the following lemma by adapting [28] to MDPs.

▶ **Lemma 8.** *Consider an MDP $\mathcal{M}$, and assume that there is a vector $(\mu_s)_{s \in S}$ of values such that all schedulers $\mathfrak{S}$ satisfy $\forall s \in S, \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(\Diamond goal) = \mu_s$. Then, $(\mathbb{V}^{\inf}_{\mathcal{M},s}(\Diamond goal))_{s \in S}$ is the unique solution of the following equation:*

$$V_s = \begin{cases} 0 & \textit{if } s = goal, \\ \min_{a \in Act(s)} \sum_{t \in S} P(s,a,t)\left((wgt(s,a) + \mu_t - \mu_s)^2 + V_t\right) & \textit{otherwise.} \end{cases} \qquad (**)$$

Note that the equation system $(**)$ is the same as the equation system used to minimize the expected accumulated weight before reaching *goal* under the weight function wgt' that assigns the non-negative weight $(wgt(s,a) + \mu_t - \mu_s)^2$ to the transition (s, α, t). So, this

equation system is solvable in polynomial time [8]. Using that all schedulers in $\mathcal{M}'$ achieve an expected accumulated weight of μ_s when starting in state s, the results of this section can be combined to the following theorem.

▶ **Theorem 9.** *Given an MDP $\mathcal{M}$ such that $\mathbb{E}^{\max}_{\mathcal{M}}[\Diamond goal] < \infty$, a memoryless deterministic, expectation-optimal scheduler $\mathfrak{S}$ such that $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M},s}[\Diamond goal]$ is minimal among all expectation-optimal schedulers for any state s is computable in polynomial time.*

4 Variance-penalized expectation

The goal of this section is to develop an algorithm to compute the optimal *variance-penalized expectation* (VPE). Given a rational $\lambda > 0$, we define the VPE with parameter λ under a scheduler $\mathfrak{S}$ as

$$\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} \stackrel{\text{def}}{=} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) - \lambda \cdot \mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) - \lambda \cdot \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal^2) + \lambda \cdot (\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal))^2.$$

Task. Compute the maximal variance-penalized expectation

$$\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} \stackrel{\text{def}}{=} \sup_{\mathfrak{S}} \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$$

where the supremum ranges over all proper schedulers. Furthermore, compute an optimal scheduler $\mathfrak{S}$ with $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$.

Throughout this section, we will restrict ourselves to MDPs $\mathcal{M} = (S, Act, P, s_{init}, wgt, goal)$ with a weight function $wgt\colon S \times Act \to \mathbb{N}$, i.e., we only consider MDPs with non-negative weights. Key results established in this section do not hold in the general setting with arbitrary weights and further complications arise. In the conclusions we will briefly discuss these complications.

As before, we are only interested in schedulers that reach the goal with probability 1. If the maximal expectation $\mathbb{E}^{\max}_{\mathcal{M}}(\Diamond goal) < \infty$, it is well-known that in this case of non-negative weights, all end components of $\mathcal{M}$ are 0-end components [3,8]. Hence, w.l.o.g., we can assume that $\mathcal{M}$ has no end components throughout this section by Lemma 6. In this case, $\Diamond goal$ is defined on almost all paths under any scheduler. So, in particular the values $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal)$ and $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal)$ are defined for all schedulers $\mathfrak{S}$. Furthermore, as we have seen in Corollary 3, it is sufficient to consider weight-based schedulers for the optimization of VPEs. The main result of this section is the following:

▶ **Main result.** Given an MDP $\mathcal{M}$ and λ as above, the optimal value $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$ and an optimal scheduler $\mathfrak{S}$ can be computed in exponential space. Given a rational ϑ, the threshold problem whether $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} \geq \vartheta$ is in NEXPTIME and EXPTIME-hard.

To obtain the main result, we will first prove that the maximal VPE is obtained by a deterministic scheduler (Section 4.1). This result can then be used for the EXPTIME-hardness proof for the threshold problem (Section 4.2). The key step to obtain the upper bounds of the main result is to show that optimal schedulers have to *minimize* the weight that is expected to still be accumulated after a computable bound of accumulated weight has been exceeded. We call such a bound a *saturation point* (Section 4.3). Finally, we show how to utilize the saturation point result to solve the threshold problem and to compute the optimal VPE (Section 4.4). Proofs omitted in this section can be found in [22].

▶ Remark 10. In the formulation presented here, the goal is to maximize the expected accumulated weight with a penalty for the variance. All results and proofs in this section, however, hold analogously for the variant $\sup_{\mathfrak{S}} -\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) - \lambda \cdot \mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal)$ of the maximal VPE in which the goal is to minimize the expected accumulated weight while receiving a penalty for the variance. In particular, the same saturation point works and optimal schedulers still have to minimize the expected accumulated weight as soon as the accumulated weight exceeds the saturation point. ⌟

4.1 Existence of optimal deterministic schedulers

We begin this section with a lemma describing how the variance of accumulated weight behaves under convex combinations of schedulers. This will allow us to show that the maximal VPE can be approximated by deterministic schedulers with the help of Lemma 14 describing a connection between randomization and convex combinations. This first lemma follows via basic arithmetic form the fact that the expected values of $\Diamond goal$ and $\Diamond goal^2$ depend linearly on the expected frequencies of the state-weight pairs $(goal, w)$ with $w \in \mathbb{N}$.

▶ **Lemma 11.** *Let $\mathcal{M} = (S, Act, P, s_{init}, wgt, goal)$ be an MDP with non-negative weights and no end components. Let $\mathfrak{S}$ and $\mathfrak{T}$ be two schedulers for $\mathcal{M}$. Let $p \in (0,1)$. The scheduler $\mathfrak{R} \stackrel{def}{=} p \cdot \mathfrak{S} \oplus (1-p) \cdot \mathfrak{T}$ satisfies*

$$\mathbb{V}^{\mathfrak{R}}_{\mathcal{M}}(\Diamond goal) = p \cdot \mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) + (1-p) \cdot \mathbb{V}^{\mathfrak{T}}_{\mathcal{M}}(\Diamond goal) + p \cdot (1-p) \cdot (\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) - \mathbb{E}^{\mathfrak{T}}_{\mathcal{M}}(\Diamond goal))^2.$$

Proof sketch. The claim follows from straight-forward calculations using that the expected values of $\Diamond goal$ and $\Diamond goal^2$ depend linearly on the expected frequencies of the state-weight pairs $(goal, w)$ with $w \in \mathbb{N}$. ◀

▶ Remark 12. Given two schedulers $\mathfrak{S}$ and $\mathfrak{S}'$ under which the expectation and variance are (η, ν) and (η', ν'), respectively, such that $\eta < \eta'$, there is a unique convex combination $\mathfrak{T}$ of the two schedulers with expectation x for all $x \in [\eta, \eta']$. Viewing the variance of these convex combinations as a function $V : [\eta, \eta'] \to \mathbb{R}$, we can observe the following using the previous Lemma 11:

$$V(x) = \nu + \frac{x-\eta}{\eta'-\eta} \cdot (\nu' - \nu) + \frac{x-\eta}{\eta'-\eta} \cdot \frac{\eta'-x}{\eta'-\eta} \cdot (\eta'-\eta)^2 = \nu + \frac{x-\eta}{\eta'-\eta} \cdot (\nu'-\nu) + (x-\eta) \cdot (\eta'-x).$$

The coefficient before x^2 in this quadratic polynomial hence is always -1. ⌟

The following lemma stating the continuity of the VPE will be useful in several ways: If we manipulate schedulers at one state-weight pair at a time, we can reason about the scheduler we obtain in the limit after manipulating the scheduler at all state-weight pairs, e.g., in the proof of Theorem 15 below. Further, it will allow us to prove that there is an optimal scheduler, i.e., that the supremum in the definition of $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$ is in fact a maximum.

▶ **Lemma 13** (Continuity of VPE). *Let $\mathcal{M}$ and $\lambda > 0$ be as above. The variance-penalized expectation as a function from weight-based schedulers to $\mathbb{R}$ is (uniformly) continuous in the following sense: Given $\varepsilon > 0$, there is a natural number N_ε such that for all weight-based schedulers $\mathfrak{S}$ and $\mathfrak{T}$ that agree on all state-weight pairs (s, w) with $w \leq N_\varepsilon$, we have*

$$\left| \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} - \mathbb{VPE}[\lambda]^{\mathfrak{T}}_{\mathcal{M}} \right| < \varepsilon.$$

Proof sketch. The claim follows from the fact that the probability that a high amount of weight w is accumulated under any scheduler decreases exponentially as w tends to ∞. ◀

For the final ingredient to show that deterministic schedulers approximate the optimal VPE, we take a closer look at the relation of randomization to convex combinations of schedulers. For the following lemma, let $\mathfrak{S}$ be a weight-based scheduler for an MDP $\mathcal{M}$ as before. Assume that there is a state-weight pair $(s, w) \in S \times \mathbb{N}$ reachable under $\mathfrak{S}$ such that $\mathfrak{S}$ chooses two different actions α and β with probabilities q and $1-q$, respectively, for some $q \in (0,1)$. Let $\mathfrak{S}_\alpha$ be the scheduler that agrees with $\mathfrak{S}$ on all state-weight pairs except for (s, w) and that chooses α with probability 1 at (s, w). Define $\mathfrak{S}_\beta$ analogously. The technical proof of the following lemma can be found in [22].

▶ **Lemma 14.** *Let $\mathcal{M}$, $\mathfrak{S}$, $\mathfrak{S}_\alpha$, $\mathfrak{S}_\beta$, and q be as above. There is a value $p \in (0,1)$ such that the expected frequencies of all state-weight pairs are the same under $\mathfrak{S}$ and $p \cdot \mathfrak{S}_\alpha \oplus (1-p) \cdot \mathfrak{S}_\beta$.*

▶ **Theorem 15** (Deterministic schedulers approximate optimal VPE)**.** *Let $\mathcal{M}$ be an MDP with non-negative weights and without end components and let $\lambda > 0$. For each scheduler $\mathfrak{S}$, there is a deterministic weight-based scheduler $\mathfrak{T}$ with*

$$\mathbb{VPE}[\lambda]^{\mathfrak{T}}_{\mathcal{M}} \geq \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}.$$

Proof sketch. W.l.o.g., we can assume that $\mathfrak{S}$ is weight-based by Corollary 3. At a single state-weight pair (s, w) at which $\mathfrak{S}$ makes use of randomization between, we can (potentially repeatedly) apply Lemma 14 and Lemma 11 to find a scheduler $\mathfrak{S}'$ that does not make use of this randomization but satisfies $\mathbb{VPE}[\lambda]^{\mathfrak{S}'}_{\mathcal{M}} \geq \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$. Going through all state-weight pairs in this fashion, we can construct an infinite sequence of schedulers with non-decreasing VPE in which randomization is successively removed at all state-weight pairs. In the limit, we obtain a well defined deterministic weight-based scheduler $\mathfrak{T}$. Lemma 13 allows us to conclude that $\mathbb{VPE}[\lambda]^{\mathfrak{T}}_{\mathcal{M}} \geq \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$. ◀

In the definition of the maximal variance-penalized expectation $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} = \sup_{\mathfrak{S}} \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$, it is sufficient to let the supremum range over deterministic weight-based schedulers $\mathfrak{S}$ in the light of this theorem. For the proof of the existence of optimal schedulers, we make use of an analytic argument: Continuous functions on compact space obtain their maximum. The continuity shown in Lemma 13 applied to the space of deterministic weight-based schedulers can be reformulated as continuity with respect to a metric on this space. Namely, we define the metric $d_{\mathcal{M}}$ on the set of deterministic weight-based schedulers as follows: Given two deterministic weight-based schedulers $\mathfrak{S}$ and $\mathfrak{T}$ for $\mathcal{M}$, first let

$$m(\mathfrak{S}, \mathfrak{T}) \stackrel{\text{def}}{=} \min\{w \mid \text{there is a state } s \in S \text{ with } \mathfrak{S}(s, w) \neq \mathfrak{T}(s, w)\}.$$

We then define $d_{\mathcal{M}}(\mathfrak{S}, \mathfrak{T}) \stackrel{\text{def}}{=} 2^{-m(\mathfrak{S}, \mathfrak{T})}$. This metric indeed turns the set of deterministic weight-based schedulers into a compact space as shown in [20]:

▶ **Lemma 16** (Compactness of the space of deterministic weight-based schedulers [20])**.** *Let $\mathcal{M}$ be as above. The space of all deterministic weight-based schedulers with the topology induced by the metric $d_{\mathcal{M}}$ is compact.*

▶ **Theorem 17** (Existence of an optimal deterministic weight-based scheduler)**.** *Let $\mathcal{M}$ and $\lambda > 0$ be as above. There is a deterministic weight-based scheduler $\mathfrak{S}$ with*

$$\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}.$$

Proof. The claim follows from Lemma 13, Theorem 15, and Lemma 16, as continuous functions on compact spaces obtain their maximum. ◀

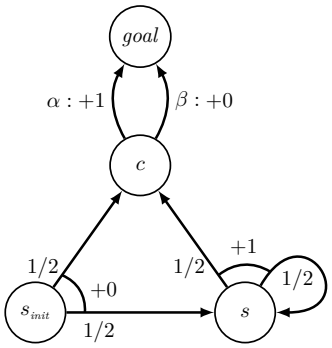


Figure 2 The MDP $\mathcal{M}$ used in Example 19.

4.2 Hardness of the threshold problem

The result that the maximal variance-penalized expectation can be achieved by a deterministic scheduler can be used for the following hardness result:

▶ **Theorem 18.** *Given an MDP $\mathcal{M}$ with non-negative weights and two rationals $\lambda, \vartheta > 0$, deciding whether $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} \geq \vartheta$ is EXPTIME-hard. Furthermore, for acyclic MDPs $\mathcal{M}$, the problem is PSPACE-hard.*

Proof sketch. We reduce from the following problem which is shown to be EXPTIME-hard in general and PSPACE-hard for acyclic MDPs in [11]: Given an MDP $\mathcal{M}$ and a natural number $T > 0$ such that *goal* is reached in $\mathcal{M}$ almost surely under all schedulers, decide whether there is a scheduler $\mathfrak{S}$ such that $\Pr^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal = T) = 1$.

The idea is to construct an MDP $\mathcal{M}'$ that reaches *goal* with weight T with probability $1/2$ directly and otherwise behaves like $\mathcal{M}$. By choosing λ sufficiently large, we can show that $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}'} \geq T$ is only possible if and only if there is a scheduler with $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = 0$. This scheduler then has to reach *goal* with weight T on all paths. The necessary technical calculations can be found in [22]. ◀

4.3 Saturation Point

In the sequel, we will provide a series of results that allow us to further restrict the class of deterministic schedulers that we have to consider when maximizing the variance-penalized expectation. In the end, we obtain a finite set of deterministic finite-memory schedulers among which there is a scheduler achieving the optimal variance-penalized expectation. In particular, this means that the optimum is computable.

The key step is the insight that we can provide a natural number K computable in polynomial time such that an optimal scheduler $\mathfrak{S}$ for the variance-penalized expectation has to minimize the expected accumulated weight before reaching *goal* once a weight of at least K has already been accumulated on a run. Furthermore, the behavior of $\mathfrak{S}$ after a weight of at least K has been accumulated must minimize the variance of the weight that will still be accumulated among all expectation-minimal schedulers. We call this value K a *saturation point*.

▶ **Example 19.** The MDP $\mathcal{M}$ in Figure 2 aims to provide some intuition on the results of this section. The state c in this MDP is reached with accumulated weight n with probability $(1/2)^{n+1}$ for all $n \in \mathbb{N}$. Then, the choice has to be made whether to collect weight $+1$ or

0 before moving to *goal*. We want to take a closer look at a family of special weight-based deterministic finite-memory schedulers for $\mathcal{M}$: Let $\mathfrak{S}_k$ be the scheduler that chooses action α in c if the accumulated weight is less than k and otherwise chooses action β.

For these schedulers, we can explicitly provide expectation and variance: The probability that the scheduler $\mathfrak{S}_k$ chooses α in c is $1-(1/2)^k$. As the expected accumulated weight before reaching c is 1, we obtain a total expected accumulated weight of

$$\mathbb{E}^{\mathfrak{S}_k}_{\mathcal{M}}(\Diamond goal) = 2 - \frac{1}{2^k}.$$

To obtain the variance, we compute $\mathbb{E}^{\mathfrak{S}_k}_{\mathcal{M}}(\Diamond goal^2)$:

$$\mathbb{E}^{\mathfrak{S}_k}_{\mathcal{M}}(\Diamond goal^2) = \sum_{n=0}^{k-1} \frac{1}{2^{k+1}} \cdot (n+1)^2 + \sum_{n=k}^{\infty} \frac{1}{2^{k+1}} \cdot n^2$$
$$= \sum_{n=0}^{\infty} \frac{1}{2^{k+1}} \cdot (n+1)^2 - \sum_{n=k}^{\infty} \frac{1}{2^{k+1}} \cdot (2n+1) = 6 - \frac{2k+3}{2^k}.$$

We can then easily compute the variance

$$\mathbb{V}^{\mathfrak{S}_k}_{\mathcal{M}}(\Diamond goal) = \mathbb{E}^{\mathfrak{S}_k}_{\mathcal{M}}(\Diamond goal^2) - (\mathbb{E}^{\mathfrak{S}_k}_{\mathcal{M}}(\Diamond goal))^2 = 2 + \frac{1-2k}{2^k} - \frac{1}{4^k}.$$

For $\lambda = 1$, we obtain the following VPE:

$$\mathbb{VPE}[\lambda]^{\mathfrak{S}_k}_{\mathcal{M}} = \frac{k-1}{2^{k-1}} + \frac{1}{4^k}.$$

Comparing scheduler $\mathfrak{S}_k$ to $\mathfrak{S}_{k+1}$, we obtain: $\mathbb{VPE}[\lambda]^{\mathfrak{S}_{k+1}}_{\mathcal{M}} - \mathbb{VPE}[\lambda]^{\mathfrak{S}_k}_{\mathcal{M}} = 2 - k/2^k - 3/4^k$. This difference is negative for $k \geq 2$ and positive for $k = 1$. We conclude that among the schedulers $\mathfrak{S}_k$, the scheduler $\mathfrak{S}_2$ is VPE-optimal. Interestingly, this means choosing not to accumulate the additional weight $+1$ by choosing α is better already for small amounts of accumulated weight. Intuitively, the reason is that choosing α for an accumulated weight ≥ 2 has a larger effect on the variance than on the expectation. Increasing the expectation in particular also increases the squared deviation of the path that reach *goal* with weight 1 which has probability $1/2$ under $\mathfrak{S}_k$ for $k \geq 2$. The saturation point result of this section will tell us that an optimal scheduler always has to minimize the weight that is expected to still be accumulated once sufficiently much weight has already been accumulated. ⌟

Let $\mathcal{M} = (S, Act, P, s_{init}, wgt, goal)$ be an MDP without end components and with non-negative weights as above and let $\lambda > 0$ be a rational. Before we define K and show that it can be computed in polynomial time, we need some additional notation.

For each state $s \in S$, define $e_s \stackrel{\text{def}}{=} \mathbb{E}^{\min}_{\mathcal{M},s}(\Diamond goal)$. For each state $s \in S \setminus \{goal\}$, we define the subset $Act^{\min}(s) \subseteq Act(s)$ of actions allowing to minimize the expectation analogously to $Act^{\max}$ before: $Act^{\min}(s) \stackrel{\text{def}}{=} \{\alpha \in Act(s) \mid e_s = wgt(s,\alpha) + \sum_{t \in S} P(s,\alpha,t) \cdot e_t\}$. Choosing an action not belonging to $Act^{\min}(s)$ in state s ensures that the expected accumulated weight before reaching *goal* is higher than the minimal possible value. Further, we can define the minimal amount by which choosing a non-minimizing action increases the expectation:

$$\delta \stackrel{\text{def}}{=} \min\{(wgt(s,\alpha) + \sum_{t \in S} P(s,\alpha,t) \cdot e_t) - e_s \mid s \in S \setminus \{goal\} \text{ and } \alpha \in Act(s) \setminus Act^{\min}(s)\}.$$

If the set on the right hand side is empty, all schedulers minimize the expected accumulated weight before reaching *goal* and the claims of this section hold trivially. So, we can assume that this set is non-empty. By the definition of $Act^{\min}$, we observe that $\delta > 0$.

Next, we can compute an upper bound U_1 for $\mathbb{E}^{\max}_{\mathcal{M},s}(\diamondsuit goal)$ for all states s by computing the maximal value $U_1 \stackrel{\text{def}}{=} \max_{s\in S} \mathbb{E}^{\max}_{\mathcal{M},s}(\diamondsuit goal)$.

Finally, let ε be the minimal transition probability present in $\mathcal{M}$. As $\mathcal{M}$ has no end components, the only trap state *goal* is reached within $n \stackrel{\text{def}}{=} |S|$ steps under each scheduler with probability at least ε^n. Let W be the largest weight in $\mathcal{M}$. Within n steps at most a weight of $n \cdot W$ is accumulated. We use these observations for the following two values:

- First, we can provide a value $B_{1/2}$ such that the probability that a weight above $B_{1/2}$ is accumulated under any scheduler is at most $1/2$: For this, let $b_{1/2}$ be such that $((1-\varepsilon^n))^{b_{1/2}} \leq 1/2$. This is the case if and only if $b_{1/2}$ is at least

$$\frac{\log(1/2)}{\log(1-\varepsilon^n)} = -\frac{1}{\log(1-\varepsilon^n)} < \frac{1}{\varepsilon^n}.$$

  So, we can choose $b_{1/2}$ to be $\frac{1}{\varepsilon^n}$. Then, with probability at most $1/2$, a path has length at least $n \cdot b_{1/2}$. This allows us to defined $B_{1/2} \stackrel{\text{def}}{=} b_{1/2} \cdot n \cdot W$.
- Second, we compute an upper bound U_2 for $\max_{s\in S} \mathbb{E}^{\max}_{\mathcal{M},s}(\diamondsuit goal^2)$: With probability ε^n a path has weight at most $n \cdot W$; with probability $(1-\varepsilon^n)\cdot\varepsilon^n$ it has weight at most $2 \cdot n \cdot W$; with probability $(1-\varepsilon^n)^2 \cdot \varepsilon^n$ it has weight at most $3 \cdot n \cdot W$; and so on. So, we get that $\max_{s\in S} \mathbb{E}^{\max}_{\mathcal{M},s}(\diamondsuit goal^2) \leq \sum_{i=0}^{\infty}(1-\varepsilon^n)^i \cdot \varepsilon^n \cdot ((i+1)\cdot n \cdot W)^2$. This allows us to define

$$U_2 \stackrel{\text{def}}{=} \frac{2\cdot n^2 \cdot W^2}{\varepsilon^{2n}} \geq \frac{(2-\varepsilon^n)\cdot n^2 \cdot W^2}{\varepsilon^{2n}} = \sum_{i=0}^{\infty}(1-\varepsilon^n)^i \cdot \varepsilon^n \cdot ((i+1)\cdot n \cdot W)^2.$$

We are now in the position to define the saturation point K: Let K be the least natural number with

$$K \geq B_{1/2} = \frac{n\cdot W}{\varepsilon^n} \qquad \text{and} \qquad K \geq \frac{U_1/\lambda + U_2 + 2U_1 + U_1^2/2}{\delta} + 1.$$

The definition of K is arguably a bit cumbersome, but the choices will become clear in the proof of Theorem 20. All values involved except for δ and U_1 can be computed directly from n, W, and ε in polynomial time. The values δ and U_1 require to maximize or minimize the expected value of $\diamondsuit goal$ from all states, i.e., to solve an SSPP which can be done in polynomial time by linear programming [5, 8].

▶ **Theorem 20** (Saturation point). *Let $\mathcal{M}$, $\lambda > 0$ and K be as above. Let $\mathfrak{S}$ be a scheduler with $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$. Then, for each finite $\mathfrak{S}$-path π with $wgt(\pi) \geq K$, the residual scheduler $\mathfrak{S}{\uparrow}\pi$ satisfies*

$$\mathbb{E}^{\mathfrak{S}\uparrow\pi}_{\mathcal{M},last(\pi)}(\diamondsuit goal) = \mathbb{E}^{\min}_{\mathcal{M},last(\pi)}(\diamondsuit goal).$$

Proof sketch. Let $\mathfrak{S}$ be a scheduler with $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$. Suppose there is a $\mathfrak{S}$-path π' with $wgt(\pi') \geq K$ such that $\mathbb{E}^{\mathfrak{S}\uparrow\pi'}_{\mathcal{M},last(\pi')}(\diamondsuit goal) > \mathbb{E}^{\min}_{\mathcal{M},last(\pi')}(\diamondsuit goal)$. Then, there must be an $\mathfrak{S}$-path π that extends π' such that $\mathfrak{S}$ chooses an action $\alpha \notin Act^{\min}(last(\pi))$ with positive probability.

The residual scheduler $\mathfrak{T}$ of $\mathfrak{S}$ after π in case $\mathfrak{S}$ chooses α then satisfies $\mathbb{E}^{\mathfrak{T}}_{\mathcal{M},last(\pi)}(\diamondsuit goal) \geq \mathbb{E}^{\min}_{\mathcal{M},last(\pi)}(\diamondsuit goal) + \delta$. We let $\mathfrak{S}'$ be a scheduler that behaves like $\mathfrak{S}$ unless $\mathfrak{S}$ chooses α after π. In this case, $\mathfrak{S}'$ minimizes the expected value of $\diamondsuit goal$ from then on. We consider the difference $\mathbb{VPE}[\lambda]^{\mathfrak{S}'}_{\mathcal{M}} - \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$. Using the bounds U_1 and U_2 and that the probability of π is at most $1/2$ as $K \geq B_{1/2}$, we obtain a lower bound for this difference that consists of an expression in terms of U_1, U_2, and λ plus the term

$$\lambda \cdot wgt(\pi) \cdot (\mathbb{E}^{\mathfrak{T}}_{\mathcal{M},last(\pi)}(\diamondsuit goal) - \mathbb{E}^{\min}_{\mathcal{M},last(\pi)}(\diamondsuit goal)).$$

Observing that this term is greater or equal to $\lambda \cdot K \cdot \delta$, the definition of K was chosen exactly so that we can conclude that $\mathbb{VPE}[\lambda]^{\mathfrak{S}'}_{\mathcal{M}} - \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} > 0$. So, $\mathfrak{S}$ was not VPE-maximal yielding a contradiction. ◀

By the results of Section 3, there is a memoryless deterministic scheduler $\mathfrak{V}$ that minimizes the variance among all schedulers minimizing the expected accumulated weight before reaching *goal*. More precisely, for all states s, the scheduler $\mathfrak{V}$ satisfies $\mathbb{E}^{\mathfrak{V}}_{\mathcal{M},s}(\Diamond goal) = \mathbb{E}^{\min}_{\mathcal{M},s}(\Diamond goal)$ and $\mathbb{V}^{\mathfrak{V}}_{\mathcal{M},s}(\Diamond goal) = \inf_{\mathfrak{M}} \mathbb{V}^{\mathfrak{M}}_{\mathcal{M},s}(\Diamond goal)$ where the infimum ranges over all schedulers $\mathfrak{M}$ with $\mathbb{E}^{\mathfrak{M}}_{\mathcal{M},s}(\Diamond goal) = \mathbb{E}^{\min}_{\mathcal{M},s}(\Diamond goal)$. We use the existence of this scheduler in the following theorem.

▶ **Theorem 21.** *Let $\mathcal{M}$, $\lambda > 0$, K, and $\mathfrak{V}$ be as above. Let $\mathfrak{S}$ be a deterministic scheduler with $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$. Let $\mathfrak{T}$ be the scheduler that agrees with $\mathfrak{S}$ on all paths π with weight less than K and that chooses actions according to the memoryless deterministic scheduler $\mathfrak{V}$ after paths π' with $wgt(\pi') \geq K$. This scheduler $\mathfrak{T}$ satisfies $\mathbb{VPE}[\lambda]^{\mathfrak{T}}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$, too.*

Proof sketch. Given a scheduler $\mathfrak{S}$ with $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$, and a path π with $wgt(\pi) \geq K$, we compare the scheduler $\mathfrak{S}$ to the scheduler $\mathfrak{S}'$ that behaves like $\mathfrak{S}$, but switches to the behavior of $\mathfrak{V}$ after π. We obtain that $\mathbb{VPE}[\lambda]^{\mathfrak{S}'}_{\mathcal{M}} \geq \mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$ is equivalent to $\mathbb{E}^{\mathfrak{V}}_{\mathcal{M}}(\Diamond goal^2) \leq \mathbb{E}^{\mathfrak{S}\upharpoonright\pi}_{\mathcal{M}}(\Diamond goal^2)$. This holds because $\mathbb{V}^{\mathfrak{V}}_{\mathcal{M}}(\Diamond goal) \leq \mathbb{V}^{\mathfrak{S}\upharpoonright\pi}_{\mathcal{M}}(\Diamond goal)$ as $\mathfrak{V}$ and $\mathfrak{S}\upharpoonright\pi$ achieve the same expectation. Using a continuity argument as before, we show that changing the behavior of $\mathfrak{S}$ to $\mathfrak{V}$ after all paths with weight at least K does not decrease the VPE. ◀

Put together, we have shown that the maximal VPE is obtained by a weight-based deterministic scheduler that switches to the memoryless behavior of $\mathfrak{V}$ as soon as a weight of at least K has been accumulated, which also means that it uses only finite memory.

4.4 Computation of the optimal VPE

Given an MDP $\mathcal{M} = (S, Act, P, s_{init}, wgt, goal)$ with non-negative weights and without end components and $\lambda > 0$ as before, let K be the saturation point given above. Note that K is computable in polynomial time and that hence its numerical value is at most exponential in the size of $\mathcal{M}$. We construct the following MDP $\mathcal{M}'$ that encodes the weights that are accumulated until the saturation point is exceeded into the state space: Let W be the maximal weight occurring in $\mathcal{M}$. The set of states is $S' = S \times \{0, 1, \ldots, K + W - 1\}$. The set of actions remains unchanged. The new probability transition function is given by $P'((s,w),(t,w + wgt(s,\alpha))) \stackrel{\text{def}}{=} P(s,\alpha,t)$ for all $s,t \in S$, all $w < K$, and all $\alpha \in Act(s)$. All remaining transition probabilities are 0. Note that this means that all states of the form $(goal, w)$ with $w \in \{0, 1, \ldots, K + W - 1\}$ and of the form (s, w) with $s \in S$ and $w \in \{K, K + 1, \ldots, K + W - 1\}$ are trap states in $\mathcal{M}'$. The initial state is $s'_{init} \stackrel{\text{def}}{=} (s_{init}, 0)$. The weight function is not relevant in $\mathcal{M}'$.

Let $\mathfrak{V}$ be the memoryless deterministic scheduler for $\mathcal{M}$ as in Theorem 21 that specifies the optimal behavior in order to maximize the variance-penalized expectation as soon as a weight of at least K has been accumulated. Let us call the set of weight-based deterministic schedulers for $\mathcal{M}$ that behave like $\mathfrak{V}$ after a weight of at least K has been accumulated by $\mathrm{WD}_K(\mathcal{M})$. Clearly, there is a natural one-to-one-correspondence between memoryless deterministic schedulers for $\mathcal{M}'$ and schedulers in $\mathrm{WD}_K(\mathcal{M})$.

By the results of Section 3, for each state $s \in S$, we can compute the values

$$e_s \stackrel{\text{def}}{=} \mathbb{E}^{\mathfrak{V}}_{\mathcal{M},s}(\Diamond goal) \qquad \text{and} \qquad q_s \stackrel{\text{def}}{=} \mathbb{E}^{\mathfrak{V}}_{\mathcal{M},s}(\Diamond goal^2) = \mathbb{V}^{\mathfrak{V}}_{\mathcal{M},s}(\Diamond goal) + e_s^2$$

in polynomial time. The following lemma now allows us to express the VPE in $\mathcal{M}$ in terms of reachability probabilities in $\mathcal{M}'$.

▶ **Lemma 22.** *Let $\mathcal{M}$, $\mathcal{M}'$, K and λ be as above. Given a scheduler $\mathfrak{S} \in \mathrm{WD}_K(\mathcal{M})$ also viewed as a memoryless deterministic scheduler for $\mathcal{M}'$, let*

$$\mu \stackrel{def}{=} \sum_{w<K} \Pr^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond(goal, w)) \cdot w + \sum_{s\in S, w\geq K} \Pr^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond(s,w)) \cdot (w + e_s). \tag{†}$$

Then,

$$\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} = \mu - \lambda \cdot \Big(\sum_{w<K} \Pr^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond(goal, w)) \cdot (w-\mu)^2) + \sum_{s\in S, w\geq K} \Pr^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond(s,w)) \cdot ((w-\mu)^2 + 2(w-\mu)e_s + q_s)) \Big). \tag{‡}$$

Proof. It is clear that $\mu = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal)$. So, we have to show that

$$\begin{aligned} &\mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal) = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}((\Diamond goal - \mu)^2) \\ =& \sum_{w<K} \Pr^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond(goal, w)) \cdot (w-\mu)^2) \\ &+ \sum_{s\in S, w\geq K} \Pr^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond(s,w)) \cdot ((w-\mu)^2 + 2(w-\mu)e_s + q_s)). \end{aligned} \tag{∘}$$

The event $\Diamond(s,w)$ that (s,w) is reached in $\mathcal{M}'$ corresponds to the event that a path in $\mathcal{M}$ has a prefix of weight w ending in s. We denote this event in $\mathcal{M}$ also by $\Diamond(s,w)$. If $\Pr^{\mathfrak{S}}_{\mathcal{M}'}(\Diamond(s,w)) > 0$ for $w \geq K$, then

$$\begin{aligned} &\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}((\Diamond goal - \mu)^2 | \Diamond(s,w)) \\ =&\mathbb{E}^{\mathfrak{V}}_{\mathcal{M},s}((\Diamond goal + w - \mu)^2) \\ =&(w-\mu)^2 + 2(w-\mu)\cdot \mathbb{E}^{\mathfrak{V}}_{\mathcal{M},s}(\Diamond goal) + \mathbb{E}^{\mathfrak{V}}_{\mathcal{M},s}(\Diamond goal^2). \end{aligned}$$

So, the sums in equation (∘) sum up the conditional expectation of $(\Diamond goal - \mu)^2$ in $\mathcal{M}$ under the conditions that $(goal, w)$ is reached for $w < K$ or that the state s is the first one reached when the accumulated weight exceeds K with weight $w \geq K$, multiplied by the respective probabilities of the conditions. ◀

Putting everything together, we arrive at the main result.

▶ **Theorem 23.** *Let $\mathcal{M}$ and λ be as above. Given a rational ϑ, the threshold problem whether $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} \geq \vartheta$ is in NEXPTIME. The optimal value $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$ and an optimal scheduler can be computed in exponential space.*

Proof. The threshold problem can be decided in non-deterministic exponential time as follows: Given $\mathcal{M}$ and λ, compute K in polynomial time and construct $\mathcal{M}'$ as above (of exponential size) in exponential time. Guess a memoryless deterministic scheduler $\mathfrak{S}$ for $\mathcal{M}'$ also viewed as a scheduler in $\mathrm{WD}_K(\mathcal{M})$. The reachability probabilities for all trap states in $\mathcal{M}'$ under $\mathfrak{S}$ can then be computed in time polynomial in the size of $\mathcal{M}'$. With the help of equations (†) and (‡) from Lemma 22, $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$ can be computed from these reachability probabilities in time polynomial in the size of $\mathcal{M}'$. If $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} \geq \vartheta$, accept. By Theorem 21, $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} \geq \vartheta$ iff there is a scheduler $\mathfrak{S}$ in $\mathrm{WD}_K(\mathcal{M})$ with $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}} \geq \vartheta$. Due to the one-to-one correspondence between schedulers in $\mathrm{WD}_K(\mathcal{M})$ and memoryless deterministic schedulers for $\mathcal{M}'$, this establishes the correctness of the algorithm.

To compute the optimal value $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$, we compute $\mathbb{VPE}[\lambda]^{\mathfrak{S}}_{\mathcal{M}}$ for all schedulers $\mathfrak{S}$ in $\mathrm{WD}_K(\mathcal{M})$ in the same fashion and always store the highest value found so far. As the memoryless schedulers for $\mathcal{M}'$ have an exponentially large representation, this can be done in exponential space and the optimal scheduler can be returned as well. ◀

5 Conclusion

In our results, there remains a complexity gap between the EXPTIME-lower bounds and the exponential-space and NEXPTIME-upper bounds for the optimization of the VPE in MDPs with non-negative weights and the corresponding threshold problem, respectively. Here, we want to shed some light on this complexity gap: It is well-known that the possible vectors of expected frequencies of all states in an MDP can be characterized by a linear equation system (see, e.g., [13]). Using this linear equation system for the exponentially large MDP constructed in Section 4.4 and equations (†) and (‡) from that section, the threshold problem for the maximal VPE can be reformulated as the satisfiability problem of an exponentially sized system of quadratic inequalities. The optimization problem can likewise be formulated as an exponentially large *quadratically constrained quadratic program* (QCQP). This satisfiability problem and QCQPs are NP-hard in general. The question whether the inequality system of exponential size we obtain here has a special structure which allows it to be solved in exponential time remains open here.

This observation stands in contrast to conceptually similar saturation point results straight-forwardly leading to exponential time algorithms (see, e.g., [21]). For example, the threshold problem for conditional expectations: "Given a set $T \subseteq S$, is there a scheduler $\mathfrak{S}$ with $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond goal \mid \Diamond T) \geq \vartheta$?" admits a saturation point result in MDPs with non-negative weights as well [4]. Deriving a system of inequalities as above, however, leads to a system of linear inequalities after straight-forward transformations. Hence, this approach directly leads to an exponential time algorithm for the threshold problem for conditional expectations. For the VPE, the system of inequalities seems to be inherently of a polynomial nature which can be seen as an indication that the situation here is fundamentally more difficult.

Further, we restricted our attention to MDPs with non-negative weights. When allowing positive and negative weights, the key result, the existence of a saturation point, does not hold anymore. For conditional expectations and other problems relying on the existence of a saturation point, the switch to integer weights makes the problems even at least as hard as the Positivity problem for linear recurrence sequences, a number theoretic problem whose decidability has been open for many decades (see [19, 21]). The question whether such a hardness result for the threshold problem of the VPE, rendering decidability impossible without a breakthrough in number theory, can be established remains as future work.

Further possible directions of research include the investigation of the following multi-objective threshold problem: Given η and ν, is there a scheduler with expectation at least η and variance at most ν? As the variance treats good and bad outcomes symmetrically, replacing the variance in the VPE by a one-sided deviation measure, such as the lower semi-variance that only takes the outcomes worse than the expected value into account, constitutes another natural extension of this work.

References

1 Mohamadreza Ahmadi, Anushri Dixit, Joel W Burdick, and Aaron D Ames. Risk-averse stochastic shortest path planning. *arXiv*, 2021. `arXiv:2103.14727`.

2 Kenneth J. Arrow. *Essays in the Theory of Risk-Bearing*. Amsterdam, North-Holland Pub. Co., 1970.

3 Christel Baier, Nathalie Bertrand, Clemens Dubslaff, Daniel Gburek, and Ocan Sankur. Stochastic shortest paths and weight-bounded properties in Markov decision processes. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 86–94. ACM, 2018.

4 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Maximizing the conditional expected reward for reaching the goal. In Axel Legay and Tiziana Margaria, editors, *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10206 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2017.

5 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

6 Tomáš Brázdil, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Trading performance for stability in Markov decision processes. *Journal of Computer and System Sciences*, 84:144–170, 2017. doi:10.1016/j.jcss.2016.09.009.

7 EJ Collins. Finite-horizon variance penalised Markov decision processes. *Operations-Research-Spektrum*, 19(1):35–39, 1997.

8 Luca de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *10th International Conference on Concurrency Theory (CONCUR)*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81, 1999.

9 Jerzy A Filar, Lodewijk CM Kallenberg, and Huey-Miin Lee. Variance-penalized Markov decision processes. *Mathematics of Operations Research*, 14(1):147–161, 1989.

10 William N Goetzmann, Stephen J Brown, Martin J Gruber, and Edwin J Elton. *Modern portfolio theory and investment analysis*. John Wiley & Sons, 2014.

11 Christoph Haase and Stefan Kiefer. The odds of staying on budget. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9135 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2015.

12 Stratton C. Jaquette. Markov Decision Processes with a New Optimality Criterion: Discrete Time. *The Annals of Statistics*, 1(3):496–505, 1973. doi:10.1214/aos/1176342415.

13 Lodewijk Kallenberg. *Markov Decision Processes*. Lecture Notes. University of Leiden, 2011.

14 Jan Kretínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 609–618. ACM, 2018. doi:10.1145/3209108.3209176.

15 Masami Kurano. Markov decision processes with a minimum-variance criterion. *Journal of mathematical analysis and applications*, 123(2):572–583, 1987.

16 Petr Mandl. On the variance in controlled Markov chains. *Kybernetika*, 7(1):1–12, 1971.

17 Shie Mannor and John N. Tsitsiklis. Mean-variance optimization in Markov decision processes. In *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, pages 177–184, Madison, WI, USA, 2011. Omnipress.

18 Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. URL: http://www.jstor.org/stable/2975974.

19 Jakob Piribauer. *On Non-Classical Stochastic Shortest Path Problems*. PhD thesis, Technische Universität Dresden, Germany, 2021.

20 Jakob Piribauer and Christel Baier. Partial and conditional expectations in Markov decision processes with integer weights. In Mikolaj Bojanczyk and Alex Simpson, editors, *22nd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 11425 of *Lecture Notes in Computer Science*, pages 436–452. Springer, 2019.

21 Jakob Piribauer and Christel Baier. On Skolem-hardness and saturation points in Markov decision processes. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *LIPIcs*, pages 138:1–138:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.138.

22 Jakob Piribauer, Ocan Sankur, and Christel Baier. The variance-penalized stochastic shortest path problem. *arXiv:2204.12280*, 2022. doi:10.48550/ARXIV.2204.12280.

23 John W. Pratt. Risk aversion in the small and in the large. *Econometrica*, 32(1/2):122–136, 1964. URL: http://www.jstor.org/stable/1913738.

24 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

25 Matthew J. Sobel. The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982. doi:10.2307/3213832.

26 Matthew J. Sobel. Mean-variance tradeoffs in an undiscounted mdp. *Operations Research*, 42(1):175–183, 1994. doi:10.1287/opre.42.1.175.

27 Michael Ummels and Christel Baier. Computing quantiles in Markov reward models. In Frank Pfenning, editor, *16th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 7794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2013. doi:10.1007/978-3-642-37075-5_23.

28 Tom Verhoeff. Reward variance in Markov chains: A calculational approach. In *Proceedings of Eindhoven FASTAR Days*. Citeseer, 2004.

29 Xiao Wu and Xianping Guo. First passage optimality and variance minimisation of Markov decision processes with varying discount factors. *Journal of Applied Probability*, 52(2):441–456, 2015. doi:10.1239/jap/1437658608.

30 Li Xia. Optimization of Markov decision processes under the variance criterion. *Automatica*, 73:269–278, 2016. doi:10.1016/j.automatica.2016.06.018.

31 Li Xia. Mean–variance optimization of discrete time discounted Markov decision processes. *Automatica*, 88:76–82, 2018.

32 Li Xia. Variance minimization of parameterized Markov decision processes. *Discrete Event Dynamic Systems*, 28(1):63–81, 2018. doi:10.1007/s10626-017-0258-5.

33 Li Xia. Risk-sensitive Markov decision processes with combined metrics of mean and variance. *Production and Operations Management*, 29(12):2808–2827, 2020.

Functions and References in the Pi-Calculus: Full Abstraction and Proof Techniques

Enguerrand Prebet
Université de Lyon, ENS de Lyon, UCB Lyon 1, CNRS, INRIA, LIP

Abstract

We present a fully abstract encoding of $\lambda^{\mathtt{ref}}$, the call-by-value λ-calculus with references, in the π-calculus. By contrast with previous full abstraction results for sequential languages in the π-calculus, the characterisation of contextual equivalence in the source language uses a labelled bisimilarity. To define the latter equivalence, we refine existing notions of typed bisimulation in the π-calculus, and introduce in particular a specific component to handle divergences.

We obtain a proof technique that allows us to prove equivalences between $\lambda^{\mathtt{ref}}$ programs via the encoding. The resulting proofs correspond closely to normal form bisimulations in the λ-calculus, making proofs in the π-calculus expressible as if reasoning in $\lambda^{\mathtt{ref}}$.

We study how standard and new up-to techniques can be used to reason about diverging terms and simplify proofs of equivalence using the bisimulation we introduce. This shows how the π-calculus theory can be used to prove interesting equivalences between $\lambda^{\mathtt{ref}}$ programs, using algebraic reasoning and up-to techniques.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning

Keywords and phrases Call-by-value λ-calculus, imperative Programming, π-calculus, Bisimulation, Type System

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.130

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding This work has been supported by the Université Franco-Italienne under the programme Vinci 2020.

Acknowledgements Many thanks to Daniel Hirschkoff and Davide Sangiorgi for fruitful discussions and helpful comments on earlier versions of this article.

1 Introduction

Milner [11] described the first encodings from the λ-calculus to the π-calculus for both call-by-name and call-by-value. These encodings are shown to be sound with respect to contextual equivalence, enabling the use of the π-calculus as a model to prove equivalence between λ-terms. In sequential languages like the λ-calculus or extensions thereof, contextual equivalence is often considered as the canonical equivalence. The π-calculus offers a rich theory of behavioural equivalences and preorders to reason coinductively and algebraically about processes. In the π-calculus, several powerful up-to techniques can be used and combined in a modular way to make bisimulation proofs shorter and more readable [10, 16]. Labelled bisimulations can also be refined by means of type systems.

Nevertheless, for sequential languages, while such equivalences in the π-calculus lead to sound encodings with respect to contextual equivalence, completeness does not hold, due to the parallel nature of the π-calculus. Full abstraction is obtained for stronger equivalences in the source language, generally based on trees: Lévy-Longo Trees for the call-by-name λ-calculus [18], and η-eager normal form bisimilarity for call-by-value [3]. More generally, we are not aware of full abstraction results relating a contextual equivalence in a sequential

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 130; pp. 130:1–130:19


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

language and a labelled bisimilarity between process terms in the encoding. Existing results either use a finer relation than contextual equivalence in the source language, or obtain full abstraction for a contextually-defined equivalence in the π-calculus [1, 21].

In this work, we study $\lambda^{\mathtt{ref}}$, the call-by-value λ-calculus with references. Our first main contribution is to characterise contextual equivalence in $\lambda^{\mathtt{ref}}$ using a labelled bisimilarity in the π-calculus. The encoding of functions is extended with references as described in [4]. To define our labelled bisimulation, we rely on *well-bracketed bisimulation* [5], wb-bisimulation for short. Wb-bisimulations enforce a well-bracketing discipline between function calls and returns, and is formalised by a type system. Using a type system limits the interactions a process can have with its environment, and thus makes the corresponding bisimulation coarser than an untyped one.

This study allows us to understand the expressiveness of wb-bisimulation and to improve it. As mentioned in [2], to capture contextual equivalence in $\lambda^{\mathtt{ref}}$, *deferred divergent terms* need to be taken into account. Intuitively such terms hide a divergence behind stuck terms, like e.g. in $(\lambda y.\,\Omega)(xV)$, where Ω is an always diverging term and V is a value: the stuck term xV prevents the β-reduction yielding Ω to be applied. Still, this term is contextually equivalent to Ω. We define an equivalent notion for typeable π-terms, called π-divergence, which is used to refine wb-bisimulation yielding a notion of *bisimulation with divergence* in the π-calculus.

Another main contribution of this work is to define this new bisimulation with divergence and to study how up-to techniques, existing and new, can be used to simplify bisimulation proofs for this new equivalence. As π-divergence is defined coinductively, up-to techniques can also be defined to enhance the proofs of divergence. We also introduce a new up-to technique, to which we return below; this technique is *compatible*, meaning that it fits in the existing theory of up-to techniques for the π-calculus.

We do not prove full abstraction with respect to contextual equivalence in $\lambda^{\mathtt{ref}}$ directly, but rather exploit its characterisation using *normal form bisimilarity* [2]. Normal form bisimulations in [2], which we call nfb in the sequel, and bisimulations with divergence in the π-calculus are strongly connected, and we can establish the following result:

> If $\mathcal{R}$ is a nfb, then the encoding of $\mathcal{R}$ is a bisimulation with divergence up to expansion in the π-calculus. (1)

This property allows us to prove that bisimilarity with divergence is complete w.r.t. nfb. Thanks to (1), when proving equivalences of $\lambda^{\mathtt{ref}}$-terms, the π-calculus machinery running under the hood is almost invisible: the bisimulations we manipulate are mainly built using (the encoding of) $\lambda^{\mathtt{ref}}$-terms. This makes π-calculus a powerful environment to prove equivalences between $\lambda^{\mathtt{ref}}$ programs. We illustrate this power via several examples throughout the paper. In some cases, we are able to compare $\lambda^{\mathtt{ref}}$ programs using equivalences which are finer, and simpler to use, than bisimulation with divergence.

We now give more details about the encoding from $\lambda^{\mathtt{ref}}$ to the π-calculus. Milner's original encoding of the untyped call-by-value λ-calculus does not handle equivalences between terms with free variables well. This issue is amended in [3] by moving to the internal π-calculus [19], where only fresh names can be transmitted. For instance, the resulting encodings of $(\lambda z.\,z)(xV)$ and xV are equivalent, which is not the case for Milner's original encoding.

Our results are based on the optimised version of the encoding of [3], where administrative reductions introduced in the encoding of the application are removed. In the optimised encoding, a stuck term like xV is translated into a process that cannot reduce. Additionally,

the translation of an evaluation context always yields an evaluation context, which is not the case in the encoding of [3]. These two properties lead to a tight statement of operational correspondence: the encoding of a $\lambda^{\mathtt{ref}}$-term can perform an internal communication only if the source term has a reduction.

When considering non-internal transitions, the picture becomes more complicated because residual processes appear. We write $[\![M]\!]_p$ for the encoding of M at p – as usual, the π-calculus encoding of a term is parametric over some name p. To illustrate operational correspondence for non-internal transitions, we have for instance $[\![E[xV]]\!]_p \xrightarrow{\overline{x}(y,q)} \gtrsim [\![V]\!]_y \mid q(z).\,[\![E[z]]\!]_p$, where E is an evaluation context and $\gtrsim$ stands for the expansion preorder. The encoding of the stuck term $E[xV]$ sends two links at x, one to its argument V (via y) and one to its continuation E (via q). Both processes are then accessible by the context, and can be executed (possibly in parallel). Intuitively, the value and continuation belong to some environment. This is close to what happens in a nfb, whereby terms are related in the presence of an environment, containing both values and contexts. Thus, it is natural to encode a nfb as a binary relation on π-calculus processes.

To prevent the context from performing unwanted transitions (like parallel calls), we use the type system for well-bracketing from [5]. Typing constraints give rise to *type-allowed* transitions. With this restriction, we obtain a typed version of operational correspondence which is in one-to-one correspondence with the clauses defining nfb. In particular, in the transition of $[\![E[xV]]\!]_p$ above, the resulting process is the encoding of the corresponding state in the nfb. Soundness and completeness of the encoding, as well as property (1), are obtained as consequences of such operational correspondence.

We explain briefly the new up-to technique we introduce. Intuitively, transitions in $\lambda^{\mathtt{ref}}$ may generate several copies of the same value, which are stored in the environment. In [2], the environment is a set, and thus duplicates are removed for free. In the standard π-calculus, these copies would be the same replicated process, and duplicates could be removed using the standard law $!P \mid !P \sim !P$. In the internal π-calculus, each copy is accessible via a distinct name. However, when these copies are only accessible by the context, it is sound to remove duplicates. The new up-to technique we introduce, called *up-to body*, formalises this idea. Together with standard up-to techniques for the pi-calculus, we use the up-to body technique to prove several examples of equivalences between $\lambda^{\mathtt{ref}}$ programs.

Paper outline

We start by presenting the necessary background: we describe $\lambda^{\mathtt{ref}}$ and normal form bisimulations from [2] in Section 2; in Section 3, we introduce the Asynchronous Internal π-calculus, AIπ, and the type system for well-bracketing. We continue with the contributions of this work, describing the encoding for $\lambda^{\mathtt{ref}}$ terms in Section 4. We also show an operational correspondence and examples that can be proved with untyped equivalences, using algebraic reasoning and standard algebraic laws. The extension of the encoding to bisimulation states, together with the definition of bisimulation with divergence, are given in Section 5, where we establish full abstraction. We present up-to techniques for wb-bisimulation in Section 6, and show how they can be used on some examples.

2 Normal Form Bisimulation for a λ-Calculus with References

We define the syntax of the λ-calculus with references, noted $\lambda^{\mathtt{ref}}$:

Terms:	$M, N ::= V \mid M\ N \mid \ell := M; N \mid\ !\ell \mid \mathtt{new}\ \ell := V\ \mathtt{in}\ M$
Values:	$V, W ::= x \mid \lambda x.\, M$
Evaluation contexts:	$E, F ::= [\cdot] \mid E\ M \mid V\ E \mid \ell := E; M$

Free variables for terms and contexts are defined as usual with $\lambda x.\, M$ as binder. We write $M\{^V/_x\}$ for the usual capture-avoiding substitution of x by V in M. And we let $\int$ range over simultaneous substitutions $\{^{V_1}/_{x_1}\} \dots \{^{V_n}/_{x_n}\}$ where $x_1, \dots, x_n$ are pairwise different variables.

Free references for terms and contexts, noted $\mathrm{fr}(M)$ and $\mathrm{fr}(E)$, are defined similarly, with $\texttt{new}\ \ell \coloneqq V\ \texttt{in}\ M$ binding ℓ in M. Notice that ℓ is not a value, meaning that in $\texttt{new}\ \ell \coloneqq V\ \texttt{in}\ M$, ℓ is local to the term M. To give access to a local reference, M may pass functions $\lambda x.\, !\ell$ and $\lambda x.\, \ell \coloneqq x$ instead of ℓ.

A store, noted $h, g, \dots$, is a partial fonction with finite domain from references to values. We write $\emptyset$ for the empty store and $\mathrm{dom}(h)$ for the set of references on which h is defined. For any ℓ, V, we write $h \uplus \ell = V$ for the store h extended with a reference ℓ containing V and $h[\ell \coloneqq V]$ for the store h with the content of ℓ updated to the value V.

The semantics are defined on *configurations* $\langle h \mid M \rangle$ where h is a store.

We assume that for all configurations $\langle h \mid M \rangle$, we have $\mathrm{fr}(M) \subseteq \mathrm{dom}(h)$ and for all $\ell \in \mathrm{dom}(h)$, $\mathrm{fr}(h(\ell)) \subseteq \mathrm{dom}(h)$. This assumption ensures that any free reference used in terms is defined in h.

Reductions between configurations are defined as follows:

$$\langle h \mid (\lambda x.\, M)V \rangle \to \langle h \mid M\{^V/_x\} \rangle \qquad (\beta)$$

$$\langle h \mid !\ell \rangle \to \langle h \mid h(\ell) \rangle \qquad \text{(Read)}$$

$$\langle h \mid \ell \coloneqq v; M \rangle \to \langle h[\ell \coloneqq v] \mid M \rangle \qquad \text{(Write)}$$

$$\langle h \mid \texttt{new}\ \ell \coloneqq v\ \texttt{in}\ M \rangle \to \langle h \uplus \ell = v \mid M \rangle \qquad \text{(Alloc)}$$

$$\langle h \mid E[M] \rangle \to \langle g \mid E[N] \rangle \qquad \text{if } \langle h \mid M \rangle \to \langle g \mid N \rangle \qquad \text{(Eval)}$$

A term M in a configuration $\langle h \mid M \rangle$ which cannot reduce is called a normal form. It can either be a value, or a stuck term of the form $E[yV]$.

▶ **Lemma 1.** *For any $\langle h \mid M \rangle$ we have:*
1. *either $\langle h \mid M \rangle \to \langle h' \mid M' \rangle$ for some $\langle h' \mid M' \rangle$*
2. *or M is a value*
3. *or M is of the form $E[yV]$.*

We write $\Rightarrow$ for $\to^*$ and we say that $\langle h \mid M \rangle$ and $\langle g \mid N \rangle$ *co-terminate* when $\langle h \mid M \rangle \Rightarrow \langle h' \mid M' \rangle$ with M' being a normal form iff $\langle g \mid N \rangle \Rightarrow \langle g' \mid N' \rangle$ with N' being a normal form. A term (resp. context) is *closed* (resp. *reference-closed*) if its set of free variables (resp. free references) is empty. A substitution $\int$ is *closing terms* $M, N, \dots$ if $M\int, N\int, \dots$ are closed.

▶ **Definition 2.** *Two reference-closed terms are contextually equivalent, written $M \asymp N$, if for all closing substitutions $\int$ and reference-closed contexts E, $\langle \emptyset \mid E[M\int] \rangle$ and $\langle \emptyset \mid E[N\int] \rangle$ co-terminate.*

To define normal form bisimulations, we need to introduce the notion of *triples*, used in [2].

We use a tilde, like in $\widetilde{V_i}$, to denote a (possibly empty) tuple. Triples are of the form $(\widetilde{V_i}, \sigma, c)$ and $(\widetilde{V_i}, \sigma, h)$ where: $\widetilde{V_i}$ is a tuple of values, σ is a stack of evaluation contexts, c is a configuration and h is a store. These are the elements being compared in a normal form bisimulation. We provide some comments about the role of triples.

The tuple $\widetilde{V_i}$ stores the values accumulated by the environment, and we write $(\widetilde{V_i}, V)$ for the tuple $\widetilde{V_i}$ extended with the additional value V. A stack of evaluation contexts σ corresponds to interrupted contexts that must be executed to complete the computation. ⊙

stands for the empty stack and $E :: \sigma$ for the stack obtained by adding E on top of σ. The store is accessible by all the other objects of the triple, values or contexts – in $(\widetilde{V_i}, \sigma, c)$, the store is part of c.

Intuitively, a triple of the form $(\widetilde{V_i}, \sigma, c)$ is active, meaning it is computing up until a normal form term is obtained, at which point the computation proceeds to triples of the form $(\widetilde{V_i}, \sigma, h)$ where the environment is carrying on the computation, deciding to call one of the accumulated functions or to resume the computation by evaluating the context at the top of the stack.

For instance, in the triple $(\widetilde{V_i}, \sigma, \langle h \mid E[yV]\rangle)$, the environment is about to get access to V and the function corresponding to y will run before eventually (in absence of divergence) returning the result that will be used by E. Thus, the "next" triple is $((\widetilde{V_i}, V), E :: \sigma, h)$: value V and context E are added to the corresponding tuple and stack, and h is kept identical while the environment is deciding for the next move. This evolution can be seen in Definitions 4 and 5.

Following [8], we first need to introduce an auxiliary relation which performs an immediate beta reduction in a term of the form VW whenever possible.

▶ **Definition 3** (Relation $\succ$). *We write $VW \succ N$ when $V = \lambda y.\, N'$ and $N = N'\{^W/_y\}$ or when $V = z$ and $N = VW$.*

We say that a term is deferred diverging if it hides a diverging behaviour behind a stuck term, e.g $(\lambda x.\, \Omega)(yV)$. This notion can be extended to triples to capture all diverging behaviours, including deferred ones, as defined below. Intuitively, a triple is diverging if it contains a diverging, possibly deferred, context or configuration.

Formally, the set of diverging triples is defined coinductively using a diverging set.

▶ **Definition 4** (Diverging set). *A set S of triples is* diverging *if the two following conditions hold:*

1. $(\widetilde{V_i}, \sigma, c) \in S$ *implies*
   - **a.** *if $c \to c'$, then $(\widetilde{V_i}, \sigma, c') \in S$*
   - **b.** *if $c = \langle h \mid V\rangle$, then $\sigma \neq \emptyset$ and $((\widetilde{V_i}, V), \sigma, h) \in S$*
   - **c.** *if $c = \langle h \mid E[yV]\rangle$, then $((\widetilde{V_i}, V), E :: \sigma, h) \in S$*
2. $(\widetilde{V_i}, \sigma, h) \in S$ *implies*
   - **a.** *for every j and fresh x, $(\widetilde{V_i}, \sigma, \langle h \mid M\rangle) \in S$ with $V_j x \succ M$*
   - **b.** *if $\sigma = E :: \sigma'$, then $(\widetilde{V_i}, \sigma', \langle h \mid E[x]\rangle) \in S$ for x fresh*

We note λ_{div} the largest diverging set.

We say that a relation $\mathcal{R}$ is a *triples relation* if the two elements of any pair of triples in $\mathcal{R}$ both contain a tuple of values and stack of the same size, and either both contain a configuration or both contain a store.

We can now define the normal form bisimulation:

▶ **Definition 5** (nfb). *A* symmetric *triples relation $\mathcal{R}$ is a normal form bisimulation when there exists a diverging set S such that:*

1. $(\widetilde{V_i}, \sigma_1, c)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, d)$ *implies*
   - **a.** *if $c \to c'$, then $d \Rightarrow d'$ and $(\widetilde{V_i}, \sigma_1, c')\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, d')$*
   - **b.** *if $c = \langle h \mid V\rangle$, then either*
     - **i.** *$d \Rightarrow \langle g \mid W\rangle$ and $((\widetilde{V_i}, V), \sigma_1, h)\ \mathcal{R}\ ((\widetilde{W_i}, W), \sigma_2, g)$, or*
     - **ii.** *$\sigma_1 \neq \emptyset$ and $((\widetilde{V_i}, V), \sigma_1, h) \in S$*
   - **c.** *if $c = \langle h \mid E[yV]\rangle$, then either*
     - **i.** *$d \Rightarrow \langle g \mid F[yW]\rangle$ and $((\widetilde{V_i}, V), E :: \sigma_1, h)\ \mathcal{R}\ ((\widetilde{W_i}, W), F :: \sigma_2, g)$, or*
     - **ii.** *$((\widetilde{V_i}, V), E :: \sigma_1, h) \in S$*

2. $(\widetilde{V_i}, \sigma_1, h)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, g)$ *implies*
   a. *for every* j *and fresh* x, $(\widetilde{V_i}, \sigma_1, \langle h \mid M\rangle)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, \langle g \mid N\rangle)$ *with* $V_j x \succ M$ *and* $W_j x \succ N$
   b. *if* $\sigma_1{=}E :: \sigma_1'$ *and* $\sigma_2{=}F :: \sigma_2'$, *then* $(\widetilde{V_i}, \sigma_1', \langle h \mid E[x]\rangle)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2', \langle g \mid F[x]\rangle)$ *for* x *fresh*

Normal form bisimilarity, $\approx_\lambda$, *is the largest normal form bisimulation.*

Definition 5 reformulates the bisimulation from [2]. This gives the same equivalence while being more suitable to establish the operational correspondence of the encoding (Theorem 17). We can thus rely on the following result.

▶ **Theorem 6** (Full abstraction [2, Theorems 3,4]). *For all* $\lambda^{\mathtt{ref}}$*-terms* M, N, *we have* $(\emptyset, \odot, \langle \emptyset \mid M\rangle) \approx_\lambda (\emptyset, \odot, \langle \emptyset \mid N\rangle)$ *iff* $M \asymp N$.

This means that to prove that two reference-closed terms are contextually equivalent, we can provide a normal form bisimulation $\mathcal{R}$ relating the two terms and its corresponding diverging set S.

3 AIπ, the Asynchronous Internal π-Calculus

We present the Asynchronous Internal π-calculus, AIπ, and the type system for well-bracketing from [5]. The syntax is given by the following grammar:

Processes	$P, Q ::= \mathbf{0} \mid \overline{a}(\widetilde{b})\ P \mid a(\widetilde{b}).\, P \mid !a(\widetilde{b}).\, P \mid P \mid Q \mid (\boldsymbol{\nu} a)P \mid K\lfloor\widetilde{a}\rfloor$
Recursive definitions	$K \triangleq (\widetilde{a})\ P$

As for $\lambda^{\mathtt{ref}}$ values, $\widetilde{b}$ denotes a tuple of names. We also write $a(\_).\, P$ for $a(z).\, P$ when $z \notin \mathrm{fn}(P)$. Here, $\overline{a}(\widetilde{b})\ P$ is a bound asynchronous output, introduced in [1]. This corresponds to $(\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{b}\rangle.\, \mathbf{0} \mid P)$ in the standard π-calculus, that is, a non-blocking output which carries bound names used in P. Constants are defined as an abstraction $(K \triangleq (\widetilde{a})\ P)$, where $\widetilde{a}$ are distinct names, bound in P. Given an abstraction $(\widetilde{a})\ P$, we note $((\widetilde{a})\ P)\lfloor\widetilde{b}\rfloor$ the process $P\{\widetilde{b}/\widetilde{a}\}$. Thus $K\lfloor\widetilde{a}\rfloor$ represents the process in the definition of K substituted with the names $\widetilde{a}$ (as expressed by the rule Cst in the operational semantics).

The full set of rules defining the Labelled Transition System (LTS) is given in Figure 1. Symmetric rules for Par and Comm have been omitted. It is similar to that of Iπ with the additional rules:

$$\textsc{Async}\ \frac{P \xrightarrow{\mu} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\mu} \overline{a}(\widetilde{b})\ P}\quad \widetilde{b} \cap (\mathrm{fn}(\mu) \cup \mathrm{bn}(\mu)) = \emptyset,\ a \notin \mathrm{bn}(\mu) \qquad \textsc{AComm}\ \frac{P \xrightarrow{a(\widetilde{b})} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{b})P'}$$

AComm allows P to communicate with the bound output, and Async allows P to perform any action that does not involve the names bound by the output.

Our type system is based on Milner's *sorting*. Names are split into *sorts*. In our case, we will use 3 sorts: F,R,C. The sorting function Σ is defined by $\Sigma(\mathtt{F}) = (\mathtt{F}, \mathtt{C})$ and $\Sigma(\mathtt{R}) = \Sigma(\mathtt{C}) = \mathtt{F}$. We call *function names*, ranged over with $w, x, y, z, \ldots$, names in F, *reference names*, ranged over with $\ell, \ell', \ldots$, names in R, and *continuation names*, ranged over with $p, q, r, \ldots$, names in C.

On top of the sorting, we adapt the type system for well-bracketing from [5] to AIπ. The type system imposes that continuation names are *receptive linear*, meaning that they can be used only once in output and once in input (hence linear) and the input is immediately

$$\textsc{Inp}\ \frac{}{a(\widetilde{b}).\,P \xrightarrow{a(\widetilde{b})} P} \qquad \textsc{Out}\ \frac{}{\overline{a}(\widetilde{b})\ P \xrightarrow{\overline{a}(\widetilde{b})} P} \qquad \textsc{Async}\ \frac{P \xrightarrow{\mu} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\mu} \overline{a}(\widetilde{b})\ P}\ \begin{array}{l}\widetilde{b} \cap (\mathrm{fn}(\mu) \cup \mathrm{bn}(\mu)) = \emptyset \\ a \notin \mathrm{bn}(\mu)\end{array}$$

$$\textsc{AComm}\ \frac{P \xrightarrow{a(\widetilde{b})} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{b})P'} \qquad \textsc{Rep}\ \frac{a(\widetilde{b}).\,P \xrightarrow{\mu} P'}{!a(\widetilde{b}).\,P \xrightarrow{\mu} P' \mid\, !a(\widetilde{b}).\,P} \qquad \textsc{Res}\ \frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu}a)P \xrightarrow{\mu} (\boldsymbol{\nu}a)P'}\ a \notin \mathrm{fn}(\mu) \cup \mathrm{bn}(\mu)$$

$$\textsc{Par}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset \qquad \textsc{Comm}\ \frac{P \xrightarrow{a(\widetilde{b})} P' \qquad Q \xrightarrow{\overline{a}(\widetilde{b})} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{b})(P' \mid Q')} \qquad \textsc{Cst}\ \frac{P\{\widetilde{b}/\widetilde{a}\} \xrightarrow{\mu} P'}{K\lfloor \widetilde{b} \rfloor \xrightarrow{\mu} P'} K \triangleq (\widetilde{a})\ P$$

Figure 1 Labelled Transition Semantics for AIπ.

available as soon as it is created (hence receptive). Additionally, creations and communications at continuation names behave in a well-bracketed manner, meaning that the communication will first occur at the name created last.

We use two constants, noted $\triangleright_{\mathtt{F}}$ and $\triangleright_{\mathtt{C}}$, defined as follows:

$$\begin{array}{ll} \triangleright_{\mathtt{C}} \triangleq (p,q)\ p(x).\overline{q}(y)\ y \triangleright_{\mathtt{F}} x & \text{with } p,q \text{ being continuation names} \\ \triangleright_{\mathtt{F}} \triangleq (x,y)\ !x(z,p).\overline{y}(w,q)\ (q \triangleright_{\mathtt{C}} p \mid w \triangleright_{\mathtt{F}} z) & \text{with } x,y \text{ being function names} \end{array}$$

These constants represent a link, also called forwarder or dynamic wire [17], which transforms outputs at the first name into outputs at the second. As their usage only differs in the linearity of continuation names and the arity, we often use the same symbol $\triangleright$ to denote both $\triangleright_{\mathtt{F}}$ and $\triangleright_{\mathtt{C}}$.

$$\textsc{wb-Nil}\ \frac{}{\emptyset \vdash_{\mathrm{wb}} \mathbf{0}} \qquad \textsc{wb-AOutC}\ \frac{\rho \vdash_{\mathrm{wb}} P}{p : \mathtt{o}, \rho \vdash_{\mathrm{wb}} \overline{p}(y)\ P} \qquad \textsc{wb-AOutF}\ \frac{p : \mathtt{i}, \rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} \overline{x}(y,p)\ P} \qquad \textsc{wb-AOutR}\ \frac{\rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} \overline{\ell}(y)\ P}$$

$$\textsc{wb-InpC}\ \frac{p : \mathtt{o} \vdash_{\mathrm{wb}} P \qquad p \neq q}{q : \mathtt{i}, p : \mathtt{o} \vdash_{\mathrm{wb}} q(y).\,P} \qquad \textsc{wb-InpF}\ \frac{p : \mathtt{o} \vdash_{\mathrm{wb}} P}{\emptyset \vdash_{\mathrm{wb}} x(y,p).\,P,\ !x(y,p).\,P} \qquad \textsc{wb-InpR}\ \frac{p : \mathtt{o} \vdash_{\mathrm{wb}} P}{p : \mathtt{o} \vdash_{\mathrm{wb}} \ell(y).\,P}$$

$$\textsc{wb-ResC1}\ \frac{\rho, p : \star, \rho' \vdash_{\mathrm{wb}} P}{\rho, \rho' \vdash_{\mathrm{wb}} (\boldsymbol{\nu}p)P} \qquad \textsc{wb-ResC2}\ \frac{\rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} (\boldsymbol{\nu}p)P}\ p \notin \rho \qquad \textsc{wb-ResFR}\ \frac{\rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} (\boldsymbol{\nu}y)P,\ (\boldsymbol{\nu}\ell)P}$$

$$\textsc{wb-Par}\ \frac{\rho \vdash_{\mathrm{wb}} P \qquad \rho' \vdash_{\mathrm{wb}} Q}{\rho'' \vdash_{\mathrm{wb}} P \mid Q}\ \rho'' \in \mathtt{inter}(\rho;\rho') \qquad \textsc{wb-FwC}\ \frac{}{p : \mathtt{i}, q : \mathtt{o} \vdash_{\mathrm{wb}} p \triangleright_{\mathtt{C}} q} \qquad \textsc{wb-FwF}\ \frac{}{\emptyset \vdash_{\mathrm{wb}} x \triangleright_{\mathtt{F}} y}$$

Figure 2 Type system for well-bracketing.

We present the typing rules for well-bracketing in Figure 2. Judgements are of the form $\rho \vdash_{\text{wb}} P$ with ρ being a stack. In rules WB-INPF and WB-RESFR, the conclusion is of the form $\rho \vdash_{\text{wb}} P, Q$ to indicate that both $\rho \vdash_{\text{wb}} P$ and $\rho \vdash_{\text{wb}} Q$ can be inferred from the premise.

A stack is a sequence of *tagged* names, the tag being either $\mathtt{i}, \mathtt{o}$ or $\star$ denoting respectively input, output and both capabilities. Upon creation, a continuation comes with both capabilities (WB-RESC1), that are used once as input and output (WB-INPC and WB-AOUTC). In WB-AOUTF, a continuation is created and its input capability is passed to P, while its output capability is sent by the communication. This capability can then be used after the input in WB-INPF. As continuation names are receptive, inputs cannot appear after another input (WB-INPF and WB-INPC).

Intuitively, a stack expresses the expected usage of the free continuation names in a process. Stacks are given by the following grammars:

$$\rho \ ::= \ \rho^{\mathtt{o}} \mid \rho^{\mathtt{i}} \qquad \rho^{\mathtt{o}} \ ::= \ p : \mathtt{o}, \rho^{\mathtt{i}} \mid p : \star, \rho^{\mathtt{o}} \mid \emptyset \qquad \rho^{\mathtt{i}} \ ::= \ p : \mathtt{i}, \rho^{\mathtt{o}} \mid \emptyset$$

Moreover, a name may appear at most once in a stack, so we will say that a name is $\mathtt{o}$-*tagged* in ρ when the name appears with tag $\mathtt{o}$ in ρ and similarly for $\mathtt{i}$ and $\star$.

As a $\star$-tagged name represents both output and input capabilities, a stack can be seen as an alternation of input- and output-tagged names. For instance, if we have

$$p_1 : \mathtt{o}, p_2 : \mathtt{i}, p_3 : \star, p_4 : \mathtt{o} \vdash_{\text{wb}} P$$

then $p_1, \ldots, p_4$ are the free continuation names in P; among these, p_1 will be used first, as an output at p_1; then p_2 will be used, in an input interaction with the environment. P possesses both the output and the input capability on p_3, and will use both capabilities by performing a reduction at p_3; the computation for P terminates with an output at p_4.

Thus, to compose two processes P and Q in parallel, the usage of continuations of $P \mid Q$ is an interleaving of the ones of P and Q (WB-PAR). Names with capabilities shared between P and Q can be used as synchronisation point. Formally, the interleaving of two stacks is described by the following definition:

▶ **Definition 7.** *The interleaving relation is defined as a ternary relation between stacks, written* $\rho \in \mathtt{inter}(\rho_1 \ ; \ \rho_2)$*, and defined as follows:*

- $\emptyset \in \mathtt{inter}(\emptyset; \emptyset)$
- $\rho \in \mathtt{inter}(\rho_1 \ ; \ \rho_2)$ *implies* $\rho \in \mathtt{inter}(\rho_2 \ ; \ \rho_1)$*.*
- $\rho \in \mathtt{inter}(\rho_1 \ ; \ \rho_2)$ *implies* $p : \eta, \rho \in \mathtt{inter}(p : \eta, \rho_1 \ ; \ \rho_2)$*, where* $\eta \in \{\mathtt{o}, \mathtt{i}, \star\}$ *and* $p : \eta, \rho_1$ *is a stack.*
- *Whenever* $\rho \in \mathtt{inter}(\rho^{\mathtt{i}}; \rho^{\mathtt{o}})$*, we have* $p : \star, \rho \in \mathtt{inter}(p : \mathtt{o}, \rho^{\mathtt{i}} \ ; \ p : \mathtt{i}, \rho^{\mathtt{o}})$*.*

Even though the grammar allows it, it is not possible to type a process $\rho \vdash P$ with ρ ending with $p : \mathtt{i}$ or $p : \star$.

As continuation names are used linearly, when both input and output are present in the process, i.e., when the name is $\star$-tagged in the stack, this name should not be observable. We call *clean* a stack where no name is $\star$-tagged. We note $\mathtt{c}(\rho)$ the clean stack obtained by removing all $\star$-tagged names from ρ, and $\rho \vDash_{\text{wb}} P$ when there exists ρ' with $\rho' \vdash_{\text{wb}} P$ and $\mathtt{c}(\rho') = \rho$. Using clean stacks, we can define *typed transitions*.

▶ **Definition 8.** *When $\rho \vDash_{\text{wb}} P$, we write $[\rho; P] \xrightarrow{\mu} [\rho'; P']$ if $P \xrightarrow{\mu} P'$ and one of the following holds:*

1. $\mu = \overline{p}(x)$ *and* $\rho = p : \mathtt{o}, \rho'$
2. $\mu = p(x)$ *and* $\rho = p : \mathtt{i}, \rho'$
3. $\mu = \overline{x}(y, p)$ *and* $\rho' = p : \mathtt{i}, \rho$
4. $\mu = x(y, p)$ *and* $\rho' = p : \mathtt{o}, \rho$
5. $\mu \in \{\ell(x), \overline{\ell}(x), \tau\}$ *and* $\rho' = \rho$.

▶ **Lemma 9** (Subject reduction)**.** *If $[\rho; P] \xrightarrow{\mu} [\rho'; P']$, then for any Q with $\rho \vDash_{\text{wb}} Q$ and $Q \xrightarrow{\mu} Q'$, we have $\rho' \vDash_{\text{wb}} Q'$.*

A relation $\mathcal{R}$ is *wb-typed* if for any $(\rho, P, Q) \in \mathcal{R}$, we have $\rho \vDash_{\text{wb}} P$ and $\rho \vDash_{\text{wb}} Q$. We define the typed version of the expansion preorder, written $\gtrsim_{wb}$, by restricting to typed transitions. As usual, we write $\xrightarrow{\widehat{\tau}}$ for $\xrightarrow{\tau} \cup \mathit{id}$ and $\xrightarrow{\widehat{\mu}}$ for $\xrightarrow{\mu}$ otherwise. The weak variants of the transitions are defined by chaining with $\Rightarrow \stackrel{\text{def}}{=} \xrightarrow{\tau}^*$: for instance, $\stackrel{\mu}{\Longrightarrow} \stackrel{\text{def}}{=} \Rightarrow \xrightarrow{\mu} \Rightarrow$.

▶ **Definition 10** (Wb-expansion)**.** *A wb-typed relation $\mathcal{R}$ is a* wb-expansion *when $(\rho, P, Q) \in \mathcal{R}$ implies:*

- *If we have $[\rho; P] \xrightarrow{\mu} [\rho'; P']$, then there exists Q' such that $Q \xrightarrow{\widehat{\mu}} Q'$ and $(\rho', P', Q') \in \mathcal{R}$.*
- *If we have $[\rho; Q] \xrightarrow{\mu} [\rho'; Q']$, then there exists P' such that $P \stackrel{\mu}{\Longrightarrow} P'$ and $(\rho', P', Q') \in \mathcal{R}$.*

We note $\gtrsim_{wb}$ the largest wb-expansion.

We omit ρ, writing $P \gtrsim_{wb} Q$ when ρ is obvious from the context. This typed expansion is coarser than its untyped variant, which is written $\gtrsim$ [19].

Wb-bisimulation, and wb-bisimilarity noted $\approx_{\text{wb}}$, are defined as wb-expansion by replacing $Q \xrightarrow{\widehat{\mu}} Q'$ with $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ in Definition 10. The untyped version is written $\approx$.

4 Encoding Terms and Values in AIπ

In this section, we describe the encoding of $\lambda^{\mathtt{ref}}$-terms into AIπ processes. This leads to a clean operational correspondence where the encoding of a term may perform a unique transition according to the case distinction of Lemma 1.

We define the encoding in Figure 3. The encoding $[\![M]\!]$ of a term M is defined as an abstraction $(p)\ P$, and we use the notation $[\![M]\!]_q$ (resp. $[\![V]\!]^{\mathtt{v}}_y$) to note $[\![M]\!] \lfloor q \rfloor$ (resp. $[\![V]\!]^{\mathtt{v}} \lfloor y \rfloor$). Intuitively, $[\![M]\!]_p$ is a computation that returns a value at p while $[\![V]\!]^{\mathtt{v}}_y$ is a value that can be accessed at y.

This encoding extends the one from [3] with the additional constructs for handling references. A reference is represented by an output transmitting the stored value. Access to a reference is performed by receiving that output and emitting it back (possibly updated).

To remove any ambiguity in the rules, we consider that M (appearing in $[\![MN]\!]$, $[\![\ell \coloneqq M; N]\!]$ and $[\![VM]\!]$) cannot be a value. This distinction enables optimisations that intuitively remove some communications signaling the end of a subcomputation. This is reminiscent of the colon translation from [15], in the setting of continuation-passing style translations. For any reference-closed configurations $\langle h \mid M \rangle$, we have $p : \mathtt{o} \vdash_{\text{wb}} [\![\langle h \mid M \rangle]\!]_p$.

We extend the encoding to evaluation contexts as shown in Figure 4. When an evaluation context is applied to a non-value term, its encoding correspond to encode the context first and then apply it to the encoding of the term:

▶ **Lemma 11.** *For any E M such that M is not a value, we have $[\![E[M]]\!]_p = [\![E]\!][\![M]\!]]_p$.*

Functions

$$[\![V]\!] \stackrel{\text{def}}{=} (p)\ \overline{p}(y)\ [\![V]\!]^{\mathrm{v}}_y \qquad [\![xV]\!] \stackrel{\text{def}}{=} (p)\ \overline{x}(z,q)\ ([\![V]\!]^{\mathrm{v}}_z \mid q \rhd p)$$

$$[\![(\lambda x.\, N)V]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} y, w)([\![\lambda x.\, N]\!]^{\mathrm{v}}_y \mid [\![V]\!]^{\mathrm{v}}_w \mid \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))$$

$$[\![VM]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid (\boldsymbol{\nu} r)([\![M]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

$$[\![MN]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![M]\!]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

Imperative constructs

$$[\![!\ell]\!] \stackrel{\text{def}}{=} (p)\ \ell(w).\, (\overline{\ell}(y)\ y \rhd w \mid \overline{p}(z)\ z \rhd w) \qquad [\![\ell \coloneqq V; N]\!] \stackrel{\text{def}}{=} (p)\ \ell(\_).\, (\overline{\ell}(y)\ [\![V]\!]^{\mathrm{v}}_y \mid [\![N]\!]_p)$$

$$[\![\ell \coloneqq M; N]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![M]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p))$$

$$[\![\texttt{new}\ \ell \coloneqq V\ \texttt{in}\ N]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![V]\!]_q \mid q(z).\, (\boldsymbol{\nu} \ell)(\overline{\ell}(y)\ y \rhd z \mid [\![N]\!]_p))$$

Configurations

$$[\![h]\!] \stackrel{\text{def}}{=} \prod_{\ell_0 \in \widetilde{\ell}} (\overline{\ell_0}(y)\ [\![h(\ell_0)]\!]^{\mathrm{v}}_y) \text{ with } \widetilde{\ell} = \mathrm{dom}(h) \qquad [\![\langle h \mid N \rangle]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} \widetilde{\ell})([\![h]\!] \mid [\![N]\!]_p)$$

where $[\![V]\!]^{\mathrm{v}}$ is defined as:

$$[\![\lambda x.\, N]\!]^{\mathrm{v}} \stackrel{\text{def}}{=} (y)\ !y(x, q).\, [\![N]\!]_q \qquad [\![x]\!]^{\mathrm{v}} \stackrel{\text{def}}{=} (y)\ y \rhd x$$

Figure 3 Encoding of terms and configurations into AIπ.

$$[\![[\cdot]]\!] \stackrel{\text{def}}{=} [\cdot] \qquad [\![FN]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![F]\!]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

$$[\![\ell \coloneqq F; N]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![F]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p))$$

$$[\![VF]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid (\boldsymbol{\nu} r)([\![F]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

Figure 4 Encoding of evaluation contexts.

Proof. We proceed by induction on E:

1. when $E = [\cdot]$, the result is immediate.
2. when $E = FN$, we have

$$[\![E[M]]\!]_p = (\boldsymbol{\nu} q)([\![F[M]]\!]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))).$$

By induction, we have that $[\![F[M]]\!]_q = [\![F]\!][[\![M]\!]]_q$ and thus:

$$\begin{aligned}[\![E[M]]\!]_p &= (\boldsymbol{\nu} q)([\![F]\!][[\![M]\!]]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))) \\ &= [\![E]\!][[\![M]\!]]_p\end{aligned}$$

3. when $E = VF$, we have

$$[\![E[M]]\!]_p = (\boldsymbol{\nu} y, r)([\![V]\!]^{\mathrm{v}}_y \mid [\![F[M]]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)).$$

   By induction, we have that $[\![F[M]]\!]_r = [\![F]\!][[\![M]]\!]_r$.
   So $[\![E[M]]\!]_p = (\boldsymbol{\nu} y, r)([\![V]\!]^{\mathrm{v}}_y \mid [\![F]\!][[\![M]]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) = [\![E]\!][[\![M]]\!]_p$.
4. when $E = \ell \coloneqq F; N$, then $[\![E[M]]\!]_p = (\boldsymbol{\nu} q)([\![F[M]]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p))$.
   By induction, we have that $[\![F[M]]\!]_q = [\![F]\!][[\![M]]\!]_q$.
   So $[\![E[M]]\!]_p = (\boldsymbol{\nu} q)([\![F]\!][[\![M]]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p)) = [\![E]\!][[\![M]]\!]_p$. ◀

Because we distinguish between values and non-values in the encoding, the previous lemma does not hold when M is a value. In that case, the encoding performs some "optimisations". To relate the two processes, we need that on the encoding, forwarders act like substitutions.

▶ **Lemma 12.** *We have*

1. $(\boldsymbol{\nu} x)([\![M]\!]_p \mid x \rhd y) \gtrsim [\![M\{y/\!/x\}]\!]_p$
2. $(\boldsymbol{\nu} x)([\![V]\!]^{\mathrm{v}}_z \mid x \rhd y) \gtrsim [\![V\{y/\!/x\}]\!]^{\mathrm{v}}_z$
3. $(\boldsymbol{\nu} p)([\![M]\!]_p \mid p \rhd q) \gtrsim [\![M]\!]_q$
4. $(\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid x \rhd y) \gtrsim [\![V]\!]^{\mathrm{v}}_x$

Lemma 12 is proved by induction on the encoding of M or V to prove the four properties in conjonction. Indeed, there are dependencies between these properties which prevent us from treating them separately. This result is proved for the optimised encoding of the plain call-by-value λ-calculus [3] and it extends to the additional constructs of $\lambda^{\mathtt{ref}}$. As a result of the optimisation, we have the following lemma when applying a context to a value:

▶ **Lemma 13.** *For any value V and context E, $[\![E]\!][[\![V]\!]]_p \gtrsim [\![E[V]]\!]_p$.*

Proof. We proceed by induction on E:

- when $E = [\cdot]$, this is trivial.
- When $E = (\lambda x.\, M)\ [\cdot]$, we have

$$\begin{aligned}[\![E]\!][[\![V]\!]]_p &= (\boldsymbol{\nu} y, r)([\![\lambda x.\, M]\!]^{\mathrm{v}}_y \mid [\![V]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) \\ &\to (\boldsymbol{\nu} y, w)([\![\lambda x.\, M]\!]^{\mathrm{v}}_y \mid [\![V]\!]^{\mathrm{v}}_w \mid \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) = [\![E[V]]\!]_p\end{aligned}$$

  As this transition is deterministic by construction, we obtain that $[\![E]\!][[\![V]\!]]_p \gtrsim [\![E[V]]\!]_p$. This also holds for the three following cases.
- When $E = x\ [\cdot]$, we have

$$\begin{aligned}[\![E]\!][[\![V]\!]]_p &= (\boldsymbol{\nu} y, r)([\![x]\!]^{\mathrm{v}}_y \mid [\![V]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) \\ &\to^2 (\boldsymbol{\nu} y, w, w', r')([\![x]\!]^{\mathrm{v}}_y \mid \overline{x}(w'', r'')\ (w'' \rhd w' \mid r'' \rhd r') \mid [\![V]\!]^{\mathrm{v}}_w \mid w' \rhd w \mid r' \rhd p) \\ &\gtrsim (\boldsymbol{\nu} y)[\![x]\!]^{\mathrm{v}}_y \mid \overline{x}(w'', r'')\ (\boldsymbol{\nu} w, w', r')(w'' \rhd w' \mid w' \rhd w \mid [\![V]\!]^{\mathrm{v}}_w \mid r'' \rhd r' \mid r' \rhd p) \\ &\gtrsim (\boldsymbol{\nu} w)(\overline{x}(w'', r'')\ ([\![V]\!]^{\mathrm{v}}_{w''} \mid r'' \rhd p)) = [\![xV]\!]_p\end{aligned}$$

- When $E = [\cdot]\ M$, we have

$$\begin{aligned}[\![E]\!][[\![V]\!]]_p &= (\boldsymbol{\nu} q)([\![V]\!]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))) \\ &\to (\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))) = [\![xV]\!]_p\end{aligned}$$

- When $E = \ell \coloneqq [\cdot]; M$, we have

$$\begin{aligned}[\![E]\!][[\![V]\!]]_p &= (\boldsymbol{\nu} q)([\![V]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p)) \\ &\to (\boldsymbol{\nu} w)([\![V]\!]^{\mathrm{v}}_w \mid \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p)) \\ &\gtrsim \ell(\_).\, (\overline{\ell}(y)\ (\boldsymbol{\nu} w)([\![V]\!]^{\mathrm{v}}_w \mid y \rhd w) \mid [\![N]\!]_p) \\ &\gtrsim \ell(\_).\, (\overline{\ell}(y)\ [\![V]\!]^{\mathrm{v}}_y \mid [\![N]\!]_p) = [\![E[V]]\!]_p\end{aligned}$$

- when $E = F\ M$ or $V\ F$ or $\ell := F; M$ with $F \neq [\cdot]$, then $F[V]$ is not a value, so $[\![E[V]]\!]_p = [\![E']\!][[\![F[M]]\!]]_p$ for some E' and the result follows by induction as $\gtrsim$ is a congruence. ◀

We can now establish operational correspondence.

▶ **Proposition 14** (Untyped Operational Correspondence). *For any M, h with $\mathrm{dom}(h) = \widetilde{\ell}$ and fresh q_0, $[\![\langle h \mid M\rangle]\!]_{q_0}$ has exactly one immediate transition, and exactly one of the following clauses holds:*

1. *$\langle h \mid M\rangle \to \langle h' \mid N\rangle$ and $[\![\langle h \mid M\rangle]\!]_{q_0} \xrightarrow{\tau} P$ with $P \gtrsim [\![\langle h' \mid N\rangle]\!]_{q_0}$*
2. *M is a value, $[\![\langle h \mid M\rangle]\!]_{q_0} \xrightarrow{\overline{q_0}(x_0)} P$ and $P = (\boldsymbol{\nu}\widetilde{\ell})([\![h]\!] \mid [\![M]\!]^{\mathtt{v}}_{x_0})$.*
3. *M is of the form $E_0[yV_0]$ for some E_0, y and V_0, and we have $[\![\langle h \mid M\rangle]\!]_{q_0} \xrightarrow{\overline{y}(x_0,p_0)} P$ with $P \gtrsim (\boldsymbol{\nu}\widetilde{\ell})([\![h]\!] \mid [\![V_0]\!]^{\mathtt{v}}_{x_0} \mid p_0(z).\,[\![E_0[z]]\!]_{q_0})$.*

In the first case, the τ transition is deterministic, so we can prove that the following holds: $[\![\langle h \mid M\rangle]\!]_{q_0} \gtrsim [\![\langle h' \mid N\rangle]\!]_{q_0}$.

Using the existing equivalences in AIπ from Section 3, we can prove that the encodings of two $\lambda^{\mathtt{ref}}$-terms are equivalent. The equivalence and preorder used here are finer than bisimilarity with divergence which is sound as stated in Section 5. These examples show how the standard theory of the π-calculus is enough to prove interesting properties of $\lambda^{\mathtt{ref}}$-terms. These properties do not require us to use the triples defined in Section 5 nor to take into account the possibility of deferred divergence.

▶ **Example 15** (Unused reference). For any reference ℓ, value V and any term M with $\ell \notin \mathrm{fr}(M)$, we have $[\![\mathtt{new}\ \ell := V\ \mathtt{in}\ M]\!]_p \gtrsim [\![M]\!]_p$.

Proof. We write

$$\begin{aligned}[\![\mathtt{new}\ \ell := V\ \mathtt{in}\ M]\!]_p &\gtrsim [\![\langle \emptyset \mid \mathtt{new}\ \ell := V\ \mathtt{in}\ M\rangle]\!]_p \\ &\gtrsim [\![\langle \ell = V \mid M\rangle]\!]_p && \text{by Proposition 14} \\ &\gtrsim (\boldsymbol{\nu}\ell)(\overline{\ell}(y)\ [\![V]\!]^{\mathtt{v}}_y) \mid [\![M]\!]_p \gtrsim [\![M]\!]_p\end{aligned}$$

We use $\gtrsim$ to perform a deterministic reduction and then to use simple laws and remove inaccessible processes. This result can be extended in presence of store by congruence, giving $[\![\langle h \mid \mathtt{new}\ \ell := V\ \mathtt{in}\ M\rangle]\!]_p \gtrsim [\![\langle h \mid M\rangle]\!]_p$. ◀

▶ **Example 16** (One-use context). Let $f_1 \stackrel{\mathrm{def}}{=} \lambda x.\,\mathtt{if}\ !\ell = \mathtt{tt}\ \mathtt{then}\ \ell := \mathbf{ff}; \mathtt{tt}\ \mathtt{else}\ \mathbf{ff}$ and $f_2 \stackrel{\mathrm{def}}{=} \lambda x.\,\mathtt{tt}$ and $E = [\cdot]\ (x\lambda y.\,y)$. Then $[\![\mathtt{new}\ \ell := \mathtt{tt}\ \mathtt{in}\ E[f_1]]\!]_p \approx [\![E[f_2]]\!]_p$

Proof. By Proposition 14 and congruence of $\gtrsim$ we have:

$$\begin{aligned}[\![E[f_2]]\!]_p &\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![f_2 w]\!]_p) \\ &\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\mathtt{tt}]\!]_p)\end{aligned}$$

Using standard laws for $\gtrsim$, we relate the encoding of the first program to the same process:

$$\begin{aligned}[\![\mathtt{new}\ \ell := \mathtt{tt}\ \mathtt{in}\ E[f_1]]\!]_p &\gtrsim (\boldsymbol{\nu}\ell)([\![\ell = \mathtt{tt}]\!] \mid \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![f_1 w]\!]_p)) \\ &\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\langle \ell = \mathtt{tt} \mid f_1 w\rangle]\!]_p) \\ &\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\langle \ell = \mathbf{ff} \mid \mathtt{tt}\rangle]\!]_p) \\ &\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\mathtt{tt}]\!]_p)\end{aligned}$$

Note that we are outside AIπ because we are using constants ($\mathbf{tt}$, $\mathbf{ff}$). Adapting our setting to simple types can be done by having sorts $\mathtt{F}, \mathtt{R}, \mathtt{C}$ and forwarders $\triangleright_{\mathtt{C}}, \triangleright_{\mathtt{F}}$ for each type T, the forwarders being defined inductively on T instead of being constants. ◀

$$[\![\widetilde{V}_i]\!]^{\mathtt{v}}_{\widetilde{x}} \stackrel{\text{def}}{=} \prod_i [\![V_i]\!]^{\mathtt{v}}_{x_i} \qquad \text{with } \widetilde{x} = \widetilde{x_i}$$

$$[\![E_1 :: \cdots :: E_n]\!]_{\widetilde{pq}} \stackrel{\text{def}}{=} \prod_{i\leq n} p_i(z).\,[\![E_i[z]]\!]_{q_i} \qquad \text{with } \widetilde{pq} = p_1, q_1, \ldots, p_n, q_n$$

$$[\![(\widetilde{V}_i, \sigma, h)]\!]_{\widetilde{x};\widetilde{pq}} \stackrel{\text{def}}{=} (\boldsymbol{\nu}\widetilde{\ell})([\![\widetilde{V}_i]\!]_{\widetilde{x}} \mid [\![\sigma]\!]_{\widetilde{pq}} \mid [\![h]\!]) \qquad \text{with } \widetilde{\ell} = \mathrm{dom}(h)$$

$$[\![(\widetilde{V}_i, \sigma, \langle h \mid M\rangle)]\!]_{\widetilde{x};q_0,\widetilde{pq}} \stackrel{\text{def}}{=} (\boldsymbol{\nu}\widetilde{\ell})([\![\widetilde{V}_i]\!]_{\widetilde{x}} \mid [\![\sigma]\!]_{\widetilde{pq}} \mid [\![h]\!] \mid [\![M]\!]_{q_0}) \qquad \text{with } \widetilde{\ell} = \mathrm{dom}(h)$$

Figure 5 Encoding for triples.

5 A π-Calculus Characterisation of Contextual Equivalence in $\lambda^{\mathtt{ref}}$

We can now move on to show our full abstraction result. To do so, we first extend the encoding to the triples defined in Section 2. This leads to an operational correspondence similar to Proposition 14 but for triples (Theorem 17). However, thanks to triples, it is possible to state Theorem 17 without using explicit AIπ constructs, so that each transition relates the encoding of triples. The bisimulation with divergence can then be defined and shown fully abstractly using mainly the operational correspondence theorem.

We describe in Figure 5 the encoding of triples (Section 2). It builds on the encoding in Figure 3, with values being encoded as expected. To encode σ, every evaluation context E in σ is encoded as the process $p(z).\,[\![E[z]]\!]_q$ so that, intutively, $E[z]$ can be executed as soon as the input at p is triggered. Both $[\![(\widetilde{V}_i, \sigma, c)]\!]_{\widetilde{x};q_0,\widetilde{pq}}$ and $[\![(\widetilde{V}_i, \sigma, h)]\!]_{\widetilde{x};\widetilde{pq}}$ are typeable with stacks that we write $\rho_{q_0,\widetilde{pq}}$ and $\rho_{\widetilde{pq}}$ respectively.

▶ **Theorem 17** (Operational Correspondence)**.**
We relate transitions for the encoding of both kind of triples:
When $[\rho; [\![(\widetilde{V}_i, \sigma, c)]\!]_{\widetilde{x};q_0,\widetilde{pq}}] \xrightarrow{\mu} [\rho'; P]$ *with* $\widetilde{x} = \widetilde{x_i}$ *and* $\widetilde{pq} = p_1, q_1, \ldots, p_n, q_n$ *then:*

1. *either* $c \to c'$, $\mu = \tau$ *and* $P \gtrsim [\![(\widetilde{V}_i, \sigma, c')]\!]_{\widetilde{x};q_0,\widetilde{pq}}$
2. *or* $c = \langle h \mid V_0\rangle$, $\mu = \overline{q_0}(x_0)$ *and* $P \gtrsim [\![((\widetilde{V}_i, V_0), \sigma, h)]\!]_{x_0,\widetilde{x};\widetilde{pq}}$
3. *or* $c = \langle h \mid E_0[y\ V_0]\rangle$, $\mu = \overline{y}(x_0, p_0)$ *and* $P \gtrsim [\![((\widetilde{V}_i, V_0), E_0 :: \sigma, h)]\!]_{x_0,\widetilde{x};p_0,q_0,\widetilde{pq}}$

When $[\rho; [\![(\widetilde{V}_i, \sigma, h)]\!]_{\widetilde{x};\widetilde{pq}}] \xrightarrow{\mu} [\rho'; P]$ *with* $\widetilde{x} = \widetilde{x_i}$ *and* $\widetilde{pq} = p_1, q_1, \ldots, p_n, q_n$ *then:*

1. *either* $\mu = x_j(z, q_0)$ *and* $P \gtrsim [\![(\widetilde{V}_i, \sigma, \langle h \mid N\rangle)]\!]_{\widetilde{x};q_0,\widetilde{pq}}$ *with* $V_j z \succ N$.
2. *or* $\sigma = E_1 :: \sigma'$ *and* $\mu = p_1(z)$ *and* $P \gtrsim [\![(\widetilde{V}_i, \sigma', \langle h \mid E_1[z]\rangle)]\!]_{\widetilde{x};q_1,p_2,q_2,\ldots,p_n,q_n}$

The following result is useful for the full abstraction proof.

▶ **Corollary 18.** *If* $[\![(\widetilde{V}_i, \sigma, c)]\!]_{\widetilde{x};q_0,\widetilde{pq}} \Rightarrow P'$, *then there exists a configuration* c' *with* $c \Rightarrow c'$ *and* $P' \gtrsim [\![(\widetilde{V}_i, \sigma, c')]\!]_{\widetilde{x};q_0,\widetilde{pq}}$.

Note that we rely on untyped expansion, which does not make any assumption about sequentiality of processes, in Theorem 17 and Corollary 18. This shows the robustness of the encoding.

The following example shows that by contrast with the encoding of configurations, the τ transition in the first case of Theorem 17 is not deterministic.

▶ **Example 19.** $[\![(\{\lambda z.\,\ell \coloneqq z\}, \odot, \langle \ell = y \mid !\ell\rangle)]\!]_{x_1;q_0} \not\gtrsim [\![(\{\lambda z.\,\ell \coloneqq z\}, \odot, \langle \ell = y \mid y\rangle)]\!]_{x_1;q_0}$.

Indeed, as the π-calculus is concurrent, the encoding of $\lambda z.\, \ell := z$ may be executed to change the content of ℓ before the read is executed. However, we can recover this result by using $\gtrsim_{wb}$ which forbids the concurrent transitions. This makes $\gtrsim_{wb}$ useful to handle reductions for triples.

We now introduce the notion of divergence for π-terms. This leads to the definition of bisimulation with divergence which coarsens $\approx_{\mathrm{wb}}$ to account for divergent terms. The induced equivalence for $\lambda^{\mathtt{ref}}$-terms corresponds to nfb.

A *wb-typed set* is a set of pairs (ρ, P) with ρ clean and $\rho \vDash_{\mathrm{wb}} P$.

▶ **Definition 20** (π_{div}). *A wb-typed set S is π-divergent if whenever we have $(\rho, P) \in S$, then $\rho \neq \emptyset$ and for all μ, ρ', P' with $[\rho; P] \xrightarrow{\mu} [\rho'; P']$, we have $(\rho', P') \in S$.*

We write π_{div} for the largest π-divergent set.

Intuitively, the computation ends when the stack gets empty, meaning there is no pending continuation. Processes in π_{div} thus correspond to processes which cannot terminate, hence the name of π-divergence.

The following example shows that the divergence of a process depends on the stack used to type it.

▶ **Example 21.** Let us consider

$$P \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu} x)(p_1(z).\, \overline{x}(y, r_1)\ r_1 \rhd q_1 \mid p_2(z).\, \overline{q_2}(y)\ x(y', r).\, \overline{r}(z')\ \mathbf{0}),$$

$$\rho_1 = p_1 : \mathtt{i}, q_1 : \mathtt{o}, p_2 : \mathtt{i}, q_2 : \mathtt{o}, \quad \rho_2 = p_2 : \mathtt{i}, q_2 : \mathtt{o}, p_1 : \mathtt{i}, q_1 : \mathtt{o}.$$

We have both $\rho_1 \vDash_{\mathrm{wb}} P$ and $\rho_2 \vDash_{\mathrm{wb}} P$.

$[\rho_1; P] \xrightarrow{p_1(z)} [\rho'; P']$ is the only typed-allowed transition and P' has no typed-allowed transition. Thus, $(\rho_1, P) \in \pi_{\mathrm{div}}$.

On the other hand, $[\rho_2; P] \xrightarrow{p_2(z)} \xrightarrow{\overline{q_2}(y)} \xrightarrow{p_1(z_0)} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\overline{q_1}(z')} [\emptyset; P']$. So $(\rho_2, P) \notin \pi_{\mathrm{div}}$.

▶ **Definition 22.** *A wb-typed symmetric relation $\mathcal{R}$ is a* bisimulation with divergence *if there exists a π-divergent set S such that whenever we have $(\sigma, P, Q) \in \mathcal{R}$ and $[\sigma; P] \xrightarrow{\mu} [\sigma'; P']$, then one of the following holds:*

1. *there exists Q' with $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $(\sigma', P', Q') \in \mathcal{R}$;*
2. *μ is an output and $(\sigma', P') \in S$.*

We write $\approx_{\mathrm{div}}$ for the largest bisimulation with divergence. We write $P \approx^{\rho}_{\mathrm{div}} Q$ when $(\rho, P, Q) \in \approx_{\mathrm{div}}$.

Bisimilarity with divergence is coarser than wb-bisimilarity and thus also coarser than the untyped bisimilarity.

To establish soundness, we rely on Theorem 17 (operational correspondence). The set of triples whose encoding is π-diverging is itself diverging, so we can prove that the relation induced by $\approx_{\mathrm{div}}$ is a nfb.

▶ **Theorem 23** (Soundness). *If $[\![M]\!]_p \approx^{p:\mathtt{o}}_{\mathrm{div}} [\![N]\!]_p$, then $M \asymp N$.*

The completeness is proved by showing that the encoding of a divergent set is π-divergent up to $\gtrsim$ and then Property (1) from the introduction, namely that the encoding of a nfb is a bisimulation with divergence up to $\gtrsim$.

▶ **Theorem 24** (Completeness). *If $M \asymp N$, then $[\![M]\!]_p \approx^{p:\mathtt{o}}_{\mathrm{div}} [\![N]\!]_p$.*

6 Up-to Techniques for $\approx_{\text{div}}$ in AIπ, and Applications

Up-to techniques are defined as functions from relations to relations. The idea of up-to techniques is to weaken the requirement by applying the up-to technique to the relation $\mathcal{R}$ in clause 1 of Definition 22. Standard up-to techniques like up-to expansion or up-to context can be adapted to our typed setting as in [5].

Similar up-to techniques for sets can also be defined and exploited for π-divergent sets. In order for up-to context to be sound, we must forbid contexts where the hole is guarded by a replicated input. Indeed, replicated input may only be typed with an empty set so they cannot be diverging. This is similar to $\lambda^{\mathtt{ref}}$, where Ω is divergent but $\lambda x.\,\Omega$ is not.

Thanks to up-to techniques, to prove that two processes are equivalent, we can give $\mathcal{R}$, a bismulation with divergence up to using S, a π-divergent set up to. This is used in the equivalences proven in Section 6.2.

6.1 A new up-to technique for AIπ: up-to body

In AIπ, functions are encoded as a replicated process of the form $!x(y,p).\,T$, which we denote T^x. When identical calls result in the same value being sent, it creates copies of that value, leading to processes of the form $T^x \mid T^y \mid \ldots \mid T^z$, with $x, y, \ldots, z$ being all different names. This behaviour makes bisimulations infinite, as they would need to contain processes with an arbitrary number of processes in parallel, despite all of them sharing the same body. We introduce a new technique, up-to body, which allows us to remove these duplicated copies. Indeed, it is sound to keep only one copy when comparing processes: any discriminating interaction with the environment involving multiple copies can be mimicked by a similar interaction with only one copy. The up-to body technique is defined by the following rule:

$$\frac{(\rho,\ E[T_1^x],\ F[T_2^x]) \in \mathcal{R}}{(\rho,\ E[T_1^z \mid T_1^x],\ F[T_2^z \mid T_2^x]) \in \mathtt{body}(\mathcal{R})}\, x, z \notin \mathrm{n}(E) \cup \mathrm{n}(F) \cup \mathrm{fn}(T_1) \cup \mathrm{fn}(T_2)$$

Up-to body differs from up-to context because we keep the possibly different evaluation contexts, E and F. These contexts correspond to private resources shared among the copies. In our case, the private resource is the local store, but this technique can be exported to the plain π-calculus. The technique for sets is defined similarly, with $(\rho, E[T^z \mid T^x]) \in \mathtt{body}(S)$ whenever $(\rho, E[T^x]) \in S$ and $x, z \notin \mathrm{n}(E) \cup \mathrm{fn}(T)$.

Using up-to body, it is possible to prove the analog of (1) from Section 1 but using normal form bisimulations from [2] instead of the formulation we have given in Definition 5.

We now present a simple example that demonstrates the use of up-to body.

▶ **Example 25.** $\mathtt{new}\ \ell \coloneqq z\ \mathtt{in}\ \lambda x.\,\lambda y.\,!\ell \asymp \lambda x.\,\lambda y.\,z$

Proof. We reason using the soundness of the encoding, and define

$$\begin{aligned}\mathcal{R} = \{&(\rho_q, [\![\mathtt{new}\ \ell \coloneqq z\ \mathtt{in}\ \lambda x.\,\lambda y.\,!\ell]\!]_q, [\![\lambda x.\,\lambda y.\,z]\!]_q),\\ &(\emptyset, [\![(\lambda x.\,\lambda y.\,!\ell,), \odot, \ell = z]\!]_{x_0}, [\![(\lambda x.\,\lambda y.\,z,), \odot, \emptyset]\!]_{x_0})\\ &(\emptyset, [\![(\lambda x.\,\lambda y.\,!\ell, \lambda y.\,!\ell), \odot, \ell = z]\!]_{x_0,x_1}, [\![(\lambda x.\,\lambda y.\,z, \lambda y.\,z), \odot, \emptyset]\!]_{x_0,x_1})\}.\end{aligned}$$

We can show that $\mathcal{R}$ is a bisimulation with divergence up to $\gtrsim_{wb}$, context and body. The up-to body technique makes it possible to stop the relation after the third pair. ◀

6.2 Examples

We now present an example that involves the encoding of triples, but does not require us to take into account deferred divergences. To validate the laws below, we thus rely on $\approx_{\text{wb}}$ which is included in $\approx_{\text{div}}$ (Definition 22).

▶ **Example 26** (Optimised access)**.** Two consecutive read and/or write operations can be transformed into an equivalent single operation.

For any pair (f_1, f_2) from the following (() is the unique inhabitant of the unit type):

$$\begin{array}{ll}
\big(f_1 = \lambda().\, \ell \coloneqq !\ell, & f_2 = \lambda().\, ()\big) \\
\big(f_1 = \lambda n.\, \lambda m.\, \ell \coloneqq n; \ell \coloneqq m, & f_2 = \lambda n.\, \lambda m.\, \ell \coloneqq m\big) \\
\big(f_1 = \lambda n.\, \ell \coloneqq n; !\ell, & f_2 = \lambda n.\, \ell \coloneqq n; n\big) \\
\big(f_1 = \lambda().\, \texttt{let}\ x = !\ell\ \texttt{in}\ \texttt{let}\ y = !\ell\ \texttt{in}\ M, & f_2 = \lambda().\, \texttt{let}\ x = !\ell\ \texttt{in}\ M\{x/y\}\big)
\end{array}$$

we have for any E, y, V_0, we have $[\![\texttt{new}\ \ell \coloneqq V_0\ \texttt{in}\ E[y\ f_1]]\!]_{q_1} \approx_{\text{wb}} [\![\texttt{new}\ \ell \coloneqq V_0\ \texttt{in}\ E[y\ f_2]]\!]_{q_1}$.

Proof. First, we have that $[\![((f_1,), E :: \odot, \ell \coloneqq V_0)]\!]_{x;p_1,q_1} \approx_{\text{wb}} [\![((f_2,), E :: \odot, \ell \coloneqq V_0)]\!]_{x;p_1,q_1}$ using $(f_i,)$ to denote the singleton containing f_i. Indeed, we define a relation $\mathcal{R}$ as follows:

$$\mathcal{R} = \{(\rho_{\widetilde{pq}},\ [\![((\widetilde{V_i}, f_1), \sigma, h \uplus \ell = V)]\!]_{\widetilde{x};\widetilde{pq}}, [\![((\widetilde{V_i}, f_2), \sigma, h \uplus \ell = V)]\!]_{\widetilde{x};\widetilde{pq}}) \mid \text{ for all } \widetilde{V_i}, \sigma, h, V, \widetilde{x}, \widetilde{pq}\}$$

and we show that $\mathcal{R}$ is a *wb*-bisimulation up to $\gtrsim_{wb}$ and evaluation context.

Then we use Theorem 17 to show

$$[\![\texttt{new}\ \ell \coloneqq V_0\ \texttt{in}\ E[x\ f_i]]\!]_{q_1} \gtrsim [\![(\emptyset, \odot, \langle \ell = V_0 \mid E[y\ f_i]\rangle)]\!]_{q_1} \xrightarrow{\overline{y}(x,p_1)} \gtrsim [\![(f_i,), E :: \odot, \ell = V_0)]\!]_{x;p_1,q_1}$$

with the output being the only transition that the intermediate process can perform. ◀

The proof of this law could become even simpler by adopting the type system of [4]: we could prove prove directly that $[\![f_1]\!]_p$ and $[\![f_2]\!]_p$ are equivalent.

Relation $\mathcal{R}$ above would have the same base if we were to reason in the source language. If we were to show $\texttt{new}\ \ell \coloneqq V\ \texttt{in}\ E[y\ f_1] \asymp \texttt{new}\ \ell \coloneqq V\ \texttt{in}\ E[y\ f_2]$ using nfb, we would need to add triples with the same normal form term on both sides.

We present some examples involving deferred divergence from the literature.

▶ **Example 27.** $\langle \emptyset \mid x\ V\ \Omega\rangle \asymp \langle \emptyset \mid \Omega\rangle$, where V is a value and Ω is an always diverging term.

Proof. Take $\mathcal{R} = \{(\rho_q, [\![\langle \emptyset \mid x\ V\ \Omega\rangle]\!]_q, [\![\langle \emptyset \mid \Omega\rangle]\!]_q)\}$ and $S = \{(\rho_r, [\![\Omega]\!]_r)\}$.

S is π-divergent, so $\gtrsim_{wb}\texttt{ctxt}(S)$ is too, where $\texttt{ctxt}$ stands for the up-to context technique. Then we show that $\mathcal{R}$ is a bisimulation with divergence up to context with $\gtrsim_{wb}\texttt{ctxt}(S)$ as the π-divergent set.

$[\rho_q; [\![\langle \emptyset \mid x\ V\ \Omega\rangle]\!]_q] \xrightarrow{\overline{x}(y,p)} [\rho_{p,q}; P]$ is the only type-allowed transition.

By Theorem 17, we know that $P \gtrsim [\![(\{V\}, [\cdot]\ \Omega :: \odot, \emptyset)]\!]_{y;pq}$.

As $(\rho_{p,q}, [\![(\{V\}, [\cdot]\ \Omega :: \odot, \emptyset)]\!]_{y;pq}) \in \texttt{ctxt}(S)$, we indeed have $(\rho_{p,q}, P) \in \gtrsim_{wb}\texttt{ctxt}(S)$. ◀

▶ **Example 28** (Example 9 from [2])**.**

$$V_1 = \lambda x.\, \texttt{if}\ !\ell\ \texttt{then}\ \Omega\ \texttt{else}\ k \coloneqq \mathbf{tt} \qquad W_1 = \lambda x.\, \Omega$$

$$V_2 = \lambda f.\, f\ V_1; \texttt{if}\ !k\ \texttt{then}\ \Omega\ \texttt{else}\ \ell \coloneqq \mathbf{tt} \qquad W_2 = \lambda f.\, f\ W_1$$

We have $\texttt{new}\ \ell \coloneqq \mathbf{ff}\ \texttt{in}\ \texttt{new}\ k \coloneqq \mathbf{ff}\ \texttt{in}\ V_2 \asymp W_2$.

Proof. Given a context E, we write E^n for $E :: E :: \cdots :: E :: \odot$ with n occurences of E.

Let $E \stackrel{\text{def}}{=} (\lambda().\,\texttt{if}\ !k\ \texttt{then}\ \Omega\ \texttt{else}\ \ell \coloneqq \mathbf{tt})[\cdot]$, and $F \stackrel{\text{def}}{=} (\lambda().\,())[\cdot]$. We define $\mathcal{R}$ as

$$
\begin{array}{lll}
\{(\rho_q, & [\![\texttt{new}\ \ell \coloneqq \mathbf{ff}\ \texttt{in new}\ k \coloneqq \mathbf{ff}\ \texttt{in}\ V_2]\!]_q, & [\![W_2]\!]_q), \\
(\emptyset, & [\![(V_2,), \odot, \ell = \mathbf{ff} \uplus k = \mathbf{ff}]\!]_{x_2}, & [\![(W_2,), \odot, \emptyset]\!]_{x_2})\} \bigcup \\
\{(\rho_{\widetilde{pq}}, & [\![(V_1, V_2), E^n, \ell = \mathbf{ff} \uplus k = \mathbf{ff}]\!]_{\widetilde{x_i};\widetilde{pq}}, & [\![(W_1, W_2), F^n, \emptyset]\!]_{\widetilde{x_i};\widetilde{pq}}), \\
(\rho_{q_0,\widetilde{pq}}, & [\![(V_1, V_2), E^n, \langle \ell = \mathbf{ff} \uplus k = \mathbf{tt} \mid ()\rangle]\!]_{\widetilde{x_i};q_0,\widetilde{pq}}, & [\![(W_1, W_2), F^n, \langle \emptyset \mid \Omega\rangle]\!]_{\widetilde{x_i};q_0,\widetilde{pq}}), \\
(\rho_{\widetilde{pq}}, & [\![(V_1, V_2), E^n, \ell = \mathbf{tt} \uplus k = \mathbf{ff}]\!]_{\widetilde{x_i};\widetilde{pq}}, & [\![(W_1, W_2), F^n, \emptyset]\!]_{\widetilde{x_i};\widetilde{pq}}) \\
 & & \mid \text{ for all } \widetilde{pq}, n \text{ with } |\widetilde{pq}| = 2n\ \}
\end{array}
$$

and S as $\{(\rho_q, [\![\Omega]\!]_q),\ (\rho_{\widetilde{pq}}, [\![(V_1, V_2), E^n, \ell = \mathbf{ff} \uplus k = \mathbf{tt}]\!]_{\widetilde{x_i};\widetilde{pq}})$ for all $\widetilde{pq}, n$ with $|\widetilde{pq}| = 2n\}$.

$\mathcal{R}$ is a bisimulation up to $\gtrsim_{wb}$, context and body. Multiple calls to V_1, W_1 create multiples continuations, but thanks to up-to body, multiples calls to V_2, W_2 do not create multiples copies of V_1, W_1. All in all, we have the same number of pairs in the candidate bisimulation relation as in the relation in [2]. ◀

By modifying this example so as to allocate the references inside V_2, we get Example 15 from [8]. In that case, V_2 does not have free reference names. This makes it possible to use up-to parallel composition between the encoding of V_2 and V_1 preventing multiples calls to V_2. This usage of up-to parallel composition is similar to the up-to separation technique introduced in [8].

7 Related and Future Works

7.1 Related works

The equivalence in [8] is similar to nfb, for an extension of $\lambda^{\texttt{ref}}$. It is also fully abstract w.r.t. contextual equivalence. Up-to techniques for nfb are defined in [2], and used to prove several equivalences between $\lambda^{\texttt{ref}}$ programs. We can redo essentially the same proofs in our setting. Both works use techniques that are specific to nfb or its variant, and are arguably less standard than the up-to techniques we exploit in the π-calculus. In particular, the *up-to separation* from [8] is expressible using up-to context in the π-calculus.

A full abstraction result for PCF programs in AIπ is presented in [1], using a contextual equivalence in a typed setting. The type system captures closely the behaviour of the encoding of PCF terms, in the sense that any process whose type is the translation of a PCF type is behaviourally equivalent to the encoding of a source term, and in particular cannot be stateful. It seems difficult to find a labelled bisimilarity for this equivalence, and thus to use proof techniques as in our paper.

Our encoding is based on the one of [3], which has been used to show a close connection between operational game semantics and the π-calculus for call-by-value in [7]. Both works focus on this stateless calculus to show the correspondence with Lassen trees.

The type system of Section 3 is inspired by evaluation stacks used to ensure the bracketed condition in game semantics [9]. Game semantics can also be used to provide a fully abstract model for RefML, which is a language similar to $\lambda^{\texttt{ref}}$ [12, 13]. By being more operational, our approach is usable more directly to reason about $\lambda^{\texttt{ref}}$ programs. This is similar to operational game semantics which are complete for recursion-free programs with integer references [6].

7.2 Future works

The type system of [4] makes it possible to reason about references in a parallel setting. We believe that its addition to our type system would allow us to extend our full abstraction result to *open references*, i.e., references that can be returned by functions.

It would be interesting to see whether $\approx_{\text{div}}$ characterises a contextually-defined equivalence in AIπ. In comparison, such a result holds for $\approx_{\text{wb}}$ [5]: this equivalence corresponds to a typed barbed equivalence with a notion of *typed barb* to restrict the visibility of observables. One lead would be limiting barbs to be only outputs on continuation names. This may work only for the image of the encoding, but not for all typeable processes.

We would like to see whether the techniques we have developed can be exploited in other languages. Idealized ALGOL has both functional and imperative aspects, so our techniques may adapt to it. When extending $\lambda^{\mathtt{ref}}$ with control operators [20], well-bracketing is not required, so we can weaken the type system to simply ensure sequentiality. Because of the links with game semantics, object-oriented languages [14] can be interesting too.

References

1 Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the pi-calculus. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Proceedings*, volume 2044 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2001. doi:10.1007/3-540-45413-6_7.

2 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity for state. In Mikolaj Bojanczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2019. doi:10.1007/978-3-030-17127-8_6.

3 Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Eager Functions as Processes (long version). working paper or preprint, December 2021. URL: https://hal.archives-ouvertes.fr/hal-03466150.

4 Daniel Hirschkoff, Enguerrand Prebet, and Davide Sangiorgi. On the representation of references in the pi-calculus. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020*, volume 171 of *LIPIcs*, pages 34:1–34:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.34.

5 Daniel Hirschkoff, Enguerrand Prebet, and Davide Sangiorgi. On sequentiality and well-bracketing in the π-calculus. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470559.

6 Guilhem Jaber. Syteci: automating contextual equivalence for higher-order programs with references. *Proc. ACM Program. Lang.*, 4(POPL):59:1–59:28, 2020. doi:10.1145/3371127.

7 Guilhem Jaber and Davide Sangiorgi. Games, mobile processes, and functions. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022).*, Göttingen, Germany, February 2022. URL: https://hal.archives-ouvertes.fr/hal-03407123.

8 Vasileios Koutavas, Yu-Yang Lin, and Nikos Tzevelekos. From bounded checking to verification of equivalence via symbolic up-to techniques. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part II*, volume 13244 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2022. doi:10.1007/978-3-030-99527-0_10.

9 James Laird. A fully abstract trace semantics for general references. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 667–679. Springer, 2007. doi:10.1007/978-3-540-73420-8_58.

10 Jean-Marie Madiot, Damien Pous, and Davide Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2014. doi:10.1007/978-3-662-44584-6_8.

11 Robin Milner. Functions as processes. *Math. Struct. Comput. Sci.*, 2(2):119–141, 1992. doi:10.1017/S0960129500001407.

12 Andrzej S. Murawski and Nikos Tzevelekos. Game semantics for good general references. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 75–84. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.31.

13 Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic games for full ground references. *Formal Methods Syst. Des.*, 52(3):277–314, 2018. doi:10.1007/s10703-017-0292-9.

14 Andrzej S. Murawski and Nikos Tzevelekos. Game Semantics for Interface Middleweight Java. *J. ACM*, 68(1):4:1–4:51, 2021. doi:10.1145/3428676.

15 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

16 Damien Pous. Coinduction all the way up. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.

17 Davide Sangiorgi. Locality and interleaving semantics in calculi for mobile processes. *Theor. Comput. Sci.*, 155(1):39–83, 1996. doi:10.1016/0304-3975(95)00020-8.

18 Davide Sangiorgi. Lazy functions and mobile processes. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 691–720. The MIT Press, 2000.

19 Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

20 Kristian Støvring and Søren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 161–172. ACM, 2007. doi:10.1145/1190216.1190244.

21 Bernardo Toninho and Nobuko Yoshida. On polymorphic sessions and functions: A tale of two (fully abstract) encodings. *ACM Trans. Program. Lang. Syst.*, 43(2):7:1–7:55, 2021. doi:10.1145/3457884.

What Can Oracles Teach Us About the Ultimate Fate of Life?

Ville Salo
Department of Mathematics and Statistics, University of Turku, Finland

Ilkka Törmä
Department of Mathematics and Statistics, University of Turku, Finland

Abstract

We settle two long-standing open problems about Conway's Life, a two-dimensional cellular automaton. We solve the Generalized grandfather problem: for all $n \geq 0$, there exists a configuration that has an nth predecessor but not an $(n+1)$st one. We also solve (one interpretation of) the Unique father problem: there exists a finite stable configuration that contains a finite subpattern that has no predecessor patterns except itself. In particular this gives the first example of an unsynthesizable still life. The new key concept is that of a spatiotemporally periodic configuration (agar) that has a unique chain of preimages; we show that this property is semidecidable, and find examples of such agars using a SAT solver.

Our results about the topological dynamics of Game of Life are as follows: it never reaches its limit set; its dynamics on its limit set is chain-wandering, in particular it is not topologically transitive and does not have dense periodic points; and the spatial dynamics of its limit set is non-sofic, and does not admit a sublinear gluing radius in the cardinal directions (in particular it is not block-gluing). Our computability results are that Game of Life's reachability problem, as well as the language of its limit set, are PSPACE-hard.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Game of Life, cellular automata, limit set, symbolic dynamics

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.131

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2202.07346`

Supplementary Material *Software (Source Code)*: `https://github.com/ilkka-torma/gol-agars` [30] archived at `swh:1:dir:395e8074fa419c9af6799cc0b60c9c80a9947eae`

Funding *Ville Salo*: Research supported by Academy of Finland grant 2608073211.

Acknowledgements We thank the anonymous referees for their useful suggestions, and the nice people of the ConwayLife forum for their interest.

1 Introduction

Conway's Game of Life is a famous two-dimensional cellular automaton defined by John Horton Conway in 1970 and popularized by Martin Gardner [14]. A cellular automaton can be thought of as zero-player game: the board is set up, and a simple rule determines the dynamics. In the case of Game of Life, the board is the two-dimensional infinite grid, where some grid cells are *live*, and some are *dead* (or *empty*); the evolution rule, executed simultaneously in all cells, is that a dead cell becomes live if it has exactly three live (cardinal or diagonal) neighbors, and a live cell stays live if and only if it has two or three live neighbors.

Iterating this rule gives rise to very complicated dynamics. Engineering patterns with interesting behaviors, and searching for such patterns by computer, has been an ongoing effort since the invention of the rule. For readers interested in delving into this world, we cite the very recent (and freely available) book [19] of Johnston and Greene. One result that

exemplifies the complexity of Game of Life is that it is *intrinsically universal* [10], meaning that Game of Life can simulate any two-dimensional cellular automaton f (including proper self-simulation), so that the states of f correspond to large blocks with special content, and one step of f is simulated in multiple steps of Game of Life.

Game of Life can be thought of as a mathematical *complex system*, namely it is a system where complex global behavior arises from interacting (simple) local rules. Such systems can be notoriously difficult to study. We can often use computer simulations to make empirical observations about typical and eventual behavior, but it can be very difficult to actually prove that a particular behavior persists on larger scales (even if it seems like its failure would require a massive conspiracy). Usually one can only successfully analyze systems that are very simple [13], or their behavior simulates a phenomenon that is mathematically well-understood, say of an algebraic [8] or number theoretic [20] nature, or the systems are specifically constructed for some purpose [23]. Due to intrinsic universality, it seems unlikely that Game of Life fits in any of these classes.

Indeed, for Game of Life, despite decades of study by enthusiasts, almost no non-trivial mathematical results exist that state limitations on its eventual behavior. In other words, as a dynamical system, we know very little about it. From computer simulations, one can conclude that Game of Life is highly "chaotic", and one can make educated guesses about things like the typical population density after a large number of iterations; however, it is very hard to make such claims rigorous. Rigorous results about Game of Life do exist, but they concern mostly the behavior of Game of Life on nice configurations (the engineering feats discussed above are of this type), and no known pattern behaves predictably in a general context; alternatively, they deal with one-step or static behavior [12].

In this paper, we study Game of Life through its *agars*, which are the Game of Life community's term for spatiotemporally periodic points. More specifically, we observe that a simple algorithm (essentially Wang's partial algorithm from [32]) can be used to find all agars with small enough periodicity parameters that have a unique chain of predecessors. We then study finite patches of these agars, and find some with interesting backwards forcing properties. Namely, these patterns behave deterministically in the (a priori nondeterministic) backwards dynamics of Game of Life. This intuitively allows us to study the "last iterations" of Game of Life (after an unknown number of steps), and leads to a wealth of results about how Game of Life behaves "in the limit".

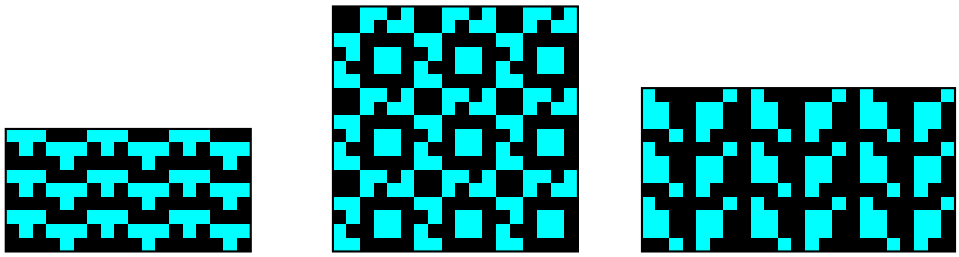
One practical difficulty is that the algorithm we use is not of the usual kind, but rather it is in the class $\mathrm{FP}^{\mathrm{NP}}$ of problems that are solvable in polynomial time with an NP oracle. Throughout this work, modern SAT solvers have constantly impressed and even humbled us by how freely they can be used as such oracles.[1] While their role is not very explicit in the final write-up of the paper, this work would not have been possible without them.

We do not expect the method of studying the eventual dynamics through self-enforcing patterns to be specific to Game of Life. Indeed, one can apply it directly to any cellular automaton rule (with any number of dimensions), and the idea can presumably be adapted to other systems as well. The reason we study a single example cellular automaton is that the results require us to find a "witness", usually a self-enforcing agar, and there is no *a priori* bound on how long this pattern-crunching will take – or whether it will succeed at all – for a particular rule. A single agar also tends to only work for a single rule or trivial

[1] For example, in our experience, general-purpose constraint-solvers and our own CA-specific solvers often fail even on the basic problem of finding a Game of Life preimage, while SAT solvers happily tell us, say, whether a preimage exists with particular constraints, and can find preimages for higher powers of Life.

modifications thereof. The choice of precisely Game of Life as the example rule to study is not mathematically motivated, it is simply a well-known simple rule that has already been studied extensively. Some of our programs are available on GitHub at [30], for readers interested in trying the methods out on other rules.

1.1 The protagonists



(a) Köynnös. **(b)** Kynnös. **(c)** Marching band.

Figure 1 Patches of the agars. A 3-by-3 grid of the repeating patterns is shown for each. Cyan cells are live.

We begin with a brief discussion of the agars that we use to prove our results. These will be explained in more detail in separate sections.

Figure 1a shows the pattern we call köynnös[2]. The infinite agar obtained by repeating this pattern infinitely in each direction has no preimage other than itself; we say it is *self-enforcing*. Up to symmetries there are exactly 11 self-enforcing agars of size 6×3. Köynnös has the special property that it is impossible to stabilize a finite difference to this configuration: if one modifies the agar in finitely many cells, the difference spreads at the speed of light (one column of cells per time step, which is the maximal speed at which information can be transmitted by Game of Life). We say it *cannot be stabilized from the inside*.

Figure 1b shows the pattern we call kynnös[3]. The corresponding infinite agar is again self-enforcing. Up to symmetries there are at least 52 self-enforcing agars of size 6×6 (we were unable to finish the search, so it is possible that more exist). Kynnös has two special properties. First, it contains a finite patch such that if a configuration has this patch in its image, then the configuration already had that patch in place, i.e. one cannot synthesize it from any other patch. Second, unlike köynnös, it can be stabilized from the inside.

Figure 1c shows the pattern we call marching band[4]. This agar has temporal period two. Its most important property is that an infinite south half-plane of this pattern must shrink if there is a difference on its border, meaning that in the nondeterministic inverse dynamics of Game of Life, an infinite south half-plane of this pattern "marches" to the north.

To find the marching band, we searched through all $w \times h$-patterns such that the corresponding agar with periods $(w, 0)$ and $(0, h)$ is temporally (exactly) t-periodic, for the parameter range $2 \leq w \leq 9$, $2 \leq h \leq 5$, $2 \leq t \leq 3$. There were no self-enforcing agars with temporal period 3 in this range, and there were exactly 14 self-enforcing agars with period 2. The marching band is the only one that has the marching property in any direction.

[2] Finnish for vine.
[3] Finnish for that which is tilled.
[4] English for marssiorkesteri.

1.2 Results

Denote by $g : \{0,1\}^{\mathbb{Z}^2} \to \{0,1\}^{\mathbb{Z}^2}$ the Game of Life cellular automaton, where dead cells are represented by 0 and live cells by 1. In this section, we list all our new technical contributions about g. The reader should consult Section 2 for precise definitions of terms used in this section. First, we solve the Generalized grandfather problem: for all $n \geq 0$, there exists a configuration that has an nth predecessor but not an $(n+1)$st one.

▶ **Theorem 1** (Generalized grandfather problem). *For each $n \geq 0$, there exists $x \in \{0,1\}^{\mathbb{Z}^2}$ with $g^{-n}(x) \neq \emptyset$ and $g^{-(n+1)}(x) = \emptyset$.*

The case of $n = 0$ (that g is not surjective) was resolved by R. Banks in 1971, only a year after the introduction of Game of Life. Conway stated the Grandfather problem, namely the case $n = 1$ of the above, in 1972, and promised \$50 in the *Lifeline* newsletter [31] for its solution. This stayed open until 2016, when the cases $n \in \{1, 2, 3\}$ were proved by the user mtve of the ConwayLife forum. We note (see Lemma 16 for the proof) that while Theorem 1 refers to infinite configurations, the analogous statements for finite patterns or finite-population configurations are equivalent to it. Cellular automata satisfying the conclusion of Theorem 1 are sometimes called "unstable" [24], though we avoid this terminology here, as "stable" has another meaning in Game of Life jargon.

More specifically, we prove the following two results, which strengthen Theorem 1 in different directions. The first result is proved using köynnös and is based on the fact it cannot be stabilized from the inside. The notation $g^{-n}(p)$ for a finite pattern p of shape $D \subset \mathbb{Z}^2$ stands for the set of patterns of shape $D + [-n, n]^2$ that evolve into p in n steps.

▶ **Theorem 1.1.** *There exists a polynomial time algorithm that, given $n \geq 0$ in unary, produces a finite pattern p with $g^{-n}(p) \neq \emptyset$ and $g^{-(n+1)}(p) = \emptyset$.*

The algorithm is very simple: change the value of one cell in the agar, apply the Game of Life rule n times, and pick the central $[-30-6n, 30+6n] \times [-27-8n, 27+8n]$-patch of the resulting configuration as p. An example with $n = 28$ is shown in Figure 2 (with insufficient padding: the periodic background should extend 164 cells further to the left and right, and 220 cells up and down).

Figure 2 A "level-29 orphan" obtained by perturbing köynnös: these angry deities could be found 28 seconds after the Big Bang, then went extinct.

The second result is proved using kynnös, and is based on the facts that kynnös admits a self-enforcing patch and that it can be stabilized from the inside. The following was first pointed out by Adam Goucher [16].

▶ **Theorem 1.2.** *For any large enough n, there exists an $n \times n$ pattern which appears in the kth image of Game of Life, but does not appear in its $(k+1)$st image, where $2^{n^2/368-O(n)} \leq k \leq 2^{n^2}$.*

This is (up to a suitable equivalence relation) the optimally slow growth rate for higher level orphans. The same idea can be used to obtain the following results about the limit set of Game of Life. Also called the eventual image, it is the set of configurations with arbitrarily long chains of predecessors. The language of the limit set refers to the set of finite patterns occurring in it.

▶ **Theorem 2.** *The limit set of Game of Life has PSPACE-hard language.*

The language might well be much harder. Even for one-dimensional cellular automata it can be Π_1^0-complete [6, 18]; we do not know if Game of Life reaches this upper bound.

We also obtain information about the symbolic dynamical nature of the limit set. A set of configurations is sofic if it can be defined by Wang tiles, or squares with colored edges: in a valid tiling of $\mathbb{Z}^2$, colors of adjacent edges are required to match, and the tiles can additionally be marked with 0 and 1 to project each valid tiling to a binary configuration. The set of those projections is called a sofic shift. Sofic systems form a large and varied class of subshifts, for example their one-dimensional projections can be essentially arbitrary (subject only to an obvious computability condition) [11, 1]. We show that the limit set of Game of Life cannot be defined by a tile set in this way.

▶ **Theorem 3.** *The limit set of Game of Life is not sofic.*

Besides illuminating the iterated images of Game of Life and its limit set, the self-enforcing kynnös patch itself solves a second open problem, namely the Unique father problem stated by John Conway in [31, 5]: is there a still life (a finite-population configuration that is a fixed point of g) whose only predecessor is itself, "with some fading junk some distance away not being counted"? We solve one interpretation of this problem.

▶ **Theorem 4** (Unique father problem)**.** *There exists a finite still life configuration x that contains a finite subpattern p such that every preimage of x also has subpattern p.*

One can also imagine stronger variants of the Unique father problem: for example, we could require p to contain all live cells of x, or all cells in their convex hull. These stay open.

Theorem 4 also tells us something about the dynamics of Game of Life *restricted to* its limit set, i.e. its limit dynamics. The chain-wandering property essentially means that there is a finite pattern that occurs in the limit set of Game of Life, but never returns to itself under the dynamics no matter how we fill the surrounding infinite plane. In fact, we are even allowed to completely rewrite the entire configuration on every step, apart from the domain of the pattern.

▶ **Theorem 5.** *Game of Life is chain-wandering on its limit set.*

Much is known about the kinds of things that can happen in Game of Life orbits, in particular it is well known that Game of Life is computationally universal and can simulate any cellular automaton. Nevertheless, to our knowledge all existing methods of simulating unbounded computation require the rest of the configuration to be empty (or at least stay out of the way). With our methods, we can enforce computations in a finite region (conditioned on its end state) even when it is completed into an infinite configuration by an adversary.

▶ **Theorem 6.** *The reachability problem of Game of Life is PSPACE-hard, i.e. given two finite patterns $p, q \in \{0,1\}^D$ whose domain D has polynomial extent, it is PSPACE-hard to tell whether there exists a configuration x with $x|D = p$ and $g^n(x)|D = q$ for some $n \geq 0$.*

Finally, the properties of the marching band's backwards dynamics imply that the limit set contains patterns that cannot be "glued" together too close: there are two $n \times n$ patterns such that no configuration of the limit set contains both of them separated by a distance less than $n/15$.

▶ **Theorem 7.** *For all large enough n there exist patterns $p, q \in \{0,1\}^{[0,n-1]^2}$ such that p and q appear in the limit set, but $p \sqcup \sigma_{(0,\lfloor n/15 \rfloor)}(q)$ does not.*

▶ **Corollary 8.** *The limit set of Game of Life is not block-gluing (thus has none of the gluing properties listed in [4]).*

1.3 Programs

Some of the programs we used can be found on GitHub at [30]. We have included Python scripts enumerating self-enforcing agars, and scripts checking the claimed properties of our three agars. In particular one can find implementations of Algorithms 1 and 2. The scripts use the PySAT [28] library to call the Minisat [26] SAT solver (the library supports many other solvers as well).

2 Definitions

Our intervals are discrete. To simplify formulas, we denote by

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix} = ([-a, b] \times [-c, d]) \setminus ([-e, f] \times [-g, h])$$

a rectangular discrete annulus when the second rectangle fits fully inside the first, that is, $-a \leq -e \leq f \leq b$ and $-c \leq -g \leq h \leq d$.

We assume some familiarity with topological and symbolic dynamics and give only brief definitions, see e.g. [22] for a basic reference. We denote by S a finite *alphabet*. A *configuration* or *point* is an element of $S^{\mathbb{Z}^d}$. More generally, a *pattern* (or sometimes *patch* in more informal contexts) is a function $p : \mathrm{dom}(p) \to S$, where $\mathrm{dom}(p) \subset \mathbb{Z}^d$ is the *domain* of p. If $S \subset \mathbb{N}$, then by $\sum p$ we denote the sum $\sum_{\vec{v} \in \mathrm{dom}(p)} p(\vec{v})$. For $\vec{v} \in \mathbb{Z}^d$, a pattern p and $D \subset \mathbb{Z}^d$, we write $q = p|D$ for the restriction $\mathrm{dom}(q) = D \cap \mathrm{dom}(p), q(\vec{v}) = p(\vec{v})$. A pattern is *finite* if its domain is, and a configuration is *finite* if its sum as a pattern is finite. If p, q are patterns with disjoint domains, define $p \sqcup q = r$ by $\mathrm{dom}(r) = \mathrm{dom}(p) \cup \mathrm{dom}(q), r|\mathrm{dom}(p) = p, r|\mathrm{dom}(q) = q$. The *extent* of a pattern is the minimal hypercube containing the origin and its domain. For two patterns, write $\mathrm{eq}(q, q')$ for the set of vectors $\vec{v} \in \mathrm{dom}(q) \cap \mathrm{dom}(q')$ such that $q(\vec{v}) = q'(\vec{v})$, and $\mathrm{diff}(q, q')$ for those that satisfy $q(\vec{v}) \neq q'(\vec{v})$. For computer science purposes, we note that patterns with polynomial extent have an efficient encoding as binary strings.

The *full shift* is the set of all configurations $S^{\mathbb{Z}^d}$ with the product topology (where S has the discrete topology), under the action of $\mathbb{Z}^d$ by homeomorphisms $\sigma_{\vec{v}}(x)_{\vec{u}} = x_{\vec{v}+\vec{u}}$ called *shifts*. We use the same formula to define $\sigma_{\vec{v}}(p)$ for patterns p (of course shifting the domain correspondingly). A pattern p defines a *cylinder* $[p] = \{x \in S^{\mathbb{Z}^d} \mid x|\mathrm{dom}(p) = p\}$. Cylinders defined by finite patterns are a base of the topology, and their finite unions are exactly the clopen sets. The *symbol partition* is the clopen partition $\{[s] \mid s \in S\}$ where s is

identified with the pattern $p : \{\vec{0}\} \to S$ with $p(\vec{0}) = s$. The space $S^{\mathbb{Z}^d}$ is homeomorphic to the Cantor space, and is metrizable. One possible metric is $\mathrm{dist} : (S^{\mathbb{Z}^2})^2 \to \mathbb{R}$, $\mathrm{dist}(x, y) = 2^{-\sup\{n \mid x|[-n,n]\times[-n,n]=y|[-n,n]\times[-n,n]\}}$ with $2^{-\infty} = 0$.

A *cellular automaton* (or *CA*) is a continuous self-map $f : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ that commutes with the shifts. The *neighborhood* is a set $N \subset \mathbb{Z}^d$ such that $f(x)_{\vec{0}}$ is determined by $x|N$; a finite neighborhood always exists by the Curtis-Hedlund-Lyndon theorem [17]. It is easy to show that there is always a unique *minimal neighborhood* under inclusion. A state $0 \in S$ is *quiescent* if $f(0^{\mathbb{Z}^d}) = 0^{\mathbb{Z}^d}$. A *subshift* is a closed subset X of $S^{\mathbb{Z}^d}$ invariant under shifts. Its *language* is the set of finite patterns p such that $[p] \cap X \neq \emptyset$, and we say these patterns *appear* or *occur* in the subshift. Patterns that do not appear in $f(S^{\mathbb{Z}^d})$ are usually called *orphans*. We say p is a *level-n orphan* if it appears in $f^{n-1}(S^{\mathbb{Z}^d})$ but not in $f^n(S^{\mathbb{Z}^d})$ (so the usual orphans are level-1). The *limit set* of a cellular automaton f is $\Omega(f) = \bigcap_n f^n(S^{\mathbb{Z}^d})$. It is a subshift invariant under f. A *subshift of finite type* is a subshift of the form $\bigcap \sigma_{\vec{v}}(C)$ where C is clopen. A *sofic shift* is a subshift which is the image of a subshift of finite type under a shift-commuting continuous function.

We are mainly interested in $d = 2$, $S = \{0, 1\}$, and the *Game of Life* cellular automaton $g : \{0, 1\}^{\mathbb{Z}^2} \to \{0, 1\}^{\mathbb{Z}^2}$ defined by

$$g(x)_{\vec{v}} = 1 \iff (x_{\vec{v}} = 0 \wedge \sum(x|\vec{v} + K) = 3) \\ \vee(x_{\vec{v}} = 1 \wedge \sum(x|\vec{v} + K) \in \{2, 3\}),$$

where $K = [-1, 1]^2 \setminus \{(0, 0)\}$.

A *fixed point* (of a CA f) is $x \in S^{\mathbb{Z}^d}$ such that $f(x) = x$. In the context of Game of Life these are also called *stable configurations* or *still lifes*. *Spatial* and *temporal* generally refer respectively to the $\mathbb{Z}^d$-action of shifts and the action of a CA. In particular a *spatially periodic point* is a configuration $x \in S^{\mathbb{Z}^d}$ which has a finite orbit under the shift dynamics, and *temporal periodicity* means $f^n(x) = x$ for some $n \geq 1$. Spatiotemporal periodicity means that both hold; in the Game of Life context spatiotemporal points are also called *agars*.

If $f : X \to X$ is a continuous function, an *ϵ-chain from x to y* is $x = x_0, x_1, \ldots, x_k = y$ with $k \geq 1$ such that $\mathrm{dist}(f(x_i), x_{i+1}) < \epsilon$ for $0 \leq i < k$. We say f is *chain-nonwandering* if for all $\epsilon > 0$ and $x \in X$ there is an ϵ-chain from x to itself; otherwise f is *chain-wandering*. (In the literature, chain-nonwandering is more commonly known as chain-recurrence, but both terms are logical.) We say f is *topologically transitive* if for all nonempty open sets U, V we have $f^n(U) \cap V \neq \emptyset$ for some n. It is *sensitive* (to initial conditions) if there exists $\epsilon > 0$ such that for all $x \in X$ and $\delta > 0$ there exists $y \in X$ with $\mathrm{dist}(x, y) < \delta$ and $n \in \mathbb{N}$ such that $\mathrm{dist}(f^n(x), f^n(y)) \geq \epsilon$. We say f has *dense periodic points* if its set of temporally periodic points is dense.

3 Proofs

We begin by introducing a formalism for forced cells in the preimages of a given pattern or configuration. The general topological idea is the following: if we have a zero-dimensional space X and a family of closed sets $\mathcal{I}$ which is closed under arbitrary intersections and contains the empty set, then to any continuous $f : X \to X$ we can associate a map $\hat{f} : \mathcal{I} \to \mathcal{I}$ by

$$\hat{f}(A) = \bigcap\{B \in \mathcal{I} \mid f^{-1}(A) \subset B\}. \tag{1}$$

We call this the *dual map* of f with respect to $\mathcal{I}$.

In our situation, $X = S^{\mathbb{Z}^d}$ and $f : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ is a cellular automaton. We define three families of subsets of $S^{\mathbb{Z}^d}$:

- $\mathcal{I}$ consists of the cylinders $[p] \subset S^{\mathbb{Z}^d}$ defined by all patterns p, plus the empty set $\emptyset$, which we denote by $\top$. The entire space $S^{\mathbb{Z}^d}$, which is the cylinder of the empty pattern, is denoted by $\bot$.
- $\mathcal{F} \subset \mathcal{I}$ consists of all cylinders $[p]$ defined by finite patterns p.
- $\mathcal{C} \subset \mathcal{I}$ consists of the singletons $[x] = \{x\}$ for full configurations $x \in S^{\mathbb{Z}^d}$.

Note that $\mathcal{F} \cap \mathcal{C} = \emptyset$. The family $\mathcal{F}$ is naturally stratified into finite subsets $\mathcal{F}_M = \{[p] \mid p \in S^M\}$, where $M \subset \mathbb{Z}^d$ ranges over finite sets. For a cylinder $[p] \in \mathcal{I}$, equation (1) defines $\hat{f}([p]) \in \mathcal{I}$ as the cylinder $[q]$, where q contains exactly those cells whose values are the same in all f-preimages of p, or $\top$ if p has no f-preimages (i.e. is an orphan). Also, $\hat{f}(\top) = \top$.

▶ **Example 9.** Consider $S = \{0, 1, 2\}$ and the cellular automaton $f : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ defined by

$$f(x)_0 = \begin{cases} 2, & \text{if } x_0 = 2, \\ \min(x_0, x_1), & \text{otherwise.} \end{cases}$$

The minimal neighborhood of f is $N = \{0, 1\}$. The pattern $p = 002$ of domain $\{0, 1, 2\}$ has preimages $f^{-1}(p) = \{0020, 0021, 0022, 1020, 1021, 1022\}$ of domain $\{0, 1, 2, 3\}$. Thus $\hat{f}([p]) = [q]$, where $q = 02$ has domain $\{1, 2\}$, since the values of these cells are the same in all preimages. The pattern $p' = 102$ has no preimages, so $\hat{f}([p']) = \top$.

We define a partial order on $\mathcal{I}$ by $[p] \leq [q]$ whenever $[q] \subset [p]$, and $\alpha \leq \top$ for all $\alpha \in \mathcal{I}$. The intuition is that $[p] \leq [q]$ corresponds to the pattern q specifying more cells than p, and thus containing more information. As the empty set $\top$ in a sense specifies the maximal amount of information – a contradiction – it is the largest element. Note that $\mathcal{C}$ consists of the maximal elements of $\mathcal{I} \setminus \{\top\}$.

We give $\mathcal{I}$ the topology with basis sets $U_p = \{\alpha \in \mathcal{I} \mid [p] \leq \alpha\}$ for $[p] \in \mathcal{F}$ as well as $\{\top\}$, making $\top$ an isolated point. This space is not Hausdorff (T_2), indeed it only satisfies the Kolmogorov (T_0) separation axiom. The induced topology on $\mathcal{C}$ is the standard compact Cantor topology, and $\mathcal{F}$ is a dense subset of $\mathcal{I} \setminus \{\top\}$. Every nonempty open set contains $\top$: "the contradiction is dense".

▶ **Lemma 10.** *The dual map $\hat{f} : \mathcal{I} \to \mathcal{I}$ is continuous.*

We are simply saying that if a (possibly infinite) pattern forces some particular value in some cell in the preimage, then actually some finite patch already forces it. The proof is a straightforward compactness argument.

Proof. Continuity at $\top$ is obvious. We show continuity at a cylinder $[p] \in \mathcal{I}$. Suppose first that $\hat{f}([p]) = [q]$, and let $[r] \in \mathcal{F}$ be such that $[q] \in U_r$. This means that p forces the pattern q in its f-preimages, and r is a finite subpattern of q. There exists a finite subpattern s of p that forces r, for otherwise we could take larger and larger subpatterns of p along with two preimages that disagree on $\mathrm{dom}(r)$, and in the limit obtain two preimages of p that disagree on $\mathrm{dom}(r)$. Hence $\hat{f}(U_s) \subset U_r$.

Suppose then that $\hat{f}([p]) = \top$, meaning that p is an orphan. It is well known that p contains a finite subpattern r that is also an orphan. Then $\hat{f}(U_r) = \{\top\}$. ◀

We list some other easy properties of $\hat{f}$. For $\alpha, \beta \in \mathcal{I}$ write $\alpha \parallel \beta$ for $\alpha \cap \beta \neq \top$. In the case of cylinders, this means that the corresponding patterns agree on the intersection of their domains. For a pattern p, write $f(p)$ for the pattern obtained by applying the local rule

of f (with the minimal neighborhood) in every position whose neighborhood is contained in the domain of p (and only those positions are included in the domain of $f(p)$). Note that with this definition $f([p]) \subset [f(p)]$, and the inclusion may be strict.

▶ **Lemma 11.**

- *$\mathcal{F} \cup \{\top\}$ is preserved under $\hat{f}$. Indeed, we have $\hat{f}(\mathcal{F}_M) \subset \mathcal{F}_{M+N} \cup \{\top\}$ where N is the minimal neighborhood of f.*
- *$\hat{f}$ is monotone, i.e. $\alpha \leq \beta$ implies $\hat{f}(\alpha) \leq \hat{f}(\beta)$.*
- *$\hat{f} \circ \hat{g} \leq \widehat{f \circ g}$ pointwise.*
- *for all $\alpha \in \mathcal{I}$, either $\hat{f}(\alpha) = \top$ or $\hat{f}(\alpha) = [p]$ with $[f(p)] \leq \alpha$; in particular $[f(p)] \parallel \alpha$ in the latter case.*
- *for all $\alpha \in \mathcal{I}$, we have $\hat{f}(\sigma_{\vec{v}}(\alpha)) = \sigma_{\vec{v}}(\hat{f}(\alpha))$.*

The next few results refer to $\mathrm{FP}^{\mathrm{NP}}$, the class of function problems solvable in deterministic polynomial time with the help of an oracle that can solve an NP decision problem in one step. Of course, the oracle can be invoked repeatedly to construct NP certificates in a polynomial number of steps. This class naturally captures the method of using SAT solvers as black boxes to compute preimages of finite patterns.

▶ **Lemma 12.** *For a fixed CA f, given $p \in \mathcal{F}$, the image $\hat{f}([p])$ can be computed in $\mathrm{FP}^{\mathrm{NP}}$. It remains computable if f is also given as input.*

Proof. Since $\hat{f}(\mathcal{F}_M) \subset \mathcal{F}_{M+N}$, we only need to determine which coordinates in $M + N$ are forced in preimages. This requires at most $1 + |M + N|$ calls to an NP oracle: one to request a preimage, and for each $\vec{v} \in M + N$, one to request a pair of preimages which differ at $\vec{v}$. ◀

The proof above is the easiest way to get the theoretical result, but for practical purposes we give Algorithm 1, which tends to find the $\hat{f}$-image much quicker (and is just as quick to implement). It is written for an "incremental oracle", meaning we can only *add* constraints to it (represented by the set F) when we make a new query. In this case, we compute a single f-preimage q of the input pattern p, and then compute additional preimages that differ from q on progressively smaller sets of cells. Modern SAT solvers tend to support such incremental access – of course, on the side of theory it is easy to see that the class $\mathrm{FP}^{\mathrm{NP}}$ is the same whether or not queries are restricted to be incremental.

Algorithm 1 Finding $\hat{f}([p])$ for a finite pattern $p \in S^M$.

```
function HatCA(f, p)
    Let O ← NP oracle.
    if O finds a pattern q ∈ f^{-1}(p) then
        Let D ← M + N.
        Let F ← {(q, D)}.
        loop
            if O finds a pattern q' ∈ f^{-1}(p) with q'|E ≠ r|E for all (r, E) ∈ F then
                Let D ← D ∩ eq(q, q').
                Let F ← F ∪ {(q', D)}
            else
                return q|D
    else
        return ⊤
```

Our results rely on the existence of patterns p that force large patterns into their preimages, meaning that $\hat{f}([p])$ is large in the sense of $\leq$. We say a pattern p is *self-enforcing* under f if $[p] \leq \hat{f}([p])$. In a slight abuse of terminology, we also say that a temporally t-periodic configuration $x \in S^{\mathbb{Z}^d}$ is *self-enforcing* if $\widehat{(f^t)}([x]) = [x]$. A self-enforcing agar is then a spatially and temporally periodic configuration that has a unique chain of preimages.

▶ **Lemma 13.** *The set of all pairs (f, x) such that f is a CA on $S^{\mathbb{Z}^d}$ and $x \in S^{\mathbb{Z}^d}$ is a self-enforcing agar is recursively enumerable.*

Proof. Let $x \in S^{\mathbb{Z}^d}$ be a self-enforcing agar with spatial periods $n_1\vec{e}_1, \ldots, n_d\vec{e}_d$ and temporal period t. Denote the iterated CA by $h = f^t$, and let $B = [0, n_1 - 1] \times \cdots \times [0, n_d - 1]$. We need to find a certificate for $\hat{h}([x]) = [x]$. For this, observe that by continuity of $\hat{h}$ there is a finite subpattern p of x such that $\hat{h}([y]) \geq [x|B]$ for every configuration $y \in [p]$. This implies $\hat{h}([p]) \geq [x|B]$. By Lemma 12, this latter inequality can be checked in $\mathrm{FP}^{\mathrm{NP}}$.

We claim that p is a certificate that x is a self-enforcing agar. Let $\vec{v} \in V = \langle n_1\vec{e}_1, \ldots, n_d\vec{e}_d \rangle$ be arbitrary. We compute

$$\hat{h}([x]) = \sigma_{\vec{v}}(\hat{h}([x])) \geq \sigma_{\vec{v}}(\hat{h}([p])) \geq \sigma_{\vec{v}}([x|B]) = [x|\vec{v} + B],$$

and since $\mathbb{Z}^d = \bigcup_{\vec{v} \in V}(\vec{v} + B)$, this implies $\hat{h}([x]) = [x]$. ◀

▶ Remark 14. The semi-algorithm described in the proof is not very practical: given an agar, we have no information about how large the certificate could be, so for each agar we either need to guess some certificate size, or we have to keep trying increasingly large certificates. Our implementation runs in parallel a search for other periodic preimages for the agar – if such a preimage exists, then clearly the agar does not enforce itself, and we can stop looking for a certificate. We omit the pseudocode.

Most agars in the range we searched were either self-enforcing or had another periodic preimage. There exist two-dimensional cellular automata whose set of self-enforcing agars is not computable (by a relatively simple reduction from the tiling problem of Wang tiles [2], which we omit), but we do not know whether this is the case for Game of Life.

Say a pattern $p \subset S^M$ is *locally fixed* for the CA f if there exists a pattern $q \in S^{M+N}$ (where N is the minimal neighborhood of f) such that $p = q|M = f(q)|M$.

▶ **Lemma 15.** *For every CA f on $S^{\mathbb{Z}^d}$, every locally fixed pattern $p \in S^M$ admits a unique maximal self-enforcing subpattern. For a fixed CA g, given p, a vector $\vec{v} \in \mathbb{Z}^d$ and $n \geq 1$ in unary, it can be computed in $\mathrm{FP}^{\mathrm{NP}}$ for the CA $f = \sigma_{\vec{v}} \circ g^n$.*

Proof. Since p has finitely many subpatterns and the empty pattern is trivially self-enforcing, p admits at least one maximal self-enforcing subpattern. If $D, D' \subset M$ satisfy $\hat{f}([p|D]) \geq [p|D]$ and $\hat{f}([p|D']) \geq [p|D']$, then $\hat{f}([p|D \cup D']) \geq [p|D \cup D']$ by monotonicity of $\hat{f}$. Thus $q = p|E$ for $E = \bigcup\{D \subset M \mid \hat{f}([p|D]) \geq [p|D]\}$ is the unique self-enforcing subpattern.

Then fix g, and let p, $\vec{v}$ and n be given. We apply Algorithm 2 to the CA $f = \sigma_{\vec{v}} \circ g^n$. On each iteration of the loop, the algorithm replaces p with the maximal subpattern forced by p (here we use the fact that p is locally fixed). Since q is a subpattern forced by itself, by monotonicity it is also forced by each of these subpatterns, and thus remains a subpattern on each iteration. Since q is maximal and p has finitely many subpatterns, the algorithm eventually converges on q.

Finally, Algorithm 2 is in $\mathrm{FP}^{\mathrm{NP}}$, since the number of iterations of the loop is at most $|M|$, and $\textsc{HatCA}(f, p)$ is in $\mathrm{FP}^{\mathrm{NP}}$ with respect to these parameters. ◀

Algorithm 2 Finding the maximal self-enforcing subpattern of a locally fixed pattern $p \in S^M$.

```
function SelfEnforcingSubpattern(p)
    loop
        Let q ← HatCA(f, p)|M.
        if q = p then
            return q
        else
            Let p ← q.
```

To conclude this section, we show that in the formulation of the Generalized grandfather problem (which we prove as Theorem 1), it makes no difference whether we consider unrestricted configurations, finite configurations or finite patterns. This is well-known in cellular automata theory.

▶ **Lemma 16.** *Let $f : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ be a cellular automaton with a quiescent state $0 \in S$, and $n \in \mathbb{N}$. The following are equivalent:*

1. *There exists a finite configuration $x \in S^{\mathbb{Z}^d}$ such that $f^{-n}(x)$ contains a finite configuration, and $f^{-(n+1)}(x) = \emptyset$.*
2. *There exists $x \in S^{\mathbb{Z}^d}$ such that $f^{-n}(x) \neq \emptyset$ and $f^{-(n+1)}(x) = \emptyset$.*
3. *There exists a finite pattern p such that $f^{-n}(p) \neq \emptyset$ and $f^{-(n+1)}(p) = \emptyset$.*

Proof. The implication 1 $\implies$ 2 is clear, and 2 $\implies$ 3 is the classical compactness argument that we used to prove Lemma 10.

We prove 3 $\implies$ 1. Take an arbitrary $q \in f^{-n}(p)$, and complete it into a finite configuration $y \in S^{\mathbb{Z}^d}$ by setting $y_{\vec{v}} = 0$ for all $\vec{v} \in \mathbb{Z}^d \setminus \mathrm{dom}(q)$. Then $x = f^n(y)$ satisfies the conditions of item 1: $f^{-n}(x)$ contains the finite configuration y, while $f^{-(n+1)}(x) = \emptyset$ since x contains an occurrence of p. ◀

3.1 Köynnös

We begin by studying köynnös, which we recall is obtained from the 6×3 pattern

$$P = \begin{matrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{matrix}$$

by repeating P horizontally and vertically to define an infinite 6×3-periodic configuration $x^P \in \{0,1\}^{\mathbb{Z}^2}$. Observe that every 0 in x^P is surrounded by exactly four 1s, and every 1 by exactly three 1s. Thus we have $g(x^P) = x^P$, so that x^P is indeed an agar. Moreover, we claim that x^P has no other predecessors than itself: $g^{-1}(x^P) = \{x^P\}$. This is due to the following lemma.

▶ **Lemma 17.** *Let x be in the spatial orbit of köynnös. Then $\hat{g}(x|[-12,17] \times [-12,14]) \geq x|[-8,13] \times [-9,10]$.*

Proof. Applying Algorithm 1 to $\sigma_{\vec{v}}(x^P)|[-12,17] \times [-12,14]$ for all $\vec{v} \in [0,5] \times [0,2]$ gives the result. The intersection of the domains of the patterns $\hat{g}(x|[-12,17] \times [-12,14])$ for such $x = \sigma_{\vec{v}}(x^P)$ is shown in Figure 3, and clearly contains the rectangle $[-8,13] \times [-9,10]$. ◀

Put concretely, the lemma states that if R is a periodic continuation of P of size 30×27 and Q is its predecessor, then P must occur at the center of Q (and indeed many more cells are forced, even beyond what we state in the lemma). This is indeed a certificate for x^P

being a self-enforcing agar, as in the proof of Lemma 13: for any predecessor $y \in g^{-1}(x^P)$ and cell $\vec{v} \in \mathbb{Z}^d$, Lemma 17 gives $\sigma_{\vec{v}}(y)|[-8,13] \times [-9,10] = \sigma_{\vec{v}}(x^P)|[-8,13] \times [-9,10]$, so in particular $y_{\vec{v}} = x^P_{\vec{v}}$.

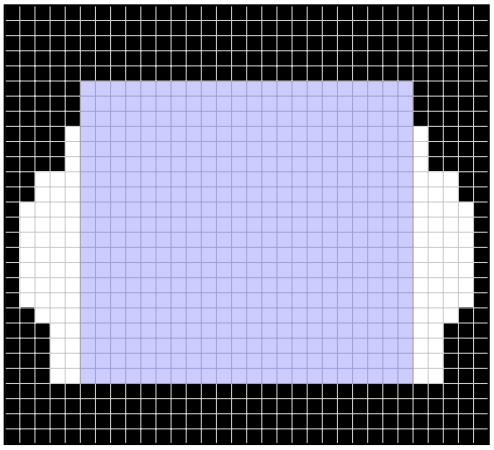


Figure 3 The intersection of the domains of $\hat{g}(x|[-12,17] \times [-12,14])$ for x in the spatial orbit of köynnös, drawn in white inside $[-13,18] \times [-13,15]$. The area $[-8,13] \times [-9,10]$ is highlighted in blue.

As a corollary of Lemma 17, finite perturbations of x^P can never be erased by g. We prove a stronger claim: all finite perturbations spread to the left and right at a speed of one column per time step. In particular, köynnös cannot be stabilized from the inside. We note that, as the agar kynnös studied in the next section does not possess this property, we cannot use it to prove Theorem 1.1, at least with the same method.

▶ **Lemma 18.** *Consider a rectangle $R = [-n_W, n_E] \times [-n_S, n_N]$ and the surrounding annulus $A = \left[\begin{smallmatrix} n_W+1 & n_E+1 & n_S+1 & n_N+1 \\ n_W & n_E & n_S & n_N \end{smallmatrix}\right]$ of thickness 1. Let p be a pattern such that the domain of $g(p)$ contains $A \cup R$, and suppose $p|A = g(p)|A = x^P|A$. If $\mathrm{diff}(x^P, g(p)) \cap R \subset [a,b] \times \mathbb{Z}$, then $\mathrm{diff}(x^P, p) \cap R \subset [a+1, b-1] \times \mathbb{Z}$.*

Note that we may have $b - a \leq 1$, in which case the conclusion becomes $\mathrm{diff}(x^P, p) \cap R = \emptyset$, or equivalently, $x^P|R = p|R$.

Proof. Since the orbit of köynnös and g are left-right symmetric, it is enough to prove that $\mathrm{diff}(x^P, p) \cap R \subset (-\infty, b-1]$. We prove the contrapositive: suppose there exists $(i,j) \in \mathrm{diff}(x^P, p) \cap R$ for some $i \geq b$, and let i be maximal. We split into cases based on the congruence class of i modulo 6, that is, the column of P that i lies in. Note that the bottom left cell of P is at the origin in x^P, and the domain of P is the rectangle $[0,5] \times [0,2]$. If $i \in \{0,2,3\} + 6\mathbb{Z}$, we choose j as maximal, and otherwise we choose it as minimal.

We handle the case $i \in 2 + 6\mathbb{Z}$, the others being similar or easier. If $j \in 3\mathbb{Z}$, then $p_{(i+1,j+1)} = x^P_{(i+1,j+1)} = 1$ has four other 1s in its neighborhood, and becomes 0 in $g(p)$. If $j \in 1 + 3\mathbb{Z}$, then $p_{(i+1,j)} = x^P_{(i+1,j+1)} = 1$ has four or five other 1s in its neighborhood, and becomes 0 in $g(p)$. If $j \in 2 + 3\mathbb{Z}$, then $p_{(i+1,j+1)} = x^P_{(i+1,j+1)} = 0$ has three 1s in its neighborhood, and becomes 1 in $g(p)$. In each case $\mathrm{diff}(x^P, g(p))$ intersects $\{i+1\} \times \mathbb{Z}$. ◀

For any $D \subset \mathbb{Z}^2$, the subpattern of köynnös of shape $D + [0,29] \times [0,26]$ forces the subpattern of shape $D + [4,25] \times [3,22]$ to occur in its g-preimage, by Lemma 17. By Lemma 18, we force more: a non-köynnös area inside a hollow patch of köynnös expands horizontally under g, so under $\hat{g}$ the horizontal extent of the hole must shrink. We do not give a precise statement for this general fact, and only apply the lemma in the case of annuli.

▶ **Lemma 19.** *Let x be in the orbit of köynnös. Suppose the following inequalities hold:*

$$m_W - n_W \geq 30, m_E - n_E \geq 30, m_S - n_S \geq 27, m_N - n_N \geq 27.$$

Denote $Q = x|\left[\begin{smallmatrix} m_W & m_E & m_S & m_N \\ n_W & n_E & n_S & n_N \end{smallmatrix}\right]$. If $n_E + n_W \geq 2$, then

$$\hat{g}(Q) \geq x|\begin{bmatrix} m_W - 4 & m_E - 4 & m_S - 3 & m_N - 4 \\ n_W - 1 & n_E - 1 & n_S + 4 & n_N + 3 \end{bmatrix}$$

while if $n_E + n_W \in \{0, 1\}$ we have

$$\hat{g}(Q) \geq x|[-(m_W - 4), m_E - 4] \times [-(m_S - 3), m_N - 4]$$

One may consider the latter case a special case of the former: there too, the hole shrinks horizontally by two steps, and since its width is at most two it disappears.

Proof. The inequalities simply state that the annulus Q is thick enough that each of its cells is part of a 30×27 rectangle contained in Q. From Lemma 17, we get $\hat{g}(Q) \geq x|A$, where $A = \left[\begin{smallmatrix} m_W - 4 & m_E - 4 & m_S - 3 & m_N - 4 \\ n_W + 4 & n_E + 4 & n_S + 4 & n_N + 3 \end{smallmatrix}\right]$ is a slightly thinner annulus. If there is no g-preimage for Q, then $\hat{g}(Q) = \top$ and we are done. Suppose then that it has a preimage R. Since $R \geq \hat{g}(Q) \geq x|A$, both Q and R agree with x on the thickness-1 annulus $\left[\begin{smallmatrix} n_W + 5 & n_E + 5 & n_S + 5 & n_N + 4 \\ n_W + 4 & n_E + 4 & n_S + 4 & n_N + 3 \end{smallmatrix}\right] \subset A$. Lemma 18 implies that R agrees with x on $\left[\begin{smallmatrix} m_W - 4 & m_E - 4 & m_S - 3 & m_N - 4 \\ n_W - 1 & n_E - 1 & n_S + 4 & n_N + 3 \end{smallmatrix}\right]$, as claimed. ◀

We now prove Theorem 1.1, and thus give the first proof of Theorem 1. In fact, we give a simple formula that produces configurations that have an nth preimage, but no $(n+1)$st one.

▶ **Lemma 20.** *Let x be in the orbit of köynnös, and suppose $\emptyset \neq \operatorname{diff}(y, x) \subset B = [0, a] \times [0, n]$ where $a \in \{0, 1\}$. Then*

$$p = g^k(y)|[-30 - 6k, 30 + a + 6k] \times [-27 - 8k, 27 + n + 8k]$$

appears in the kth image of g, but not in the $(k+1)$st.

Proof. By definition, p appears in the kth image of g. It suffices to show its $\hat{g}^{k+1}$-image is $\top$. Namely, we then have $\widehat{g^{k+1}}(p) \geq \hat{g}^{k+1}(p) = \top$ by Lemma 11, which means precisely that p has no g^{k+1}-preimage.

Let q be the restriction of p to

$$\begin{bmatrix} 30 + 6k & 30 + a + 6k & 27 + 8k & 27 + n + 8k \\ k & a + k & k & n + k \end{bmatrix}.$$

Observe that q agrees with x because g has radius 1, so by Lemma 19 and induction, we can deduce that

$$\hat{g}^j(q) \geq x|\begin{bmatrix} 30 + 6k - 4j & 30 + a + 6k - 4j & 27 + 8k - 3j & 27 + n + 8k - 4j \\ k - j & a + k - j & k + 4j & n + k + 3j \end{bmatrix}$$

for all $j \leq k$. This is because for $j \leq k - 1$ we have

$$\begin{aligned} 30 + 6k - 4j - (k - j) &\geq 30, & 30 + a + 6k - 4j - (a + k - j) &\geq 30, \\ 27 + 8k - 3j - (k + 4j) &\geq 27, & 27 + n + 8k - 4j - (n + k + 3j) &\geq 27, \end{aligned}$$

and thus we can inductively apply the lemma. But in

$$\hat{g}^k(q) \geq x|\begin{bmatrix} 30 + 2k & 30 + a + 2k & 27 + 5k & 27 + n + 4k \\ 0 & a & 5k & n + 4k \end{bmatrix}$$

the annulus still has sufficient thickness (i.e. the inequalities still hold for $j = k$), so we can apply the second case of the lemma to get

$$\hat{g}^{k+1}(q) \geq x|[-(26+2k), 26+2k] \times [-(24+5k), 23+n+4k] = r.$$

By Lemma 11 we have $\hat{g}^{k+1}(p) \geq \hat{g}^{k+1}(q) \geq r$, and by the same lemma we either have $\hat{g}^{k+1}(p) = \top$ (as desired), or

$$g^{k+1}(\hat{g}^{k+1}(p)) \leq g^{k+1}(\widehat{g^{k+1}}(p)) \parallel p.$$

But since the speed of light is 1 and x is a fixed point, we have

$$g^{k+1}(r) \geq x|[-(25+k), 25+k] \times [-(23+4k), 22+n+3k] \geq x|[-k, a+k] \times [-k, n+k]$$

and $x|[-k, a+k] \times [-k, n+k] \parallel p$. But Lemma 18 applied k times to y implies that $\mathrm{diff}(x, g^k(y))$, and thus $\mathrm{diff}(x, p)$, intersects $[-k, a+k] \times [-k, n+k]$, a contradiction. Thus we indeed must have $\hat{g}^{k+1}(p) = \top$. ◀

3.2 Kynnös

Denote by

$$Q = \begin{array}{cccccc} 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

the fundamental domain of kynnös, and by $x^Q \in \{0,1\}^{\mathbb{Z}^2}$ the associated 6×6-periodic configuration with $g(x^Q) = x^Q$. The following lemma states that it contains a self-enforcing patch (it is essentially a more precise stement of Theorem 4).

▶ **Lemma 21.** *There is a finite set $D \subset \mathbb{Z}^2$ such that $p = x^Q|D$ satisfies $\hat{g}(p) = p$. Furthermore, there is a finite-support configuration $x \in [p]$ with $g(x) = x$.*

The patch p is shaped like a 22×28 rectangle with 8 cells missing from each corner. It is depicted in Figure 4, together with the still life x containing it. The patch was found by simply applying the function of Algorithm 2 to the 70×70-patches of the agars we found during our searches. Kynnös was the first configuration that yielded a nonempty self-enforcing patch, which we then optimized to its current size. This lemma directly implies Theorem 4, and almost directly Theorem 5.

Proof of Theorem 5. Let y be the finite-support configuration obtained by taking x from the previous lemma and adding a glider that is just about to hit the kynnös patch. It can be checked by simulation that the patch can be annihilated this way. Observe that y is in the limit set $\Omega(g)$: simply shoot the glider from infinity. If $\epsilon > 0$ is very small, in any ϵ-chain starting from y we see the patch destroyed. It is impossible to reinstate it, as the existence of a first step in the chain where it appears again contradicts Lemma 21. ◀

As stated, kynnös can be stabilized from both inside and outside. Figure 5 shows a still life configuration containing a "ring" of kynnös with a hole of 0-cells inside it. From the figure it is easy to deduce the existence of such rings of arbitrary size and thickness.

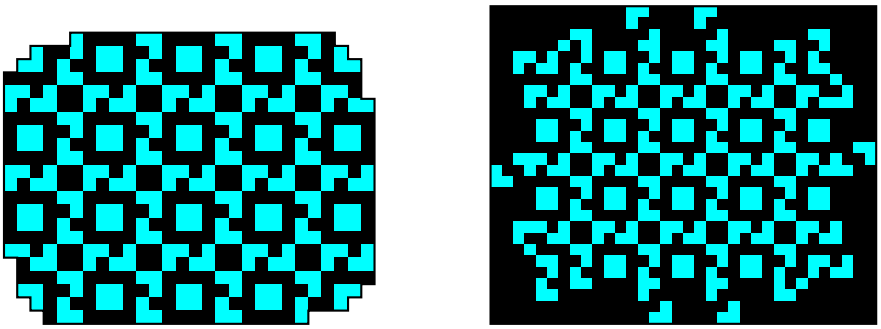


Figure 4 A self-enforcing patch of kynnös and a still life containing it. The still life has the minimal number of live cells, 306, of any still life containing the patch. The number was minimized by Oscar Cunningham. [7]

Note that if the ring is at least 22 cells thick, then its interior is completely surrounded by a ring-shaped self-enforcing pattern consisting of translated, rotated and partially overlapping copies of the 28×22 self-enforcing patch, through which no information can pass without destroying it forever. If we then replace the empty cells inside the ring with an arbitrary pattern, the resulting finite pattern P occurs in the limit set $\Omega(g)$ if and only if the interior pattern evolves periodically under g. Namely, if the pattern occurs in $\Omega(g)$, then it has an infinite sequence of preimages, each of which must contain the self-enforcing kynnös ring. The interior has a finite number of possible contents (2^m for an interior of m cells), so it must evolve into a periodic cycle, of which P is part. From this idea, and some engineering with gadgets found by other researchers and Life enthusiasts, we will obtain Theorems 1.2, 2 and 3. The first one was essentially proved by Adam Goucher [16]. Note the difference between these rings and the köynnös annuli of Section 3.1: the latter force strictly smaller versions of themselves in their preimages, and do not admit nontrivial periodically evolving interiors.

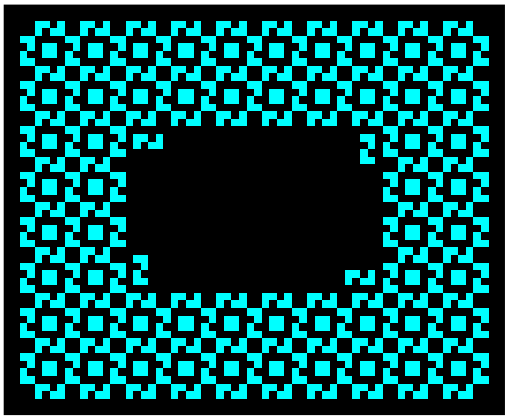


Figure 5 A stable ring of kynnös.

Proof of Theorem 1.2. Given integers $k, m \geq 1$ with k odd, we construct a configuration $x \in \{0,1\}^{\mathbb{Z}^2}$ such that the support of $g^n(x)$ is contained in $[0, 32k+73] \times [0, 46m]$ for all $n \geq 0$, and $g^{48 \cdot 4^{(2k+1)m}}(x)$ is not g-periodic. When the support of $g^{48 \cdot 4^{(2k+1)m}}(x)$ is surrounded by a kynnös ring of width 22, the resulting pattern has a $48 \cdot 4^{(2k+1)m}$th preimage, but not arbitrarily old preimages. If we choose $k = 23s$ and $m = 16s$ for some $s \geq 0$, the resulting

pattern has size $(736s + O(1)) \times (736s + O(1))$, and the chain of preimages has length $48 \cdot 4^{736s^2+16s}$. Choosing $s = n/736 - O(1)$ yields the lower bound, and the upper bound is the trivial one (even ignoring the fact we do not modify the boundaries).

The main components of x are the *period-48 glider gun* [27], which produces one glider every 48 time steps, and the *quadri-snark* [29], which emits one glider at a 90 degree angle for every 4 gliders it receives. The support of x consists of a single period-48 gun aimed at a sequence of $(2k+1)m$ quadri-snarks, each of which receives the gliders the previous one emits. They effectively implement a quaternary counter with values in $[0, 4^{(2k+1)m} - 1]$. The glider emitted by the final quasi-snark will collide with the kynnös ring, ensuring that the pattern right before the impact does not occur in the limit set $\Omega(g)$, but has a chain of preimages of length at least $48 \cdot 4^{(2k+1)m}$. An example pattern and a schematic for $m = n = 2$ are given in Figure 6. It is easy to extrapolate to arbitrary $m, n \geq 1$ from the figure. ◀

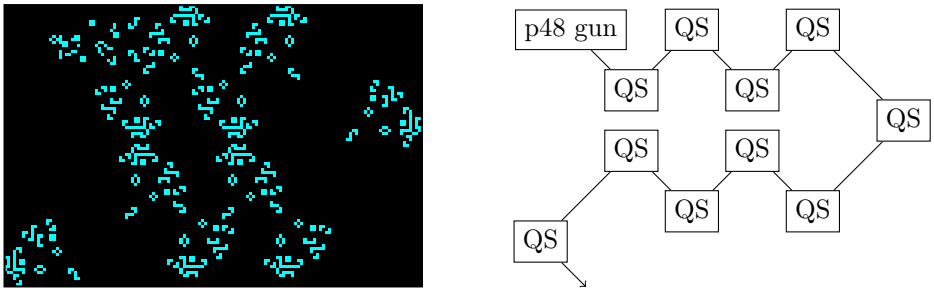


Figure 6 A configuration corresponding to $k = m = 2$ in the proof of Theorem 1.2.

To implement more complex Life patterns with desired properties, we use the fact that Life is *intrinsically universal*, that is, capable of simulating all $\mathbb{Z}^2$ cellular automata. Formally, for any other CA $f : \Sigma^{\mathbb{Z}^2} \to \Sigma^{\mathbb{Z}^2}$, there are numbers $K, T \geq 1$ and an injective function $\tau : \Sigma \to \{0,1\}^{K \times K}$ such that for all configurations $x \in \Sigma^{\mathbb{Z}^2}$ we have $g^T(\tau(x)) = \tau(f(x))$, where τ is applied cellwise in the natural way. We use the simulation technique of [10], which allow us to easily simulate patterns with *fixed boundary conditions*. This means that any rectangular pattern $R \in \Sigma^{a \times b}$ can be simulated by a finite-support configuration of g in such a way that simulated cells whose f-neighborhood is not completely contained in the rectangle $[0, a-1] \times [0, b-1]$ are forced to retain their value.

Proof of Theorem 2. Let $L \subset \{0,1\}^*$ be a PSPACE-hard language decidable in linear space, such as TQBF. Define a Turing machine M as follows. Given input $w \in \{0,1\}^*$, M determines whether $w \in L$ using at most $|w|$ additional tape cells and without modifying w. If $w \in L$, then it erases the additional tape cells and returns to its initial state, thus looping forever. If $w \notin L$, then M stays in a rejecting state forever. We simulate M by a cellular automaton f in a standard way: each cell is either empty, or contains a tape symbol and possibly the state of the computation head.

Next, we simulate the CA f by g as described above. Given a word $w \in \{0,1\}^*$, let $P(w)$ be the pattern corresponding to a simulated initial configuration of M on input w with $|w|$ additional tape cells and fixed boundary conditions, surrounded by a kynnös ring. If $w \in L$, then $P(w)$ occurs in the limit set $\Omega(g)$, since it can be completed into a g-periodic configuration in which the simulated M repeatedly computes $w \in L$. If $w \notin L$, then $P(w)$ does not occur in $\Omega(g)$, since the interior of the ring eventually evolves into a simulated configuration with M in a rejecting state, never returning to $P(w)$. ◀

Extending $P(w)$ by zeroes on all sides (resp. repeating it periodically), we obtain that it is PSPACE-hard whether a given finite-support configuration (resp. periodic configuration) appears in the limit set.

Proof of Theorem 6. Let L be as in the previous proof, and let M be a Turing machine that, on input $w \in \{0,1\}^*$, decides $w \in L$ using no additional tape cells. Then M erases the entire tape and enters an accepting or rejecting state depending on the result of the computation. We simulate M by g as in the previous proof. Given $w \in \{0,1\}^*$, let p be the pattern corresponding to a tape of M containing w and an initial state, and q the one corresponding to $|w|$ blank tape cells and an accepting state of M, both surrounded by a ring of kynnös of the same dimensions. Then q is reachable from p if and only if $w \in L$: if q is to be reached, the ring of p must stay intact, enclosing a correct simulation of M. ◀

Of course, again by extending the resulting patterns by 0-cells (resp. repeating them periodically), we obtain PSPACE-hardness of reachability between two given finite-support (resp. periodic) configurations, i.e. given the full descriptions of two configurations $x, y \in \{0,1\}^{\mathbb{Z}^2}$, the question of whether $g^n(x) = y$ for some $n \geq 0$. However, this reachability problem is in fact even Σ_1^0-complete (resp. PSPACE-complete) directly by intrinsic universality. For the case of finite configurations, one needs a variant of intrinsic universality where the zero state of an arbitrary cellular automaton is represented by an all-zero pattern; such a variant was proved in [15].

Proof of Theorem 3. Let M be a two-dimensional Turing machine whose tape alphabet has two distinguished values, denoted a and b. When M is initialized on a rectangular tape containing only as and bs, it repeatedly checks whether its left and right halves are equal, destroying the tape if they are not. We again simulate M by a CA f, and then f by g. Then a simulated rectangular tape with the head of M in its initial state, surrounded by a kynnös ring, is in $\Omega(g)$ if and only if the two halves of the tape are equal.

It was proved in [21] that for all sofic shifts $X \subset S^{\mathbb{Z}^2}$ there exists an integer $C > 1$ with the following property. For all $n \geq 1$ and configurations $x^1, \dots, x^{C^n} \in X$, there exist $i \neq j$ such that the configuration $y = (x^i|[0,n-1]^2) \sqcup (x^j|\mathbb{Z}^2 \setminus [0,n-1]^2)$ is in X. Assuming for a contradiction that $\Omega(g)$ is sofic, consider the configurations $x(P) \in \Omega(g)$ for $P \in \{a,b\}^{n \times n}$ that contain a kynnös ring and a simulated tape of M with two identical P-halves. Based on the above, when n is large enough that $2^{n^2} > C^{Kn}$, we can swap the right half of one $x(P)$ with that of another to obtain a configuration $y \in \Omega(g)$ containing a simulated tape of M with unequal halves inside a kynnös ring, a contradiction. ◀

We remark that a weaker version of Theorem 1.2 (where 1/368 is replaced by a much smaller, or even implicit, constant) could also be proved by intrinsic universality.

3.3 The marching band

Let $h = g^2$. Denote by

$$R = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{matrix}$$

the fundamental domain of the marching band, and by $x^R \in \{0,1\}^{\mathbb{Z}^2}$ the associated 8×4-periodic configuration with $h(x^R) = x^R$. The following is proved just like Lemma 17. Note that the forced region extends outside the original pattern.

▶ **Lemma 22.** *Let x be in the spatial orbit of the marching band. Then $\hat{h}(x|[0,47]\times[0,43]) \geq x|[10,29]\times[-1,44]$.*

Proof of Theorem 7. Let x be in the orbit of x^R, and let $p = x|[-a,b]\times[-c,d]$. By the previous lemma, as long as $a+b \geq 48$ and $c+d \geq 44$ we have $\hat{h}(p) = x|[-(a-10),b-18]\times[-(c+1),d+1]$. Iterating this we get

$$\hat{h}^n(x|[-10n,18n+47]\times[0,43]) \geq x|[0,47]\times[-n,n+43].$$

Denote $S = [-10n,18n+47]\times[0,43]$. Let $P = x|S$ and $Q = \sigma_{\vec{u}}(x)|\vec{v}+S$ for some $\vec{u}\in\mathbb{Z}^2$ and $\vec{v} = (0,2n)$. Both patterns appear in the limit set of g, since they are extracted from a fixed point of $h = g^2$. Observe that since the domains of $\hat{h}^n(P)$ and $\hat{h}^n(Q)$ intersect, we can pick the shift $\vec{u}$ so that one of the forced bits is different in some position in $\hat{h}^n(P)$ and $\hat{h}^n(Q)$, which clearly means $\hat{h}^n(P\sqcup Q) = \top$.

Now, P and Q each fit inside a $29n\times 29n$ rectangle (if $n\geq 47$), and the patterns cannot be glued in the limit set with gluing distance at most $2n$, since the glued pattern should have an nth h-preimage. This gives the statement. ◀

4 Chaotic conclusions

There are several definitions of topological chaos. We refer the reader to [3] for a survey. Briefly, a system is called Auslander-Yorke chaotic if it is topologically transitive and is sensitive to initial conditions, and Devaney chaotic if it is Auslander-Yorke chaotic and additionally has dense periodic points. As far as we know, before our results it was open whether Game of Life exhibits these types of chaos on its limit set; the following corollary shows that it does not.

▶ **Theorem 23.** *The Game of Life restricted to its limit set is not topologically transitive, and does not have dense periodic points.*

Proof. Either of these properties clearly implies chain-nonwanderingness, contradicting Theorem 5. ◀

Two other standard notions of chaos are Li-Yorke chaos and positive entropy (we omit the definitions). Game of Life exhibits these trivially, since it admits a glider. More generally, intrinsic universality implies that it exhibits any property of spatiotemporal dynamics of cellular automata that is inherited from subsystems of finite-index subactions of the spacetime subshift. Sensitivity in itself is also sometimes considered a notion of chaos. This remains wide open.

▶ **Question 1.** *Is Game of Life sensitive to initial conditions?*

One can also ask about chaos on "typical configurations". For example, take the uniform Bernoulli measure (or some other distribution) as the starting point, and consider the trajectories of random configurations. We can say essentially nothing about this setting.

In our topological dynamical context, a natural way to formalize this problem is through the *generic limit set* as defined in [25]. It is a subset of the phase space of a dynamical system that captures the asymptotic behavior of topologically large subsets of the space. We omit the exact definition, but for a cellular automaton f, this is a nonempty subshift invariant under f [9]. It follows that the generic limit set is contained in the limit set, and that the language of the generic limit set of g contains the letter 0 (because the singleton subshift $\{1^{\mathbb{Z}^2}\}$ is not g-invariant).

We say a cellular automaton f on $S^{\mathbb{Z}^d}$ is *generically nilpotent* if its generic limit set contains only one configuration, which must then be the all-s configuration for a quiescent state $s \in S$. This is equivalent to the condition that every finite pattern can be extended into some larger pattern p such that for large enough $n \in \mathbb{N}$, we have $f^n(x)_{\vec{0}} = s$ for all $x \in [p]$. By the previous observation, if Game of Life were generically nilpotent, we would have $s = 0$. We strongly suspect that it is not generically nilpotent, i.e. the symbol 1 occurs in the generic limit set. However, we have been unable to show this.

▶ **Question 2.** *Is Game of Life generically nilpotent?*

Chaos is usually discussed for one-dimensional dynamical system, but we find its standard ingredients, such as topological transitivity and periodic points, quite interesting. We have been unable to resolve most of these.

▶ **Question 3.** *Is the limit set of Game of Life topologically transitive as a subshift?*

▶ **Question 4.** *Does the limit set of Game of Life have dense totally periodic points as a subshift?*

References

1. Nathalie Aubrun and Mathieu Sablik. Simulation of effective subshifts by two-dimensional subshifts of finite type. *Acta Appl. Math.*, 126(1):35–63, August 2013. doi:10.1007/s10440-013-9808-5.
2. Robert Berger. The undecidability of the domino problem. *Mem. Amer. Math. Soc. No.*, 66, 1966. 72 pages.
3. François Blanchard. Topological chaos: what may this mean? *Journal of Difference Equations and Applications*, 15(1):23–46, 2009. doi:10.1080/10236190802385355.
4. Mike Boyle, Ronnie Pavlov, and Michael Schraudner. Multidimensional sofic shifts without separation and their factors. *Transactions of the American Mathematical Society*, 362(9):4617–4653, 2010. doi:10.1090/s0002-9947-10-05003-8.
5. John H. Conway. Email to Dean Hickerson. Private email group *LifeCA*, 1992. Provided by Dave Greene.
6. Karel Culik, II, Jan Pachl, and Sheng Yu. On the limit sets of cellular automata. *SIAM J. Comput.*, 18(4):831–842, 1989. doi:10.1137/0218057.
7. Oscar Cunningham. Response on ConwayLife forum (username macbi). https://conwaylife.com/forums/viewtopic.php?f=7&t=3180&start=125#p140295. Accessed: 2022-02-09.
8. Alberto Dennunzio, Enrico Formenti, Darij Grinberg, and Luciano Margara. From Linear to Additive Cellular Automata. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 125:1–125:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.125.
9. Saliha Djenaoui and Pierre Guillon. The generic limit set of cellular automata. *Journal of Cellular Automata*, 14(5-6):435–477, 2019. URL: https://www.oldcitypublishing.com/journals/jca-home/jca-issue-contents/jca-volume-14-number-5-6-2019/jca-14-5-6-p-435-477/.
10. Bruno Durand and Zsuzsanna Róka. The game of life: universality revisited. In *Cellular automata (Saissac, 1996)*, volume 460 of *Math. Appl.*, pages 51–74. Kluwer Acad. Publ., Dordrecht, 1999.
11. Bruno Durand, Andrei Romashchenko, and Alexander Shen. Effective closed subshifts in 1D can be implemented in 2D. In *Fields of logic and computation*, volume 6300 of *Lecture Notes in Comput. Sci.*, pages 208–226. Springer, Berlin, 2010.

12 Noam D Elkies. The still-life density problem and its generalizations. *arXiv*, 1999. arXiv:math/9905194.

13 Henryk Fukś. Explicit solution of the cauchy problem for cellular automaton rule. *J. Cell. Autom.*, 12(6):423–444, 2017. URL: http://www.oldcitypublishing.com/journals/jca-home/jca-issue-contents/jca-volume-12-number-6-2017/jca-12-6-p-423-444/.

14 Martin Gardner. Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game "Life". *Scientific American*, 223(4):120–123, 1970.

15 Adam Goucher. Fully self-directed replication. https://cp4space.hatsya.com/2018/11/12/fully-self-directed-replication/. Accessed: 2022-04-20.

16 Adam Goucher. Response on ConwayLife forum (username calcyman). https://conwaylife.com/forums/viewtopic.php?f=7&t=3180&start=100#p140273. Accessed: 2022-02-09.

17 Gustav A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Math. Systems Theory*, 3:320–375, 1969.

18 Lyman P. Hurd. Formal language characterizations of cellular automaton limit sets. *Complex Systems*, 1(1):69–80, 1987.

19 N. Johnston and D. Greene. *Conway's Game of Life: Mathematics and Construction*. Lulu.com, 2022. URL: https://books.google.fi/books?id=xSJlEAAAQBAJ.

20 Jarkko Kari. Universal pattern generation by cellular automata. *Theoret. Comput. Sci.*, 429:180–184, 2012. doi:10.1016/j.tcs.2011.12.037.

21 Steve Kass and Kathleen Madden. A sufficient condition for non-soficness of higher-dimensional subshifts. *Proceedings of the American Mathematical Society*, 141(11):3803–3816, 2013. doi:10.1090/S0002-9939-2013-11646-1.

22 Douglas Lind and Brian Marcus. *An introduction to symbolic dynamics and coding*. Cambridge University Press, Cambridge, 1995. doi:10.1017/CBO9780511626302.

23 Ville Lukkarila. Sensitivity and topological mixing are undecidable for reversible one-dimensional cellular automata. *J. Cell. Autom.*, 5(3):241–272, 2010. URL: http://www.oldcitypublishing.com/journals/jca-home/jca-issue-contents/jca-volume-5-number-3-2010/jca-5-3-p-241-272/.

24 Alejandro Maass. On the sofic limit sets of cellular automata. *Ergodic Theory and Dynamical Systems*, 15, 1995. doi:10.1017/S0143385700008609.

25 John Milnor. On the concept of attractor. *Communications in Mathematical Physics*, 99(2):177–195, 1985. doi:10.1007/BF01212280.

26 MiniSat 2.2. http://minisat.se/. Accessed: 2022-02-09.

27 Period-48 glider gun – LifeWiki. https://conwaylife.com/wiki/Period-48_glider_gun. Accessed: 2022-02-09.

28 PySAT 0.1.7.dev15. https://pysathq.github.io/. Accessed: 2022-02-09.

29 Quadri-Snark – LifeWiki. https://conwaylife.com/wiki/Quadri-Snark. Accessed: 2022-02-09.

30 Ville Salo and Ilkka Törmä. Game of Life agars. https://github.com/ilkka-torma/gol-agars, 2022. GitHub repository.

31 Robert Wainwright. Lifeline vol. 6, 1972.

32 Hao Wang. Proving theorems by pattern recognition II. *Bell System Technical Journal*, 40:1–42, 1961.

Processes Parametrised by an Algebraic Theory

Todd Schmid
Department of Computer Science, University College London, UK

Wojciech Rozowski
Department of Computer Science, University College London, UK

Alexandra Silva
Department of Computer Science, Cornell University, Ithaca, NY, USA

Jurriaan Rot
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Abstract

We develop a (co)algebraic framework to study a family of process calculi with monadic branching structures and recursion operators. Our framework features a uniform semantics of process terms and a complete axiomatisation of semantic equivalence. We show that there are uniformly defined fragments of our calculi that capture well-known examples from the literature like regular expressions modulo bisimilarity and guarded Kleene algebra with tests. We also derive new calculi for probabilistic and convex processes with an analogue of Kleene star.

2012 ACM Subject Classification Theory of computation → Program reasoning

Keywords and phrases process algebra, program semantics, coalgebra, regular expressions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.132

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/2202.06901`

Funding *Todd Schmid*: Partially supported by ERC grant Autoprobe (grant agreement 101002697). *Wojciech Rozowski*: Partially supported by ERC grant Autoprobe (grant agreement 101002697). *Alexandra Silva*: Partially supported by ERC grant Autoprobe (grant agreement 101002697) and a Royal society Wolfson fellowship.

1 Introduction

The theory of processes has a long tradition, notably in the study of concurrency, pioneered by seminal works of Milner [30, 29] and many others [2]. In labelled transition systems, a popular model of computation in process theory, processes branch nondeterministically. This means that any given action or observation transitions a starting state into any member of a predetermined set of states. In Milner's CCS [29], nondeterminism appears as a binary operation that constructs from a pair of programs e and f the program $e + f$ that nondeterministically chooses between executing either e or f. This acts precisely like the join operation in a semilattice. In fact, elements of a free semilattice are exactly sets, as the free semilattice generated by a collection X is the set $\mathcal{P}^+_\omega X$ of finite nonempty subsets of X [26]. This is our first example of a more general phenomenon: the type of branching in models of process calculi can often be captured with an algebraic theory.

A second example appears in the probabilistic process algebra literature, where the process denoted $e +_p f$ flips a weighted coin and runs e with probability p and f with probability $1 - p$. The properties of $+_p$ are axiomatised and studied in convex algebra, an often revisited algebraic theory of probability [1, 47, 35, 46]. The free convex algebra on a set X is the set $\mathcal{D}_\omega X$ of finitely supported probability distributions on X [46, 11, 23].


A third example is guarded Kleene algebra with tests (GKAT), where the process $e +_b f$ proceeds with e if a certain Boolean predicate b holds and otherwise proceeds with f, emulating the `if-then-else` constructs of imperative programming languages [27, 8, 28]. If the predicates are taken from a finite Boolean algebra $2^{\mathtt{At}}$, the free algebra of `if-then-else` clauses on a set X is the function space $X^{\mathtt{At}}$. This explains why adjacency sets for tree models of GKAT programs take the form of functions $\mathtt{At} \to X$.

This paper proposes a framework in which these languages can be uniformly described and studied. We use the *algebra of regular behaviours* (or ARB) introduced in [30] as a prototypical example. ARB employs nondeterministic choice as a branching operation, prefixing of terms by atomic actions, a constant representing deadlock, variables, and a recursion operator for each variable. Specifications are interpreted using structural operational semantics in the style of [34], which sees the set `Exp` of all process terms as one large labelled transition system. This is captured succinctly as a *coalgebra*, in this case a function

$$\beta : \mathtt{Exp} \to \mathcal{P}(V + A \times \mathtt{Exp}) \tag{1}$$

Only finitely branching processes can be specified in ARB, so we will replace $\mathcal{P}$ with $\mathcal{P}_\omega$ in (1). From a technical point of view, $\mathcal{P}_\omega$ is the monad on the category **Sets** of sets and functions presented by the algebraic theory of semilattices with bottom.

By substituting the finite powerset functor in (1) with other monads presented by algebraic theories, we obtain a parametrised family of process types that covers the examples above and a general framework for studying the processes of each type. Instantiating the framework with an algebraic theory gives a fully expressive specification language for processes and a complete axiomatisation of behavioural equivalence for specifications.

One striking feature of many of the specification languages we construct is that they contain a fragment consisting of nonstandard analogues of regular expressions. We call these expressions *star expressions* and the fragment composed of star expressions the *star fragment*. Star fragments extend several existing analogues of basic regular algebra found in the process theory literature, including basic process algebra [5] and Andova's probabilistic basic process algebra [1], by adding recursion operators modelled after the Kleene star.

Milner is the first to notice the star fragment of ARB in [30]. He observes that the algebra of processes denoted by star expressions is more unruly than Kleene's algebra of regular languages, and that it is not clear what the appropriate axiomatisation should be. He offers a reasonable candidate based on Salomaa's first axiomatisation of Kleene algebra [39], but ultimately leaves completeness as an open problem. This problem has been subjected to many years of extensive research [15, 14, 16, 3, 20, 19]. A potential solution has recently been announced by Clemens Grabmayer and will appear in the upcoming LICS.

Replacing nondeterministic choice with the `if-then-else` branching structure of GKAT, we obtain the process behaviours explored in the recent rethinking of the language [40]. This makes the open problem of axiomatising GKAT (without the use of extremely powerful axioms like the *Uniqueness Axiom* of [44]), stated first in [44] and again in [40], yet another problem of axiomatising an algebra of star expressions. Our general characterisation of star expressions puts all these languages under one umbrella, and shows how they are derived canonically from a single abstract framework.

In summary, the contributions of this paper are as follows:

- We present a family of process types parametrised by an algebraic theory (Section 2) together with a uniform syntax and operational semantics (Section 3). We show how these can be instantiated to concrete algebraic theories, including guarded semilattices and pointed convex algebras. These provide, respectively, a calculus of processes capturing control flow of simple imperative programs and a calculus of probabilistic processes.

- We define an associated denotational semantics and show that it agrees with the operational semantics (Section 4). This coincidence result is important in order to prove completeness of the uniform axiomatisation we propose for each process type (Section 5).
- Finally, we study the star fragment of our parameterised family and propose a sound axiomatisation for this fragment (Section 6). We show that star fragments of concrete instances of our calculi yield known examples in the literature, e.g. Guarded Kleene Algebra with tests (GKAT) [44, 40] and probabilistic processes of Stark and Smolka [45].

Related work is surveyed in Section 7, and future research directions are discussed in Section 8.

2 A Parametrised Family of Process Types

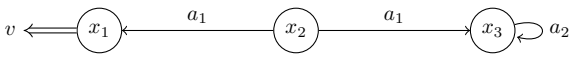
In this section, we present a family of process types parametrised by a certain kind of algebraic theory. The processes we care about are stateful, meaning they consist of a set of states and a suitably structured set of transitions between states. Stateful systems fit neatly into the general framework of *universal coalgebra* [37], which stipulates that the type of structure carried by the transitions can be encoded in an endofunctor on the category **Sets** of sets and functions. Formally, given a functor $B : \mathbf{Sets} \to \mathbf{Sets}$, a *$B$-coalgebra* is a pair (X, β) consisting of a set X of *states* and a *structure* map $\beta : X \to BX$. A *coalgebra homomorphism* $h : (X, \beta) \to (Y, \vartheta)$ is a function $h : X \to Y$ satisfying $\vartheta \circ h = B(h) \circ \beta$. Many types of processes found in the literature coincide with B-coalgebras for some B, and so do their homomorphisms. For example, finitely branching labelled transition systems are $\mathcal{P}_\omega(A \times \mathrm{Id})$-coalgebras, and deterministic Moore automata are $O \times \mathrm{Id}^A$-coalgebras [36].

In this paper, we consider coalgebras for functors of the form

$$B_M := M(V + A \times \mathrm{Id}) \tag{2}$$

for fixed sets V and A and a specific kind of functor $M : \mathbf{Sets} \to \mathbf{Sets}$. Intuitively, there are two layers to the process behaviours we care about: one layer consists of either an *output variable* in V or an *action* from A that moves on to another state, and the other layer (encoded by M) combines output variables and action steps in a structured way.

▶ **Example 1.** When $M = \mathcal{P}_\omega$, we obtain Milner's nondeterministic processes [30]. Coalgebras for $B_{\mathcal{P}_\omega}$ are functions of the form $\beta : X \to \mathcal{P}_\omega(V + A \times X)$, or labelled transition systems with an additional decoration by variables. Write $x \xrightarrow{a} y$ to mean $(a, y) \in \beta(x)$ and $x \Rightarrow v$ to mean $v \in \beta(x)$. The image below posits a well-defined $B_{\mathcal{P}_\omega}$-coalgebra



Its state space is $\{x_1, x_2, x_2\}$, A includes a_1 and a_2, and v is a variable in V.

Algebraic Theories and Their Monads

We are particularly interested in B_M-coalgebras when M is the functor component of a monad (M, η, μ) that is presented by an algebraic theory capturing a type of branching. A monad consists of natural transformations $\eta : \mathrm{Id} \Rightarrow M$ and $\mu : MM \Rightarrow M$, called the *unit* and *multiplication* respectively, satisfying two laws: $\mu \circ \eta_M = \mathrm{id}_M = \mu \circ M(\eta)$ and $\mu_M \circ \mu = M(\mu) \circ \mu$. For our purposes, an *algebraic theory* is a pair (S, E) consisting of a polynomial endofunctor $S = \coprod_{\sigma \in I} \mathrm{Id}^{n_\sigma}$ on **Sets** called an *algebraic signature* and a set E of equations in the signature S. An element σ of I should be thought of as an operation

with arity n_σ. An algebraic theory (S, E) *presents* a monad (M, η, μ) if there is a natural transformation $\rho : SM \Rightarrow M$ such that for any set X, (MX, ρ_X) is the free (S, E)-algebra on X. That is, (MX, ρ_X) satisfies E and for any S-algebra (Y, φ) also satisfying E and any function $h : X \to Y$, there is a unique S-algebra homomorphism $\hat{h} : (MX, \rho_X) \to (Y, \varphi)$ such that $h = \hat{h} \circ \eta$. This universal property implies that any two presentations of a given algebraic theory are isomorphic, so we speak simply of "the" monad presented by an algebraic theory.

▶ **Example 2.** The finite powerset functor is part of the monad $(\mathcal{P}_\omega, \{-\}, \bigcup)$ that is presented by the theory of semilattices (with bottom). The theory of semilattices is the pair $(1+\mathrm{Id}^2, \mathsf{SL})$, since the arity of a constant operation is 0 and $+$ is a binary operation, and SL consists of

$$x + 0 \overset{(\mathsf{SL1})}{=} x \qquad x + x \overset{(\mathsf{SL2})}{=} x \qquad x + y \overset{(\mathsf{SL3})}{=} y + x \qquad x + (y + z) \overset{(\mathsf{SL4})}{=} (x + y) + z$$

Not every algebraic theory has such a familiar presentation as the theory of semilattices, but it is nevertheless true that every algebraic theory presents a monad. If we let S^*X denote the set of S-*terms*, expressions built from X and the operations in S, then (S, E) automatically presents the monad (M, η, μ) where $MX = (S^*X)/\mathsf{E} := \{[q]_\mathsf{E} \mid q \in S^*X\}$ is the set of E-congruence classes of S-terms, η computes congruence classes of variables, and μ evaluates terms. This is witnessed by letting the transformation ρ be the restriction of μ to the operations of S on S-terms. We take this to be the default presentation of an arbitrary algebraic theory.

Our aim is to develop a (co)algebraic framework for studying B_M-coalgebras when M is the functor part of a monad presented by an algebraic theory. We will make three assumptions about the algebraic theories. First, we rule out the case of M being the constant 1 functor.

▶ **Assumption 1.** *The theory* E *is* nontrivial*, meaning that the equation* $x = y$ *is not a consequence of* E *for distinct* x *and* y*.*

This is equivalent to requiring that the unit η is injective. That is, the E-congruence classes $[x]_\mathsf{E}$ and $[y]_\mathsf{E}$ in MX are distinct for distinct variables x and y in X.

Second, we assume the existence of a constant symbol denoting deadlock, which might occur when recursing on unguarded programs.

▶ **Assumption 2.** *Algebraic theories contain a designated constant* 0*.*

Finally, to keep the specifications of processes finite, we make the following assumption despite the fact that it has no bearing on the results presented before Section 5.

▶ **Assumption 3.** *Each operation from* S *has a finite arity.*

We conclude this section with examples of algebraic theories and the monads they present.

▶ **Example 3.** For a fixed finite set $\mathtt{At}$ of *atomic tests*, the algebraic theory of *guarded semilattices* is the pair $(1 + \coprod_{b \subseteq \mathtt{At}} \mathrm{Id}^2, \mathsf{GS})$, where GS consists of the equations

$$x +_b x \overset{(\mathsf{GS1})}{=} x \qquad x +_\mathtt{At} y \overset{(\mathsf{GS2})}{=} x \qquad x +_b y \overset{(\mathsf{GS3})}{=} y +_{\bar{b}} x \qquad (x +_b y) +_c z \overset{(\mathsf{GS4})}{=} x +_{bc} (y +_c z)$$

Here, $+_b$ is the binary operation associated with the subset $b \subseteq \mathtt{At}$, $\bar{b} := \mathtt{At} \setminus b$, and $bc := b \cap c$. The theory of guarded semilattices is presented by the monad $((1 + \mathrm{Id})^\mathtt{At}, \lambda\xi.(-), \Delta^*)$, where $(\lambda\xi.x)(\xi) = x$ and $\Delta^*(F)(\xi) = F(\xi)(\xi)$. The idea is that $+_b$ acts like an `if-then-else` clause in an imperative program. This is reflected in a free guarded semilattice $((1 + X)^\mathtt{At}, \rho_X)$, where for a pair of maps $h_1, h_2 : \mathtt{At} \to X$ we define

$$\rho_X(h_1 +_b h_2)(\xi) := \begin{cases} h_1(\xi) & \text{if } \xi \in b \\ h_2(\xi) & \text{otherwise} \end{cases}$$

The theory of guarded semilattices dates back to the algebras of `if-then-else` clauses studied in [28, 27, 8, 7]. For instance, guarded semilattices are examples of *McCarthy algebras*, introduced by Manes in [27].[1]

▶ **Example 4.** The theory of *pointed convex algebras* studied in [12] is $(1 + \coprod_{p\in[0,1]} \mathrm{Id}^2, \mathsf{CA})$, where CA consists of the equations

$$x +_p x \overset{(\mathsf{CA1})}{=} x \qquad x +_1 y \overset{(\mathsf{CA2})}{=} x \qquad x +_p y \overset{(\mathsf{CA3})}{=} y +_{\bar{p}} x \qquad (x +_p y) +_q z \overset{(\mathsf{CA4})}{=} x +_{pq} (y +_{\frac{q\bar{p}}{1-pq}} z)$$

Here, $+_p$ is the binary operation with index $p \in [0,1]$, $\bar{p} := 1 - p$, and $pq \neq 1$. This theory presents the pointed finite subprobability distribution monad $(\mathcal{D}_\omega(1 + \mathrm{Id}), \delta_{(-)}, \sum)$, where

$$\mathcal{D}_\omega(1 + X) = \left\{ \theta : X \to [0,1] \;\middle|\; \begin{array}{c} \{x \mid \theta(x) > 0\} \text{ is finite} \\ \sum_{x\in X} \theta(x) \leq 1 \end{array} \right\}$$

for any set X, and for any $x \in X$, $\theta \in \mathcal{D}_\omega(1 + X)$, and $\Theta \in \mathcal{D}_\omega(1 + \mathcal{D}_\omega(1 + X))$,

$$\delta_x(y) = [x = y?] \qquad \sum(\Theta)(\theta) = \sum_{y\in X} \Theta(\theta) \cdot \theta(y)$$

This is witnessed by the transformation ρ that takes 0 to the trivial subdistribution and computes the Minkowski sum $\rho_X(\theta +_p \psi) = p\cdot\theta + (1-p)\cdot\psi$ for each $p \in [0,1]$, $\theta, \psi \in \mathcal{D}_\omega(1+X)$.

▶ **Example 5.** The theory of *pointed convex semilattices* studied in [12, 49, 10] combines the theory of semilattices and the theory of convex algebras. It has both a binary operation $+$ mimicking nondeterministic choice and the probabilistic choice operations $+_p$ indexed by $p \in [0,1]$. Formally, it is given by the pair $(1 + \mathrm{Id}^2 + \coprod_{p\in[0,1]} \mathrm{Id}^2, \mathsf{CS})$, where CS is the union of SL, CA, and the distributive law

$$(x + y) +_p z \overset{(\mathsf{D})}{=} (x +_p z) + (y +_p z)$$

This theory presents the pointed convex powerset monad $(\mathcal{C}, \eta^{\mathcal{C}}, \mu^{\mathcal{C}})$, where $\mathcal{C}X$ is the set of finitely generated convex subsets of $\mathcal{D}_\omega(1 + X)$ containing δ_0, and for $x \in X$ and $Q \in \mathcal{C}\mathcal{C}X$,

$$\eta^{\mathcal{C}}(x) = \{p\cdot\delta_x \mid p \in [0,1]\} \qquad \mu^{\mathcal{C}}(Q) = \bigcup_{\Theta\in Q} \left\{ \sum_{U\in\mathcal{C}_0 X} \Theta(U) \cdot \theta_U \;\middle|\; (\forall U \in \mathcal{C}_0 X)\ \theta_U \in U \right\}$$

The witnessing transformation $\rho^{\mathcal{C}}$ takes 0 to $\{\delta_0\}$, computes the Minkowski sum (extended to subsets) in place of $+_p$, and interprets the $+$ operation as the convex union

$$\rho^{\mathcal{C}}_X(U + V) = \{p\cdot\theta_1 + (1-p)\cdot\theta_2 \mid p \in [0,1], \theta_1 \in U, \theta_2 \in V\}$$

3 Specifications of Processes

Fix an algebraic theory (S, E) presenting a monad (M, η, μ). In this section, we give a syntactic and uniformly defined specification system for B_M-coalgebras and an associated operational semantics. We are primarily concerned with the specifications of finite processes, and indeed the process terms we construct below denote processes with finitely many states. The converse is also true, that every finite B_M-coalgebra admits a specification in the form of a process term, but we defer this result to Section 5 because of its relevance to the completeness theorem there.

[1] More information on the theory of guarded semilattices can be found in [42, Appendix A].

$$\epsilon(v) = \eta(v) \qquad \epsilon(\sigma(e_1, \ldots, e_n)) = \sigma(\epsilon(e_1), \ldots, \epsilon(e_n))$$
$$\epsilon(ae) = \eta((a, e)) \qquad \epsilon(\mu v\ e) = \epsilon(e)[\mu v\ e // v]$$

Figure 1 Operational semantics of process terms. Here, $v \in V$, $a \in A$, and $e, e_i \in \mathtt{Exp}$.

The syntax of our specifications consists of variables from an infinite set V, actions from a set A, and operations from S. The set $\mathtt{Exp}$ of *process terms* is given with the grammar

$$e, e_i ::= 0 \mid v \mid \sigma(e_1, \ldots, e_n) \mid ae \mid \mu v\ e$$

where $v \in V$, $a \in A$, and σ is an S-operation. Abstractly, process terms form the initial Σ_M-algebra $(\mathtt{Exp}, \alpha)$, where $\Sigma_M : \mathbf{Sets} \to \mathbf{Sets}$ is the functor defined by

$$\Sigma_M := S + V + A \times \mathrm{Id} + V \times \mathrm{Id}$$

and the algebra map $\alpha : \Sigma_M \mathtt{Exp} \to \mathtt{Exp}$ evaluates Σ_M-terms.

Intuitively, the symbol 0 is the designated constant of S denoting the *deadlock* process, which takes no action. Output variables are used in one of two ways, depending on the expression in which they appear. A variable v is *free* in an expression e if it does not appear within the scope of μv and *bound* otherwise. If v is free in e, then v denotes "output v". Otherwise, v denotes a *goto* statement that returns the computation to the μv that binds v. The process $\sigma(e_1, \ldots, e_n)$ is the process that branches into $e_1, \ldots, e_n$ using an n-ary operation σ as the branching constructor. The expression ae denotes the process that performs the action a and then proceeds with e. Finally, $\mu v\ e$ denotes recursion in the variable v.

Small-step Semantics

Next we give a small-step (operational) semantics to process terms that is uniformly defined for the process types in our parametrised family. Many of the algebraic theories we consider lack a familiar presentation, which ultimately prevents the corresponding semantics from taking the traditional form of a set of inference rules describing transition relations. We take an abstract approach instead by defining a B_M-coalgebra structure $\epsilon : \mathtt{Exp} \to B_M \mathtt{Exp}$ that mirrors the intuitive descriptions of the executions of process terms above. The formal description of ϵ is summarised in Figure 1.

The operational interpretation of the recursion operators requires further explanation. Intuitively, $\mu v\ e$ performs the process denoted by e until it reaches an exit in channel v, at which point it loops back to the beginning. However, this is really only an accurate description of recursion in v when e performs an action before exiting in v. For example, the process $\mu v\ v$ not only never exits in channel v, but it also never performs any action at all. Thus, the operational interpretation of $\mu v\ v$ is indistinguishable from that of deadlock. We deal with this issue as follows: if an exit in channel v is immediately reached by a branch of e, then we replace that exit with deadlock in $\mu v\ e$. Formally, we say that a variable v is *guarded* in a process term e if (i) $e \in V \setminus \{v\}$, (ii) $e = af$ or (iii) $e = \mu v\ f$ for some $f \in \mathtt{Exp}$, or (iv) either $e = \mu u\ e_1$ or (v) $e = \sigma(e_1, \ldots, e_n)$ and v is guarded in e_i for each $i \leq n$. In our calculus, we syntactically allow for recursion in unguarded variables, but one should keep in mind that those variables are ultimately deadlock under the recursion operator.

The operational interpretation of recursion is formally defined using a *guarded syntactic substitution operator* $[g//v] : B_M\mathtt{Exp} \to B_M\mathtt{Exp}$,[2] a variant of the usual syntactic substitution of variables. Given $g \in \mathtt{Exp}$, we first define $[g//v]$ by induction on $S^*(V + A \times \mathtt{Exp})$ as

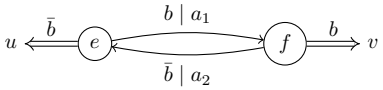
$$u[g//v] = \begin{cases} \eta(u) & u \neq v \\ \eta(0) & u = v \end{cases} \qquad \begin{aligned} (a, f)[g//v] &= (a, f[g/v]) \\ \sigma(p_1, \ldots, p_n)[g//v] &= \sigma(p_1[g//v], \ldots, p_n[g//v]) \end{aligned}$$

where $u \in V$, $p_i \in S^*(V + A \times \mathtt{Exp})$, $f \in \mathtt{Exp}$, and $[g/v]$ replaces free occurrence of v with g. The following lemma completes the description of the operational semantics of process terms.

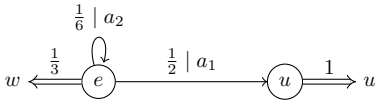
▶ **Lemma 6.** *For any $g \in \mathtt{Exp}$ and $v \in V$, the map $[g//v]$ factors uniquely through $B_M\mathtt{Exp}$.*

Formally, the map ϵ assigns to each process term e an E-congruence class $\epsilon(e)$ of terms from $S^*(V + A \times \mathtt{Exp})$. A term from $S^*(V + A \times \mathtt{Exp})$ is a combination of variables v and transition-like pairs (a, e_i), so there is often only a small conceptual leap from the coalgebra structure ϵ to a more traditional representation of transitions as decorated arrows. We provide the following examples as illustrations of this phenomenon, as well as the specification languages and operational semantics of terms defined above.[3]

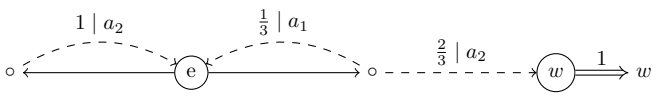
▶ **Example 7.** The *algebra of control flows*, or ACF, is obtained from the theory of guarded semilattices of Example 3 and $M = (1 + Id)^{\mathtt{At}}$. Given a structure map $\beta : X \to B_{(1+\mathrm{Id})^{\mathtt{At}}}X$ and $b \subseteq \mathtt{At}$, write $x \xrightarrow{b|a} y$ if $\beta(x)(\xi) = (a, y)$ for all $\xi \in b$, and $x \overset{b}{\Rightarrow} v$ if $\beta(x)(\xi) = v$ for all $\xi \in b$. The operational semantics returns the constant map $\lambda\xi.v$ given a variable $v \in V$ and interprets conditional choice as guarded union. For example, let $e = \mu w\ (a_1(v +_b a_2 w) +_b u)$ and $f = v +_b a_2 e$. The process denoted by e is



▶ **Example 8.** The *algebra of probabilistic actions*, or APA, is obtained from the theory of pointed convex algebras of Example 4 and $M = \mathcal{D}_\omega(1 + Id)$. For a structure map $\beta : X \to B_{\mathcal{D}_\omega(1+\mathrm{Id})}X$, write $x \xrightarrow{k|a} y$ when $\beta(x)(a, y) = k$ and $e \overset{k}{\Rightarrow} v$ when $\beta(e)(v) = k$. The operational semantics returns the Dirac distribution δ_v for $v \in V$ and interprets probabilistic choice as the Minkowski sum. The process denoted by $e = \mu v\ (a_1 u +_{\frac{1}{2}} (a_2 v +_{\frac{1}{3}} w))$ is



▶ **Example 9.** The *algebra of nondeterministic probabilistic actions*, or ANP, is obtained from the theory of pointed convex semilattices of Example 5. For a structure map $\beta : X \to B_{\mathcal{C}}X$, write $x \to\!\circ\xrightarrow{k|a} y$ to mean there is a $\theta \in \beta(x)$ such that $\theta(a, y) = k$, and $x \overset{k}{\Rightarrow} v$ to mean there is a $\theta \in \beta(x)$ with $\theta(v) = k$. The operational semantics returns $\eta^{\mathcal{C}}(v)$ given $v \in V$, interprets nondeterministic choice as convex union, and replaces probabilistic choice with Minkowski sum. For example, $e = \mu v\ ((a_1 v +_{\frac{1}{3}} a_2 w) + a_2 v)$ denotes



[2] Technically, it is only partially defined. See [42, Appendix C] for details.
[3] See [42, Appendix B] in the full version of the paper.

$$\zeta(\gamma(v)) = [v]_{\mathsf{E}} \qquad \zeta(\gamma(\sigma(t_1,\ldots,t_n))) = [\sigma(\zeta(t_1),\ldots,\zeta(t_n))]_{\mathsf{E}}$$
$$\zeta(\gamma(a,t)) = [(a,t)]_{\mathsf{E}} \qquad \zeta(\gamma(\mu v\ t)) = \zeta(t)\{\gamma(\mu v\ t)//v\}$$

Figure 2 The Σ_M-algebra structure of (Z,γ). Here, $v \in V$, $a \in A$, $t, t_i \in Z$ for $i \leq n$, and σ is an n-ary operation from S. By Lambek's lemma [25], $\zeta : Z \to B_M Z$ is a bijection, so the first three equations determine $\gamma : V + SZ + A \times V \to Z$. The fourth is a behavioural differential equation [36].

4 Behavioural Equivalence and the Final Coalgebra

In this section, we relate the operational semantics arising from the coalgebra structure on $\mathtt{Exp}$ in the previous section to a denotational semantics, which arises through the definition of a suitable algebra structure on the domain of process behaviours.

For an arbitrary functor $B : \mathbf{Sets} \to \mathbf{Sets}$, a *behaviour* is a state of the *final* B-coalgebra (Z,ζ), the unique (up to isomorphism) coalgebra (if it exists) such that there is exactly one homomorphism $!_\beta : (X,\beta) \to (Z,\zeta)$ from every B-coalgebra (X,β). It follows from general considerations that the functor B_M admits a final coalgebra [37]. The universal property of the final B_M-coalgebra produces the homomorphism $!_\epsilon : (\mathtt{Exp},\epsilon) \to (Z,\zeta)$. The behaviour $!_\epsilon(e)$ is called the *final (coalgebra) semantics* of e, also known as its operational semantics [38].

For example, the final $B_{\mathcal{P}_\omega}$-coalgebra consists of bisimulation equivalence classes of finite and infinite labelled trees of a certain form [4]. In this setting, $(\mathtt{Exp},\epsilon)$ is a labelled transition system and the final semantics $!_\epsilon$ constructs a tree from a process term by unrolling. Intuitively, this captures the behaviour of a specification by encoding all possible actions and outgoing messages at each time-step in its execution.

In addition to forming the state space of the final B_M-coalgebra, the set of process behaviours also carries the structure of a Σ_M-algebra (Z,γ), summarised in Figure 2. Now, $(\mathtt{Exp},\alpha)$ is the *initial* Σ_M-algebra, which in particular means there is a unique algebra homomorphism $[\![-]\!] : (\mathtt{Exp},\alpha) \to (Z,\gamma)$. The behaviour $[\![e]\!]$ is called the *initial (algebra) semantics* of e [18], and provides a denotational semantics to our process calculus.

The algebra structure $\gamma : \Sigma_M Z \to Z$ of (Z,γ) can be seen as a reinterpretation of the programming constructs of the language $\mathtt{Exp}$ that mimics the operational semantics of process terms. The basic constructs are the content of the first three equations in Figure 2: output variables are evaluated so as to behave like the variables of $(\mathtt{Exp},\epsilon)$, the behaviour at performs a and moves on to t, and $\sigma(t_1,\ldots,t_n)$ branches into the behaviours $t_1,\ldots,t_n$ with additional structure determined by the operation σ. Interpreting recursive behaviours like $\mu v\ t$ requires coalgebraic analogues of syntactic and guarded syntactic substitution from Section 3.

For a given behaviour $s \in Z$ and a variable $v \in V$, the *behavioural substitution* of s for v is the map $\{s/v\} : Z \to Z$ defined by the identity

$$\zeta(t\{s/v\}) = \begin{cases} \zeta(s) & \zeta(t) = [v]_{\mathsf{E}} \\ [u]_{\mathsf{E}} & \zeta(t) = [u]_{\mathsf{E}} \neq [v]_{\mathsf{E}} \\ [(a, r\{s/v\})]_{\mathsf{E}} & \zeta(t) = [(a,r)]_{\mathsf{E}} \\ \sigma(\zeta(t_1\{s/v\}),\ldots,\zeta(t_n\{s/v\})) & \zeta(t) = [\sigma(\zeta(t_1),\ldots,\zeta(t_n))]_{\mathsf{E}} \end{cases}$$

for any $t \in Z$. The *guarded behavioural substitution* of s for v is constructed in analogy with guarded syntactic substitution from the previous section. We start by defining guarded behavioural substitution in $S^*(V + A \times Z)$ as

$$u\{s//v\} = \begin{cases} u & u \neq v \\ 0 & u = v \end{cases} \qquad \begin{aligned} (a,r)\{s//v\} &= (a, r\{s/v\}) \\ \sigma(r_1,\ldots,r_n)\{s//v\} &= \sigma(r_1\{s//v\},\ldots,r_n\{s//v\}) \end{aligned}$$

where $u \in V$, $a \in A$, and $r, r_i \in Z$ for $i \leq n$. This map lifts to an operator $B_M Z \to B_M Z$ for the same reason as the guarded syntactic substitution operator. This completes the description of the algebraic structure of (Z, γ) in Figure 2.

▶ **Theorem 10.** *Let* $[\![-]\!]$ *be the unique algebra homorphism* $(\mathtt{Exp}, \alpha) \to (Z, \gamma)$. *For any process term* $e \in \mathtt{Exp}$, *we have* $!_\epsilon(e) = [\![e]\!]$.

In other words, the final semantics given with respect to operational rules in Exp coincides with the initial semantics given with respect to the programming constructs in Z. Consequently, we write $[\![-]\!]$ in place of $!_\epsilon$ and simply refer to $[\![e]\!]$ as the semantics of e.

5 An Axiomatisation of Behavioural Equivalence

An important corollary of Theorem 10 is that behavioural equivalence is a Σ_M-congruence on $(\mathtt{Exp}, \alpha)$, meaning that it is preserved by all the program constructs of Σ_M. This opens the door to the possibility of deriving behavioural equivalences between process terms from just a few axioms. The purpose of this section is to show that all behavioural equivalences between process terms can be derived from the equations in E presenting (M, η, μ) as well as three axiom schemas concerning the recursion operators.

The first two out of the three recursion axiom schemas are

$$\text{(R1)} \quad \mu v\ e = e[\mu v\ e//v] \qquad\qquad \text{(R2)} \quad \frac{w \text{ not free in } e}{\mu v\ e = \mu w\ (e[w/v])}$$

Above, $e[\mu v\ e//v]$ is the expression obtained by replacing every guarded free occurrence of v in e with the expression $\mu v\ e$ and every unguarded occurrence of v in e with 0, in analogy with the operator on $B_M\mathtt{Exp}$ of the same name.[4]

The axiom (R1) essentially allows for a sort of guarded unravelling of recursive terms. This has the effect of identifying $\mu v\ v$ with 0, for example, as well as $\mu v\ av$ with $a(\mu v\ av)$. The latter satisfies our intuition that $\mu v\ av$ should solve the recursive specification $x = ax$ in the indeterminate x. The axiom (R2) allows for recursion variables to be swapped for fresh variables. This amounts to the observation that pairs of terms like $\mu v\ av$ and $\mu w\ aw$ should both denote the unique solution to $x = ax$.

The third recursion axiom schema can be stated in the form of the proof rule

$$\text{(R3)} \quad \frac{g = e[g/v] \quad v \text{ guarded in } e}{g = \mu v\ e}$$

We let R denote the set of equations derived from (R1)-(R3), and we let $\equiv$ denote the smallest congruence in $(\mathtt{Exp}, \alpha)$ containing the set of equations derived from E and R. When we refer to examples like ARB, ACF, APA, and ANP, we are often identifying each of these with their associated algebras $(\mathtt{Exp}/{\equiv}, \hat{\alpha})$ of process terms modulo $\equiv$.

[4] Indeed, the identity $\epsilon(e[\mu v\ e//v]) = \epsilon(e)[\mu v\ e//v]$ holds for all $e \in \mathtt{Exp}$ and $v \in V$ [42, Lemma C.9].

Soundness

We would like to argue that $\equiv$ is *sound* with respect to behavioural equivalence, meaning that $[\![e]\!] = [\![f]\!]$ whenever $e \equiv f$. This is indeed the case, and can be derived from the fact that the set of congruence classes of process terms itself forms a B_M-coalgebra. For an arbitrary function $h : X \to Y$, call the set $\ker(h) := \{(x, y) \mid h(x) = h(y)\}$ the *kernel* of h.

▶ **Lemma 11.** *The congruence $\equiv$ is the kernel of a coalgebra homomorphism.*

We write $[-]_\equiv : \mathtt{Exp} \to \mathtt{Exp}/{\equiv}$ for the quotient map and $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$ for the coalgebra structure on $\mathtt{Exp}/{\equiv}$ making $[-]_\equiv$ a coalgebra homomorphism (there is at most one such coalgebra structure [37]). As $[\![-]\!]$ is the unique coalgebra homomorphism $(\mathtt{Exp}, \epsilon) \to (Z, \zeta)$, and because there is also a coalgebra homomorphism $!_{\bar{\epsilon}} : (\mathtt{Exp}/{\equiv}, \bar{\epsilon}) \to (Z, \zeta)$, it must be the case that $!_{\bar{\epsilon}} \circ [-]_\equiv = [\![-]\!]$. By Lemma 11, if $e \equiv f$, then $[\![e]\!] = !_{\bar{\epsilon}}([e]_\equiv) = !_{\bar{\epsilon}}([f]_\equiv) = [\![f]\!]$. This establishes the following.

▶ **Theorem 12** (Soundness). *Let $e, f \in \mathtt{Exp}$. If $e \equiv f$, then $[\![e]\!] = [\![f]\!]$.*

Soundness allows us to derive at least a subset of all the behavioural equivalences between process terms from the axioms in E and R. If our aspiration were simply to have a set of behaviour-preserving code-transformations, then we could simply stop here and be satisfied, since in principle we could see the axioms of E and R as rewrite rules that satisfy this purpose.

Completeness

Aiming a bit higher than deriving only a subset of the behavioural equivalences between process terms, we move on to show the converse of Theorem 12, that $\equiv$ is *complete* with respect to behavioural equivalence. We use [41, Lemma 5.1], which can be stated as follows.

▶ **Lemma 13.** *Let $B : \mathbf{Sets} \to \mathbf{Sets}$ be an endofunctor with a final coalgebra (Z, ζ), and let $\mathbf{C}$ be a class of B-coalgebras. If $\mathbf{C}$ is closed under homomorphic images*[5] *and has a final object (E, ε), then $!_\varepsilon : E \to Z$ is injective.*

A *subcoalgebra* of a B-coalgebra (X, β) is an injective map $\iota : U \hookrightarrow X$ such that $\beta|_U$ factors through $B(\iota)$. A B-coalgebra is *locally finite* if every of its states is contained in (the image of) a finite subcoalgebra. We instantiate Lemma 13 in the case where $B = B_M$, $(E, \varepsilon) = (\mathtt{Exp}/{\equiv}, \bar{\epsilon})$, and $\mathbf{C}$ is the class of locally finite B_M-coalgebras. Completeness of $\equiv$ with respect to behavioural equivalence follows shortly after, for if $[\![e]\!] = [\![f]\!]$, then $!_{\bar{\epsilon}}([e]_\equiv) = !_{\bar{\epsilon}}([f]_\equiv)$. By Lemma 13, $!_{\bar{\epsilon}}$ is injective, so $[e]_\equiv = [f]_\equiv$ or equivalently $e \equiv f$. To establish the converse of Theorem 12, it suffices to show that our choices of (E, ε) and $\mathbf{C}$ satisfy the hypotheses of Lemma 13.

Before we continue, we would like to remind the reader of Assumption 3, that S only has operations of finite arity, as up until now it has not been strictly necessary.

▶ **Lemma 14.** *The coalgebra $(\mathtt{Exp}, \epsilon)$ is locally finite.*

Proof. Given $e \in \mathtt{Exp}$, we construct a subcoalgebra of $(\mathtt{Exp}, \epsilon)$ that has a finite set of states that includes e. To this end, define $U : \mathtt{Exp} \to \mathcal{P}_\omega(\mathtt{Exp})$ by

$$U(v) = \{v\} \qquad U(ae) = \{ae\} \cup U(e) \qquad U(\sigma(e_1, \ldots, e_n)) = \{\sigma(e_1, \ldots, e_n)\} \cup \bigcup_{i<n} U(e_i)$$

$$U(\mu v\ e) = \{\mu v\ e\} \cup U(e)[\mu v\ e//v] = \{\mu v\ e\} \cup \{f[\mu v\ e//v] \mid f \in U(e)\}$$

[5] Ie., if $(X, \beta) \in \mathbf{C}$ and $h : (X, \beta) \to (Y, \vartheta)$, then $(h[X], \vartheta|_{h[X]}) \in \mathbf{C}$.

Note that $e \in U(e)$ for all $e \in \mathtt{Exp}$ and that $U(e)$ is finite. We begin with following claim, which says that the outgoing transitions of e are given in terms of expressions from $U(e)$: for any $e \in \mathtt{Exp}$, there is a representative S-term $p \in \epsilon(e)$ such that $p \in S^*(V + A \times U(e))$. This can be seen by induction on the construction of e, the only interesting case being in the inductive step $\mu v\ e$. Here, let $p \in \epsilon(e)$ and observe that $p[\mu v\ e//v]$ is a representative of $\epsilon(\mu v\ e)$ in $S^*(V + A \times U(\mu v\ e))$.

To finish the proof of the lemma, fix an $e \in \mathtt{Exp}$ and define a sequence of sets beginning with $U_0 = \{e\}$ and proceeding with

$$U_{n+1} = U_n \cup \bigcup_{e_0 \in U_n} \{g \mid (\exists a \in A)(\exists p \in \epsilon(e_0) \cap S^*(V + A \times U(e_0)))\ (a, g) \text{ appears in } p\}$$

Then $U_0 \subseteq U_1 \subseteq \cdots \subseteq U(e)$ and the latter set is finite, so $U := \bigcup U_n$ is finite and contained in $U(e)$. We define a coalgebra structure $\epsilon_U : U \to B_M U$ by taking $\epsilon_U(e) = [p]_{\mathsf{E}}$ where if $e \in U_n$, then p is a representative of $\epsilon(e)$ in $S^*(V + A \times U_{n+1})$. Since $S^*(V + A \times U_{n+1}) \subseteq S^*(V + A \times U)$, this defines a B_M-coalgebra structure on U. Where $\iota : U \hookrightarrow \mathtt{Exp}$, we have $\epsilon(\iota(e)) = \epsilon(e) = B_M(\iota)\ \epsilon_U(e)$, so (U, ϵ_U) is a finite subcoalgebra of $(\mathtt{Exp}, \epsilon)$ containing e. ◀

The class of locally finite coalgebras is closed under homomorphic images: if (X, β) is locally finite and $h : (X, \beta) \to (Y, \vartheta)$ is a surjective homomorphism, then for any $y \in Y$ and $x \in X$ such that $h(x) = y$, and for any finite subcoalgebra U of (X, β) containing x, $h[U]$ is a finite subcoalgebra of (Y, ϑ) containing y [21]. Since y was arbitrary, it follows from Lemma 11 that $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$ is locally finite.

What remains to be seen among the hypotheses of Lemma 13 is that $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$ is the *final* locally finite coalgebra, meaning that for any locally finite coalgebra (X, β) there is a unique coalgebra homomorphism $(X, \beta) \to (\mathtt{Exp}/{\equiv}, \bar{\epsilon})$. Every homomorphism from a locally finite coalgebra is the union of its restrictions to finite subcoalgebras, so it suffices to see that every finite subcoalgebra of (X, β) admits a unique coalgebra homomorphism into $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$.

To this end, we make use of an old idea, possibly originating in the work of Salomaa [39]. We associate with every finite coalgebra a certain system of equations whose solutions (in $\mathtt{Exp}/{\equiv}$) are in one-to-one correspondence with coalgebra homomorphisms into $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$. Essentially, if a system admits a unique solution, then its corresponding coalgebra admits a unique homomorphism into $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$. This would then establish finality.

▶ **Definition 15.** *A* (finite) system of equations *is a sequence of the form* $\{x_i = e_i\}_{i \le n}$ *where* $x_i \in V$ *and* $e_i \in \mathtt{Exp}$ *for* $i \le n$*, and none of* $x_1, \ldots, x_n$ *appear as bound variables in any of* $e_1, \ldots, e_n$*. A system of equations* $\{x_i = e_i\}_{i \le n}$ *is* guarded *if* $x_1, \ldots, x_n$ *are guarded in* e_i *for each* $i \le n$*. A* solution *to* $\{x_i = e_i\}_{i \le n}$ *is a function* $\phi : \{x_1, \ldots, x_n\} \to \mathtt{Exp}$ *such that*

$$\phi(x_i) \equiv e_i[\phi(x_1)/x_1, \ldots, \phi(x_n)/x_n]$$

for all $i \le n$ *and* $x_1, \ldots, x_n$ *do not appear free in* $\phi(x_i)$ *for any* $i \le n$*.*

Every finite B_M-coalgebra (X, β) gives rise to a guarded system of equations in the following way: for each $p \in S^*(V + A \times X)$, define $p^\dagger$ inductively as

$$v^\dagger = v \qquad (a, e)^\dagger = ae \qquad \sigma(f_1, \ldots, f_n)^\dagger = \sigma(f_1^\dagger, \ldots, f_n^\dagger)$$

and for each $x \in X$, let p_x be a representative of $\beta(x)$. The[6] system of equations *associated with* (X, β) is then defined to be $\{x = p_x^\dagger\}_{x \in X}$. We treat the elements of X as variables in these equations, and note that by definition every $y \in X$ is guarded in $p_x^\dagger$.

[6] Technically speaking, there could be many systems of equations associated with a given coalgebra. We say "the" system of equations because any two have the same set of solutions up to E.

▶ **Theorem 16.** *Let (X, β) be a finite B_M-coalgebra and $\phi : X \to \mathtt{Exp}$ a function. Then the composition $[-]_{\equiv} \circ \phi : X \to \mathtt{Exp}/{\equiv}$ is a B_M-coalgebra homomorphism if and only if ϕ is a solution to the system of equations associated with (X, β).*

Proof. We begin by observing that $\bar{\epsilon} : \mathtt{Exp}/{\equiv} \to B_M\mathtt{Exp}/{\equiv}$ is a bijection. Indeed, the map $(-)^{\heartsuit} : B_M\mathtt{Exp} \to \mathtt{Exp}/{\equiv}$ defined to be the unique map satisfying

$$[v]_{\mathsf{E}}^{\heartsuit} = [v]_{\equiv} \qquad [(a, e)]_{\mathsf{E}}^{\heartsuit} = [ae]_{\equiv} \qquad [\sigma(p_1, \ldots, p_n)]_{\mathsf{E}}^{\heartsuit} = \sigma([p_1]_{\mathsf{E}}^{\heartsuit}, \ldots, [p_n]_{\mathsf{E}}^{\heartsuit})$$

is its inverse. By construction, $\bar{\epsilon}([p]_{\mathsf{E}}^{\heartsuit}) = [p]_{\mathsf{E}}$ for any $p \in S^*(V + A \times \mathtt{Exp})$, so it suffices to see that $\bar{\epsilon}([e]_{\equiv})^{\heartsuit} = [e]_{\equiv}$ for all $e \in \mathtt{Exp}$. This can be done by induction on the construction of e, and again the only interesting case is $\mu v\ e$. For this case, observe that

$$\begin{aligned}
\bar{\epsilon}([\mu v\ e]_{\equiv})^{\heartsuit} &= (B_M([-]_{\equiv})(\epsilon(e)[\mu v\ e//v]))^{\heartsuit} \\
&= (B_M([-]_{\equiv})(\epsilon(e[\mu v\ e//v])))^{\heartsuit} \\
&= (\bar{\epsilon}([e[\mu v\ e//v]]_{\equiv}))^{\heartsuit} && ([-]_{\equiv} \text{ a coalg. homom.}) \\
&= (\bar{\epsilon}([e]_{\equiv})[[\mu v\ e]_{\equiv}//v])^{\heartsuit} \\
&= \bar{\epsilon}([e]_{\equiv})^{\heartsuit}[[\mu v\ e]_{\equiv}//v] && (*) \\
&= [e[\mu v\ e//v]]_{\equiv} && (\text{induct. hyp.}) \\
&= [\mu v\ e]_{\equiv} && (\mathsf{R1})
\end{aligned}$$

We have used several properties of syntactic substitution in the above calculation; see [42, Appendix C] for details. We have also used that guarded syntactic substitution commutes with $(-)^{\heartsuit}$ in (*), but this follows from an easy induction on terms in $S^*(V + A \times (\mathtt{Exp}/{\equiv}))$.

Now let $\{x = p_x^{\dagger}\}_{x \in X}$ be the system of equations associated with the coalgebra (X, β). Observe that for any $x, y \in X$, if y appears in p_x, then it is guarded in $p_x^{\dagger}$. This means that $\phi : X \to \mathtt{Exp}$ is a solution to $\{x = p_x^{\dagger}\}_{x \in X}$ if and only if $\phi(x) \equiv p_x^{\dagger}[\phi(y)//y]_{y \in X}$. Now, if $\beta(x) = [p_x]_{\mathsf{E}}$, we see that

$$\begin{aligned}
(B_M([-]_{\equiv} \circ \phi)(\beta(x)))^{\heartsuit} &= (B_M([-]_{\equiv}) \circ B_M(\phi)([p_x]_{\mathsf{E}}))^{\heartsuit} && (\text{functoriality}) \\
&= (B_M([-]_{\equiv})([p_x]_{\mathsf{E}}[\phi(y)//y]_{y \in X}))^{\heartsuit} && (B_M(\phi) \text{ an alg. homom.}) \\
&= ([p_x]_{\mathsf{E}}[[\phi(y)]_{\equiv}//y]_{y \in X})^{\heartsuit} \\
&= [p_x^{\dagger}[\phi(y)//y]_{y \in X}]_{\equiv}
\end{aligned}$$

Thus, ϕ is a solution to the system $\{x = p_x^{\dagger}\}_{x \in X}$ if and only if

$$[-]_{\equiv} \circ \phi(x) = (-)^{\heartsuit} \circ B_M([-]_{\equiv} \circ \phi) \circ \beta(x) \tag{3}$$

for every $x \in X$. The maps $(-)^{\heartsuit}$ and $\bar{\epsilon}$ are inverse to one another, so Equation (3) is equivalent to the identity $\bar{\epsilon} \circ [-]_{\equiv} \circ \phi = B_M([-]_{\equiv} \circ \phi) \circ \beta$. This identity is the defining property of a coalgebra homomorphism of the form $[-]_{\equiv} \circ \phi$. ◀

As a direct consequence of Theorem 16, we see that a finite subcoalgebra $U \hookrightarrow \mathtt{Exp}$ of $(\mathtt{Exp}, \epsilon)$ is a solution to the system of equations associated with $(U, \epsilon|_U)$.

▶ **Example 17.** The system of equations associated with the automaton in Example 7 is the two-element set $\{x_1 = a_1x_2 +_b u, x_2 = v +_b a_2x_1\}$. The map $\phi : \{x_1, x_2\} \to \mathtt{Exp}$ defined by $\phi(x_1) = \mu w\ (a_1(v +_b a_2w) +_b u)$ and $\phi(x_2) = v +_b a_2\ \phi(x_1)$ is a solution.

Theorem 16 establishes a one-to-one correspondence between solutions to systems and coalgebra homomorphisms as follows. Say that two solutions ϕ and ψ to a system $\{x_i = e_i\}_{i \leq n}$ are $\equiv$-*equivalent* if $\phi(x_i) \equiv \psi(x_i)$ for all $i \leq n$. Starting with a solution $\phi : X \to \mathtt{Exp}$ to the system associated with (X, β), we obtain the homomorphism $[-]_\equiv \circ \phi$ using Theorem 16. A pair of solutions ϕ and ψ are $\equiv$-equivalent if and only if $[-]_\equiv \circ \phi = [-]_\equiv \circ \psi$, so up to $\equiv$-equivalence the correspondence $\phi \mapsto [-]_\equiv \circ \phi$ is injective. Going in the opposite direction and starting with a homomorphism $\psi : (X, \beta) \to (\mathtt{Exp}/{\equiv}, \bar{\epsilon})$, let e_x be a representative of $\psi(x)$ for each $x \in X$ and define $\phi := \lambda x.e_x$. Then ϕ is a solution to (X, β), and $[-]_\equiv \circ \phi = \psi$. Thus, up to $\equiv$-equivalence, solutions to systems are in one-to-one correspondence with coalgebra homomorphisms into $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$.

Say that a system *admits a unique solution up to* $\equiv$ if it has a solution and any two solutions to the system are $\equiv$-equivalent. Since, up to $\equiv$-equivalence, solutions to a system associated with a coalgebra (X, β) are in one-to-one correspondence with coalgebra homomorphisms $(X, \beta) \to (\mathtt{Exp}/{\equiv}, \bar{\epsilon})$, it suffices for the purposes of satisfying the hypotheses of Lemma 13 to show that every finite guarded system of equations admits a unique solution up to $\equiv$. The following theorem is a generalisation of [30, Theorem 5.7].

▶ **Theorem 18.** *Every finite guarded system of equations admits a unique solution up to* $\equiv$.

The proof is a recreation of the one that appears under [30, Theorem 5.7] with the more general context of our paper in mind. Remarkably, the essential details of the proof remain unchanged despite the jump in the level of abstraction between the two results.

Completeness of $\equiv$ with respect to behavioural equivalence is now a direct consequence of Lemma 13 and Theorems 16 and 18.

▶ **Corollary 19** (Completeness). *Let* $e, f \in \mathtt{Exp}$. *If* $[\![e]\!] = [\![f]\!]$, *then* $e \equiv f$.

One way to interpret this theorem is that the algebra $(\mathtt{Exp}/{\equiv}, \hat{\alpha})$ of process terms modulo $\equiv$ is isomorphic to a subalgebra of (Z, γ), or dually $(\mathtt{Exp}/{\equiv}, \bar{\epsilon})$ is a subcoalgebra of (Z, ζ). It is in this sense that ARB, ACF, APA, and ANP are algebras of behaviours.

6 Star Fragments

In this section we study a fragment of our specification languages consisting of *star expressions*. These include primitive actions from A, a form of sequential composition, and analogues of the Kleene star. We do not aim to give a complete axiomatisation of behavioural equivalence for star expressions, as even in simple cases this is notoriously difficult. Nevertheless, we think it is valuable to extrapolate from known examples a speculative axiomatisation independent of the specification languages from previous sections.

Fix an algebraic theory (S, E) and assume S consists of only constants and binary operations. Its *star fragment* is the set $\mathtt{SExp}$ of expressions given by the grammar

$$e, e_i ::= c \mid \mathsf{1} \mid a \mid e_1 +_\sigma e_2 \mid e_1 e_2 \mid e^{(\sigma)}$$

where $a \in A$, c is a constant in S, and σ is a binary S-operation.

The star fragment of an algebraic theory is a fragment of $\mathtt{Exp}$ in the sense that star expressions can be thought of as shorthands for process terms, as we explain next. In this translation, we fix a distinguished variable $\underline{u} \in V$, called the *unit*, which will denote successful termination, and we also fix a variable v distinct from the unit, which will appear in the fixpoint. The translation of star expressions to process terms is defined to be

$$\mathsf{1} \mapsto \underline{u} \qquad a \mapsto a\underline{u} \qquad e_1 +_\sigma e_2 \mapsto \sigma(e_1, e_2) \qquad e_1 e_2 \mapsto e_1[e_2/\underline{u}] \qquad e^{(\sigma)} \mapsto \mu v\, (e[v/\underline{u}] +_\sigma \underline{u})$$

$$\ell(c) = [c]_{\mathsf{E}} \qquad \ell(e_1 +_\sigma e_2) = \sigma(\ell(e_1), \ell(e_2))$$
$$\ell(1) = [\checkmark]_{\mathsf{E}} \qquad \ell(ef) = p(\ell(f), [(a_1, e_1 f)]_{\mathsf{E}}, \dots, [(a_n, e_n f)]_{\mathsf{E}})$$
$$\ell(a) = [(a, 1)]_{\mathsf{E}} \qquad \ell(e^{(\sigma)}) = p([0]_{\mathsf{E}}, [(a_1, e_1 e^{(\sigma)})]_{\mathsf{E}}, \dots, [(a_n, e_n e^{(\sigma)})]_{\mathsf{E}}) +_\sigma [\checkmark]_{\mathsf{E}}$$

Figure 3 The coalgebra structure map $\ell : \mathtt{SExp} \to L_M \mathtt{SExp}$. Here, c is a constant of S, σ is a binary operation of S, $a \in A$, and $e, e_i \in \mathtt{SExp}$. In the last two equations, $\ell(e) = [p(\checkmark, (a_1, e_1), \dots, (a_n, e_n))]_{\mathsf{E}}$ for some $p \in S^*(\{\checkmark\} + A \times \mathtt{SExp})$.

Sequential composition of terms is associative and distributes over branching operations on the right-hand side[7]: for any $e_1, e_2, f \in \mathtt{SExp}$, $(e_1 +_\sigma e_2)f$ and $e_1 f +_\sigma e_2 f$ translate to the same process term. Similarly, the intuitively correct identities $\mathtt{1}e = e = e\mathtt{1}$ hold modulo translation, as well as the identity $0e = 0$.[8]

The operational semantics for star expressions is given by an L_M-coalgebra $(\mathtt{SExp}, \ell)$ in Figure 3, where $L_M = M(\{\checkmark\} + A \times \mathrm{Id})$. Abstractly, the operational interpretation $\ell(e)$ of a star expression e is obtained by translating e into a process term (also called e) and then identifying $\underline{a}$ with $\checkmark$ in $\epsilon(e)$. While the notation is somewhat opaque at this level of generality, in specific instances the map ℓ amounts to a familiar transition structure.

▶ **Example 20.** The star fragment of ACF from Example 3 and Example 7 coincides with GKAT, the algebra of programs introduced in [24] and studied further in [44, 40]. Instantiating $\mathtt{SExp}$ in this context reveals the syntax

$$e_i ::= 0 \mid \mathtt{1} \mid a \mid e_1 +_b e_2 \mid e_1 e_2 \mid e^{(b)}$$

for $b \subseteq \mathtt{At}$ and $a \in A$. This is nearly the syntax of GKAT, the only difference being the presence of $\mathtt{1}$ and 0 instead of Boolean constants $b \subseteq \mathtt{At}$. This is merely cosmetic, as we can just as well define $b := \mathtt{1} +_b 0$.

In this context, $M = (1 + \mathrm{Id})^{\mathtt{At}}$, and so $L_M \cong (2 + A \times \mathrm{Id})^{\mathtt{At}}$, which is the precise coalgebraic signature of the automaton models of GKAT expressions. It is readily checked that the operational semantics of GKAT also coincides with the operational semantics of the star fragment of ACF given above.

▶ **Example 21.** The star fragment of APA from Example 4 and Example 8 is a subset of the calculus of programs introduced in [11], but with an iteration operator for each $p \in [0, 1]$. Instantiating $\mathtt{SExp}$ in this context reveals the syntax

$$e_i ::= 0 \mid \mathtt{1} \mid a \mid e_1 +_p e_2 \mid e_1 e_2 \mid e^{(p)}$$

for $p \in [0, 1]$ and $a \in A$. The process $e^{(p)}$ can be thought of as a generalised Bernoulli process that runs e until it reaches $\checkmark$ and then flips a weighted coin to decide whether to start from the beginning of e or to terminate successfully.

We now provide a candidate axiomatisation for the star fragment while leaving the question of completeness open. Say that a star expression e is *guarded* if the unit is guarded in e as an expression in $\mathtt{Exp}$. We define E^* to be the theory consisting of E, the axiom schema

$$(\mathsf{E}^*1) \quad 1e = e1 = e \qquad (\mathsf{E}^*3) \quad e_1(e_2 e_3) = (e_1 e_2)e_3$$
$$(\mathsf{E}^*2) \quad ce = c \qquad (\mathsf{E}^*4) \quad (e +_\tau 1)^{(\sigma)} = (e +_\tau 0)^{(\sigma)}$$

[7] But not on the left-hand side! Observe the difference between the processes $a(b + c)$ and $ab + ac$ here.
[8] But not $e0 = 0$! See also the previous footnote.

and the inference rules

$$(\mathsf{E}^*5)\quad \frac{e \text{ is guarded}}{e^{(\sigma)} = ee^{(\sigma)} +_\sigma 1} \qquad (\mathsf{E}^*6)\quad \frac{g = eg +_\sigma f \qquad e \text{ is guarded}}{g = e^{(\sigma)} f}$$

In the specific cases where $\mathsf{E} = \mathsf{SL}$ and $\mathsf{E} = \mathsf{GS}$, E^* is equivalent to the candidate axiomatisations for the star fragments of ARB [30] and ACF [44, 40].[9]

There is a difference between our axioms and the axioms in [30, 44, 40]: instead of (E^*5), all equations of the form $e^{(\sigma)} = ee^{(\sigma)} +_\sigma 1$ appear in loc cit, even those where e is unguarded. We adopt (E^*5) instead because the unrestricted version of (E^*5) fails to be sound for the star expressions of APA. For example, if $e = 1 +_{\frac{1}{3}} a$, then $ee^{(1/2)} +_{\frac{1}{2}} 1 \xrightarrow{7/12} \checkmark$ while $e^{(\frac{1}{2})} \xrightarrow{1/2} \checkmark$.

We are confident that a completeness result can be obtained in several instances of the framework for the axiomatisation we have suggested above. However, in several cases this cannot happen. For example, there is no way to derive the identity $((a +_{\frac{1}{2}} 1) + b)^* = ((a +_{\frac{1}{2}} 0) + b)^*$ from CS^* (see Example 5) despite these expressions being behaviourally equivalent. What is likely missing from CS is a number of axioms that would allow 1 to be moved to the top level of every S-term (and then replaced by 0 using (E^*4)). Algebraic theories where this is doable are called *skew-associative*, which we define formally as follows.

▶ **Definition 22.** *An algebraic theory (S, E) consisting of constants and binary operations is called* skew-associative *if for any pair of binary operations σ_1, τ_1, there is a pair of binary operations σ_2, τ_2 such that $\sigma_1(x, \tau_1(y, z)) = \tau_2(\sigma_2(x, y), z)$ appears in E.*

Many of the examples we care about are skew-associative, including the theories of semilattices, guarded semilattices, and convex algebras.

▶ **Question 1.** *Assume (S, E) is a skew-associative algebraic theory. If e and f are behaviourally equivalent star expressions, is it true that $\mathsf{E}^* \vdash e = f$?*

7 Related Work

Our framework can be seen as a generalisation of Milner's ARB [30] that reaches beyond nondeterministic choice and covers several other process algebras already identified in the literature. For example, instantiating our framework in the theory of pointed convex algebras produces the algebra we have called APA (see Example 8), which only differs from the algebra PE of Stark and Smolka [45] in the axiom (R1). In loc cit, the requirement that the variable be guarded in the recursed expression is absent because recursion is computed as a least fixed point in their semantics. This is not how we interpret recursion. We have included the guardedness requirement because it is necessary for the soundness of the axiom in our semantics: for example, where $e = u +_{\frac{1}{2}} v$, we have $\mu v\ e \xrightarrow{1/2} u$ and $e[\mu v\ e/v] \xrightarrow{3/4} u$. In contrast, both $\mu v\ e$ and $e[\mu v\ e/v]$ exit in u with probability 1 in [45].

For another example, instantiating our framework in the theory CS of pointed convex semilattices gives ANP (see Example 5), which differs from the calculus of Mislove, Ouaknine, and Worrell [31] on two points. Firstly, their axiomatisation contains an unguarded version of (R1), like in [45]. Secondly, the underlying algebraic theory of [31] corresponds to CS extended with the axiom $x +_p 0 = 0$. The resulting theory is known in the literature as that of *convex semilattices with top* [11].

[9] See [42, Appendix F] for details.

Star expressions for non-deterministic processes appeared in the work of Milner [30] as a fragment of ARB and can be thought of as a bisimulation-focused analogue of Kleene's regular expressions for NFAs. While the syntaxes of Milner's star expressions and Kleene's regular expressions are the same, there are several important differences between their interpretations. For example, sequential composition is interpreted as the variable substitution $ef := e[f/\underline{u}]$ in Milner's paper, which fails to distribute over $+$ on the left. A notable insight from [30] is that, despite these differences, an iteration operator $(-)^*$ can be defined for Milner's star expressions that satisfies many of the same identities as the Kleene star. Given a variable v distinct from the unit and a process term e of ARB in which at most the unit is free,

$$e^* = \mu v\ (e[v/\underline{u}] + \underline{u})$$

defines the iteration operator in Milner's star fragment of ARB. In Section 6, we generalised this construction of Milner for the more general process types that we considered in this paper. Our proposed axiomatization is also inspired by Milner's work. We expect completeness of our general calculus will be a hard problem, as completeness in the instantiation to ARB was open for decades despite the extensive literature on the subject [15, 14, 16, 3, 20].

There are clear parallels between our work and the thesis of Silva [43], in which a family of calculi is introduced that includes one-exit versions of ARB, ACF, and APA (see Examples 2, 7, and 8). The main difference is that our framework is parametric on a finitary monad on **Sets** whereas Silva's is centered around one particular theory (semilattices). However, her work considers general polynomial functors on **Sets**, which we have not yet done in our paper. We could achieve a similar level of generality by replacing $A \times \mathrm{Id}$ in our signatures Σ_M, B_M, and L_M with an arbitrary polynomial functor.

Our results are also in the same vein as the work of Myers on coalgebraic expressions [32]. Coalgebraic expressions generalise the calculi of [43] to arbitrary finitary coalgebraic signatures on a variety of algebras, and furthermore have totally defined recursion operators similar to ours. However, the focus of the framework of coalgebraic expressions is on language semantics, achieved by lifting the coalgebraic signature to a variety. This distinguishes the framework from our approach: we focus on bisimulation semantics. This focus is also the reason we interpret our B_M-coalgebras in **Sets** and not in the Kleisli category of the monad M, as is done in [22] to capture trace semantics of coalgebras.

Finally, there is also a notable connection to the iterative theories of Elgot [13, 6, 7, 33]. Theorem 18 in particular implies that our process algebras are examples of iterative algebras.

8 Future Work

In this paper, we introduced a family of process types whose branching structure is determined by an algebraic theory. We provided each process type with a fully expressive specification language paired with a sound and complete axiomatisation of behavioural equivalence.

There are several instantiations of our framework that we have not yet explored and are of interest. For example, processes with multiset branching given by the theory of commutative monoids produces nondeterministic processes with a simplistic notion of resources. Another example is nondeterministic weighted processes with branching captured by the monad arising from the weak distributive law between the free semimodule and powerset monads [9]. Yet another instantiation arises from the theory of monoids (presenting the list monad), which produces processes related to breadth-first search algorithms.

Star fragments offer a uniform construction of Kleene-like algebras for a variety of paradigms of computing. However, our framework does not suggest an axiomatisation of the star fragment that combines nondeterministic and probabilistic choice, as the theory CS is

not skew-associative (see Definition 22). We would like to expand our framework to include this fragment as it provides an interesting but nonstandard interpretation of a part of the language ProbNetKAT used to verify probabilistic networks [17].

We would also like to investigate the question at the end of Section 6 of whether E^* is complete for skew-associative theories. In particular, we believe that a connection can be made to the work of Grabmayer and Fokkink [20] on LLEE-charts, which provides a completeness theorem for the so-called 1-free expressions of the star fragment of ARB. Our process algebras also have uniformly defined 1-free star fragments, and it is not difficult to give 1-free versions of the axiomatisation E^*. We intend to suitably generalise LLEE-charts to arbitrary skew-associative theories and prove completeness theorems for 1-free star fragments.

Finally, we would like to know whether our operational semantics for process terms is an instance of the mathematical operational semantics introduced by Turi and Plotkin [48].

References

1 Suzana Andova. Process algebra with probabilistic choice. In Joost-Pieter Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany, May 26-28, 1999. Proceedings*, volume 1601 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 1999. doi:10.1007/3-540-48778-6_7.

2 Jos C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, 2005. doi:10.1016/j.tcs.2004.07.036.

3 Jos C. M. Baeten, Flavio Corradini, and Clemens Grabmayer. On the star height of regular expressions under bisimulation (extended abstract), 2006.

4 Michael Barr. Terminal coalgebras in well-founded set theory. *Theor. Comput. Sci.*, 114(2):299–315, 1993. doi:10.1016/0304-3975(93)90076-6.

5 Jan A. Bergstra and Jan Willem Klop. Process theory based on bisimulation semantics. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 50–122. Springer, 1988. doi:10.1007/BFb0013021.

6 Stephen L. Bloom and Calvin C. Elgot. The existence and construction of free iterative theories. *J. Comput. Syst. Sci.*, 12(3):305–318, 1976. doi:10.1016/S0022-0000(76)80003-7.

7 Stephen L. Bloom and Zoltán Ésik. Varieties of iteration theories. *SIAM J. Comput.*, 17(5):939–966, 1988. doi:10.1137/0217059.

8 Stephen L. Bloom and Ralph Tindell. Varieties of "if-then-else". *SIAM J. Comput.*, 12(4):677–707, 1983. doi:10.1137/0212047.

9 Filippo Bonchi and Alessio Santamaria. Combining semilattices and semimodules. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 102–123. Springer, 2021. doi:10.1007/978-3-030-71995-1_6.

10 Filippo Bonchi, Alexandra Silva, and Ana Sokolova. The power of convex algebras. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.23.

11 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. The theory of traces for systems with nondeterminism and probability. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–14. IEEE, 2019. doi:10.1109/LICS.2019.8785673.

12 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. Presenting Convex Sets of Probability Distributions by Convex Semilattices and Unique Bases. In Fabio Gadducci and Alexandra Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, volume 211 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CALCO.2021.11.

13 Calvin C. Elgot. Monadic computation and iterative algebraic theories. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 175–230. Elsevier, 1975. doi:10.1016/S0049-237X(08)71949-9.

14 Wan Fokkink. Axiomatizations for the perpetual loop in process algebra. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, pages 571–581, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

15 Wan J. Fokkink and Hans Zantema. Basic process algebra with iteration: Completeness of its equational axioms. *Comput. J.*, 37(4):259–268, 1994. doi:10.1093/comjnl/37.4.259.

16 Wan J. Fokkink and Hans Zantema. Termination modulo equations by abstract commutation with an application to iteration. *Theor. Comput. Sci.*, 177(2):407–423, 1997. doi:10.1016/S0304-3975(96)00254-X.

17 Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic netkat. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 282–309. Springer, 2016. doi:10.1007/978-3-662-49498-1_12.

18 Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, 1977. doi:10.1145/321992.321997.

19 Clemens Grabmayer. A coinductive version of milner's proof system for regular expressions modulo bisimilarity. In Fabio Gadducci and Alexandra Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science, CALCO 2021, August 31 to September 3, 2021, Salzburg, Austria*, volume 211 of *LIPIcs*, pages 16:1–16:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CALCO.2021.16.

20 Clemens Grabmayer and Wan J. Fokkink. A complete proof system for 1-free regular expressions modulo bisimilarity. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 465–478. ACM, 2020. doi:10.1145/3373718.3394744.

21 H. Peter Gumm. Elements of the general theory of coalgebras. *LUATCS'99, Rand Afrikaans University, Johannesburg*, 1999.

22 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Log. Methods Comput. Sci.*, 3(4), 2007. doi:10.2168/LMCS-3(4:11)2007.

23 Bart Jacobs. Convexity, duality and effects. In Cristian S. Calude and Vladimiro Sassone, editors, *Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 1–19. Springer, 2010. doi:10.1007/978-3-642-15240-5_1.

24 Dexter Kozen and Wei-Lung Dustin Tseng. The böhm-jacopini theorem is false, propositionally. In Philippe Audebaud and Christine Paulin-Mohring, editors, *Mathematics of Program Construction, 9th International Conference, MPC 2008, Marseille, France, July 15-18, 2008. Proceedings*, volume 5133 of *Lecture Notes in Computer Science*, pages 177–192. Springer, 2008. doi:10.1007/978-3-540-70594-9_11.

25 Joachim Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103(2):151–161, 1968.

26 Ernest G. Manes. *Algebraic Theories*. Graduate Texts in Mathematics. Springer-Verlag New York, 1 edition, 1976. doi:10.1007/978-1-4612-9860-1.

27 Ernest G. Manes. Equations for if-then-else. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 7th International Conference, Pittsburgh, PA, USA, March 25-28, 1991, Proceedings*, volume 598 of *Lecture Notes in Computer Science*, pages 446–456. Springer, 1991. doi:10.1007/3-540-55511-0_23.

28 John McCarthy. A basis for a mathematical theory of computation, preliminary report. In Walter F. Bauer, editor, *Papers presented at the 1961 western joint IRE-AIEE-ACM computer conference, IRE-AIEE-ACM 1961 (Western), Los Angeles, California, USA, May 9-11, 1961*, pages 225–238. ACM, 1961. doi:10.1145/1460690.1460715.

29 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. doi:10.1007/3-540-10235-3.

30 Robin Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

31 Michael W. Mislove, Joël Ouaknine, and James Worrell. Axioms for probability and nondeterminism. In Flavio Corradini and Uwe Nestmann, editors, *Proceedings of the 10th International Workshop on Expressiveness in Concurrency, EXPRESS 2003, Marseille, France, September 2, 2003*, volume 96 of *Electronic Notes in Theoretical Computer Science*, pages 7–28. Elsevier, 2003. doi:10.1016/j.entcs.2004.04.019.

32 Robert S. R. Myers. Coalgebraic expressions. In Ralph Matthes and Tarmo Uustalu, editors, *6th Workshop on Fixed Points in Computer Science, FICS 2009, Coimbra, Portugal, September 12-13, 2009*, pages 61–69. Institute of Cybernetics, 2009. URL: http://cs.ioc.ee/fics09/proceedings/contrib8.pdf.

33 Evelyn Nelson. Iterative algebras. *Theor. Comput. Sci.*, 25:67–94, 1983. doi:10.1016/0304-3975(83)90014-2.

34 Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139, 2004.

35 D. Pumplün and Helmut Röhrl. Convexity theories IV. klein-hilbert parts in convex modules. *Appl. Categorical Struct.*, 3(2):173–200, 1995. doi:10.1007/BF00877635.

36 Jan J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 194–218. Springer, 1998. doi:10.1007/BFb0055624.

37 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

38 Jan J. M. M. Rutten and Daniele Turi. On the foundations of final semantics: Non-standard sets, metric spaces, partial orders. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Semantics: Foundations and Applications*, pages 477–530, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

39 Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.

40 Todd Schmid, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded kleene algebra with tests: Coequations, coinduction, and completeness. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 142:1–142:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.142.

41 Todd Schmid, Jurriaan Rot, and Alexandra Silva. On star expressions and coalgebraic completeness theorems. In Ana Sokolova, editor, *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics, MFPS 2021, Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021*, volume 351 of *EPTCS*, pages 242–259, 2021. doi:10.4204/EPTCS.351.15.

42 Todd Schmid, Wojciech Rozowski, Alexandra Silva, and Jurriaan Rot. Processes parametrised by an algebraic theory (with appendix), 2022. arXiv:2202.06901.

43 Alexandra Silva. *Kleene coalgebra*. PhD thesis, University of Nijmegen, 2010.

44 Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proc. ACM Program. Lang.*, 4(POPL):61:1–61:28, 2020. doi:10.1145/3371129.

45 Eugene W. Stark and Scott A. Smolka. A complete axiom system for finite-state probabilistic processes. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 571–596. The MIT Press, 2000.

46 Marshall Harvey Stone. Postulates for the barycentric calculus. *Annali di Matematica Pura ed Applicata*, 29(1):25–30, 1949.

47 Tadeusz Świrszcz. Monadic functors and convexity. *Bulletin de L'Académie Polonaise des Sciences*, XXII(1):39–42, 1974.

48 Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 280–291. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614955.

49 Daniele Varacca and Glynn Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006. doi:10.1017/S0960129505005074.

The Dimension Spectrum Conjecture for Planar Lines

D. M. Stull ✉
Department of Computer Science, Northwestern University, Evanston, IL, USA

Abstract

Let $L_{a,b}$ be a line in the Euclidean plane with slope a and intercept b. The dimension spectrum $\mathrm{sp}(L_{a,b})$ is the set of all effective dimensions of individual points on $L_{a,b}$. Jack Lutz, in the early 2000s posed the *dimension spectrum conjecture*. This conjecture states that, for every line $L_{a,b}$, the spectrum of $L_{a,b}$ contains a unit interval.

In this paper we prove that the dimension spectrum conjecture is true. Specifically, let (a, b) be a slope-intercept pair, and let $d = \min\{\dim(a, b), 1\}$. For every $s \in [0, 1]$, we construct a point x such that $\dim(x, ax + b) = d + s$. Thus, we show that $\mathrm{sp}(L_{a,b})$ contains the interval $[d, 1 + d]$.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Algorithmic randomness, Kolmogorov complexity, effective dimension

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.133

Category Track B: Automata, Logic, Semantics, and Theory of Programming

1 Introduction

The effective dimension, $\dim(x)$, of a point $x \in \mathbb{R}^n$ gives a fine-grained measure of the algorithmic randomness of x. Effective dimension was first defined by J. Lutz [5], and was originally used to quantify the sizes of complexity classes. Unsurprisingly, because of its strong connection to (classical) Hausdorff dimension, effective dimension has proven to be geometrically meaningful [3, 15, 1, 9]. Indeed, an exciting line of research has shown that one can prove classical results in geometric measure theory using effective dimension [7, 10, 11, 13]. Importantly, these are not effectivizations of known results, but new results whose *proofs* rely on effective methods. Thus, it is of considerable interest to investigate the effective dimensions of points of geometric objects such as lines.

Let $L_{a,b}$ be a line in the Euclidean plane with slope a and intercept b. Given the point-wise nature of effective dimension, one can study the *dimension spectrum* of $L_{a,b}$. That is, the set

$$\mathrm{sp}(L_{a,b}) = \{\dim(x, ax + b) \,|\, x \in \mathbb{R}\}$$

of all effective dimensions of points on $L_{a,b}$. In the early 2000s, Jack Lutz posed the *dimension spectrum conjecture* for lines. That is, he conjectured that the dimension spectrum of every line in the plane contains a unit interval.

The first progress on this conjecture was made by Turetsky.

▶ **Theorem 1** (Turetsky [18]). *The set of points $x \in \mathbb{R}^n$ with $\dim(x) = 1$ is connected.*

This immediately implies that $1 \in \mathrm{sp}(L_{a,b})$ for every line $L_{a,b}$. The next progress on the dimension spectrum conjecture was by Lutz and Stull [11]. They showed that the effective dimension of points on a line is intimately connected to problems in fractal geometry. Among other things, they proved that $1+d \in \mathrm{sp}(L_{a,b})$ for every line $L_{a,b}$, where $d = \min\{\dim(a, b), 1\}$. Shortly thereafter, Lutz and Stull [12] proved the dimension spectrum conjecture for the special case where the effective dimension and strong dimension of (a, b) agree.

In this paper, we prove that dimension spectrum conjecture is true. For every $s \in (0,1)$, we construct a point x such that $\dim(x, ax+b) = d + s$, where $d = \min\{\dim(a,b), 1\}$. This, combined with the results of Lutz and Stull, imply that

$$[d, 1+d] \subseteq \mathrm{sp}(L_{a,b}),$$

for every planar line $L_{a,b}$. The proof of the conjecture builds on the techniques of [11]. The primary difficulty of the conjecture is the case when the dimension of x is less than the difficulty of the line (a,b). We expand on the nature of this $\dim(x) < \dim(a,b)$ obstacle in Section 3.1. Our main technical contribution is showing how to overcome this difficulty by encoding the information of a into our point x. Further complications arise in the "high-dimensional" case, i.e., when $\dim(a,b) > 1$. In this case, we combine the encoding idea with a non-constructive argument.

Apart from its intrinsic interest, recent work has shown that the effective dimensions of points has deep connections to problems in classical analysis [10, 11, 13, 17, 8]. Lutz and Lutz [7] proved the point-to-set principle, which characterizes the Hausdorff dimension of a *set* by effective dimension of its *individual points*. Lutz and Stull [11], using the point-to-set principle, showed that lower bounds on the *effective* dimensions of points on a line are intimately related to well-known problems of classical geometric measure theory such the Kakeya and Furstenberg conjectures.

The structure of the paper is as follows. In Section 2, we recall the basic definitions and results of Kolmogorov complexity and effective dimension we need. In Section 3, we recall the strategy of Lutz and Stull [11] to give strong lower bounds on the effective dimension of points on a line. In Sections 3 and 3.1 we give intuition about this strategy, and discuss why it is not enough to settle the dimension spectrum conjecture.

In Section 4, we prove the dimension spectrum conjecture for lines with effective dimension at most one. We also give a brief overview of this proof, and how it overcomes the strategy discussed in Section 3. In Section 5, we prove the dimension spectrum conjecture for lines with effective dimension greater than one. We also give intuition of this proof, and how it overcomes the difficulties when the line is high-dimensional.

Finally, in the conclusion, we discuss open questions and avenues for future research.

2 Preliminaries

The *conditional Kolmogorov complexity* of a binary string $\sigma \in \{0,1\}^*$ given binary string $\tau \in \{0,1\}^*$ is

$$K(\sigma|\tau) = \min_{\pi \in \{0,1\}^*} \{\ell(\pi) : U(\pi, \tau) = \sigma\} ,$$

where U is a fixed universal prefix-free Turing machine and $\ell(\pi)$ is the length of π. The *Kolmogorov complexity* of σ is $K(\sigma) = K(\sigma|\lambda)$, where λ is the empty string. Thus, the Kolmogorov complexity of a string σ is the minimum length program which, when run on a universal Turing machine, eventually halts and outputs σ. We stress that the choice of universal machine effects the Kolmogorov complexity by at most an additive constant (which, especially for our purposes, can be safely ignored). See [4, 16, 2] for a more comprehensive overview of Kolmogorov complexity.

We can extend these definitions to Euclidean spaces by introducing "precision" parameters [9, 7]. Let $x \in \mathbb{R}^m$, and $r, s \in \mathbb{N}$. The *Kolmogorov complexity of x at precision r* is

$$K_r(x) = \min \{K(p) \,:\, p \in B_{2^{-r}}(x) \cap \mathbb{Q}^m\} .$$

The *conditional Kolmogorov complexity of x at precision r given $q \in \mathbb{Q}^m$* is

$$\hat{K}_r(x|q) = \min\{K(p) : p \in B_{2^{-r}}(x) \cap \mathbb{Q}^m\} .$$

The *conditional Kolmogorov complexity of x at precision r given $y \in \mathbb{R}^n$ at precision s* is

$$K_{r,s}(x|y) = \max\left\{\hat{K}_r(x|q) : q \in B_{2^{-r}}(y) \cap \mathbb{Q}^n\right\} .$$

We abbreviate $K_{r,r}(x|y)$ by $K_r(x|y)$.

The *effective Hausdorff dimension* and *effective packing dimension*[1] of a point $x \in \mathbb{R}^n$ are

$$\dim(x) = \liminf_{r\to\infty} \frac{K_r(x)}{r} \quad \text{and} \quad \mathrm{Dim}(x) = \limsup_{r\to\infty} \frac{K_r(x)}{r} .$$

Intuitively, these dimensions measure the density of algorithmic information in the point x.

By letting the underlying fixed prefix-free Turing machine U be a universal *oracle* machine, we may *relativize* the definition in this section to an arbitrary oracle set $A \subseteq \mathbb{N}$. The definitions of $K_r^A(x)$, $\dim^A(x)$, $\mathrm{Dim}^A(x)$, etc. are then all identical to their unrelativized versions, except that U is given oracle access to A. Note that taking oracles as subsets of the naturals is quite general. We can, and frequently do, encode a point y into an oracle, and consider the complexity of a point *relative* to y. In these cases, we typically forgo explicitly referring to this encoding, and write e.g. $K_r^y(x)$.

Among the most used results in algorithmic information theory is the *symmetry of information*. In Euclidean spaces, this was first proved, in a slightly weaker form in [7], and in the form presented below in [11].

▶ **Lemma 2.** *For every $m, n \in \mathbb{N}$, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, and $r, s \in \mathbb{N}$ with $r \geq s$,*

(i) $\left|K_r(x|y) + K_r(y) - K_r(x,y)\right| \leq O(\log r) + O(\log\log\|y\|)$.

(ii) $|K_{r,s}(x|x) + K_s(x) - K_r(x)| \leq O(\log r) + O(\log\log\|x\|)$.

2.1 Initial segments versus K-optimizing rationals

For $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and a precision $r \in \mathbb{N}$, let $x\restriction r = (x_1\restriction r, \ldots, x_n\restriction r)$, where each

$$x_i\restriction r = 2^{-r}\lfloor 2^r x_i \rfloor$$

is the truncation of x_i to r bits to the right of the binary point. For $r \in (0,\infty)$, let $x\restriction r = x\restriction\lceil r\rceil$.

We can relate the complexity $K_r(x)$ of x at precision r and the *initial segment complexity* $K(x\restriction r)$ of the binary representation of x. Lutz and Stull [11] proved the following lemma, and its corollaries, relating these two quantities. Informally, it shows that, up to a logarithmic error, the two quantities are equivalent.

▶ **Lemma 3.** *For every $m, n \in \mathbb{N}$, there is a constant c such that for all $x \in \mathbb{R}^m$, $p \in \mathbb{Q}^n$, and $r \in \mathbb{N}$,*

$$|\hat{K}_r(x|p) - K(x\restriction r \,|\, p)| \leq K(r) + c .$$

This has the following two useful corollaries.

[1] Although effective Hausdorff was originally defined by J. Lutz [6] using martingales, it was later shown by Mayordomo [14] that the definition used here is equivalent. For more details on the history of connections between Hausdorff dimension and Kolmogorov complexity, see [2, 15].

▶ **Corollary 4.** *For every $m \in \mathbb{N}$, there is a constant c such that for every $x \in \mathbb{R}^m$ and $r \in \mathbb{N}$,*

$$|K_r(x) - K(x\restriction r)| \leq K(r) + c\,.$$

▶ **Corollary 5.** *For every $m, n \in \mathbb{N}$, there is a constant c such that for all $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, and $r, s \in \mathbb{N}$,*

$$|K_{r,s}(x|y) - K(x\restriction r \,|\, y\restriction s)| \leq K(r) + K(s) + c\,.$$

3 Previous Work

The proof of our main theorem will use the tools and techniques introduced by Lutz and Stull [11]. In this section we will state the main lemmas needed for this paper. We will devote some time giving intuition about each lemma. In Subsection 3.1, we give an informal discussion on how to combine these lemmas to give bounds on the effective dimensions of points on a line. We will also discuss where these tools break down, motivating the techniques introduced in this paper.

The first lemma, informally, states the following. Suppose that $L_{a,b}$ intersects $(x, ax+b)$ and the complexity of (a,b) is low (item (i)). Further assume that (item (ii)), if $L_{u,v}$ is any other line intersecting $(x, ax+b)$ such that $\|(a,b) - (u,v)\| < 2^{-m}$ then either

1. u, v is of high complexity, or
2. u, v is very close to a, b.

Then it is possible to compute an approximation of (a,b) given an approximation of $(x, ax+b)$ and first m bits of (a,b). Indeed, we can simply enumerate over all low complexity lines, since we know that the only candidate is essentially (a,b).

▶ **Lemma 6** (Lutz and Stull [11])**.** *Suppose that $A \subseteq \mathbb{N}$,$a, b, x \in \mathbb{R}$, $m, r \in \mathbb{N}$, $\delta \in \mathbb{R}_+$, and $\varepsilon, \eta \in \mathbb{Q}_+$ satisfy $r \geq \log(2|a| + |x| + 5) + 1$ and the following conditions.*

(i) *$K_r^A(a,b) \leq (\eta + \varepsilon)\, r$.*

(ii) *For every $(u,v) \in B_{2^{-m}}(a,b)$ such that $ux + v = ax + b$,*

$$K_r^A(u,v) \geq (\eta - \varepsilon)\, r + \delta \cdot (r - t)\,,$$

whenever $t = -\log \|(a,b) - (u,v)\| \in (0, r]$.

Then,

$$K_r^A(a,b,x) \leq K_r(x, ax+b) + K_{m,r}(a,b \,|\, x, ax+b) + \frac{4\varepsilon}{\delta} r + K(\varepsilon, \eta) + O(\log r)\,.$$

The second lemma which will be important in proving our main theorem is the following. It is essentially the approximation version of the simple geometric fact that any two lines intersect at a single point. In other words, if $ax + b = ux + v$ and you are given an approximation of (a,b) and an approximation of (u,v), then you can compute an approximation of x. Moreover, the quality of the approximation of x depends linearly on the distance between (u,v) and (a,b).

▶ **Lemma 7** ([11])**.** *Let $a, b, x \in \mathbb{R}$. For all $u, v \in B_1(a,b)$ such that $ux + v = ax + b$, and for all $r \geq t := -\log \|(a,b) - (u,v)\|$,*

$$K_r(u,v) \geq K_t(a,b) + K_{r-t,r}(x|a,b) - O(\log r)\,.$$

The primary function of this lemma is to give a lower bound on the complexity of any line intersecting $(x, ax+b)$, i.e., ensuring condition (ii) of the previous lemma.

Finally, we also need the following oracle construction of Lutz and Stull. The purpose of this lemma is to show that we can lower the complexity of our line (a, b), thus ensuring item (i) of Lemma 6. Crucially, we can lower this complexity using only the information contained in (a, b).

▶ **Lemma 8** ([11]). *Let $r \in \mathbb{N}$, $z \in \mathbb{R}^2$, and $\eta \in \mathbb{Q} \cap [0, \dim(z)]$. Then there is an oracle $D = D(r, z, \eta)$ satisfying*

- **(i)** *For every $t \leq r$, $K_t^D(z) = \min\{\eta r, K_t(z)\} + O(\log r)$.*
- **(ii)** *For every $m, t \in \mathbb{N}$ and $y \in \mathbb{R}^m$, $K_{t,r}^D(y|z) = K_{t,r}(y|z) + O(\log r)$ and $K_t^{z,D}(y) = K_t^z(y) + O(\log r)$.*

3.1 Combining the lemmas

We now briefly discuss the strategy of [11] which combines the above lemmas to give non-trivial bounds on the effective dimension of points on a line. Suppose (a, b) is a line with $\dim(a, b) = d$, and x is a point with $\dim^{a,b}(x) = s$. We will also make the crucial assumption that $d \leq s$. Roughly, Lutz and Stull showed that, for sufficiently large r

$$K_r(x, ax + b) \geq (s + d)r.$$

The strategy is as follows. Note that to simplify the exposition, all inequalities in this discussion will be approximate. Using Lemma 8, we find an oracle D which reduces the complexity of (a, b) to some $\eta \leq d$, i.e., $K_r^D(a, b) = \eta r$. Combining this with Lemma 7, we get a lower bound on every line (u, v) intersecting $(x, ax + b)$. That is, we show for any such line,

$$K_r^D(u, v) \geq \eta t + s(r - t) - O(\log r)$$

By our choice of η, this implies that

$$K_r^D(u, v) > \eta r$$

In particular, relative to D, both conditions of Lemma 6 are satisfied and we have the sufficient lower bound.

In the previous sketch, it was crucial that the dimension of (a, b) was less than s, in order for the lower bound from Lemma 7 to be useful. In the case where $\dim(a, b)$ is much larger than $\dim^{a,b}(x)$, this strategy breaks down, and further techniques are required.

We also note that this seems to be a very deep issue. As discussed in the Introduction, the point-to-set principle of J. Lutz and N. Lutz [7] allows us to translate problems from (classical) geometric measure theory into problems of effective dimension. The same issue discussed in this section occurs when attacking the notorious Kakeya and Furstenberg set conjectures using the point-to-set principle. While resolving this obstacle in full generality is still elusive, we are able to get around it in the context of the Dimension Spectrum Conjecture.

4 Low-Dimensional Lines

In this section, we prove the spectrum conjecture for lines with $\dim(a, b) \leq 1$.

▶ **Theorem 9.** *Let $(a, b) \in \mathbb{R}^2$ be a slope-intercept pair with $\dim(a, b) \leq 1$. Then for every $s \in [0, 1]$, there is a point $x \in \mathbb{R}$ such that*

$$\dim(x, ax + b) = s + \dim(a, b).$$

We begin by giving an intuitive overview of the proof.

4.1 Overview of the proof

As mentioned in Section 3.1, the main obstacle of the Dimension Spectrum Conjecture occurs when the dimension of x is lower than the dimension of the line (a, b). As mentioned in Section 3.1, in general, this issue is still formidable. However, in the Dimension Spectrum Conjecture, we are given the freedom to specifically construct the point x, allowing us overcome this obstacle.

The most natural way to construct a sequence x with $\dim^{a,b}(x) = s$ is to start with a random sequence, and pad it with long strings of zeros. This simple construction, unfortunately, does not seem to work.

We are able to overcome the obstacle by padding the random sequence *with the bits of* a, instead of with zeros. Thus, given an approximation $(x, ax + b)$ we trivially have a decent approximation of a (formalized iin Lemma 10). This allows us, using Lemma 6, to restrict our search for (a, b) to a smaller set of candidate lines.

4.2 Proof for low-dimensional lines

Fix a slope-intercept pair (a, b), and let $d = \dim(a, b)$. Let $s \in (0, d)$. Let $y \in \mathbb{R}$ be random relative to (a, b). Thus, for every $r \in \mathbb{N}$,

$$K_r^{a,b}(y) \geq r - O(\log r).$$

Define the sequence of natural numbers $\{h_j\}_{j\in\mathbb{N}}$ inductively as follows. Define $h_0 = 1$. For every $j > 0$, let

$$h_j = \min\left\{h \geq 2^{h_{j-1}} : K_h(a, b) \leq \left(d + \frac{1}{j}\right) h\right\}.$$

Note that h_j always exists. For every $r \in \mathbb{N}$, let

$$x[r] = \begin{cases} a[r - \lfloor sh_j \rfloor] & \text{if } r \in (\lfloor sh_j \rfloor, h_j] \text{ for some } j \in \mathbb{N} \\ y[r] & \text{otherwise} \end{cases}$$

where $x[r]$ is the rth bit of x. Define $x \in \mathbb{R}$ to be the real number with this binary expansion.

One of the most important aspects of our construction is that we encode (a subset of) the information of a into our point x. This is formalized in the following lemma.

▶ **Lemma 10.** *For every $j \in \mathbb{N}$, and every r such that $sh_j < r \leq h_j$,*

$$K_{r-sh_j,r}(a, b \,|\, x, ax + b) \leq O(\log h_j).$$

Proof. By definition, the last $r - sh_j$ bits of x are equal to the first $r - sh_j$ bits of a. That is,

$$\begin{aligned} x[sh_j]\, x[sh_j + 1] \ldots x[r] &= a[0]\, a[1] \ldots a[r - sh_j] \\ &= a{\restriction}(r - sh_j). \end{aligned}$$

Therefore, since additional information cannot increase Kolmogorov complexity,

$$\begin{aligned} K_{r-sh_j,r}(a \,|\, x, ax + b) &\leq K_{r-sh_j,r}(a \,|\, x) \\ &\leq O(\log h_j). \end{aligned}$$

Note that, given $2^{-(r-sh_j)}$-approximations of a, x, and $ax + b$, it is possible to compute an approximation of b. That is,

$$K_{r-sh_j}(b \,|\, a, x, ax + b) \leq O(\log h_j).$$

Therefore, by Lemma 2 and the two above inequalities,

$$\begin{aligned} K_{r-sh_j,r}(a,b \,|\, x, ax+b) &= K_{r-sh_j,r}(a \,|\, x, ax+b) \\ &\quad + K_{r-sh_j,r}(b \,|\, a, x, ax+b) + O(\log r) \\ &\leq O(\log h_j) + K_{r-sh_j,r}(b \,|\, a, x, ax+b) + O(\log r) \\ &\leq O(\log h_j). \end{aligned}$$ ◀

The other important property of our construction is that (a, b) gives no information about x, beyond the information specifically encoded into x.

▶ **Lemma 11.** *For every $j \in \mathbb{N}$, the following hold.*

1. $K_t^{a,b}(x) \geq t - O(\log h_j)$, *for all* $t \leq sh_j$.
2. $K_r^{a,b}(x) \geq sh_j + r - h_j - O(\log h_j)$, *for all* $h_j \leq r \leq sh_{j+1}$.

Proof. We first prove item (1). Let $t \leq sh_j$. Then, by our construction of x, and choice of y,

$$\begin{aligned} K_t^{a,b}(x) &\geq K_t^{a,b}(y) - h_{j-1} - O(\log t) \\ &\geq t - O(\log t) - \log h_j - O(\log t) \\ &\geq t - O(\log h_j). \end{aligned}$$

For item (2), let $h_j \leq r \leq sh_{j+1}$. Then, by item (1), Lemma 2 and our construction of x,

$$\begin{aligned} K_r^{a,b}(x) &= K_{h_j}^{a,b}(x) + K_{r,h_j}^{a,b}(x) - O(\log r) && \text{[Lemma 2]} \\ &\geq sh_j + K_{r,h_j}^{a,b}(x) - O(\log r) && \text{[Item (1)]} \\ &\geq sh_j + K_{r,h_j}^{a,b}(y) - O(\log r) \\ &\geq sh_j + r - h_j - O(\log r), \end{aligned}$$ ◀

and the proof is complete.

We now prove bounds on the complexity of our constructed point. We break the proof into two parts. In the first, we give lower bounds on $K_r(x, ax+b)$ at precisions $sh_j < r \leq h_j$. Intuitively, the proof proceeds as follows. Since $r > sh_j$, given $(x, ax + b)$ to precision r immediately gives a 2^{-r+sh_j} approximation of (a, b). Thus, we only have to search for candidate lines (u, v) which satisfy $t = \|(a, b) - (u, v)\| < 2^{-r+sh_j}$. Then, because of the lower bound on t, the complexity $K_{r-t}(x)$ is maximal. In other words, we are essentially in the case that the complexity of x is high. Thus, we are able to use the method described in Section 3.1. We now formalize this intuition in the proof.

▶ **Lemma 12.** *For every $\gamma > 0$ and all sufficiently large $j \in \mathbb{N}$,*

$$K_r(x, ax+b) \geq (s+d)r - \gamma r,$$

for every $r \in (sh_j, h_j]$.

Proof. Let $\eta \in \mathbb{Q}$ such that

$$d - \gamma/4 < \eta < d - \gamma^2.$$

Let $\varepsilon \in \mathbb{Q}$ such that

$$\varepsilon < \gamma(d - \eta)/16.$$

Note that

$$\frac{4\varepsilon}{1-\eta} \leq \frac{\gamma}{4}$$

We also note that, since η and ε are constant,

$$K(\eta, \varepsilon) = O(1).$$

Let $D = D(r, (a,b), \eta)$ be the oracle of Lemma 8 and let $\delta = 1 - \eta$.

Let (u, v) be a line such that $t := \|(a,b) - (u,v)\| \geq r - sh_j$, and $ux + v = ax + b$. Note that $r - t \leq sh_j$. Then, by Lemma 7, Lemma 8 and Lemma 11(1),

$$\begin{aligned}
K_r^D(u,v) &\geq K_t^D(a,b) + K_{r-t,r}^D(x \,|\, a,b) - O(\log r) && \text{[Lemma 7]}\\
&\geq K_t^D(a,b) + K_{r-t,r}(x \,|\, a,b) - O(\log r) && \text{[Lemma 8]}\\
&\geq K_t^D(a,b) + r - t - O(\log r). && \text{[Lemma 11(1)]}
\end{aligned}$$

There are two cases by Lemma 8. For the first, assume that $K_t^D(a,b) = \eta r$. Then

$$\begin{aligned}
&\geq \eta r + r - t - O(\log r) && \text{[Definition of dim]}\\
&= (\eta - \varepsilon) r + r - t && [r \text{ is large}]\\
&\geq (\eta - \varepsilon) r + (1 - \eta)(r - t)
\end{aligned}$$

For the second, assume that $K_t^D(a,b) = K_t(a,b)$. Then

$$\begin{aligned}
K_r^D(u,v) &\geq K_t(a,b) + r - t - O(\log r)\\
&\geq dt - o(t) + r - t - O(\log r) && \text{[Definition of dim]}\\
&= \eta r + (1-\eta) r - t(1-d) - \varepsilon r && [r \text{ is large}]\\
&\geq \eta r - \varepsilon r + (1-\eta)(r-t) && [d > \eta]\\
&\geq (\eta - \varepsilon) r + (1-\eta)(r-t). && (1)
\end{aligned}$$

Therefore, in either case, we may apply Lemma 6,

$$\begin{aligned}
K_r(x, ax+b) &\geq K_r^D(a,b,x) - K_{r-sh_j,r}(a,b\,|, x, ax+b) && \text{[Lemma 6]}\\
&\qquad - \frac{4\varepsilon}{1-\eta} r - K(\eta,\varepsilon) - O(\log r)\\
&\geq K_r^D(a,b,x) - K_{r-sh_j,r}(a,b \,|\, x, ax+b) - \frac{\gamma}{4} r - \frac{\gamma}{8} r\\
&= K_r^D(a,b,x) - K_{r-sh_j,r}(a,b \,|\, x, ax+b) - \frac{3\gamma}{8} r. && (2)
\end{aligned}$$

By Lemma 11(1), our construction of oracle D, and the symmetry of information,

$$\begin{aligned}
K_r^D(a,b,x) &= K_r^D(a,b) + K_r^D(x \,|\, a,b) - O(\log r) && \text{[Lemma 2]}\\
&= K_r^D(a,b) + K_r(x \,|\, a,b) - O(\log r) && \text{[Lemma 8(ii)]}\\
&\geq \eta r + K_r(x \,|\, a,b) - O(\log r) && \text{[Lemma 8(i)]}\\
&\geq \eta r + sh_j - O(\log r) && \text{[Lemma 11(1)]}\\
&\geq \eta r + sh_j - \frac{\gamma}{4} r. && (3)
\end{aligned}$$

Finally, by Lemma 10,

$$K_{r-sh_j,r}(a,b \mid x, ax+b) \leq \frac{\gamma}{8}r. \tag{4}$$

Together, inequalities (2), (3) and (4) imply that

$$\begin{aligned}
K_r(x, ax+b) &\geq K_r^D(a,b,x) - K_{r-sh_j,r}(a,b\,|,x,ax+b) - \frac{3\gamma}{8}r \\
&\geq \eta r + sh_j - \frac{\gamma}{4}r - \frac{\gamma}{8}r - \frac{3\gamma}{8}r \\
&\geq dr - \frac{\gamma}{4}r + sh_j - \frac{3\gamma}{4}r \\
&\geq dr + sh_j - \gamma r \\
&\geq (s+d)r - \gamma r,
\end{aligned}$$

and the proof is complete. ◀

We now give lower bounds on the complexity of our point, $K_r(x, ax+b)$, when $h_j < r \leq sh_{j+1}$. Intuitively, the proof proeeds as follows. Using the previous lemma, we can, given a 2^{-h_j}-approximation of $(x, ax+b)$, compute a 2^{-h_j}-approximation of (a,b). Thus, we only have to compute the last $r-h_j$ bits of (a,b). Importantly, since $r > h_j$, the last $r-h_j$ bits of x are maximal. Hence, we can simply lower the complexity of the last $r-h_j$ bits of (a,b) to roughly $s(r-h_j)$. Thus, we are again, essentially, in the case where $\dim(x) \geq \dim(a,b)$ and the techniques of Section 3.1 work. We now formalize this intuition.

▶ **Lemma 13.** *For every $\gamma > 0$ and all sufficiently large $j \in \mathbb{N}$,*

$$K_r(x, ax+b) \geq (s+d)r - \gamma r,$$

for every $r \in (h_j, sh_{j+1}]$.

Proof. Recall that we are assuming that $s < d$. Let $\hat{s} \in \mathbb{Q} \cap (0,s)$ be a dyadic rational such that

$$\gamma/8 < s - \hat{s} < \gamma/4.$$

Let $\hat{d} \in \mathbb{Q} \cap (0, \dim(a,b))$ be a dyadic rational such that

$$\gamma/8 < \dim(a,b) - \hat{d} < \gamma/4.$$

Define

$$\alpha = \frac{s(r-h_j) + \dim(a,b)h_j}{r},$$

and $\eta \in \mathbb{Q} \cap (0, \alpha)$ by

$$\eta = \frac{\hat{s}(r-h_j) + \hat{d}h_j}{r}.$$

Finally, let $\varepsilon = \gamma^2/64$. Note that

$$\begin{aligned}
\alpha - \eta &= \frac{s(r-h_j) + dh_j - \hat{s}(r-h_j) - \hat{d}h_j}{r} \\
&= \frac{(s-\hat{s})(r-h_j) + (d-\hat{d})h_j}{r} \\
&\leq \frac{\frac{\gamma}{4}(r-h_j) + \frac{\gamma}{4}h_j}{r} \\
&= \frac{\gamma}{4}
\end{aligned} \tag{5}$$

Similarly,

$$\begin{aligned}\alpha - \eta &= \frac{s(r-h_j) + \dim(a,b)h_j - \hat{s}(r-h_j) - \hat{d}h_{j-1}}{r} \\ &= \frac{(s-\hat{s})(r-h_j) + (\dim(a,b) - \hat{d})h_j}{r} \\ &> \frac{\frac{\gamma}{8}(r-h_j) + \frac{\gamma}{8}h_j}{r} \\ &= \frac{\gamma}{8} \end{aligned} \tag{6}$$

In particular,

$$\frac{4\varepsilon}{\alpha - \eta} \leq \gamma/4. \tag{7}$$

We also note that

$$K(\varepsilon, \eta) \leq K(\gamma, \hat{s}, \hat{d}, r, h_j) \leq O(\log r), \tag{8}$$

since j was chosen to be sufficiently large and γ is constant.

Finally, let $D = D(r, (a,b), \eta)$ be the oracle of Lemma 8. Note that we chose D so that, roughly, D lowers the complexity of the last $r - h_j$ bits of (a,b) to $s(r-h_j)$.

Let (u,v) be a line such that $t := \|(a,b) - (u,v)\| \geq h_j$, and $ux + v = ax + b$. Then, by Lemmas 7, 8 and 11,

$$\begin{aligned} K_r^D(u,v) &\geq K_t^D(a,b) + K_{r-t,r}^D(x \,|\, a,b) - O(\log r) && \text{[Lemma 7]} \\ &\geq K_t^D(a,b) + K_{r-t,r}(x \,|\, a,b) - O(\log r) && \text{[Lemma 8]} \\ &\geq K_t^D(a,b) + s(r-t) - O(\log r). && \text{[Lemma 11(1)]} \end{aligned}$$

There are two cases. In the first, $K_t^D(a,b) = \eta r$. Then,

$$\begin{aligned} K_r^D(u,v) &\geq \eta r + s(r-t) - O(\log r) \\ &\geq (\eta - \varepsilon)r + s(r-t) \\ &\geq (\eta - \varepsilon)r + (\alpha - \eta)(r-t). \end{aligned}$$

In the other case, $K_t^D(a,b) = K_t(a,b)$. Then,

$$\begin{aligned} K_r^D(u,v) &\geq K_t(a,b) + s(r-t) - O(\log r) \\ &\geq dt - o(t) + s(r-t) - O(\log r) && \text{[Definition of dim]} \\ &= dh_j + d(t-h_j) + s(r-t) - o(r) \\ &= dh_j + d(t-h_j) + s(r-h_j) - s(t-h_j) - o(r) \\ &= \alpha r + (d-s)(t-h_j) - o(r) \\ &= \eta r + (\alpha - \eta)r + (d-s)(t-h_j) - o(r) \\ &\geq \eta r + (\alpha - \eta)(r-t) - o(r) \\ &\geq (\eta - \varepsilon)r + (\alpha - \eta)(r-t). \end{aligned}$$

Therefore we may apply Lemma 6, which yields

$$\begin{aligned}
K_r^D(a,b,x) &\leq K_r(x, ax+b) + K^D_{h_j,r}(a,b,x \,|\, x, ax+b) && \text{[Lemma 6]}\\
&\quad + \frac{4\varepsilon}{\alpha-\eta} r + K(\varepsilon,\eta) + O(\log r)\\
&\leq K_r(x, ax+b) + K^D_{h_j,r}(a,b,x \,|\, x, ax+b)\\
&\quad + \frac{\gamma}{4} r + \frac{\gamma}{8} r && \text{[Choice of } \eta, \varepsilon\text{]}\\
&= K_r(x, ax+b) + K^D_{h_j,r}(a,b,x \,|\, x, ax+b) + \frac{3\gamma}{8} r. && (9)
\end{aligned}$$

By Lemma 11, and our construction of oracle D,

$$\begin{aligned}
K_r^D(a,b,x) &= K_r^D(a,b) + K_r^D(x \,|\, a,b) - O(\log r) && \text{[Lemma 2]}\\
&= \eta r + K_r(x \,|\, a,b) - O(\log r) && \text{[Lemma 8]}\\
&\geq \eta r + s h_j + r - h_j - O(\log r) && \text{[Lemma 11(2)]}\\
&\geq \alpha r - \frac{\gamma}{4} r + s h_j + r - h_j - O(\log r)\\
&\geq s(r - h_j) + d h_j - \frac{\gamma}{4} r + s h_j + r - h_j - O(\log r)\\
&\geq (1+s) r - (1-d) h_j - \frac{\gamma}{4} r. && (10)
\end{aligned}$$

By Lemmas 12, and 2, and the fact that additional information cannot increase Kolmogorov complexity

$$\begin{aligned}
K_{h_j,r}(a,b,x \,|\, x, ax+b) &\leq K_{h_j,h_j}(a,b,x \,|\, x, ax+b)\\
&= K_{h_j}(a,b,x) - K_{h_j}(x, ax+b) && \text{[Lemma 2]}\\
&= K_{h_j}(a,b) + K_{h_j}(x \mid a,b)\\
&\qquad - K_{h_j}(x, ax+b) && \text{[Lemma 2]}\\
&= K_{h_j}(a,b) + s h_j - K_{h_j}(x, ax+b) && \text{[Lemma 11]}\\
&\leq K_{h_j}(a,b) + s h_j - (s+d) h_j + \frac{\gamma}{16} h_j && \text{[Lemma 12]}\\
&\leq d h_j + h_j/j - d h_j + \frac{\gamma}{16} r && \text{[Definition of } h_j\text{]}\\
&\leq \frac{\gamma}{8} r && (11)
\end{aligned}$$

Combining inequalities (9), (10) and (11) , we see that

$$\begin{aligned}
K_r(x, ax+b) &\geq K_r^D(a,b,x) - \frac{\gamma}{8} r - \frac{3\gamma}{8} r\\
&\geq (1+s) r - (1-d) h_j - \frac{\gamma}{4} r - \frac{\gamma}{4} r\\
&\geq (1+s) r - (1-d) h_j - \gamma r.
\end{aligned}$$

Note that, since $d \leq 1$, and $h_j \leq r$,

$$\begin{aligned}
(1+s) r - h_j(1-d) - (s+d) r &= r(1-d) - h_j(1-d)\\
&= (r - h_j)(1-d)\\
&\geq 0.
\end{aligned}$$

Thus,

$$\begin{aligned} K_r(x, ax+b) &\geq (1+s)r - h_j(1-d) - \gamma r \\ &\geq (s+d)r - \gamma r, \end{aligned}$$

and the proof is complete for the case $s < \dim(a,b)$. ◀

We are now able to prove our main theorem of this section.

▶ **Theorem 9.** *Let $(a,b) \in \mathbb{R}^2$ be a slope-intercept pair with $\dim(a,b) \leq 1$. Then for every $s \in [0,1]$, there is a point $x \in \mathbb{R}$ such that*

$$\dim(x, ax+b) = s + \dim(a,b).$$

Proof. Let $(a,b) \in \mathbb{R}^2$ be a slope-intercept pair with

$$d = \dim(a,b) \leq 1.$$

Let $s \in [0,1]$. If $s = 0$, then

$$\begin{aligned} K_r(a, a^2+b) &= K_r(a) + K_r(a^2+b \,|\, a) + O(\log r) \\ &= K_r(a) + K_r(b \,|\, a) + O(\log r) \\ &= K_r(a,b) + O(\log r), \end{aligned}$$

and so the conclusion holds.

If $s = 1$, then by [11], for any point x which is random relative to (a,b),

$$\dim(x, ax+b) = 1 + d,$$

and the claim follows.

If $s \geq d$, then Lutz and Stull [11] showed that for any x such that

$$\dim^{a,b}(x) = \dim(x) = s,$$

we have $\dim(x, ax+b) = s + d$.

Therefore, we may assume that $s \in (0,1)$ and $s < d$. Let x be the point constructed in this section. Let $\gamma > 0$. Let j be large enough so that the conclusions of Lemmas 12 and 13 hold for these choices of (a,b), x, s and γ. Then, by Lemmas 12 and 13,

$$\begin{aligned} \dim(x, ax+b) &= \liminf_{r\to\infty} \frac{K_r(x, ax+b)}{r} \\ &\geq \liminf_{r\to\infty} \frac{(s+d)r - \gamma r}{r} \\ &= s + d - \gamma. \end{aligned}$$

Since we chose γ arbitrarily, we see that

$$\dim(x, ax+b) \geq s + d.$$

For the upper bound, let $j \in \mathbb{N}$ be sufficiently large. Then

$$\begin{aligned} K_{h_j}(x, ax+b) &\leq K_{h_j}(x, a, b) \\ &= K_{h_j}(a,b) + K_{h_j}(x \,|\, a, b) \\ &\leq dh_j + sh_j \\ &= (d+s)h_j. \end{aligned}$$

Therefore,

$$\dim(x, ax+b) \leq s + d,$$

and the proof is complete. ◀

5 High-Dimensional Lines

In this section we prove the following theorem.

▶ **Theorem 14.** *Let $(a,b) \in \mathbb{R}^2$ be a slope-intercept pair with $\dim(a,b) > 1$.. Then for every $s \in [0,1]$, there is a point $x \in \mathbb{R}$ such that*

$$\dim(x, ax+b) = 1 + s.$$

5.1 Overview of proof

In this case, we again apply essential insight of the proof for low-dimensional lines, namely, encoding (a subset of) the information of a into x. However, when $\dim(a,b) > 1$ constructing x as before potentially causes a problem. Specifically, in this case, the previous construction might cause $\dim(x, ax+b)$ to become too large.

The overcome this, we rely on a non-constructive argument. More specifically, we begin as in the construction of x in the low-dimensional case. However at stage j of our construction, we do not add all $h_j - sh_j$ bits of a to x. Instead we consider the $m = h_j - sh_j$ strings $\mathbf{x_0}, \ldots, \mathbf{x_m}$, where

$$\mathbf{x_n}[i] = \begin{cases} 0 & \text{if } 0 \leq i < m-n \\ \frac{1}{a}[i-(m-n)] & \text{if } m-n \leq i \leq m \end{cases} \tag{*}$$

and look at the extension of x with the bits of $\mathbf{x_n}$.

Using a discrete, approximate, version of the intermediate value theorem, we are able to conclude that there is some extension $x' = x\mathbf{x_n}$ such that

$$\min_{sh_j \leq r \leq h_j} |K_r(x', ax'+b) - (1+s)r|$$

is sufficiently small. We then carry on with the argument of the low-dimensional lines until sh_{j+1}.

5.2 Proof for high-dimensional lines

In order to prove Theorem 14, we will, given any slope-intercept pair (a,b) and $s \in (0,1)$, construct a point $x \in [0,1]$ such that $\dim(x, ax+b) = 1+s$.

Our construction is best phrased as constructing an infinite binary sequence $\mathbf{x}$, and then taking x to be the unique real number whose binary expansion is $\mathbf{x}$. We now recall terminology needed in the construction. We will use bold variables to denote binary strings and (infinite) binary sequences. If $\mathbf{x}$ is a (finite) binary string and $\mathbf{y}$ is a binary string or sequence, we write $\mathbf{x} \prec \mathbf{y}$ if $\mathbf{x}$ is a prefix of $\mathbf{y}$.

Let (a,b) be a slope intercept pair and let $d = \dim(a,b)$. Define the sequence of natural numbers $\{h_j\}_{j \in \mathbb{N}}$ inductively as follows. Define $h_0 = 2$. For every $j > 0$, let

$$h_j = \min\left\{h \geq 2^{h_{j-1}} : K_h(a,b) \leq \left(d + 2^{-j}\right) h\right\}.$$

We define our sequence $\mathbf{x}$ inductively. Let $\mathbf{y}$ be a random, relative to (a,b), binary sequence. That is, there is some constant c such that

$$K^{a,b}(y\restriction r) \geq r - c, \tag{12}$$

for every $r \in \mathbb{N}$. We begin our inductive definition by setting $\mathbf{x}[0 \dots 2] = \mathbf{y}[0 \dots 2]$. Suppose we have defined $\mathbf{x}$ up to h_{j-1}. We then set

$$\mathbf{x}[r] = \mathbf{y}[r], \text{ for all } h_{j-1} < r \leq sh_j.$$

To specify the next $h_j - sh_j$ bits of $\mathbf{x}$, we use the following lemma, which we will prove in the next section.

▶ **Lemma 15.** *For every sufficiently large j, there is a binary string $\mathbf{z}$ of length $h_j - sh_j$ such that*

$$\min_{sh_j < r \leq h_j} |K_r(x, ax+b) - (1+s)r| < \frac{r}{j},$$

where x is any real such that $\mathbf{xz} \prec x$. Moreover, $\mathbf{z}$ is of the form (*) of Section 5.1.

For now, we assume the truth of this lemma. If the current j is not sufficiently large, take $\mathbf{z}$ to be the string of all zeros. Otherwise, if j is sufficiently large, we let $\mathbf{z}$ be such a binary string. We then set

$$\mathbf{x}[r] = \mathbf{z}[r - sh_j], \text{ for all } sh_j < r \leq h_j,$$

completing the inductive step. Finally, we let $x_{a,b,s}$ be the real number with binary expansion $\mathbf{x}$.

▶ **Proposition 16.** *Let $x = x_{a,b,s}$ be the real we just constructed. Then for every j,*
1. $K^{a,b}_{sh_j}(x) \geq sh_j - O(\log h_j)$, *and*
2. $K_r(x \mid a, b) \geq sh_j + r - h_j$, *for every $h_j \leq r < sh_{j+1}$.*

We now show, again assuming Lemma 15, that $\dim(x, ax+b) = 1+s$, where $x = x_{a,b,s}$ is the point we have just constructed.

We begin by proving an upper bound on $\dim(x, ax+b)$. Note that this essentially follows from our choice of $\mathbf{z}$.

▶ **Proposition 17.** *Let (a, b) be a slope intercept pair, $s \in (0,1)$ and $\gamma \in \mathbb{Q}$ be positive. Let $x = x_{a,b,s}$ be the point we have just constructed. Then*

$$\dim(x, ax+b) \leq (1+s) + \gamma.$$

Proof. Let j be sufficiently large. By our construction of x,

$$\min_{sh_j < r \leq h_j} |K_r(x, ax+b) - (1+s)r| < \frac{\gamma r}{4} \tag{13}$$

Therefore,

$$\begin{aligned}
\dim(x, ax+b) &= \liminf_r \frac{K_r(x, ax+b)}{r} \\
&\leq \liminf_j \min_{sh_j < r \leq h_j} \frac{K_r(x, ax+b)}{r} \\
&\leq \liminf_j \min_{sh_j < r \leq h_j} \frac{(1+s)r + \gamma r/4}{r} \\
&= \liminf_j \min_{sh_j < r \leq h_j} 1 + s + \gamma/4 \\
&= 1 + s + \frac{\gamma}{4}.
\end{aligned}$$

◀

We break the proof of the lower bound on $\dim(x, ax+b)$ into two parts. In the first, we give lower bounds on $K_r(x, ax+b)$ on the interval $r \in (sh_j, h_j]$. Note that this essentially follows from inequality (13).

▶ **Proposition 18.** *Let (a,b) be a slope intercept pair, $s \in (0,1)$, $\gamma \in \mathbb{Q}$ be positive and j be sufficiently large. Let $x = x_{a,b,s}$ be the point we have just constructed. Then*

$$K_r(x, ax+b) \geq (1+s-\gamma)r$$

for all $sh_j < r \leq h_j$

We now give lower bounds on $K_r(x, ax+b)$ on the interval $r \in (h_{j-1}, sh_j]$. The proof of this lemma is very similar to the analogous lemma for low-dimensional lines (Lemma 13). Intuitively, the proof is as follows. Using the previous lemma, we can, given a 2^{-h_j}-approximation of $(x, ax+b)$, compute a 2^{-h_j}-approximation of (a,b) *with a small amount of extra bits*. Having done so, we have to compute the last $r-h_j$ bits of (a,b). Importantly, since $r > h_j$, the last $r-h_j$ bits of x are maximal. Thus, we can simply lower the complexity of the last $r-h_j$ bits of (a,b) so that the complexity of these bits is roughly $s(r-h_j)$. Thus, we are again, morally, in the case where $\dim(x) \geq \dim(a,b)$ and the techniques of Section 3.1 work.

▶ **Lemma 19.** *Let (a,b) be a slope intercept pair, $s \in (0,1)$, $\gamma \in \mathbb{Q}$ be positive and j be sufficiently large. Let $x = x_{a,b,s}$ be the point we have just constructed. Then*

$$K_r(x, ax+b) \geq (1+s-\gamma)r$$

for all $h_{j-1} < r \leq sh_j$

Proof. Intuitively, we will use the approximation of $(x, ax+b)$ at precision h_{j-1} to compute (a,b) at precision h_{j-1}. Then we will only search for candidate lines within $2^{-h_{j-1}}$ of (a,b). Formally, the argument proceeds as follows.

We first show that we can compute (a,b) to within $2^{-h_{j-1}}$ with an approximation of $(x, ax+b)$, with few additional bits of information. By Lemma 2 and inequality (13)

$$\begin{aligned}
K_{h_{j-1},r}(a,b,x \mid x, ax+b) &\leq K_{h_{j-1},h_{j-1}}(a,b,x \mid x, ax+b) + O(\log h_j)) \\
&= K_{h_{j-1}}(a,b,x) - K_{h_{j-1}}(x, ax+b) && \text{[Lemma 2]} \\
&\leq K_{h_{j-1}}(a,b,x) - (1+s)h_{j-1} + \frac{\gamma}{4}h_{j-1} && \text{[(13)]} \\
&= K_{h_{j-1}}(a,b) + K_{h_{j-1}}(x \mid a,b) \\
&\qquad - (1+s)h_{j-1} + \frac{\gamma}{4}h_{j-1} && \text{[Lemma 2]} \\
&\leq dh_{j-1} + h_j 2^{-j} + K_{h_{j-1}}(x \mid a,b) \\
&\qquad - (1+s)h_{j-1} + \frac{\gamma h_{j-1}}{4} && \text{[Definition } h_j\text{]} \\
&\leq dh_{j-1} + h_j 2^{-j} + sh_{j-1} \\
&\qquad - (1+s)h_{j-1} + \frac{\gamma h_{j-1}}{4} && \text{[Proposition 16]} \\
&\leq dh_j + sh_{j-1} \\
&\qquad - (1+s)h_{j-1} + \frac{\gamma h_{j-1}}{2} && [j \text{ large}] \\
&\leq dh_j - h_j + \frac{\gamma h_{j-1}}{2}. && (14)
\end{aligned}$$

Thus, we can, given a 2^{-r} approximation of $(x, ax+b)$, compute a $2^{-h_{j-1}}$-approximation of (a,b) with

$$(d-1)h_j + \frac{\gamma h_{j-1}}{2}$$

additional bits of information. Knowing (a,b) to precision h_{j-1} allows us to search for candidate lines within $2^{-h_{j-1}}$ of (a,b), i.e., using Lemma 6 with $m = h_{j-1}$.

Let $\hat{s} \in \mathbb{Q} \cap (0,s)$ be a dyadic rational such that

$$\gamma/8 < s - \hat{s} < \gamma/4.$$

Let $\hat{d} \in \mathbb{Q} \cap (0, \dim(a,b))$ be a dyadic rational such that

$$\gamma/8 < \dim(a,b) - \hat{d} < \gamma/4.$$

Define

$$\alpha = \frac{s(r-h_{j-1}) + dh_{j-1}}{r}.$$

Define

$$\eta = \frac{\hat{s}(r-h_{j-1}) + \hat{d}h_{j-1}}{r}.$$

Finally, let $\varepsilon = \gamma^2/64$. Note that

$$\begin{aligned}
\alpha - \eta &= \frac{s(r-h_{j-1}) + dh_{j-1} - \hat{s}(r-h_{j-1}) - \hat{d}h_{j-1}}{r} \\
&= \frac{(s-\hat{s})(r-h_{j-1}) + (d-\hat{d})h_{j-1}}{r} \\
&\leq \frac{\frac{\gamma}{4}(r-h_{j-1}) + \frac{\gamma}{4}h_{j-1}}{r} \\
&= \frac{\gamma}{4} \qquad (15)
\end{aligned}$$

Similarly,

$$\begin{aligned}
\alpha - \eta &= \frac{s(r-h_{j-1}) + dh_{j-1} - \hat{s}(r-h_{j-1}) - \hat{d}h_{j-1}}{r} \\
&= \frac{(s-\hat{s})(r-h_{j-1}) + (d-\hat{d})h_{j-1}}{r} \\
&> \frac{\frac{\gamma}{8}(r-h_{j-1}) + \frac{\gamma}{4}h_{j-1}}{r} \\
&= \frac{\gamma}{8} \qquad (16)
\end{aligned}$$

In particular,

$$\frac{4\varepsilon}{\alpha - \eta} \leq \gamma/4. \qquad (17)$$

We also note that

$$K(\varepsilon, \eta) \leq K(\gamma, \hat{s}, \hat{d}, r, h_{j-1}) \leq O(\log r), \qquad (18)$$

since j was chosen to be sufficiently large and γ is constant.

Let $D = D(r, (a, b), \eta)$ be the oracle of Lemma 8. We now show that the conditions of Lemma 6 are satisfied for these choices $a, b, \eta, \varepsilon, r$ and $\delta = \alpha - \eta$, $m = h_{j-1}$ and $A = D$.

Let (u, v) be a line such that $t := \|(a, b) - (u, v)\| \geq h_{j-1}$, and $ux + v = ax + b$. Then, by Lemmas 7, 8, and Proposition 16,

$$\begin{aligned}
K_r^D(u, v) &\geq K_t^D(a, b) + K_{r-t,r}^D(x \,|\, a, b) - O(\log r) && \text{[Lemma 7]} \\
&\geq K_t^D(a, b) + K_{r-t,r}(x \,|\, a, b) - O(\log r) && \text{[Lemma 8]} \\
&\geq K_t^D(a, b) + s(r - t) - O(\log r). && \text{[Proposition 16]}
\end{aligned}$$

By Lemma 8, there are two cases. In the first, $K_t^D(a, b) = \eta r$, and so

$$\begin{aligned}
K_r^D(u, v) &\geq K_t^D(a, b) + s(r - t) - O(\log r) \\
&= \eta r + s(r - t) - O(\log r) \\
&\geq (\eta - \varepsilon)r + s(r - t) && [j \text{ is large}] \\
&\geq (\eta - \varepsilon)r + \delta(r - t) && [\gamma \text{ is small}]
\end{aligned}$$

In the second case, $K_t^D(a, b) = K_t(a, b)$, and so

$$\begin{aligned}
K_r^D(u, v) &\geq K_t^D(a, b) + s(r - t) - O(\log r) \\
&\geq dt - o(t) + s(r - t) - O(\log r) && \text{[Definition of dim]} \\
&= dh_{j-1} + d(t - h_{j-1}) + s(r - t) - o(r) \\
&= dh_{j-1} + d(t - h_{j-1}) + s(r - h_{j-1}) - s(t - h_{j-1}) - o(r) \\
&= \alpha r + d(t - h_{j-1}) - s(t - h_{j-1}) - o(r) && \text{[Definition of } \alpha] \\
&= \eta r + (\alpha - \eta)r + (d - s)(t - h_{j-1}) - o(r) \\
&\geq \eta r + (\alpha - \eta)r - o(r) && [d > 1,\, t > h_{j-1}] \\
&\geq \eta r + (\alpha - \eta)(r - t) - o(r) && [\alpha > \eta] \\
&\geq (\eta - \varepsilon)r + \delta(r - t) && [j \text{ is large}]
\end{aligned} \tag{19}$$

Therefore, in either case, we may apply Lemma 6, relative to D which yields

$$\begin{aligned}
K_r^D(a, b, x) &\leq K_r(x, ax + b) + K_{h_j,r}(a, b, x \,|\, x, ax + b) \\
&\quad + \frac{4\varepsilon}{\alpha - \eta} r + K(\varepsilon, \eta) + O(\log r) && \text{[Lemma 6]} \\
&\leq K_r(x, ax + b) + dh_j - h_j + \frac{\gamma h_{j-1}}{2} \\
&\quad + \frac{4\varepsilon}{\alpha - \eta} r + K(\varepsilon, \eta) + O(\log r) && [(14)] \\
&\leq K_r(x, ax + b) + dh_j - h_j + \frac{\gamma h_{j-1}}{2} \\
&\quad + \frac{4\varepsilon}{\alpha - \eta} r + O(\log r) && [(18)] \\
&\leq K_r(x, ax + b) + dh_j - h_j + \frac{\gamma h_{j-1}}{2} \\
&\quad + \frac{\gamma r}{4} + O(\log r) && [(17)] \\
&\leq K_r(x, ax + b) + dh_j - h_j + \frac{3\gamma r}{4} + O(\log r) && (20)
\end{aligned}$$

By Lemma 11, and our construction of oracle D,

$$\begin{aligned}
K^D_r(a,b,x) &= K^D_r(a,b) + K^D_r(x \,|\, a,b) - O(\log r) && \text{[Lemma 2]}\\
&= \eta r + K_r(x \,|\, a,b) - O(\log r) && \text{[Lemma 8]}\\
&\geq \eta r + sh_j + r - h_j - O(\log r) && \text{[Lemma 11(2)]}\\
&\geq \alpha r - \frac{\gamma}{4} r + sh_j + r - h_j - O(\log r)\\
&\geq s(r-h_j) + dh_j - \frac{\gamma}{4} r + sh_j + r - h_j - O(\log r)\\
&\geq (1+s)r - (1-d)h_j - \frac{\gamma}{4} r. && (21)
\end{aligned}$$

Rearranging (20) and combining this with (21), we see that

$$\begin{aligned}
K_r(x, ax+b) &\geq K^D_r(a,b,x) - dh_j + h_j - \frac{3\gamma r}{4} - O(\log r) && \text{[(20)]}\\
&\geq (1+s)r - (1-d)h_j - \frac{\gamma}{4} r\\
&\qquad - dh_j + h_j - \frac{3\gamma r}{4} - O(\log r) && \text{[(21)]}\\
&= (1+s)r - \gamma r - O(\log r) && \blacktriangleleft
\end{aligned}$$

We are now able to prove that the Dimension Spectrum Conjecture holds for high dimensional lines.

Proof of Theorem 14. Let $(a,b) \in \mathbb{R}^2$ be a slope-intercept pair with

$$d = \dim(a,b) > 1.$$

Let $s \in [0,1]$. In the case where $s = 0$, Turetsky showed (Theorem 1) that $1 \in \mathrm{sp}(L_{a,b})$, i.e., there is a point x such that $\dim(x, ax+b) = 1$. In the case where $s = 1$, Lutz and Stull [11] showed than any point x which is random relative to (a,b) satisfies

$$\dim(x, ax+b) = 2.$$

Therefore, we may assume that $s \in (0,1)$. Let $x = x_{a,b,s}$ be the point constructed in this section. By Propositions 17 and 18 and Lemma 19, for every γ,

$$|\dim(x, ax+b) - (1+s)| < \gamma.$$

Thus, by the definition of effective dimension,

$$\dim(x, ax+b) = 1 + s,$$

and the proof is complete. ◀

5.3 Proof Sketch of Lemma 15

To complete the proof of the main theorem of this section, we now prove Lemma 15. Recall that this states that, for every j, after setting $\mathbf{x}[h_{j-1} \ldots sh_j] = \mathbf{y}[h_{j-1} \ldots sh_j]$, the following holds.

▶ **Lemma 15.** *For every sufficiently large j there is a binary string $\mathbf{z}$ of length $h_j - sh_j$ such that*

$$\min_{sh_j < r \leq h_j} |K_r(x, ax+b) - (1+s)r| < \frac{r}{j},$$

where x is any real such that $\mathbf{xz} \prec x$. Moreover, $\mathbf{z}$ is of the form (*) of Section 5.1.

Let $m = h_j - sh_j$. For each $0 \leq n \leq m$, define the binary string $\mathbf{x_n}$ of length m by

$$\mathbf{x_n}[i] = \begin{cases} 0 & \text{if } 0 \leq i < m - n \\ \frac{1}{a}[i - (m-n)] & \text{if } m - n \leq i \leq m \end{cases}$$

Thus, for example $\mathbf{x_0}$ is the binary string of m zeros, while $\mathbf{x_m}$ is the binary string containing the m-bit prefix of $\frac{1}{a}$.

Let x be the real number such that $\mathbf{x}\mathbf{x_0} \prec x$, and whose binary expansion contains only zeros after sh_j. For each $1 \leq n \leq m$, let x_n be the real number defined by

$$x_n = x + 2^{-h_j+n}/a.$$

Therefore, for every n,

$$(x_n, ax_n + b) = (x_n, ax + b + 2^{-h_j+n}).$$

Since the binary expansion of x satisfies $x[r] = 0$ for all $r \geq sh_j$, we have, for every n,

$$\mathbf{x}\mathbf{x_n} \prec x_n \tag{22}$$

In other words, the binary expansion of x_n up to index h_j is just the concatenation of $\mathbf{x}$ and $\mathbf{x_n}$.

We now collect a few facts about our points x_n.

▶ **Lemma 20.** *For every n, r such that $0 \leq n \leq m$ and $sh_j \leq r \leq h_j$ the following hold.*

1. $K_{n,h_j}(a \,|\, x_n) \leq O(\log h_j)$.
2. *For every n and $n' > n$,*

$$|K_r(x_{n'}, ax_{n'} + b) - K_r(x_n, ax_n + b)| < n' - n + \log(r).$$

3. $K_{r-sh_j,r}(a, b \mid x_m, ax_m + b) \leq O(\log r)$.

Note that the constants implied by the big oh notation depend only on a.

6 Conclusion and Future Directions

The behavior of the effective dimension of points on a line is not only interesting from the algorithmic randomness viewpoint, but also because of its deep connections to geometric measure theory. There are many avenues for future research in this area.

The results of this paper show that, for any line $L_{a,b}$, the dimension spectrum $\mathrm{sp}(L_{a,b})$ contains a unit interval. However, this is not, in general, a tight bound. It would be very interesting to have a more thorough understanding of the "low end" of the dimension spectrum. Stull [17] showed that the Hausdorff dimension of points x such that

$$\dim(x, ax + b) \leq \alpha + \frac{\dim(a, b)}{2}$$

is at most α. Further investigation of the low-end of the spectrum is needed.

It seems plausible that, for certain lines, the dimension spectrum contains an interval of length greater than one. For example, are there lines in the plane such that $\mathrm{sp}(L)$ contains an interval of length strictly greater than 1?

Another interesting direction is to study the dimension spectrum of particular classes of lines. One natural class is the lines $L_{a,b}$ whose slope and intercept are both in the Cantor set. By restricting the lines to the Cantor set, or, more generally, self-similar fractals, might give enough structure to prove tight bounds not possible in the general case.

Additionally, the focus has been on the effective (Hausdorff) dimension of points. Very little is known about the effective *strong* dimension of points on a line. The known techniques do not seem to apply to this question. New ideas are needed to understand the strong dimension spectrum of planar lines.

Finally, it would be interesting to broaden this direction by considering the dimension spectra of other geometric objects. For example, can anything be said about the dimension spectrum of a polynomial?

References

1 Randall Dougherty, Jack Lutz, R Daniel Mauldin, and Jason Teutsch. Translating the Cantor set by a random real. *Transactions of the American Mathematical Society*, 366(6):3027–3041, 2014.

2 Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag, 2010.

3 Xiaoyang Gu, Jack H Lutz, Elvira Mayordomo, and Philippe Moser. Dimension spectra of random subfractals of self-similar fractals. *Annals of Pure and Applied Logic*, 165(11):1707–1726, 2014.

4 Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008.

5 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003.

6 Jack H. Lutz. The dimensions of individual strings and sequences. *Inf. Comput.*, 187(1):49–79, 2003.

7 Jack H. Lutz and Neil Lutz. Algorithmic information, plane Kakeya sets, and conditional dimension. *ACM Trans. Comput. Theory*, 10(2):Art. 7, 22, 2018. doi:10.1145/3201783.

8 Jack H Lutz and Neil Lutz. Who asked us? how the theory of computing answers questions about analysis. In *Complexity and Approximation*, pages 48–56. Springer, 2020.

9 Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM J. Comput.*, 38(3):1080–1112, 2008.

10 Neil Lutz. Fractal intersections and products via algorithmic dimension. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, 2017.

11 Neil Lutz and D. M. Stull. Bounding the dimension of points on a line. In *Theory and applications of models of computation*, volume 10185 of *Lecture Notes in Comput. Sci.*, pages 425–439. Springer, Cham, 2017.

12 Neil Lutz and D. M. Stull. Dimension spectra of lines. In *Unveiling dynamics and complexity*, volume 10307 of *Lecture Notes in Comput. Sci.*, pages 304–314. Springer, Cham, 2017.

13 Neil Lutz and D. M. Stull. Projection theorems using effective dimension. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, 2018.

14 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inf. Process. Lett.*, 84(1):1–3, 2002.

15 Elvira Mayordomo. Effective fractal dimension in algorithmic information theory. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 259–285. Springer New York, 2008.

16 Andre Nies. *Computability and Randomness*. Oxford University Press, Inc., New York, NY, USA, 2009.

17 D. M. Stull. Results on the dimension spectra of planar lines. In *43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 79, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.

18 Daniel Turetsky. Connectedness properties of dimension level sets. *Theor. Comput. Sci.*, 412(29):3598–3603, 2011.